System-level feature-based modeling of cyber-physical systems

A theoretical framework and methodological fundamentals

Pourtalebi, Shahab

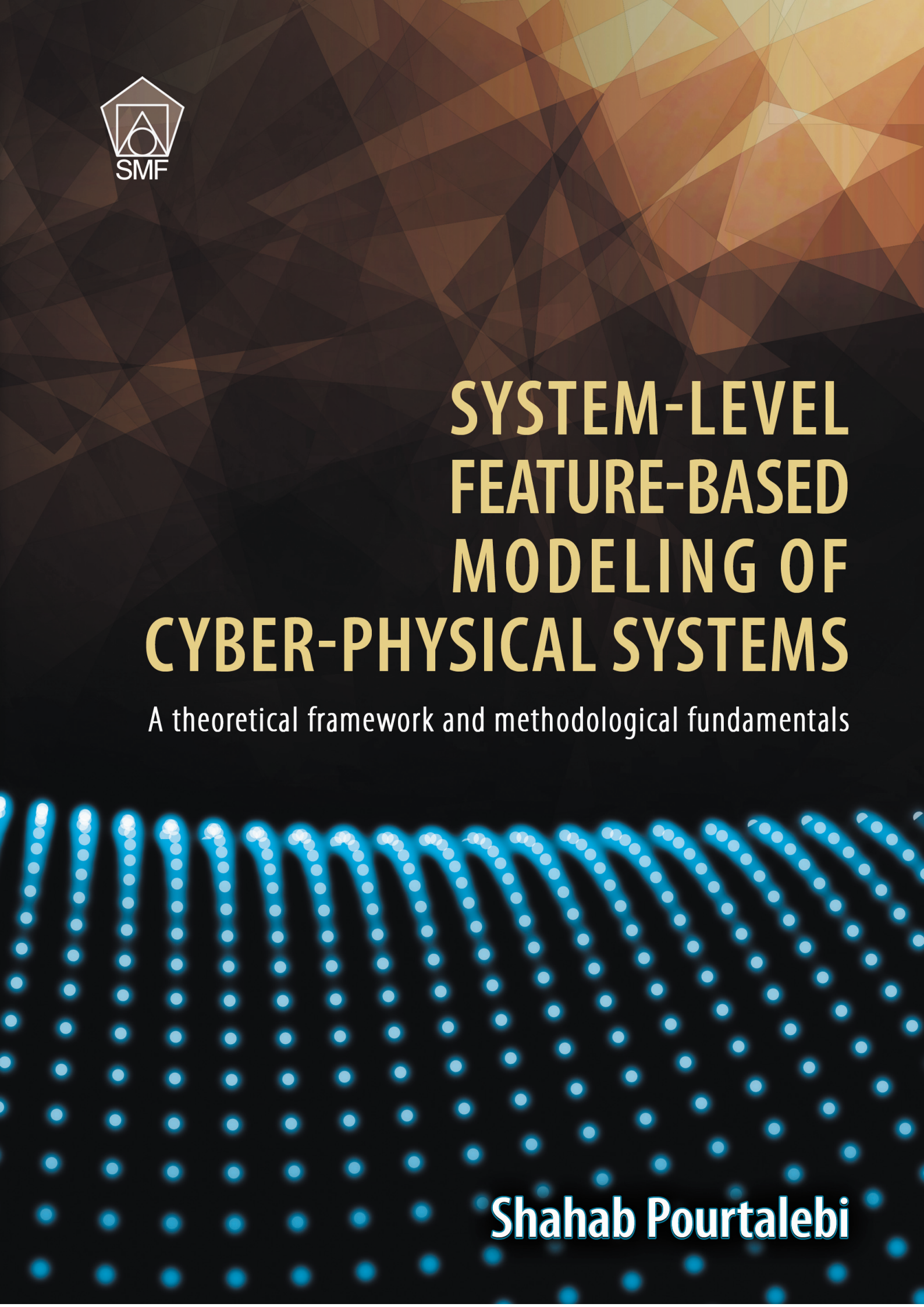
DOI
10.4233/uuid:a75df963-aede-44a7-9216-274cdc7c4278

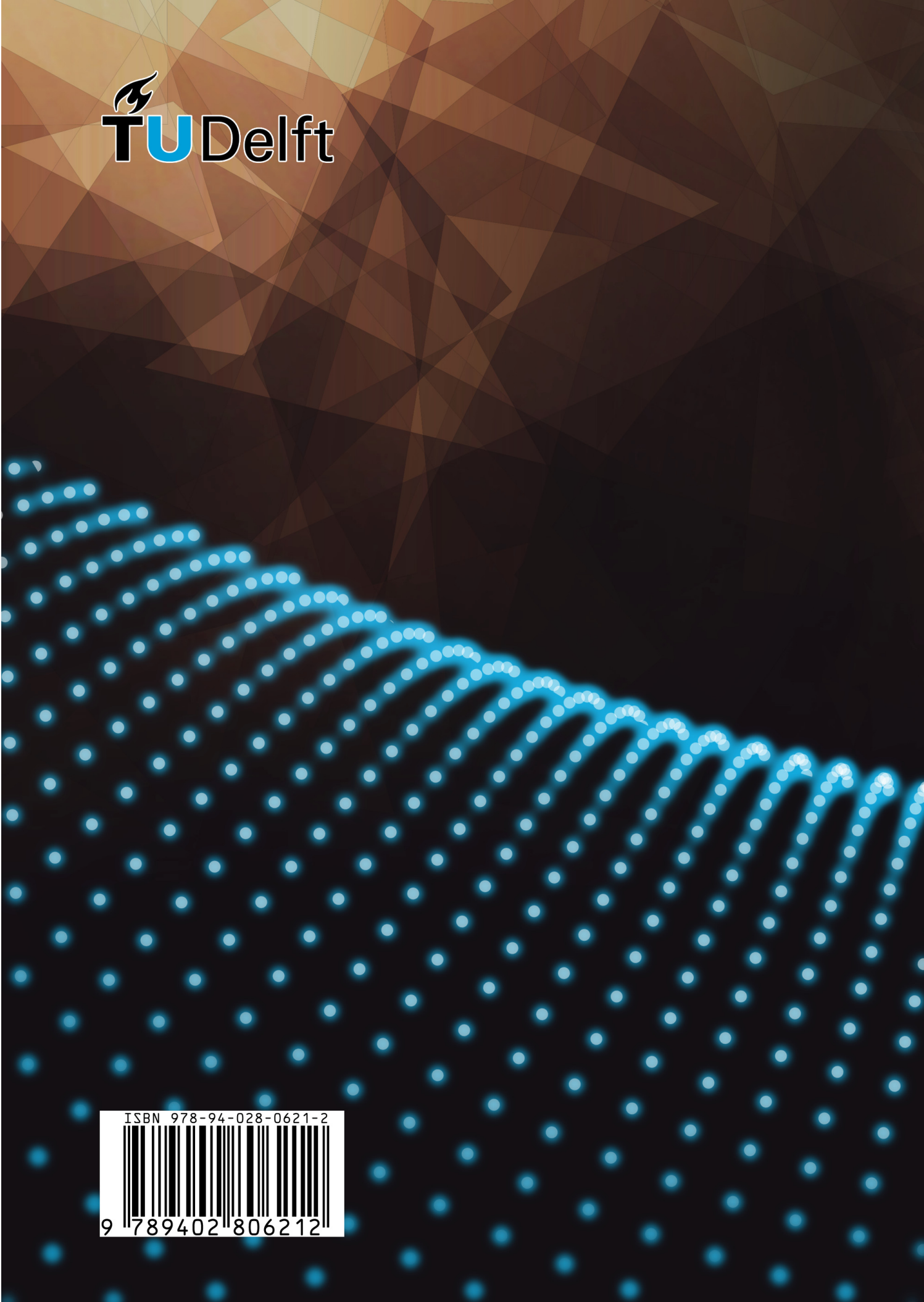
Publication date
2017

Document Version
Final published version

Citation (APA)
Pourtalebi, S. (2017). *System-level feature-based modeling of cyber-physical systems: A theoretical framework and methodological fundamentals*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:a75df963-aede-44a7-9216-274cdc7c4278

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.


SMF
SYSTEM-LEVEL FEATURE-BASED MODELING OF CYBER-PHYSICAL SYSTEMS
A theoretical framework and methodological fundamentals
Shahab Pourtalebi



ISBN 978-94-028-0621-2

System-level feature-based modeling of cyber-physical systems

A theoretical framework and methodological fundamentals

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof.ir. K.C.A.M. Luyben;
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op Woensdag 26 April 2017 om 10:00 uur

Door

Shahab POURTALEBI

Master of Science in Industrial Design
geboren te Rasht, Iran.

This dissertation has been approved by the
promotor: Prof. dr. I. Horváth

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. I. Horváth,	Technische Universiteit Delft, promotor

Independent members:

Prof. dr. J-P Pernot	Arts et Métiers ParisTech
Prof. dr. I.S. Sariyildiz	Technische Universiteit Delft
Prof. dr. F.J.A.M. van Houten	University of Twente
Prof. dr. G.W. Kortuem	Technische Universiteit Delft
Dr. S. Borgo	Consiglio Nazionale delle Ricerche
Dr. F. Derakhshan	University of Tabriz
Prof. dr. A.R. Balkenende	Technische Universiteit Delft

TU Delft Delft University of Technology

System-level feature-based modeling of cyber-physical systems
A theoretical framework and methodological fundamentals

Keywords: Cyber-physical system; system manifestation features; system-level features; system-level modeling; pre-embodiment design; mass customization;

ISBN/EAN: 978-94-028-0621-2
An electronic version of this dissertation is available at
http://repository.tudelft.nl/

shahab60p@gmail.com

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1 INTRODUCTION

1.1 PROLIFERATION OF CYBER-PHYSICAL COMPUTING

There are several interesting trends in the field of product development and distribution. The traditional consumer products (so-called consumer durables) are gradually disappearing and their place is taken over by new kinds of products and/or artifact-service combinations [1]. Typically the new products are digitally connected and offer some level of smart operation. Their implementation is facilitated by the current rapid technological advancement, which includes novel functional materials, energy sources, and information processing. In the field of technologies, we can observe a process of convergence, which entails that many of the new products are exploiting the same core technologies and show resemblance in terms of their functions. The convergence of digital hardware, software and cyberware technologies is a strong trend in our days [2]. In addition, a novel concept and approach of digital computing emerged a decade ago, which has been named as cyber-physical computing (CPC). This also has a strong effect on the implementation of practically all software integrated and controlled consumer durables.

As a successor of the concepts of ubiquitous and pervasive computing, the fundamental premise CPC offers is that digital computing will be much more integrated with the physical and social environment. This integration was described also as penetration into real life processes and having the control algorithms of computation defined in run-time, dominantly by systems which are equipped with the necessary networking, reasoning, and decision making capabilities [3]. The governing vision is cyber-physical computing-based systems will interact with humans, physical systems, and the environment in time, space, energy, tasks, and performance dimensions. Though CPC paves the way to decentralization and automation in decision making, humans will still play an important role as part of the overall computing system. CPC also entails the need for completely new theories, modeling approaches and tools, design principles, development paradigms, and run-time support in order to be able to take all the challenges successfully. However, the most intrinsic challenges originate in technological issues such as distributed sensing, system control, adaptive reasoning, and timed actuations, but in privacy, security, information distillation, robustness, system troubleshooting, energy provisioning, and sustainability, among others [4].

CPC-based systems gather data from the physical world through various sensors, process the run-time obtained data in quasi-real time, develop strategic plans for optimal operation and adaptation, and provide results that control objects, subjects and other systems in the physical world through sophisticates hardware and software actuators [5]. Exploitation of the new opportunities offered by CPC have been set as an objective in many European and foreign research initiatives, such as the Horizon 2020 Framework Program for Research and Innovation of the European Commission, the COSME 2014-2020 program for the competitiveness of enterprises and SMEs, and the Strategic Innovation Agenda for the European Institute of Innovation and Technology, just to name a few. These programs focus on cyber-physical systems, which represent a new paradigm in product and service

development. The underpinning paradigm of cyber-physical system (CPS) essentially reintegrates different concepts and paradigms such as "embedded system" [6] [7], "smart ubiquitous computing" [8] [9], and "the Internet of things" [10]. Nevertheless, it also introduces brand new concepts of system architecting and novel forms of system operation and implementation. Essentially, the paradigm of CPS is seen by many as the result of the current technological progress and physical miniaturization in the fields of mechatronics and electronics, and the evolution of network systems, embedded software and systems, wireless networked sensors and actuators, Internet of (every)things, large-scale information processing, and semantic knowledge engineering in the cyber domain [11]. Being driven by cyber-physical computing, conceptualization of CPSs requires stepping beyond the boundaries of the current system design and architecting arena.

1.2 CYBER-PHYSICAL SYSTEMS AND THEIR MAIN CHARACTERISTICS

As indicated above, CPSs are not just a combination of separate physical and computing components, but systems that intends to achieve a rather tight coupling among these two intrinsically different constituents in an interconnected concept [12] [13]. Some researchers identifies synergy as the main characteristics of CPSs [14]. There are however many more fundamental characteristics of CPSs that differentiate them from other engineered technical systems. This is an important issue since CPSs systems also manifest in a wide variety for forms, which often hide their genuine distinguishing characteristics. This may easily happen if we look at these systems at the level of their technical implementation. However, if we enforce an abstract view of the different manifestations of CPSs, we can identify and describe those characteristics that concern the whole of systems and/or their substantial constituents, rather than its specific components and elements. Through the abstract window, medical, military, transportation, aerospace, agricultural, production, home care, etc. cyber-physical systems may show similar system level characteristics. Obviously, the number of these abstract characteristics can be very large depending on the number of aspects the system is looked at. Since architectural and operational aspects are fundamental for any systems, in order to identify the main characteristics of CPSs, these two aspects are at least to be considered by model-based CPS development approaches.

From an architectural aspect, CPSs are branded by the following characteristics: (i) physically distributed and decentralized architecture, (ii) ad-hoc functional connections of heterogeneous components, (iii) high level of system complexity, (iv) fully open, extendable or adaptive system boundary, (v) hetarachically organized or runtime dynamically adapted internal architecture, (vi) high level of heterogeneity with regards to components, (vii) balance in terms of the amount of hardware, software and cyberware components and relations, and (viii) multi-level composition of scales. These characteristics are addressed by the principles of architecture driven development approaches, which facilitate development of large scale CPSs, use behavior as constraint for interfacing, and indicate the need for generation new components [15].

From an operational aspect, CPSs are characterized by: (i) services oriented overall operation, while also providing means and utilities, (ii) external functional adaptability and self-adapting/self-evolving capabilities, (iii) intense information exchange among the functional components, (iv) harmonized operating principles with regards to sensing and observing, execution and control, computing and reasoning, and activating and motivating, (v)

growing functional intelligence extending to context awareness and smart reasoning, (vi) smart control according to goal-driven operational scenarios, (vii) ability of a level of proactivity and autonomous operation, (viii) capability for unsupervised learning, and (ix) context sensitive interaction with human in the loop. Behavior driven CPS development approaches aim at implementing these characteristics, while address functionality development of individual systems and therefore inadequate for analyzing large scale systems (of systems). They typically use architecture as constraint for interfacing [16].

The impartial consideration of integration of digital cyber and analogue physical parts in a CPS clarify several opportunities that could be utilized for different purposes, and threats which can be an issue for research. The interesting point is the totally new situation resulting from the advent of CPSs that has never been experienced before in different aspects and perspectives of system considerations. The most important characteristics of CPSs that make them distinguishable are: (i) hybrid structure of components, (ii) real-time information processing capability, (iii) fixed or ad-hoc functional connection (or both) between components, (iv) different sources of providing knowledge including built-in, sensors-obtained, and reasoning based sources, and (v) ability of learning from history and situations in an unsupervised manner [17].

1.3 A NEW CHALLENGE: CROSS-BOUNDARY ARCHITECTING & OPERATION DESIGN

The typical high level of heterogeneity, complexity, and integration of CPSs introduces new design challenges in the field of system engineering. Challenges, issues, methods and principles of design and modeling CPSs have been discussed in different publications. For instance, as Grimm, C. and Ou, J. pointed at it, a particular challenge raised by designing cyber-physical systems at system level and as a whole is the formal and abstract specification and representation of whole of the system including both physical and cyber components, which is usually accompanied by the challenges of design space exploration, optimization, and verification[18]. Woo, H. et alias argued that resilience of mission-critical cyber-physical systems to all classes of failures, (emerging in hardware, software and/or cyber components) must be achieved by a top-down approach with respect to the designing a software architecture which embeds the implementations of the feedback control laws as well as the implementation and verification of the software for possible failures in the related sensors or actuators [19]. Bogdan, P. and Marculescu, R. emphasized the importance of modeling workload (i.e. the amount of measured and/or processed data per unit of time that is communicated between various CPS nodes and which affects not only various local parameters but also macroscopic metrics) and design optimization based on statistical properties of the workload and adopting a statistical physics approach for describing the interactions among the components [20].

In the context of advancing model-based design of CPSs, Mosterman, P.J. and Zande, J. proposed to describe the computation in an abstract manner and to consider modeling approximations of computational semantics [21]. To facilitate the concurrent but distributed development and integration of hardware, software and cyberware components, Raghav, G., & Gopalswamy, S. proposed to prefer non-functional analysis using architecture description, to use constraints to select combinations of the component variants, and to associate it with behavioral models, data, requirements and other properties [22]. Heterogeneity and complexity of this kind of systems have been identified as one of the

most challenging engineering issues in many scientific publications [23] Heterogeneity and complexity issues are especially critical in the pre-embodiment design phase, where system designers have to deal with both system compositionality and component composability questions. Usually the composability issues are explicitly addressed in conceptualization and architecting of CPSs, and subsequently, homogenous constituent design and development are delegated to the experts of the related fields. For example, development of the software constituents is delegated to software engineers, control engineers deal with control issues, mechanical engineers solve the hardware problems, and electronic experts design and develop circuits. Since they are working on a manageable and homogeneous part of CPS, they can tackle the design challenges by using their expertise. However, the compositionality issues of CPSs remain unsolved and challenging for system designers.

A large number of efforts have been made in research and system development to provide methods and techniques for addressing the concrete technical challenges. For instance, Rajhans, A. et alias suggested using parameters in architectural views to support heterogeneous design and verification [24]. Fallah, Y.P., and Sengupta, R. proposed to design systems in a top-down manner, by identifying system metrics such as approximate measures of awareness, and then characterizing component models and their interactions in the process of designing vehicle safety networks [25]. Zhang, J. et alias discussed the importance and application of modular verification of dynamically adaptive systems [26] Pinto, A., and Krishnamurthy, S. considered uncertainty as key characteristic of cyber-physical system and proposed analysis and design tools for specification and detailing these type of systems in an industrial setting [27] Zilic, Z., and Karajica, B. discussed various high-level design and synthesis challenges and proposed an approach with the emphasis on the numerical values passed between sensors and the rest of the system [28].

It must be noted that the above cited works are just examples of the wide ranging efforts, which have been made to address the numerous technical challenges of cross-boundary architecting and operational design of CPSs. The problem becomes more complicated when consideration of some kind of design intentions and/or optimization for multiple concerns are added to the problem. Design for X (e.g. customizability, maintainability, sustainability, interaction) traditionally has some sort of design principles and/or guidelines which are not performing well for CPSs due to the unique characteristics of CPSs. Therefore, designing CPSs in the pre-embodiment phase needs new principles, tools, techniques, or a kind of design supporting tool. The traditional 'reductionist' handling of the design and implementation concerns is not advantageous from multi-aspect optimization of complicated and complex systems.

A traditional way of addressing the problem of handling complexity and heterogeneity in pre-embodiment design phase is to apply abstraction and to simplify model of the system in a level which could be handled logically. Due to the abstraction, the complexity of the CPSs is reduced, and heterogeneous components can be represented as simple elements. In fact, the system designers are being pushed to formulate the abstract relationship among components. We refer to this formulation as logic-based modeling. The logic of the systems' functions could be modelled and simulated through the defined methods of operation (usually as mathematic equations) with available modeling and simulation tools (e.g. SysML, Simulink, Mathematica, LabVIEW). Many software tools and modeling languages have been developed recently for these purposes. These tools and languages are

able to capture required logic and relations among elements of a system to model and simulate it at a high level of abstraction to avoid dealing with complexity and heterogeneity. Multiple optimization algorithms have also been developed to assist designers in development of the system.

Another method to handle the heterogeneity of CPS in pre-embodiment design phase is to compose a team of designers with various fields of expertise in order to tackle composability challenges of architecting and conceptualization of CPSs. In this way, the experts of each field could explain to the other team members which issues should be considered during system design. Usually, the leader of such a team should be the one who has sufficient knowledge about all related knowledge and engineering fields.

1.4 DEFINITION OF THE RESEARCH PROBLEM

All available mentioned solutions suffer from the truth that intrinsic differences of hardware, software, and cyberware components negatively affect proper interdisciplinary communication. In addition, the terminology of each discipline differs from that of the others, which can cause misunderstanding and misinterpretation. The problems result in the scientific knowledge gaps are listed as:

- Current modeling tools and techniques do not sufficiently support CPS designers. Due to the necessary simplification imposed by the available tools, all technical problems are not noticed during system architecting and conceptualization.
- The abstraction used for simplification purposes eliminates physical matters (e.g. physical attributes, dimensions, constraints, cardinality) in favor of logical considerations. Ignoring physical considerations in system engineering cause serious problems afterwards.
- Interdisciplinary considerations remain unsolved both from technical and communicational point of view.
- Composability concerns remain unsolved in most system modeling tools. Solving the composability challenges in CPSs architecting requires considering architectural parameters, constraints, and values as well as the operational ones.

In the line of the currently widespread CAD/CAE systems, we considered an interactive design system, which facilitate system-level feature-based modeling on a goal driven and structured interaction of the system designer and the supporting system, rather than an automated (a.k.a. intelligent) design system. Behind this decision was the recognition that automation of the design process would have opened a new dimension for research (actually, would have pulled the research topic from the field of engineering system modeling systems to an artificial intelligence driven system development). Though the technology of machine learning-as-a-service (MLaaS), and deep learning and artificial intelligence-based creative problem solving methodologies are available, combination of these and applying them in SMFs-based system design would have meant an enormous challenge that cannot be addressed meaningfully in one single PhD research project. Therefore, the target system manifests itself in a tool box of multiple, procedurally and informationally interconnected modules of functional components. The modules support not only functional and logical design of the system, but also its structural and morphological design.

We had two alternative ways to choose in order to address the mentioned problems. The easiest way would be to improve the current modeling tools in order to adapt them to the mentioned problems. The more difficult way was to study the main challenges and to try building a novel theoretical framework for CPS modeling that considers physical matters besides other considerations such as complexity and heterogeneity. Being aware of the risks and considering all difficulties, we have chosen the second way with the hope that the outcomes would open a new door in system modeling arena.

Accordingly, we targeted a theoretical framework and methodological fundamentals for imposing purely physical view in modeling CPSs, named as system-level feature-based modeling framework (SFMF). For developing SFMF, we needed underpinning theories to support it. These theories should be made based on the knowledge provided by technical and detailed exploring the knowledge gap, and the investigation of causalities. The theoretical works resulted in introducing computational constructs that are needed for our modelling framework. The computational constructs provide sufficient support for implementation of the introduced modelling framework. The steps towards this end are generally listed as understanding the knowledge gap, exploring causalities, constructing theories to fill the knowledge gap, introducing the concept of modelling framework, proposing computational constructs for implementation of the framework, and finally benchmarking by comparing the results with other available modeling tools in the framework level.

1.5 THE OBJECTIVES AND KEY RESEARCH QUESTIONS

The objective of this research has been formulated so as: "to develop a supporting tool for designers of heterogeneous-complex-systems in order to model these systems in pre-embodiment design phase." Accordingly, the key research questions are listed as:

RQ1. What is the state of the art and the knowledge gaps in designing and development of CPSs in the pre-embodiment design phase?
RQ2. Why do the conventional design methods not sufficiently support CPS designers to tackle composability issues?
RQ3. How to consider all required interdisciplinary and multicultural aspects of CPS design and development in pre-embodiment design phase?
RQ4. What kind of information should to be considered for uniformly modelling heterogeneous CPSs?
RQ5. What is the information structure of the modeling units?
RQ6. Which pieces of information should be captured by the knowledge frames in order to computationally model components and constituents of CPSs?
RQ7. How the pieces of information are related together in order to create modeling building blocks?
RQ8. How the pieces of information and the relations among them can be captured computationally?
RQ9. What is the computational procedure of creating modeling building blocks?
RQ10. What is the procedure of creating CPS model by composing modeling building blocks?
RQ11. What are the aspects of comparison of CPS modeling tools and languages?

RQ12. What are the advantages and disadvantages of the introduced modeling framework in pre-embodiment design of CPSs, in comparison with the current modeling frameworks?

Our research plan was designed to answer the mentioned key questions in five research cycles (RCs). For this reason, the key questions are grouped and assigned to research cycles. The first research cycle aimed to answer first two questions. The questions three and four are assigned to the second research cycle. Focus of the research cycle three was on the questions five to seven. The aim of research cycle four was to answer questions eight to ten. And finally, the questions eleven and twelve are answered in the fifth research cycle. Table 1-1 summarizes the relations of research questions to the chapters.

The objective of the first research cycle was to explore the state of the art about designing CPSs in the pre-embodiment design phase. The concerned studies were done in the following contexts: (i) overall characteristics of CPSs, (ii) relevant design approaches, (iii) deriving the required design principles, (iv) relevant design-assistive tools, and (v) advantages of feature technology.

Table 1-1 Mapping of the research key questions to the research cycles

	Chapter 2	Chapter 3	Chapter 4	Chapter 5	Chapter 6
RQ1	Research cycle 1				
RQ2					
RQ3		Research cycle 2			
RQ4					
RQ5			Research cycle 3		
RQ6					
RQ7					
RQ8				Research cycle 4	
RQ9					
RQ10					
RQ11					Research cycle 5
RQ12					

The objective of the second research cycle was to develop a theoretical framework for interdisciplinary and multicultural system-level modeling of CPSs in the pre-embodiment design phase. This theoretical framework should support uniform modeling of heterogeneous CPSs through specifying the kinds of information which should be captured. The theoretical work was supported by empirical studies.

The objective of the third research cycle was to provide computation methodological fundamentals and information constructs to underpin modeling of CPSs. This also investigated the concept and implemented system-level manifestation features. The information structures, which specify the required pieces of information for computational modeling of CPSs, as well as the relations among the pieces of information, were the main contexts of the studies performed in this research cycle.

In the fourth research cycle, the objective was twofold: (i) testing computational feasibility of the results of the theoretical works, and (ii) moving towards implementation of the introduced modeling entities and procedures through designing information schema constructs and computational processes for the proposed modeling tool. Computationally

capturing the required chunks of information and the relations among them, and the procedural considerations of a transparent modeling of CPSs, were the main working context of this research cycle.

The objective of the fifth research cycle was to benchmark the proposed conceptual modeling framework against the available tools and languages in the form of a comparative study in order to measure their appropriateness for pre-embodiment design of CPSs. The study considered novelties of the proposed solution against the characteristics of the current modeling tools and languages, according to the derived comparison aspects and indicators.

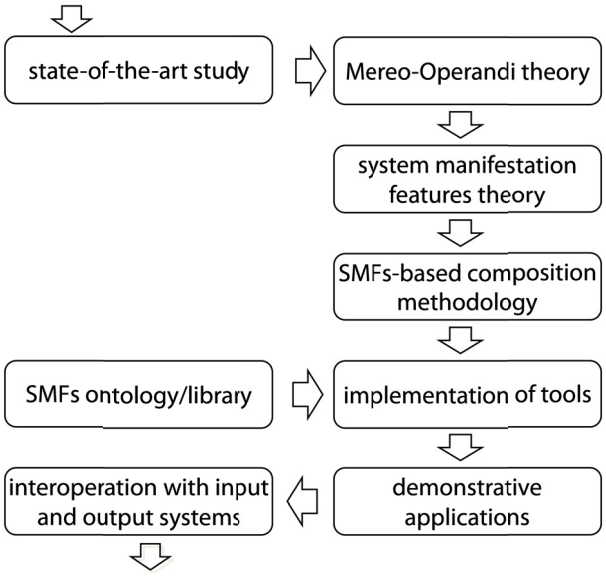


Figure 1-1 Process flow of the background research project

1.6 RESEARCH APPROACH

This research is part of a background research project, which proceeds according to the flow of activities shown in Figure 1-1. This promotion research, aiming to answer the abovementioned research questions, has been decomposed into five research cycles with two types of framing which have been introduced in [29] [30]. These include one cycles of research in design context (RIDC), followed by three cycles of design inclusive research (DIR), and a benchmarking study conducted based on RIDC. All research cycles consist of an exploration part and a confirmative part. The DIR frame also includes a constructive part for converting the available knowledge into concepts (and/or the development of prototypes). The implementation of the constructive part serves as research means in the inquiry process.

In the first research cycle, a multi-aspect reasoning model was developed based on the outcome of the forerunning literature study and knowledge synthesis. According to the developed reasoning model, five domains of study have been identified. The research activities are performed through focused literature study, as well as web-based searches and document analyses. The theory resulted from the explorative study provide sufficient detail about the knowledge gap. For some domains of study, the theory is confirmed through focus group discussions with experts.

The second research cycle concentrated on development of conceptual framework. The explorative part started with empirical studies in order to identifying requirements for constructing the conceptual framework. Studying available theories related to the requirements resulted in identifying the classic theories that could support our framework development. These theories underpinned development of the conceptual framework

named as mereo-operandi theory (MOT). The theoretical development has been tested on the cases provided in the forerunning empirical studies.

The third research cycle combined theoretical development of the information structure with demonstration in the chosen cases. In fact, the confirmation of the theory of system manifestation features (SMFs) was made through case studies and analyses. The SMFs theory is indeed an extension and complementary of MOT. Rational reasoning and literature review on system modeling theories helps to develop the theory. The constructive part of the research focuses on developing knowledge frames for modeling unites and the principles support them.

The fourth research cycle also consists of theoretical development part and constructive part. The concept of information schema construct and process of step-wise SMF creation are formulated as theoretical development part. The introduced theories are tested by constructing databases (DB) of genotype (GT) and phenotype (PT) as well as model DB. In addition procedures of SMF creation and model composition are considered as constructive part of this research cycle. The results of constructive part are used for confirmation based on critical reasoning and consistency analysis.

The fifth research cycle aimed to benchmarking of the developed modeling framework against the available modeling tools and languages. The study started with identifying the most proper modeling tools and languages as well as collecting information about their performing mechanism. Moreover, in parallel the aspects of comparison are derived based on the requirements of pre-embodiment modeling of CPSs. The comparison is performed by questionnaire answered by the experts involved in modeling CPSs. The results are analyzed and concluded as the confirmation of the whole research work.

1.7 THESIS OUTLINE

In the following paragraphs, the thesis outline is represented by explaining objective of each chapter, and the relations between the chapters. Figure 1-2 visually demonstrates the chapters as research cycles. It also briefly represents conduct of the research in each research cycle.

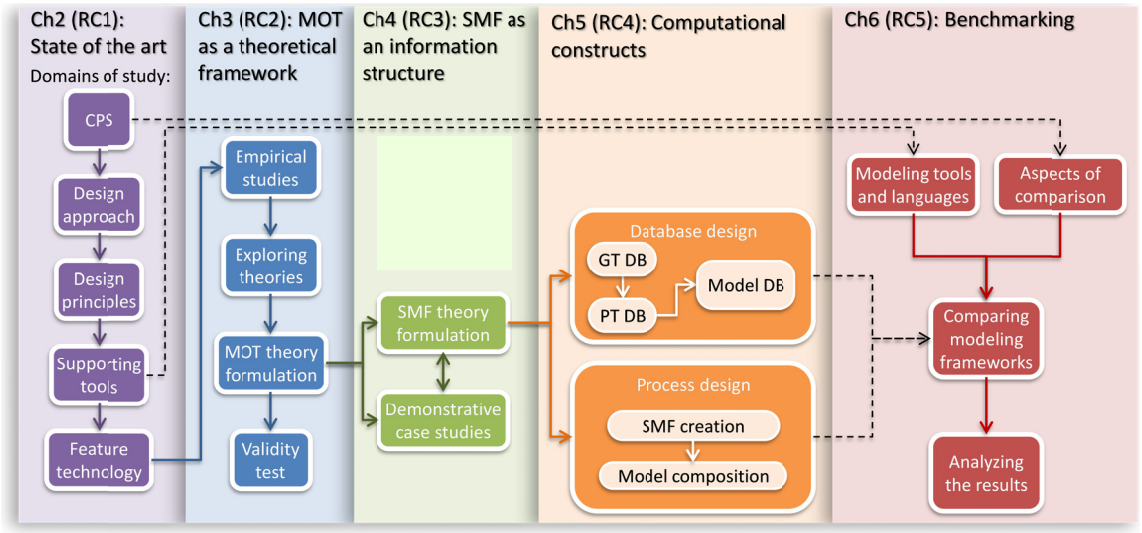


Figure 1-2 Outline of the chapters and conduct of the research

The main objective of Chapter 2 was to acquire fundamental insights in the studied general phenomenon and to provide a comprehensive and rigorous descriptive knowledge. Five domains of CPS, design approach, design principles, design supporting tools, and feature technology are studied for providing comprehensive insight about the knowledge gap from different perspectives. It is proved that the conventional design principles are not adequate for designing CPSs. The reasons of insufficiently of the current modelling tools for supporting pre-embodiment design of CPSs are also explored.

Chapter 3 aimed to propose a conceptual framework for pre-embodiment design of CPSs. It started by empirical studies. After identifying requirements of modeling heterogeneous system components, available theories that could support our conceptual framework have been explored. Accordingly, mereo-operandi theory is introduced for filling in the knowledge gap. This theory underpins the rest of our research by shifting the paradigm of modeling complex systems to system-level feature-based modeling. MOT introduces all fundamentals needed for uniformly formulating interdisciplinary and multicultural components of CPSs.

Theory of system manifestation features is introduced in the Chapter 4. As complementation and extension of MOT, SMFs theory aims to offer formal description for all required aspects of CPSs and their components and constituents in a uniform way. The information structure provided by SMF theory is represented in architectural and operational knowledge frames.

Chapter 5 explains how the information structure of SMFs could be implemented computationally. The aim was to propose computational constructs to support feasibility of the MOT and SMFs theories. Three databases of genotypes, phenotypes, and instantiated models have been designed that reflects the pieces of information and the relations among them as proposed by the SMFs theory. This chapter also introduces the procedures of SMF creation and model composition in connection with the entry form and warehouses.

The aim of Chapter 6 is to present the outcome of the validation of the results of the theoretical efforts through benchmarking. Five modelling tools were chosen to be compared with our proposed tool on framework level. The aspects of comparison were chosen according to the characteristics of CPSs and the requirements of pre-embodiment design. Comparing the modeling frameworks have been completed according to specific comparison aspects on the basis of the results of analyzing a set of indicators and the questionnaire answered by modeling experts.

Chapter 7 concludes the research by mentioning the main findings, research outcomes, and suggestions for follow up researches.

1.8 RELATED OWN PUBLICATIONS

To Chapter 2:

Pourtalebi, S., Horváth, I., & Opiyo, E. (2013). "Multi-aspect study of mass customization in the context of cyber-physical consumer durables", *In ASME 2013 International Design*

Engineering Technical Conferences and Computers and Information in Engineering Conference, Portland, Oregon, USA. pp. V004T05A006-V004T05A006.

Pourtalebi, S., Horváth, I., & Opiyo, E. Z. (2014). "New features imply new principles? Deriving design principles for mass customization of cyber-physical consumer durables", In *Proceedings of the 10th international symposium on tools and methods of competitive engineering TMCE 2014,* Budapest, Hungary. pp. 95-108.

To Chapter 3:

Pourtalebi, S., Horváth, I., & Opiyo, E. Z. (2014). "First steps towards a mereo-operandi theory for a system feature-based architecting of cyber-physical systems", *In Proceedings of GI-Jahrestagung,* ACDP 2014, Stuttgart, Germany. pp. 2001-2006.

Horváth, I., & Pourtalebi, S. (2015). "Fundamentals of a mereo-operandi theory to support transdisciplinary modeling and co-design of cyber-physical systems", *In Proceedings of the ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* Boston, Massachusetts, USA. pp. V01AT02A005-V01AT02A005.

To Chapter 4:

Pourtalebi, S., and Horváth, I., (2016), "Towards a methodology of system manifestation features-based pre-embodiment design", *Journal of Engineering Design*, Vol. 27 (4-6), pp. 232-268.

To Chapter 5:

Pourtalebi, S., and Horváth, I., (2016), "Information schema constructs for defining warehouse databases of genotypes and phenotypes of system manifestation features", *Frontiers of Information Technology & Electronic Engineering*, Vol. 17 (9), pp. 862-884.

Pourtalebi, S. and Horváth, I. (2016), "Procedure of creating system manifestation features: An information processing perspective", In *Proceedings of the 10th international symposium on tools and methods of competitive engineering TMCE 2016,* Aix-en-Provence, France. pp. 129-142.

Pourtalebi, S. and Horváth, I. (2016), "Information schema constructs for instantiation and composition of system manifestation features, *Frontiers of Information Technology & Electronic Engineering*, Vol. 17 (x), pp. xxx-xxx.

To Chapter 6:

Pourtalebi, S., & Horváth, I. (2016d). Benchmarking the conceptual framework of a system-level manifestation features-based toolbox, (in preparation for a journal publication), pp. x-x.

1.9 REFERENCES

[1] Horváth, I., Rusák, Z., Martin, E.H., Van Der Vegte, W.F., Kooijman, A., Opiyo, E.Z., and Peck, D., (2011), "An information technological specification of abstract prototyping for artifact and service combinations", Proceedings of the, Vol. 9, Washington, DC, pp. 209-223.

[2] Gantz, J., and Reinsel, D., (2012), "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east", IDC iView: IDC Analyze the future, Vol. 2007, pp. 1-16.

[3] Horváth, I., (2014), "What the Design Theory of Social-Cyber-Physical Systems Must Describe, Explain and Predict?", in: An Anthology of Theories and Models of Design, Springer, pp. 99-120.

[4] Khaitan, S.K., and McCalley, J.D., (2015), "Design techniques and applications of cyberphysical systems: A survey", IEEE Systems Journal, Vol. 9 (2), pp. 350-365.

[5] Horvath, I., (2012), "Beyond advanced mechatronics: new design challenges of social-cyber-physical systems", Proceedings of the Proceedings of the 1st Workshop on Mechatronic Design, Linz (Austria), 27-29 June, 2012, ACCM Austrian Center of Competence in Mechatronics.

[6] Edwards, S., Lavagno, L., Lee, E.A., and Sangiovanni-Vincentelli, A., (1997), "Design of embedded systems: formal models, validation, and synthesis", Proceedings of the IEEE, Vol. 85 (3), pp. 366-389.

[7] Wolf, W.H., (1994), "Hardware-software co-design of embedded systems", Proceedings of the IEEE, Vol. 82 (7), pp. 967-989.

[8] Prekop, P., and Burnett, M., (2003), "Activities, context and ubiquitous computing", Computer Communications, Vol. 26 (11), pp. 1168-1176.

[9] Want, R., Fishkin, K.P., Gujar, A., and Harrison, B.L., (1999), "Bridging physical and virtual worlds with electronic tags", Conference on Human Factors in Computing Systems - Proceedings, pp. 370-377.

[10] Atzori, L., Iera, A., and Morabito, G., (2010), "The Internet of Things: A survey", Computer Networks, Vol. 54 (15), pp. 2787-2805.

[11] Sha, L., Gopalakrishnan, S., Liu, X., and Wang, Q., (2008), "Cyber-physical systems: A new frontier", Proceedings of the, Taichung, pp. 1-9.

[12] Anonymous, (2006), "Cyber-Physical Systems", (NSF), official web site of National Science Foundation, 2006.

[13] Lee, E.A., (2008), "Cyber physical systems: Design challenges", Proceedings of the, Orlando, FL, pp. 363-369.

[14] Lee, E.A., and Seshia, S.A., (2011), Introduction to Embedded Systems: A Cyber-Physical Systems Approach, 1st ed., LeeSeshia.org.

[15] Doddapaneni, K., Ever, E., Gemikonakli, O., Malavolta, I., Mostarda, L., and Muccini, H., (2012), "A model-driven engineering framework for architecting and analysing wireless sensor networks", Proceedings of the Proceedings of the Third International Workshop on Software Engineering for Sensor Network Applications, IEEE Press, pp. 1-7.

[16] Zheng, X., Julien, C., Podorozhny, R., and Cassez, F., (2014), "Braceassertion: Behavior-driven development for cps application", Technical Report UTARISE-2015-002, 2014.

[17] Horváth, I., and Gerritsen, B.H.M., (2012), "Cyber-Physical Systems: concepts, technologies and implementation principles", Proceedings of the TMCE, Horváth, I., Rusák, Z., Albers, A., Behrendt, M. (Eds.), Organizing Committee of TMCE 2012, Germany, pp. 19-36.

[18] Grimm, C., and Ou, J., (2011), "Unifying process networks for design of cyber physical systems", Proceedings of the, San Diego, CA.

[19] Woo, H., Yi, J., Browne, J.C., Mok, A.K., Atkins, E., and Xie, F., (2008), "Design and development methodology for resilient cyber-physical systems", Proceedings of the, Beijing, pp. 525-528.

[20] Bogdan, P., and Marculescu, R., (2011), "Cyberphysical systems: Workload modeling and design optimization", IEEE Design and Test of Computers, Vol. 28 (4), pp. 78-87.

[21] Mosterman, P.J., and Zander, J., (2011), "AdvancingModel-Based Design by Modeling Approximations of Computational Semantics", Proceedings of the Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools; Zurich; Switzerland; September 5; 2011, Linköping University Electronic Press, pp. 3-7.

[22] Raghav, G., and Gopalswamy, S., (2010), "Architecture driven development for cyber physical systems", SAE International Journal of Aerospace, Vol. 3 (1), pp. 95-100.

[23] Fallah, Y.P., Huang, C., Sengupta, R., and Krishnan, H., (2010), "Design of cooperative vehicle safety systems based on tight coupling of communication, computing and physical vehicle dynamics", Proceedings of the, Stockholm, pp. 159-167.

[24] Rajhans, A., Bhave, A., Loos, S., Krogh, B.H., Platzer, A., and Garlan, D., (2011), "Using parameters in architectural views to support heterogeneous design and verification", Proceedings of the, Orlando, FL, pp. 2705-2710.

[25] Fallah, Y.P., and Sengupta, R., (2012), "A cyber-physical systems approach to the design of vehicle safety networks", Proceedings of the, Macau, pp. 324-329.

[26] Zhang, J., Goldsby, H.J., and Cheng, B.H.C., (2009), "Modular verification of dynamically adaptive systems", Proceedings of the, Charlottesville, VA, pp. 161-172.

[27] Pinto, A., and Krishnamurthy, S., (2010), "Developing design tools for uncertain systems in an industrial setting", Proceedings of the, Monticello, IL, pp. 1714-1721.

[28] Zilic, Z., and Karajica, B., (2011), "High-level design of integrated microsystems - Arithmetic perspective", Proceedings of the, Montreal, QC, pp. 77-82.

[29] Horváth, I., (2007), "Comparison of three methodological approaches of design research", Proceedings of the Proceedings of International Conference on Engineering Design ICED, Vol. 7, pp. 28-31.

[30] Horváth, I., (2006), "On the differences between research in design context and design inclusive research", Proceedings of the The International Design Research Symposium.

Chapter 2

OVERVIEW OF THE STATE OF THE ART

This chapter presents the workflow and results of the first research cycle of the research, which focused on describing the studied phenomenon and the associated knowledge gap, as well as on the main factors influencing this phenomenon and their correlations and causalities. The knowledge gap, which was already concisely circumscribed in the Introduction, needed a systematic investigation and multi-aspect elaboration. Filling in the knowledge gap was a multi-faceted issue, implying the necessity of investigating and evaluating the recent advances from all relevant aspects and perspectives. A very broad domain of research interests had to be considered since system design and engineering have been put into the center of research in multiple interrelated disciplines. Therefore, the main objective of the first research cycle was defined as identifying the various domains of research that have contributed to the description and explanation of the studied phenomenon and blending the knowledge and insights obtained from these knowledge sources.

Driven by the above objective, a survey is performed to explore and describe the various occurrences of the phenomenon. Subchapter 2.1 provides further information about the specific objectives of the survey as well as the reasoning model which facilitated the systematic execution of the survey. The reasoning model identified five knowledge domains for the study and specified the aspects to be considered. The first domain, elaborated upon in Subchapter 2.2, is the state of the art of *cyber-physical systems,* as the category of systems, the design and customization issues of which were considered in the research project. Discussed in Subchapter 2.3, the second domain of the study is *design approaches*, which are applied in design and development of CPSs. The third domain of the survey was *design principles and guidelines, which are* discussed in Subchapter 2.4. The investigations of these three domains were complemented by the investigation of *design supporting tools* and the findings are presented and analyzed in Sub-chapter 2.5. The study of the computational design and modeling tools directed our attention to the emerging research domain of *system-level features*, which was considered as the fifth domain of our study in Subchapter 2.6. Finally, sub-chapter 2.9 summarizes and blends the findings of the investigations, and discusses the implications of the findings in the context of the research.

2 OVERVIEW OF THE STATE OF THE ART

2.1 INTRODUCTION

Aim of the first research cycle was to (i) define the concerned knowledge domains of interest, and specifically to (ii) specify the addressed research phenomenon, (iii) describe the observed scientific/engineering knowledge gap related to it, and (iv) landmark the preferred direction of the promotion research. Initially, this research cycle started with the study of the wide topic of *designing of cyber-physical systems,* but it has gradually been narrowed down to the investigation of *system-level feature-based modeling of cyber-physical systems* as a result of the refinement applied by means of the insights obtained in the first research cycle. This was an implication of studying a very large number of recent academic research documents and industrial developments in the domains related to our research topic. The main guiding research question has been stated as:

What conceptual frameworks, computational approaches, and implementation methodologies are needed for the theoretical and practical realization of system-level feature-based modeling of cyber-physical systems?

Designing cyber-physical systems as a technological, epistemological and methodological concept has multi-faceted aspects. These aspects could be explored from multiple perspectives such as design methods, principles and tools, as well as CPS characteristics and other issues associated with design and modeling of CPSs. The studied phenomenon has been formulated as *advanced computer support for designing of cyber-physical systems in the pre-embodiment phase of development*. The characteristics of CPSs have a strong effect on all domains of knowledge and on the aspects of investigation concerning this phenomenon. It was a challenge to find the proper focus, arguable scope, and approach of the research. After getting more information about the unknown, the conduct of the research has been reinterpreted, narrowed down and streamlined multiple times. This thesis however includes only the part of the work that are in the mainstream of our inquiry and conceptualization, and those results which are seen as valuable outcome of this work.

Since the investigated phenomenon is multi-disciplinary (i.e. requires the knowledge and methods of multiple disciplines) it was unavoidable to find an exact focal point, which enabled us to make decision on the relevant research aspects, and to clarify the domains of interest for knowledge aggregations. The backbone of the knowledge aggregation was a structured literature survey for exploring the state of the art in the regarding domains. The knowledge domains considered for the study are explained by the reasoning model presented in sub-chapter 2.1.2. In a nutshell, the studied domains were (i) *cyber-physical systems*, (ii) *system design approaches*, (iii) *design principles*, (iv) *design supporting tools,* and (v) *feature-based design*. The survey of these domains resulted in a rather comprehensive overview of the state of the art, to define a specific knowledge gap that has been addressed in our research. The findings of the survey also helped us to define the specific objectives for the research with the intent of creating scientific and engineering novelties. In simple words, the knowledge gap has been formulated as lack of knowledge (genuine

concepts) and proper supporting tools for CPS designers in the phase of pre-embodiment. The knowledge gap was addressed from the abovementioned five perspectives.

2.1.1 Objectives of the first research cycle

The first research cycle aimed at constructing and formulating a descriptive theory that identifies the boundaries of the knowledge gap, determines what part of it can be addressed in this research, and fills in the identified part of it with new knowledge. The orientation studies have made it obvious that there is a relatively large knowledge gap related to *advanced computer support for designing of cyber-physical systems in the pre-embodiment phase of development*. Consequently, objective of the first research cycle was to study (i) the five identified knowledge domains, and synthesize a body of knowledge that describes what the current situation is, and (ii) the break out points regarding the development of novel tools. There were many useful source materials found in academic publications, but also in professional repositories on the web. However, in some fields the research was found to be still in an immature state. This especially true for the second generation (smart) CPSs. The design principles and supporting tools were more exposed and advanced in the case of the first generation of CPSs, which have their roots in, e.g. embedded systems, smart robotics, digital networking, systems control, and so forth.

In addition to the term cyber-physical systems, many similar phrases or nominations have been used in the literature to describe systems with similar or resembling paradigmatic characteristics: 'advanced embedded systems', 'smart robotics', 'intelligent sensor networks', 'complex adaptive systems', 'industrial Internet of things', and 'smart connected everything'. This resulted in an unclearness, which could not be resolved by the different definitions mushrooming in the last years. As a result, many seemingly relevant publication, claiming focus on and contribution to CPSs, were ignored. On the other hand, it made necessary for us to specify own definitions that correctly and exactly conveying the interpretations used in this thesis. Interestingly, the overwhelming majority of publications presented mono-disciplinary approaches, although the domain of CPSs is claimed to be multi-disciplinary in nature. It is fair to mention that the term cyber-physical system was used in many publications just because of its novelty and the imperative trends.

2.1.2 Reasoning model

Based on the identification of the pertinent knowledge domains, a simple reasoning model was formed. This is shown graphically in Figure 2-1. This model indicates a kind of flow of knowledge through the interrelated domains. This flow conveys and blends the knowledge associated with 'cyber-physical system' and 'system design approaches' with that of 'design principles'. This linking allows us to explore and understand causalities with regards to the aforementioned knowledge gap. The new two knowledge domains, namely 'design supporting tools' and 'system-level features' domains have been included in the study because of our intention to provide a theoretical framework and technical solution for supporting pre-embodiment design of cyber-physical systems.

The *nature of cyber-physical systems* was included in the study with the intent of collecting information about the characteristics of CPSs, which differentiate them from other technical systems and traditional products. The conducted study has explored what these characteristics are, and why and how these affect the conceptualization, embodiment and

implementations of CPSs. In the very wide field of CPS applications, we narrowed down the scope of our investigation to a specific family of CPSs, namely to cyber-physical consumer durables (CPCDs). The study of the literature associated with this family of CPSs provided evidences that there are intrinsic differences between the design challenges of CPCDs and the design challenges of traditional consumer durables. This part of the literature study also supported our assumption that CPS designers need different supporting tools than those available in the form of CAD/E systems or logics/analytics-based system architecting tools in order to be able to handle the high level complexity and heterogeneity of CPSs.

Figure 2-1 Domains of study

The study of the second domain was intended to explore and evaluate the recent design approaches and to identify mismatches caused by applying traditional design approaches for designing CPSs. We introduced a scoping here by limiting our attention to design approaches dedicated to *design for customization.* This was in line with our endeavor to concentrate on CPCDs. In the course of the literature study, all possible methodological and computational approaches of traditional design for mass customization have been identified. Through a critical comparative study of the specified design approaches and the characteristics of CPCDs, conflicting situations were identified. The reason of the conflicts was that none of the studied traditional design approaches could be applied in the architectural design and functional customization of CPCDs without a major adaptation. The third domain of study explored causalities of the identified mismatches in the design principle level.

The fourth studied knowledge domain was *supporting tools for CPS design*. We considered the tools that have been documented in the literature, as well as those which were available in web pages and repositories. Both academic implements and commercial tools were taken into consideration if they claimed to support mainstream CPS design activities. The study identified various functional categories of the tools. Some of the tools were initially created for component design, but insufficiently upgraded for system design. Some of them were based on high level abstraction and simplification, and intended for system-level design, but missed the necessary functional capabilities for multi-level system design. The results of this analysis showed that extension of the tools currently used for component design does not support system-level design of CPSs properly and sufficiently. Consequently, a (new) concept of system-level feature technology has been hypothesized as a possible solution for multi-level system design and customization.

As an implication, the past and current developments of feature technology have been studied as well as some conceptual frameworks in which it can be re-operationalized with a different objective. The traditional approaches of feature-based design have been discussed in a huge number of publications. However, elaboration of a system-level feature technology has not received enough attention so far. The reasons are such as theory-intensiveness, strong need for a multi-disciplinarily knowledge integration, and methodological symbiosis. In addition to the specification of the whole of system-level

feature-based design of CPSs, we also aggregated the requirements that should be considered at a computational implementation of the technology. Towards this ends, the findings of the various domains and sub-domains were critically analyzed and synthesized for a physically-based modeling of CPSs.

2.2 NATURE OF CPSs AND CYBER-PHYSICAL CONSUMER DURABLES

2.2.1 A brief historical overview

Cyber-physical systems emerged less than a decade ago as the result of the converging of the physical technologies and the cyber technologies, or more specifically, due to the growing functional interaction between mechatronic systems and information technological (IT) systems [1]. At the beginning, they have been developed based on concepts and paradigms such as *embedded systems* [2] [3], *distributed collaborative agent systems*, *smart ubiquitous computing* [4] [5], and *the internet of things* [6]. In 2009, the National Science Foundation (NSF) of the USA defined cyber-physical systems as "the tight conjoining of and coordination between computational and physical resources" [7]. Two years later, a more elaborated definition has been proposed, namely CPSs are "engineered systems that are built from and depend upon the synergy of computational and physical components that are coordinated, distributed, connected, robust and responsive" [8]. There have been multiple definitions proposed for CPSs by referring to keywords such as engineered systems [9], network-connected [10], tight integration of computing and physical constituents in a unified concept [11] [12], incorporation of computing and communication with monitoring and control of entities in the physical world [13], environment-coupled and diverse in capabilities [14], having feedback loops [15], monitored, coordinated, controlled and integrated operations by a computing and communication core [16], and working dependably, securely, efficiently in real-time [13].

There are different opinions about the origin of CPS due to its multi-disciplinary nature. If we want to consider the deepest roots, then we have to go back to the time of the first industrial revolution, when the early uses of mechanical feed-back in steam engines was seen as a starting point of control systems [17]. This progression was followed by the innovation of automatic aircraft guns during World War II [18], and by turning of the principles of control systems from analogue to digital by inventing and exploiting digital control in 1954 [19]. In the context of digital computing, the roots of CPS go back to the advent of the first multi-purpose computer (ENIAC) in 1946, that was used for providing close control loops around physical systems [17]. The next steps towards CPSs were facilitated by the development of a network implementation called ARPANET in 1969, which made it possible to connect large number of computers without any concern about their geographical locations [20], and the introduction of wireless communication technology as a combination of the concept of worldwide network with already developed wireless radio electromagnetic signal transmission technology [21].

In the last two decades, different kinds of wireless networks and protocols, microelectronics and processor technology, and advanced mechatronics became accessible in a cost effective manner. In addition, burgeoning of information technology as a field of both theoretical and practical science also facilitated the realization of concepts such as high level decision-making, artificial intelligence, big data processing, semantic technologies,

intelligent robotics, knowledge engineering, and context management. Putting everything together, we can argue that the advent of the paradigm of CPS is essentially the result of the progress in mechatronics and networking technologies (as the physical enabler) and the evolution and conjoining of many domains of information technology and engineering (as a cyber-enabler) [11]. This development has been complemented in recent years by innovations in the domains of energy (powering) and sustainability [22], as well as in the domain of security [23], [24].

The paradigm of CPS also entailed new functional structures and unique behavioral characteristics. Figure 2-2 shows the abstracted functional units of a typical CPS and the relations among the functional carriers. From an architectural viewpoint, CPSs are tight integrations of analogue and digital hardware (HW), control, and application software (SW), and knowledge, data, and media contents as cyberware (CW) constituents [26]. Hardware and software constituents are functionally active units of CPSs in contrast with cyberware constituents that are not functionally active units and accordingly absent in Figure 2-2. (It has to be noted that, just recently, it has become an objective of the development of smart reasoning mechanisms for CPSs and the development of active ontologies to make cyberware an active constituent of systems).

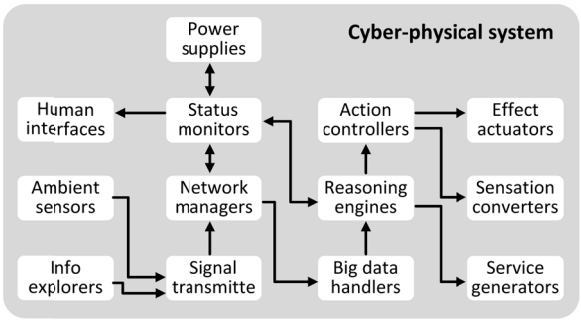


Figure 2-2 Abstract generic function of CPSs [25]

2.2.2 Main characteristics of CPSs and CPCDs

The best way for understanding the essence of the concept of CPS is to examine its distinguishing characteristics. In our literature survey towards the identification of the main characteristics of CPSs, several keywords such as “characteristics”, “attributes”, “specifications”, “features” and “properties” were used to find the most related publications. Many of the found publications dealt with extrinsic (from outside observable) characteristics, rather than with intrinsic (from inside observable) characteristics of cyber-physical systems. For instance, several of them focus on particular extrinsic characteristics such as, ‘dependability’, ‘security’, and ‘adaptability’. Some other characteristics are formulated in a high level of abstraction such as ‘environment coupling’ , ‘diversity in capabilities’, and ‘networked in nature’ [14], while some of them captured more tangible concepts such as ‘distributed’, ‘closed loop feedback control’ and ‘real time’. Though the converging technologies and system manifestations make it difficult and somewhat unnecessary to draw a strong demarcation line between CPSs and the other types of systems, we observed that many publication coming out at the end of the last decade tried to distinguish embedded systems and CPSs. The simple reason is that many of the characteristics of embedded systems are similar to or resemble those of the first generation CPSs [27]. Anyway, embedded systems are seen by many researchers as forerunners of CPSs. Moreover, embedded systems can be seen as part of a technical form in which specific (augmentative) implementations of CPSs may appear. However, the multitude of direct interactions of CPSs with the physical world is known to be a distinguishing characteristic [26] [27]. In addition, their typically intense motor, perceptive, and cognitive interaction with human (or other living)

stakeholders and the engineered and/or natural environment/systems is also a distinguishing characteristics of CPSs.

Below we present a succinct overview of characteristics extracted by the literature survey. Since the system characteristics were discussed in the literature from different aspects, we sorted the characteristics into seven groups: namely (i) architectural, (ii) processing, (iii) behavioral, (iv) networking, (v) timing, (vi) dependability, and (vii) efficiency characteristics.

The major **architectural** characteristics have been identified as: (i) closely integrated (i.e. deeply integrate computation with physical processes) [28] [29], (ii) highly varied system scales and device categories [29], (iii) heterogeneous and divers set of components [30], (iv) hybrid systems architecture (which includes analog and digital parts) [31], (v) capability to reorganize and reconfigure their internal structure [30], and (vi) open system with blurred overall system boundaries [30].

The major **processing** characteristics are: (i) distributed cooperative problem solving [30], (ii) information processing in real-time [30], (iii) high degrees of automation based on closed control loops [28], (iv) different problem solving strategies may be employed to achieve the overall objective of system [30], (v) knowledge intensiveness (and components handle different information sources) [30], and (vi) unsupervised way of learning from history based on smart agents and memorizing [30], and (vii) decentralized operation and organization of autonomous subsystems [32].

The distinguishing **behavioral** characteristics are: (i) dynamically reorganizing/ reconfiguring operation [28], (ii) having adaptive capabilities [29], (iii) offering convenient man–machine interaction [29], (iv) context aware and context dependent reasoning [30], (v) situation-based and context-dependent decision making [30], (vi) reactive system behavior in continuous interaction with the environment and executing at a pace determined by that environment [31] [33], (vii) disappearing computer: due to different type of interfaces, the user hardly recognize that information processing is involved [31], (viii) dedicated towards a certain application: due to dependability and efficiency [31], (ix) proactive in emergent internal and external situation [30].

The **networking** characteristics are as follows: (i) networked at multiple and extreme scales [28], (ii) relying on heterogeneous and distributed networks [34], (iii) using different kinds of connections (e.g., wired/wireless network, Wi-Fi, Bluetooth, and GSM) [29], and (iv) connections with other components can be predefined, ad-hoc or both [30].

The scheduling and **timing related** characteristics are: (i) meeting real-time constraints [28] [31], (ii) having unequal granularity of time and spatiality in various components [29], (iii) concurrent composition of the computing processes with the physical ones [15], and (iv) precise timing behavior with reliability that is unprecedented in any other human-engineered mechanism [15].

The **dependability** characteristics are described by the following key words: (i) trustworthiness and confidence [35], (ii) reliability [31] [28], (iii) maintainability [31], (iv) availability [31], (v) safety [31], (vi) security [31] [28], (vii) integrity (i.e., absence of inappropriate system alterations) [13], (viii) robustness under unexpected conditions [15], and (ix) adaptable to subsystem failures [15]. Finally, the efficiency characteristics have been defined as (i)

energy efficiency, (ii) run-time efficiency, (iii) code size efficiency, (iv) weight efficiency, and (v) cost efficiency [31].

Some of the mentioned characteristics are not solely belong to CPSs and can be considered as common characteristics of (engineered) technical systems. On the other hand, some of the characteristics discussed in the literature reflect a particular perspective, e.g. system security, while some other mentioned characteristics are very similar or closely related to design principles or guidelines (that point to an ideal situation). Consequently, some purification of the ideas seemed to be necessary for us.

2.2.3 A compendium of CPS characteristics

Having the above findings in mind, we developed a systematic and comprehensive compendium of CPS characteristics. This compendium includes the generic characteristics of CPSs [30], but it has also been extended to social-cyber-physical systems (SCPSs) [35], as a subcategory of CPSs that are in direct social and mental interactions with end users. The items included in the compendium are used in the rest of work as guides at the theoretical and development work presented in this thesis. The required characteristics of CPSs are:

C1. **distributed integrity:** distributed cooperative problem solving, in harmony with the techno-socio-economic environment
C2. **Openness:** functional and structural openness system (with blurred overall system boundaries)
C3. **Dynamicity:** capability to dynamically change boundaries and behaviors, and to reorganize/reconfigure internal structure
C4. **Integrity:** consisting cyber-part and physical-part in high level functional and structural synergy
C5. **Heterogeneity:** made of heterogeneous and diverse sets of components which can join and leave the collective at any time
C6. **spatiotemporal multiplicity:** manifestation in various spatial scales and temporal ranges
C7. **real-time processing:** real-time information processing capability
C8. **multiple connectedness:** connections with other components in a predefined and/or an ad-hoc manner at multiple levels
C9. **goal-oriented cooperation:** components may operate according to different problem solving strategies while achieving the overall objective of the system
C10. **source multiplicity:** knowledge-intensive components are able to handle different information sources
C11. **situated reasoning:** situated decision making, automated problem solving, and context-dependent reasoning
C12. **autonomy:** memorizing and learning from history in an unsupervised manner by smart agents
C13. **proactivity:** non-planned functional interactions, and acting proactively in emergent internal and external situations
C14. **distributed cooperation:** distributed decision making based on reflexive interaction of components and multi-criteria analysis and optimization
C15. **strategy variability:** different strategies for operation, resource management, security, reliability and integrity

C16. **reproductive intelligence:** seeds of reproductive intelligence towards the next generation (molecular and bio-computing-based) CPSs.

The required characteristics of SCPSs are:

C17. Ability to become aware of the users and their personal and social contexts, and to adapt themselves towards and optimal symbiosis.
C18. Ability to achieve the highest possible level of dependability (trustworthiness and confidence), accountability, security, accessibility, and maintainability.
C19. Operating as a self-organizing holarchic open system with a minimal environmental impact and sustainability from ecological, economic and social viewpoints.
C20. Achieving a balance between overheads and outputs, demand and usage of resources, and wastes and gains.

2.2.4 Some representative examples of CPSs

Advanced technologies are enablers of the implementation of advanced systems that are interconnected, smart and communicative, and able to sense, to make decisions, and act according to strategies [36]. There are numerous industrial applications of such systems, for example as environment monitoring system, home care surveillance systems, disaster warning systems, medical and healthcare monitoring systems, distributed robotics systems, and process control systems. Moreover, there are applications which benefit from the potential smartness of CPSs, such as smart energy grids, smart transportation infrastructure, smart medical equipment, air traffic management systems, automated manufacturing systems, intelligent buildings, autonomic vehicle convoys, and traffic prediction systems. Two often cited academic examples of CPSs are the *Distributed Robot Garden* at MIT, which includes distributed sensing and wireless networking integrated into a team of robots attending a garden of tomato plants [37], and the system developed in the CarTel project in the Boston area, in which real-time traffic information is used for calculating the fastest routes for a given time of the day [38].

Cyber-physical consumer durables represent a specific manifestation of CPSs. In a wide perspective, consumer durables are artefactual products that are (i) developed for long-term use (> 3 years), (ii) in frequent and direct interaction with the user, (iii) mass-customized for a wide range of users, (iv) equipped with specific control interfaces for users, (v) produced by batch manufacture, and (vi) widely/directly commercialized as merchandizes. Their typical representatives are such as major household appliances, personal mobility equipment, personal computing devices, and household accessories. CPCDs pave the way for the on-going transition to social-cyber-physical systems [39]. The reason is that CPCDs deeply penetrate into different domains (perceptive, cognitive, and emotional) of human life. Towards this end, they have to be able to work with the semantic of information, rather than with its syntax [35]. In general, the user aspects of CPCDs are getting more and more into the spotlight. These may give floor to the development of CPCDs, e.g., personal rehabilitation assistive systems, sport training systems, smart learning enabler systems, home facility management systems, personal safety systems. Due to their proximity to the users, CPCDs as heterogeneous and complicated CPSs, need to be developed for smart interaction with consumers, and should be made conform to consumer-oriented strategies, such as mass customization.

2.2.5 On the need for adaptation of CPSs and CPCDs

While CPSs are getting embedded human and natural environment and used in social contexts, the need for their adaptability increases. The characteristics C2, C3, C9, C11, C13, and C15 together circumscribe the *ability of adapting to emerging situations*. Adaptive CPSs are supposed to examine the changes in the whole context of their operation and reason based on the actual situations (C11), change their boundaries and behavior (C2, C3) in order to adjust themselves to the new objectives (C9), and interact with their environment proactively according to the internal and external situations and strategies (C13, C15). In different words, adaptability of CPSs entails three specific abilities: (i) recognizing changes in their working context, (ii) making a proper decision in context, and (iii) acting based on the decision. Although adaptation and customization might refer technically to the same system architecture and functionality, they imply rather different concepts from the user point of view. Customization can be considered as a kind of adaptation that is applied to a system according to the (explicit or implicit) demands of its users. Therefore, in CPSs (and particularly CPCDs) that are interacting with (functionally active or passive) human, customization is a valid type of adaptation. However, dynamicity of such systems entails other types of adaptation as well. The well-known term of 'mass customization' reflects only the marketing perspective of the same concept.

2.2.6 Customization as a form of adaptation of CPSs and CPCDs

Traditionally, mass customization (MC) is a win-win proposition for customers and producers only when producers find efficient and effective means of delivering high level of product variety [40]. Product variety fulfills either multiple demands of a user or similar demands of multiple users. The former one is sometimes referred to in the literature as multi-purpose or multi-functional products [41] [42] [43]. The latter one is alternatively called adjustable or configurable products [44] [45]. In any case, customization of a CPCD can be considered as a form of *adaptation of a CPS*. The challenge is how to apply MC principles to address CPS customization issues. Due to the limited similarities of CPSs and conventional consumer durables, there is a strong limitation with respect to the application of conventional MC approaches. In order to find opportunities, we analyzed the characteristics of CPCDs and the potentials of MC approaches, as shown in Table 2-1.

Our literature survey of this aspect resulted in the following conclusions: In the traditional practice MC, users are supposed to partly play the role of designers while customizing a system. Since CPSs have intense relationship with their embedding environment and stakeholders, they should be either self-adaptable or customizable by users. Although there are some levels of adaptability in CPSs, the role of user in customizing usually neglected. The role of user is more important in consumer durables due to their direct interactions. Therefore, mass customization receives growing attention in CPCDs.

Many of the mentioned CPS characteristics imply complexity. Complexity makes it necessary to predict the consequences of a change (even a simple one) with regards to the whole of a CPS. In addition to complexity, heterogeneity and integrity are the characteristics that raise challenges for customizing CPCDs. They both entail the necessity of a concurrent customization of hardware and software constituents in a way where changing one of them is supported by the other one.

2.3 POSSIBLE DESIGN APPROACHES TO MC OF CPSs

2.3.1 Domains and aspects of MC

In order to find concrete relations between the current MC approaches and the above-detailed CPCD characteristics, we scrutinized the known MC approaches. Our investigation took a closer look at the essence and the methodology of the design approaches of MC, considering their adaptability for MC of CPCDs. These findings are presented below. For the sake of completeness we mention that customization is essentially not the same thing as mass customization [51]. The term mass customization was coined by Davis [46]. It is usually explained as a strategy that involves integration of product variety and process efficiency [1]. The main goals of MC are to minimize "customer sacrifice" [47] and to improve customer satisfaction, which can be achieved by fulfilling their particular needs and by providing the required product properties at the price that customers are willing to pay. Cost, quality, and pace of delivery are the explicit advantages of MC that can be realized through improvement of design and manufacturing processes [52] and technology [53].

Table 2-1 Challenges and opportunities of MC of CPSs with regards to the characteristics of CPSs

Characteristics of CPSs		Challenges	Opportunities
C1:	Distributed integrity	Access to the components are limited	Distributed components imply modularity
C2:	Openness	Composition of the system can be changed unintentionally	Extension by adding components can be used for MC
C3:	Dynamicity		Reorganizing or reconfiguring internal structures is an opportunity
C4:	Integrity	Traditional MC cannot handle concurrent SW and HW variability	
C5:	Heterogeneity	Handling and responding to different protocols is challenging	
C6:	Spatiotemporal multiplicity	Access to other components is not applicable for a customer	
C7:	Real-time processing		Real-time customization could be obtained by incorporating this capability
C8:	Multiple connectedness	Traditional MC is not compatible with ad-hoc connections	Exploit different networks provides possibility for different customization
C9:	Goal-oriented cooperation	MC is limited to predefined problem solving strategies	Goal-oriented customization can be devised
C10:	Source multiplicity		It provides opportunity for multi-channel user interaction
C11:	Situation dependability	Traditional MC is not developed to deal with this capability	It could be handled in different levels to provide intelligent customization
C12:	Autonomy	It is in conflict with manual customization of features	This characteristic could be utilized to achieve auto-customization
C13:	Proactivity	MC requires predefined component's interaction	
C14:	Distributed cooperation	Applying changes requires distributed component variation	
C15:	Strategy variability	Could cause conflict in customization	
C16:	Reproductive intelligence	Cannot be handled by traditional MC	

Pine and Tseng [48,49] developed the concept of MC for use in marketing and industrial applications. They defined MC as offering products with increased variety, whilst keeping the economic efficiency of mass production. In the past two decades, many design and manufacturing principles, methods, and frameworks have been developed in order to realize MC. These principles have been applied in various contexts of product and service design, such as home appliances [50], automotive [51], fashion products [52], insurance services [47], and hotel facilities [53]. The main issues are how adaptable the MC principles are, and what can be done to adopt or adapt conventional MC principles for use in MC of CPSs.

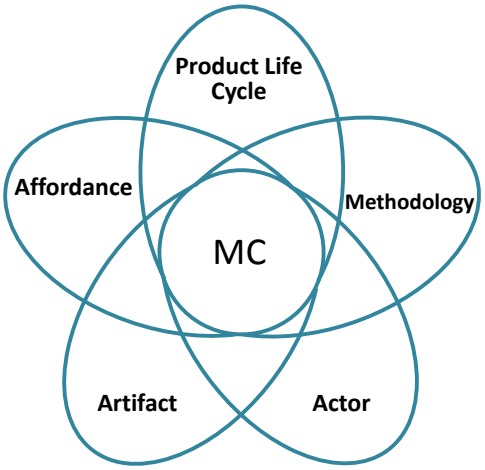


Figure 2-3 Aspects of MC

To support our assumptions and reasoning, we defined mass customization as a *methodology* (e.g. for designing, manufacturing, and marketing) of providing a variety of *affordances* (e.g. from appearance customization to function customization) in a certain phase of *product life cycle* (e.g. conceptualization, design, production, installation, and use) of an *artifact* (e.g. consumer durable, service, software) by a particular *actor* (e.g. designer, customer, retailer, company) [1]. Based on this multi-aspect definition, we considered the aspects shown Figure **2-3** as the ones we need to consider for our study. Conceptually, these aspects are orthonormal and imply different definitions and considerations of MC of CPCDs.

There are different MC approaches (types, levels, or degrees) depending on (i) deliverable product or service, (ii) provided affordances, (iii) exploited methods of design and manufacturing, (iv) actors in customization, and (v) points of actors' involvement in the product life cycle. In our research we referred to *'MC approach'* as a representative of a type (level or degree) of MC along with its associated principles, frameworks, methods, affordances, etc. as a package. It is worth mentioning that there is no clear boundary among various MC approaches, and that they can be used together, or the underlying methods can be interchangeably applied in some cases. For realization of mass customizable products, product developers need to apply one of the known MC approaches. This is recommended in the literature because MC approaches (i) are recognized as effective ways for realization of mass customizable products, (ii) have been employed and evaluated by other companies in terms of functionality and applicability, and (iii) work with methods that have been already devised and validated [1]. This also underlines the importance of studying the MC approaches towards finding principles for realization of mass customizable CPCDs.

2.3.2 Differences of the MC approaches from the aspect of product life cycle

Product life cycle (PLC) aspect is important element of many MC taxonomies for the reason that it helps distinguish various specific approaches (Table 2-2). Product life cycle refers to the phases which lead to delivery of a product. The four-stage model of PLC has been frequently used for distinguishing different approaches of MC by researchers [54-57]. This model has two parameters, namely (i) the phases of PLC and (ii) the indicator point. As

phases of PLC, it considers four phases of design, fabrication, assembly, and use that appear in all MC approaches. The differences of approaches are usually addressed with respect to the point of customer involvement in the PLC. This point is also known as the order decoupling point (ODP) [58] and has been used as an indicator for characterizing MC approaches [59] (Figure 2-4)

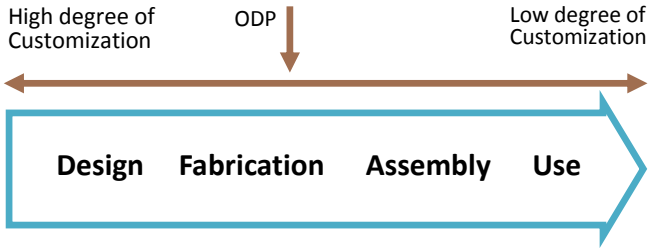


Figure 2-4 Customer involvement in the PLC

The earlier involvement of the customer in the product life cycle results in higher degree of customization because the product or service could be tailored to their specific requirements and vice versa. Many taxonomies and classifications of MC approaches have been formulated based on different reasoning and perspectives [60-64]. Table 2-2 presents the most common classifications in the literature.

Table 2-2 Forms of MC in different studies

Research	Forms of mass customization
Sharma [65]	(i) Customized product, (ii) Modified to customer spec., (iii) Customer specified options, (iv) No option
Pine [66]	(i) Modular production, (ii) Point of delivery customization, (iii) quick response providing, (iv) Customized services, (v) Embedded customization
Fisher et al. [67]	(i) Stand-alone options, (ii) Package options
Lampel & Mintzberg [68]	(i) Pure customization, (ii) Tailored customization, (iii) Customized standardization, (iv) Segmented standardization, (v) Pure standardization)
Ross [69]	(i) Core customization, (ii) High-variety manufacturing, (iii) Post-product customization, (iv) Self-customization
Spira [70]	(i) Assemble into unique configurations, (ii) Additional custom work, (iii) Additional service, (iv) Customizing packaging
Gilmore & Pine [71]	(i) Collaborative, (ii) Transparent, (iii) Cosmetic, (iv) Adaptive
Da Silveira et al. [62]	(i) Design customization, (ii) Fabrication customization, (iii) Assembly customization, (iv) Additional custom work, (v) additional services, (vi) package and distribution customization, (vii) Usage customization, (viii) Standardization
Tien et al. [72]	(i) Real-time MC, (ii) MC, (iii) Partial MC, (iv) Minor customization, (v) Mass production

The shortcomings of the current taxonomies can be discussed in terms of both the phases of PLC and the indicator point. In terms of phases of PLC, current taxonomies have ignored phases of *purchasing* and *preparation for shipping*, which are very significant in some MC strategies and approaches. Moreover, *assembly* is considered as a phase of PLC. However, we consider *assembly* as an activity, which can variously be placed before purchase in the production phase, after purchase, in the installation phase, or even during the use of product. On the other hand, customization is not only confined to ODP. Multiple MC approaches might have similar ODP. So, considering ODP as a sole indicator of categorization is not enough. We assumed that other indicator points are needed to systematically differentiate conventional MC approaches. Accordingly, in order to clarify the position of MC of CPCDs in the taxonomy, it was necessary to develop a new classification with finer specification of the phases of PLC. We have decomposed PLC into eight phases: (i) ***conceptualization***: concepts of a new product are generated, (ii) ***design***: details of a product are defined and developed for production, (iii) ***production***: a product is fabricated and assem-

bled (however, the product assembly may be postponed if assembly is considered as a means of customizing), (iv) ***launch***: introducing a product to markets, (v) ***purchase***: buying a product in a physical retail shop or via an internet store, (vi) ***preparation for shipping***: time between purchase and shipping which might be used for preparation of a product, (vii) ***installation***: setting a product to the place and condition of use (i.e. preparation before use), and (viii) ***use***: the phase in which a product performs its roles and functions.

For identifying different MC approaches, the possible indicator points have been considered and, as a result, ODP was replaced by four new indicator points (Figure 2-5). These points are attained by separating design and customization activities and by assuming that performing both (design and customization) has two sub-processes, namely decision making and executing. Therefore, the significant indicator points (actions that influence MC process) in the PLC have been identified as:

Ⓣ: (decision making in design); *Thinking about unique requirements of user(s)*: designers investigate changeable features and specifications of products to realize a product idea based on the differences in user(s)' requirements.

Ⓔ: (executing in design); *Enabling customization*: putting customization capability in a product by producer. This action is not a decision making process, but incorporation of specified affordances which have already been determined.

Ⓒ: (decision making in customization); *Customization of product/service*: decision making action to customize a product based on specified user requirements (mostly performed by the customer, but it could also be performed in consultation with designers and experts).

Ⓐ: (executing in customization); *Applying customization*: incorporating the preferences of the user into the product (this may be performed by anyone; it is simply the realization of customer's wishes).

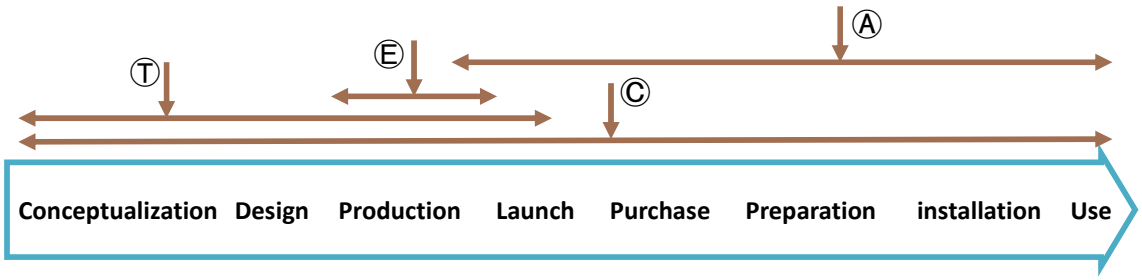


Figure 2-5 A model for taxonomizing MC approaches considering PLC phases and indicators

Figure 2-6 shows the applied classification of MC approaches. It identifies eight phases of PLC, based on the four indicator points introduced above [1]. This classification does not consider any level or special advantage of the different approaches of MC. However, the first two MC approaches can be considered as approaches of higher level customization. Each MC approach is suitable for a particular kind of products and can be adopted by companies according to their marketing strategies. In the following paragraphs we briefly describe the MC approaches shown in Figure 2-6.

	Conceptualization	Design	Production	Launch	Purchase	Preparation for shipping	Installation	Use
1. Co-customization	ⓒⓣ		Ⓐ	—	—			
2. Custom-fabrication		ⓒⓣ	Ⓐ	—	—			
3. Assembly-by-company		Ⓣ	Ⓔ		Ⓒ	Ⓐ	Ⓐ	
4. Assembly-by-customer		Ⓣ	Ⓔ				ⓒⒶ	
5. On-delivery-customization				Ⓣ		Ⓐ		
6. Embedded-customization		Ⓣ	Ⓔ					ⓒⒶ
7. Standard-customization		Ⓣ	Ⓔ		ⓒⒶ			

Figure 2-6 MC approaches

Co-customization approach implies designing and conceptualization of a product based on unique requirements of a particular customer. For example designing and tailoring an evening dress for a specific ceremony for a given customer.

Custom-fabrication approach involves customers in ordering a product in the design phase. This allows ensuring that the product is made to order. Although the concept of the product is similar for all customers, the product details and some configuration will be made based on customer's order. On the line of the previous example, it means supplying a product based on a predefined dress, considering only sizes of customers in tailoring.

Assembly-by-company approach implies applying customization by company based on the decision of the customers. Considering their own requirements, the customers should choose the components from a catalogue, and the company installs components and ships the product. An example of this approach is choosing a particular CPU, RAM, hard drive, etc. for a notebook in producer's website.

Assembly-by-customer approach means customizing an already purchased product by (i) changing the layout of components, (ii) adding or removing some modules, (iii) bending or shifting the angle of components etc. As an example, consider a food processor with different attachable components for different purposes.

On-delivery-customization approach is a restricted modification of a product usually for personalization. Engraving the owner's name or special sentences in back of the purchased tablet is a good example for this approach.

Embedded-customization approach implies controlling the capability of a product in the use phase by built-in controllers (e.g. fan speed of an air conditioner, and brightness/contrast of a TV set).

Standard-customization approach means providing various products with a range of attributes (e.g. size, color, internal memory) that can be chosen by customers during the purchase phase.

2.3.3 Differences of the MC approaches from the aspect of method

The second aspect of investigation is *'method'* which is implied by the fact that various methods, tools, and technologies have been developed in order to support MC. Normally, two groups of methods are used for these purposes: (i) product design methods, and (ii)

manufacturing methods [73]. Certain MC approaches have been developed to simultaneously address both design and manufacturing aspects of MC due to interdependences among design and fabrication tasks. In our research, only design related aspects of MC are considered. Below, we briefly overviewed the design and manufacturing methods that have been used in MC. We indicate the most important differences of the methods. Typically time and cost issues drive the designers to redesign or reuse existing designs in most of MC approaches. There are endeavors to reuse relevant design experience to achieve the economy of scale, even in design processes [74]. Lots of efforts were made to develop design methods for mass customization. For instance:

- the ***case-based evolutionary design*** (CBED) method was developed for combining the cognitive model of reasoning with experiences and technology for finding and presenting such experiences [77] [75],
- the ***design for mass customization*** (DFMC) method was developed for conducting family-based design, considering several aspects of product realization such as design, manufacturing, sale, market and service, concurrently [49],
- *the* ***modular design*** (MD) method (which focuses on consumer, functional, physical, and process domains to generate modular product architectures) was developed according to the assumptions of the axiomatic design theory [76] [77],
- the ***product family modeling*** (PFM) method was developed by including a triple-view scheme used in modeling various functional, technical and structural aspects of product families [78],
- the ***design by customer*** (DBC) method was developed to address specific individual customers' needs when the uniqueness is regarded as an important requirement [79],
- the ***concurrent design for mass customization*** (CDFMC) method was developed with the aim of integrating entrepreneurial development with disciplined systematic approach [80], and
- the ***design for mass personalization*** (DFMP) method was proposed to effectively and efficiently offer personally unique products through creation of a product ecosystem based on a design platform and active customer participation [81].

All of design methods for MC have been developed according to particular approaches. Most of them are based on *product family architecture* (PFA), which provides a unifying product platform to cover diverse customer needs while keeping the economy of scale [74]. PFA is efficient and mostly used in MC approaches #2, #3 and #4 in Figure 2-3.

As for the need to address challenges of maximizing variability and efficiency simultaneously, manufacturing methods of MC demands unique characteristics, which stem from its underlying strategies such as: (i) dynamic fabrication, (ii) flexible resources (e.g. workers, robots, machines, workstations, etc.) [82], (iii) postponement of product variety [83], (iv) consideration of PFA [49], and (v) flexible process routing (non-linear process plans, alternative routings and variable sequences). It is worthwhile to mention that the importance of these characteristics depends on the selected MC approach. For example, 'dynamic fabrication', 'flexible resources' and 'flexible process routing' are suitable for MC approaches #1 and #2 in Figure 2-3, while "postponement" is important in MC approaches #3, #4 and #5. Due to the use of mass production methods, cost effectiveness is a major concern in MC approaches #6 and #7. Customized products are expensive when produced according to the MC approaches #1 and #2, because they are designed and made individually. On

the other hand, CPCDs are components-based systems, and it is logical to develop and design them based on PFA in order to benefit from economy of scale.

2.3.4 Differences of the MC approaches from the aspect of actor

There are multiple actors involved in the life cycle of a product. In addition to the activities that are accounted for each MC approach, the roles of the involved actors are also influential. For instance, the task of assembling can be accomplished by the company, by retailers, by a third party company or by users in different MC approaches. It was mentioned earlier that in some of the MC approaches the user is supposed to play the role of a designer during the customization activity. However, most users do not have enough knowledge about products, e.g. concerning on material selection, safety issues, and functionality. They are usually unfamiliar with "designerly way of thinking" [84] and the uncertainties make them puzzled. Perception of customers about complexity, effort, and risk that they have count on during the mass customization process could prevent them from buying a customizable product [85]. Pine argued that performing the user–designer role may lead to confusion during customization. Therefore, he coined the term *mass-confusion* [24]. Many studies considered the phenomenon of mass confusion as a result of the complexity inherent in a wide assortment of options [86-88]. It has been recognized that more complex products and user interfaces increase the probability of mass-confusion.

Three main sources of mass-confusion include: (i) the large number of choices and uncertainty of finding the most appropriate choice, (ii) complexity of translating and addressing individual needs to a concrete specification of a product, and (iii) uncertainties related to the behavior of the provider (due to missing information) [100]. Therefore, there is a high probability of the occurrence of mass-confusion in the case of MC of CPCDs. Nevertheless, it can be noticed that to avoid mass-confusion in MC approaches #3, #4 and #6, the customer must have a certain level of prerequisite knowledge and skills. For customizing a product, the user needs to interact with different stakeholders in different ways. In co-customization, a direct interaction with concept designer is compulsory, while interaction among customer and detail designer is important in custom fabrication. In MC approaches #3 and #5 the users interact with sales departments of companies (e.g. through the producer's website). Finally, in MC approaches #4 and #6, the users have to customize the product thorough a predetermined interaction with the product. The role of *customization applier* in MC approaches #1, #2, #3 and #5 is played by the *company* while, in MC approaches #4, #6 and #7, the *users* have to apply customization themselves.

2.3.5 Differences of the MC approaches from the aspect of artifact

The characteristics and specifications of a*rtifacts* are another aspect which influences choosing of MC approaches. In this regard, complexity of the product is a major issue (which indirectly determines the applicability of the MC approaches). This needs attention since complexity is an intrinsic characteristic of CPCDs, which reduces the number of applicable MC approaches. The options usable for customization are also affected by the complexity of interactions. Three factors have been identified concerning the options of customization: (i) width of options, (ii) range of option, and (iii) resolution of option. *Width of options* refers to the number of available options to customize a product. It indicates how many features can be controlled by user. For instance, if for an office chair (i) the height of the seat, (ii) the tilt, (iii) the armrest height and (iv) the swivel angle are adjusta-

ble, then the width of options is four. *Range of option* can be described based on the (i) minimum value, (ii) maximum value, and (iii) the distance between these values in an option. For instance, in the office chair example, the distance between minimum and maximum seat height indicates the range of option. *Resolution of option* implies the number of offered choices in a range of option. If the seat height of the office chair example stops at five different levels, then resolution of option is five. In the case of many products and options, the resolution of option is continuous. Evidently, the three customization factors strongly depend on the product characteristics. That is why they are defined based on the targeted users and the preferred MC approach. Width of options influences the interface complexity of a product more than two times of the other factors. MC approaches #1, #2 and #3 have the capability to offer the largest width of options, while MC approaches #4 and #6 have some limitations. The MC approaches # 5 and 7 have the least capability to offer width of options.

2.3.6 Differences of the MC approaches from the aspect of affordance

Affordance is a quality of a product, which can be customized by an individual. Affordance expresses how deeply a product can functionally be customized. For MC of CPCDs it is important to choose the MC approach, which most effectively supports functional affordances. In order to study affordances of various MC approaches, we needed to (i) develop a framework for classification of affordances of MC approaches, (ii) define the significant characteristics of each type of affordances, and (iii) compare the state of different approaches based on developed criteria. The affordances of a MC approach might be superficial (such as changing the appearance of a product), or might be deep (such as customizing the functional scope and quality of a product). In order to establish a framework, four affordances have been identified, namely: (i) *appearance* which mean customizing the appearance of the product or its packaging in term of color, texture, 2D image, engraving, 3D shape, and even smell and taste, (ii) *capacity/size* which mean changing size of product or scale of output, e.g. in a vacuum cleaner, (iii) *quality of service* which refers to different services provided such as quality and speed of printing or number of USB port of a notebook, and (iv) *function*, which implies providing variety of functions in a product according to multiple functional demands or the capability of a product to perform diverse functions by replacing some components.

Different MC approaches usually offer different affordances. This means that each MC approach is efficient and effective in providing only a particular set of affordances. For instance, *embedded customization* can support affordances #2 and #3. The *functional affordance* is mostly obtained through *co-customization* or sometimes through *assembly-by-customer*. *On-delivery customization* is mostly suitable for *appearance affordances*. Additionally, *custom-fabrication* and *assembly-by-company* are the best MC approaches for the *quality of service* affordance (Table 2-3).

2.3.7 Recognized constraints and limitations of using MC in the context of CPCDs

Based on the findings of the study, we argue that the most important generic principles of conventional MC approaches are as follow:

- Postponing the '*applying customization*' action until the latest possible phase in the product life cycle [89].

- Designing products based on independent modules in order to facilitate assembling in different ways.
- Designing the manufacturing process based on independent modules in order to move and rearrange manufacturing resources easily.
- Designing supply network for providing two capabilities: (i) cost-effective supply of the basic product, and (ii) flexibility, responsiveness, and quick delivery in taking customers' order. [89].
- Process transformation through (i) significant up-front investments, and (ii) flexible information flow [90].
- Information architecture based integration of different manufacturing technologies into a structured framework capable of combining human and technological factors [90].
- Reuse of design experiments, process architecture, and supply chain structure [74].
- Unifying product platform to cover diverse customer needs while keeping the economy of scale [74].
- Developing PFA by (i) identifying optimal building blocks, (ii) formulate product family structure, and (iii) configuration of the process.
- Deriving the most economic building blocks and maximizing their applicability [49].
- Utilizing inherent flexibility in the production capability and maximizing the repeatability of the product family-based design, rather than designing one unique product at a time.
- Promotion of sharing knowledge over different products by incorporating customer needs, process experience, and learning [80].
- Planning effective information transformation in order to make the demands visible throughout the supply chain [91].
- Using highly-skilled and -trained employees especially in the information transformation and product delivery phases.
- Designing product family architecture and process concurrently.

Table 2-3 summarizes the major differences of the approaches with regards to sub-aspects of MC. It can be used as a basis for selection of a suitable approach for MC of CPCDs. However, the generic principles included in this table may be inadequate if they are used with an incompatible strategy and approach. In the case of *PLC,* Table 3 shows the differences of approaches in terms of the customization tasks. It captures the phase of customization and categorizes of the approaches as: (i) before production customization, (ii) on-purchase customization, and (iii) post purchase customization. As far as *methods* are concerned, despite the significant number of publication about product family architecture and module-based design, only three MC approaches exploits PFA. The sub-aspect *'cost effectiveness'* has a wide range in MC approaches. Regarding *actors*, the *'customization applier'* is a 'company' in MC approaches #1, #2, #3 and #5. As indicated in Table 2-3, usually *'users interact with' 'designer'*, *'retailer'* and *'product'* in order to customize a product. In approaches #4, #6 and #7, the users can customize product without any interaction with anyone else. *'Users' skill'* sub-aspect is an indicator for the knowledge and skills the users need to have for customizing products. In the MC approach 4, users should have high level of knowledge about the product they want to customize. In the *artifact* aspect, the sub-aspects are *'product complexity'* and *'width of options'*. Finally, in the *affordance* aspect, *'depth of customization'*, implies function to appearance range and the numbers stands for the different affordances (mentioned in previous part). As shown in this table, the MC approaches #5 and #7 are not proper for realization of functional affordance.

Table 2-3 Comparison of MC sub-aspects for various MC approaches

MC Aspect	MC sub-aspects	MC approaches						
		1. Co-customization	2. Custom-fabrication	3. Assembly-by-company	4. Assembly-by-customer	5. On-delivery Customization	6. Embedded Customization	7. Standard Customization
PLC	Made-to-order customization	•	•					
	On-purchase customization			•		•		•
	Post-purchase customization				•		•	
Method	Depend on PFA		•	•	•			
	Cost effectiveness	Low	Low	Medium	Medium	Medium	High	High
Actor	Customization applier	Company	Company	Company	User	Company	User	User
	For customizing, users interact with…	Designer	Designer	Retailer	Product	Retailer	Product	--
	Users' skill	Low	Low	Medium	High	Low	Medium	Low
Artifact	Product complexity	High	High	High	Low	High	High	High
	Width of options	High	High	High	Medium	Low	Medium	Low
Affordance	Depth of customization	**4**,3,2,1	4,**3**,2,1	4,**3**,2	**4**,3,**2**	**1**	4,**3**,**2**	**3**,2,1

2.3.8 Mapping the characteristics of CPSs to the findings

The characteristics of CPSs have been discussed in the previous sub-section. Based on a grouping of these characteristics, we propose to consider seven distinguishing *generalized characteristics* of CPCDs:

GC1 ***Dynamic function and architecture***: Characteristics C2, C3, C5, C7, C8, C9, C11, C15 and C16 imply to the fact that function and architecture of CPCDs are completely dynamic. Therefore, only post-purchase customization is efficient and logically possible in MC of CPCDs. MC approaches #1, #2, #3, #5 and #7 are not appropriate for MC of CPCDs.

GC2 ***High overall complexity***: Characteristics C4, C5, C9, C11, C13 and C16 imply that high complexity is an inherent generalized characteristic of CPCDs. So, MC approach #4 cannot be used in customizing CPCDs.

GC3 ***Component-based architecture***: *Characteristics* C1, C3, C4, C5, C8, C9, C10 and C14 imply the component-based architectural nature of CPCDs. MC approaches #1, #2, #6 and #7 are independent of PFA, and consequently are not economically efficient for MC of CPCDs.

GC4 ***Importance of functional affordances***: In fact, the primary objective of designing and applying CPCDs is providing functional affordances fulfilling certain known or emergent needs. All characteristics of CPCDs are associated with this provided functionality. MC approaches #5 and #7 were not initially developed for providing functional affordances, and therefore they are inappropriate for MC of CPCDs.

GC5 ***Functional complexity of components***: Functional complexity of CPCDs implied by almost all of the mentioned characteristics entails that their customization can only be performed by experts. Since the users are regarded as the customization applier in MC approaches #4, #6 and #7, these approaches are not suitable for MC of CPCDs.

GC6 ***Inaccessibly distributed components***: Characteristics C1, C6 and C14 indicate the distributed nature of CPCDs. Some components belong to common systems (e.g. GPS satellites) and this generalized characteristic makes them rather inaccessible. Therefore, the users' control and access to all components of CPCDs are rather restricted. As a result, MC approaches #4 and #6 are not efficient for MC of CPCDs.

GC7 ***Autonomous system behavior***: Characteristics C10, C12 and C14 imply intelligence and independence behavior of CPCDs, and therefore the authority of users over systems declines. Autonomous behavior of CPCDs disregards the demands of users (and even those of the manufacturer) to a large extent. For this reason, using MC approaches #1, #2, #3, #4 and #6 are not logical for MC of CPCDs.

A closer examination of the contents of Table 2-3 and the generalized characteristics of CPCDs listed above revealed that there are some mismatches (even conflicts) with regard

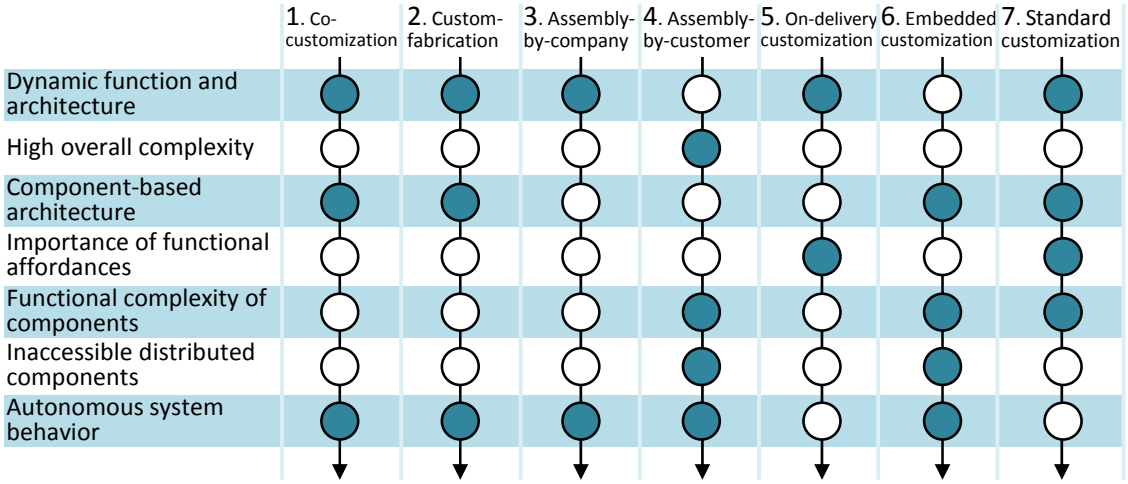


Figure 2-7 Mismatches of conventional MC approaches to CPCDs' generalized characteristics

to adopting different MC approaches. Conceivably, as the bottom line, they may lead to inefficiencies. We have developed Figure 2-7, to visualize the found mismatches. The filled circles indicate conflicts between conventional MC approaches (columns) and the generalized characteristics of CPCDs (rows).

2.3.9 Consideration of a specific form of MC for CPCDs.

Based on the conducted survey and analysis presented in this sub-chapter, it can be concluded that the currently known MC approaches were devised for conventional consumer durables, and have limitations when applied to CPCDs. The reason is the unique characteristics of CPCDs such as duality, openness, heterogeneity and distributed nature. For example, due to the heavy '*complexity*' of CPCDs there is a probability of mass-confusion when the task of customization using conventional customization approaches is assigned to users. As an inherent characteristic of CPCDs, *dynamicity* also needs an appro-

priate MC approach for customization in the later phases of product life cycle. It should be seen that the methods and principles of conventional MC approaches are much more user-oriented than product-driven, while the characteristics of CPCDs entail a function-orientated rather than usability-orientated approach. Obviously, there are some conventional MC principles, such as module-based design, reusing sources, postponing customization task, etc. that may be applied in MC of CPCDs after a necessary adaptation. Putting everything together, there is a need for novel MC approaches (as well as for new design tools and methods) for customization of CPCDs. The conceived approach should be able to fulfill the following requirements:

R1 **Providing customization opportunity after purchase:** Functional adaptation of the of purchased CPCDs to the users requirements and specificities of the embedding environment implies the need for customizations in the *use-phase*.

R2 **Appropriateness for adapting high complexities:** The chosen MC approach should be able to adapt complicated components (i.e., hardware, software, and cyberware constituents), and to consider the influence of adaptation of one constituent in the adaptation of other constituents.

R3 **Enabling exploitation of functional affordances:** It is important to have opportunity to explore and utilize functional affordances that are dependent on and can manifest in specific application contexts. MC approaches that cannot support *capacity*, '*quality of service'* and '*function'* affordances are not appropriate.

R4 **Achieving cost efficiency for all stakeholders:** The customization of CPCDs should happen in a cost effective way, and the approach of customization should facilitate achieving it.

R5 **All-constituent and system-level customization:** The approach of customization should allow for system-level adaptation, and not only adaptation of one type of constituents (e.g. hardware or software). Customization of all constituents together supports system-level customization.

R6 **Easy re-customization according to different needs:** The approach should make it possible to change the customization of CPCDs relatively easily and fast, even under dynamically changing conditions.

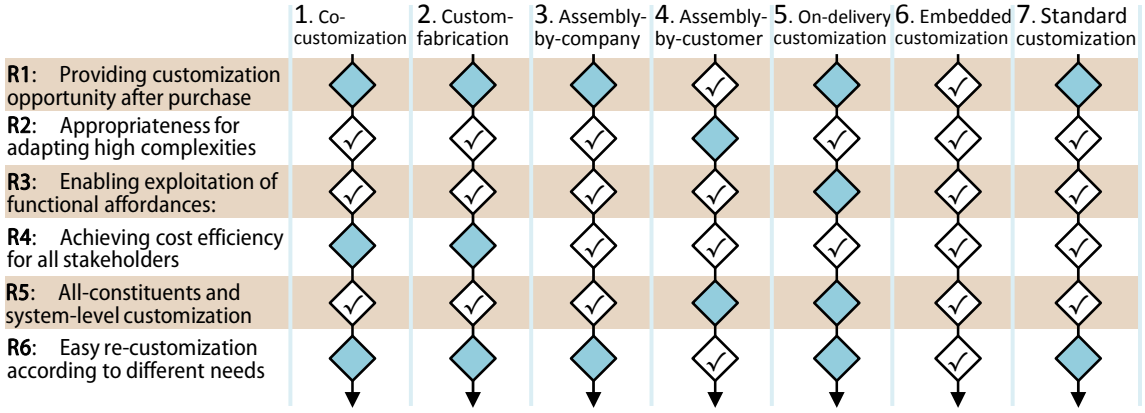


Figure 2-8 Comparison applicability of MC approaches for CPCDs customization

If we assess the MC customization approaches from the point of view of satisfying these requirements, or mostly supporting their fulfillment, then, *embedded customization* (EC) seems to be the most appropriate approach among other (Figure 2-9). Accordingly, we hypothesized that EC can be adapted for customization of CPCDs. Consequently, the next sub-chapter focuses on elaboration of the EC of CPCDs considering the design principles.

2.4 DERIVING DESIGN PRINCIPLES FOR EMBEDDED CUSTOMIZATION OF CPCDs

2.4.1 Research on design principles for EC of CPCDs

In general, d*esign principles* are pieces of knowledge that explains how to get to feasible concepts and how to realize them in the design phase. The d*esign principles of embedded-customization* specify "what designers should do" in the design phase of a product in order to enable its customization by the user in the use phase. According to our reasoning, the principles of EC that underline the design of CPCDs should be developed according to their specific characteristics. To operationalize this in the practice, we followed the scheme shown in Figure 2-9. This scheme has been created based on two assumptions. First, we assumed that there is a relation between the system features and the design principles of EC approach relevant for traditional consumer durables (TCDs). Secondly, we assumed that the system features of CPCDs determine the design principles that are applicable for their EC, which some of these principles can be derived from the design principles of TCDs. Some design principles may be taken over without any change (solid line), some others may be taken over after adaptation of various extents (dotted line) and some of them cannot be used at all for EC of CPCDs due to mismatch (wavy line). Yet another set of design principles are to be defined starting out from the system features of CPCDs (dashed line). The objective is to embed adjustable and controllable features in a CPCD, which could be changed and customized after purchase, according to the requirements of the users [92]. Without striving after exhaustiveness, the following categories of design principles for EC have been considered in our study:

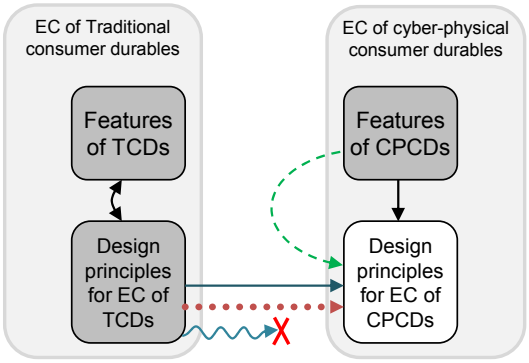


Figure 2-9 Scheme of deriving novel design principles for EC of CPCDs

Strategy principles are rooted in the nature of the EC approach and that are expected to be generally applicable (P01 & P02). ***Conduct principles*** are general but also have specific meaning in EC of CPCDs (P03-P06). ***Architecture principles*** consider components of the system, as well as their structural relations (P07-P12). ***Exploitation principles*** are related to user interaction with the system (P12-P23). The design principles for EC of TCDs are presented in detail in [93]. These design principles can be briefed as:

P01 Think about 'customization capabilities' in the 'conceptual design phase' and design them as 'product features'

P02 Consider prospectively that customization of the product will be done in the 'use phase' by various users

P03 Identify what aspects, functions or features of a product need to be customizable

P04 Choose the most straightforward 'means of customization'

P05 Identify the customizable components

P06 Identify common components

P07 Design the core-component containing common denominators

P08 Design the power connection only for the core-component

P09 Combine several products for the variable requirements of users

P10 Use available components for different task (by e.g. folding, adjusting)

P11 Design the product based on an integrated architecture

P12 Reuse internal and external components and behaviors

P13 Provide the possibility of customizing for all users with different mental and physical abilities

P14 Select proper type of adjusting and controlling interface

P15 Reduce the complexity level of user interface (UI) as much as possible

P16 Use the same (or universal) connection protocols and ports for all replaceable components

P17 Design safe and error-free connectors for replaceable components

P18 Design efficient feedback on customization of intangible features

P19 Use meaningful graphical elements (e.g. icons, color) for intangible customization

P20 Include a customization assistant for users

P21 Design controlling and customization means consistently

P22 Develop a multi-language or language-free customization enabler and instruction

P23 Reduce the costs of mistakes and misuses by corrective user interface design.

2.4.2 Correlation between design principles of EC and system-level features of TCDs

Below we address the issue of correlations between design principles of EC and system-level features of TCDs. For a comprehensive investigation of customizability and the related customization principles of TCDs, we needed to consider their system features, which could be sorted into five groups namely: (i) operational features, (ii) implementation features, (iii) application features, (iv) behavioral features and (v) emergent features. For pragmatic reasons, we decided to narrow down our investigation to a specific group of products for which embedded-customization is a typical form of customization. It is worth pointing out that several MC approaches can be applied together to a product, e.g., a laptop whose hardware components can be customized through "assembly-by-company" approach, with engraved logo of a customer's company through "on-delivery-customization" approach, and also can be customized by the user in the use phase by using the opportunities provided by the "embedded customization" approach. In our investigation, we did not consider features that are brought up through other MC approaches. Figure 2-10 represents the correlations among design principles of EC and the

system features of EC-TCDs. These features are either results of applying the EC principles, or require the application of specific EC principles that are still to be realized.

The system features of TCDs that can be customized based on an EC approach are elaborated in [93], which can be briefly mentioned as:

F01 Responsive to a wide range of requirements

F02 Embedded controllers

F03 Embedded varying mechanisms

F04 User-controlled functionality

F05 Multi-functionality (multi-capacity and multi-purpose)

F06 Component replaceability

F07 Program-based customizability

F08 Hardware-based (and/or software augmented) constituents

F09 Centralized architecture

F10 Independent predefined input/output

F11 Dependent on external power sources

F12 Limited connection possibilities

F13 Predefined scale of customization

F14 Predefined and specified diversity of functions

F15 Customizable in specific ways

F16 Simplified interfaces

F17 Easy to learn and easy to use interface

F18 Manually controllable and configurable

F19 Simple physical joints

F20 Standardized and uniform joints

F21 Possibility of undoing

F22 Multi-level interaction

F23 Built-in user notification

F24 Automated warning notification

F25 Graphical entity-based interaction

F26 Real-time prompting

F27 Updatable and upgradable constituents

F28 Comprehensive protection

F29 Misuse prevention, and

F30 Multi-lingual operations

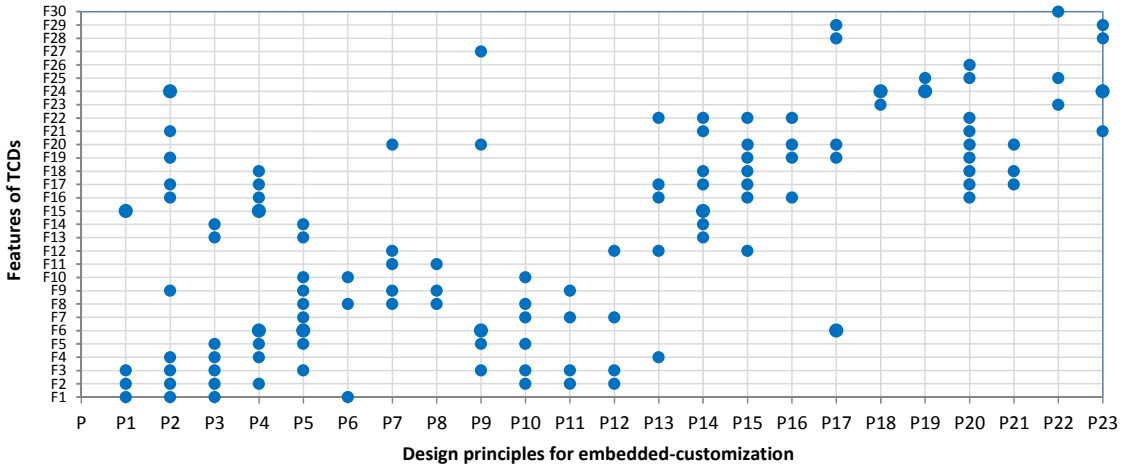


Figure 2-10 Correlation of design principles of EC and features of TCDs

We assumed that there are correlations among the design principles of EC and the system features of products. Figure 2-10 shows these correlations between the design principles of embedded-customization and the system features of EC-TCDs. It can be recognized that one system feature can in principle be related to several design principles, and a design principle can also be reflected by multiple system feature manifestations. These intertwined relationships can probably also be observed in other approaches due to the commonality of most of the MC principles. For comparison of system-level features of CPCDs and TCDs, we consider those features of CPCDs which are in one way or another in contrast with the above mentioned system features of EC-TCDs. The differences between TCDs and CPCDs have been considered from several aspects. The main consideration is *architecture* and *function* aspects. However for further elaboration, aspects such as *application*, *network*, *interaction*, and *maintenance* have been studied as well. Although the issues related to these complementary aspects can be considered as either *function* or *architecture* aspects, we group them as separate groups due to their importance.

From *application* perspective, the importance refers to different behavior of CPCDs based on the various situation; they are so called "situation based" [94]. Unlike TCDs in which input and output of components are predefined, in CPCDs, components perform their functions differently according to their situations, signals, input data, required output, and so forth. From *network* perspective, ad hoc network possibilities [95] and openness are key features of CPCDs. It is always unpredictable to know which components connect to each other, and how they behave in operation. Therefore, the functional capability of a system is not anticipated in some situations. In addition, CPCDs are combinations of networked and decentralized sensors, actuators, and processing components [96]. Decision making is distributed over system components [30]. From *user interaction* perspective, usually only expert users can deal with CPCDs due to lack of development of user-interaction. Replacing components in a CPCD require dealing with more sophisticated connection types rather TCDs. Re-programming, re-coding, and reconfiguring components are common ways of adapting CPCDs, which cannot be modified by a novice user. Additionally, in case of connecting additional components, possibility of connecting components in wrong ways is high due to high complexity of system and lack of standardization in interaction pattern [97]. From *maintenance* perspective, fault detection systems are not developed

well-enough, especially for CPCDs. Therefore, maintaining and detecting errors, faults and failures is challenging due to high complexity, integration, and uncertainty of these systems. Actually, fault detection systems in CPCDs are not as much mature as in TCDs. In the following paragraphs, the mentioned aspects of comparison are elaborated further.

2.4.3 Comparing TCDs with CPCDs based on their system-level features

By comparing system features for both TCDs and CPCDs which are correlated to the EC design principles, we sorted system features into various sets based on their relevance to the TCDs and CPCDs. These sets are: (i) features that are usually available in TCDs and do not have any similar feature in CPCDs, (ii) features that are more related to TCDs and sometimes can be found in lower level CPCDs, (iii) features that are similar and common in both groups of consumer durables, (iv) features that are more dominant and critical in CPCDs, and (v) features that belong to CPCDs without any congruence with TCDs. The mapping of these features in two-dimensional space is represented in Figure 2-11. The horizontal axis below the chart represents features of TCDs and the above horizontal axis is for features of CPCDs. Distances of each point to these axes show the relevance of the features to these consumer durables. In other words, if a feature is more related to TCDs, the point is plotted in the lower part of the graph and the points above the graph show features with more relevance to CPCDs. Points labeled "F" are the same as the features as used in this subchapter. Moreover, we found features that are more manifested in CPCDs. These features were briefly discussed in this chapter, and are labeled with "C" in Figure 2-11. The colored areas (separated by background color) show different sets of the stated features. In order to derive design principles for EC of CPCD, It could be proposed that:

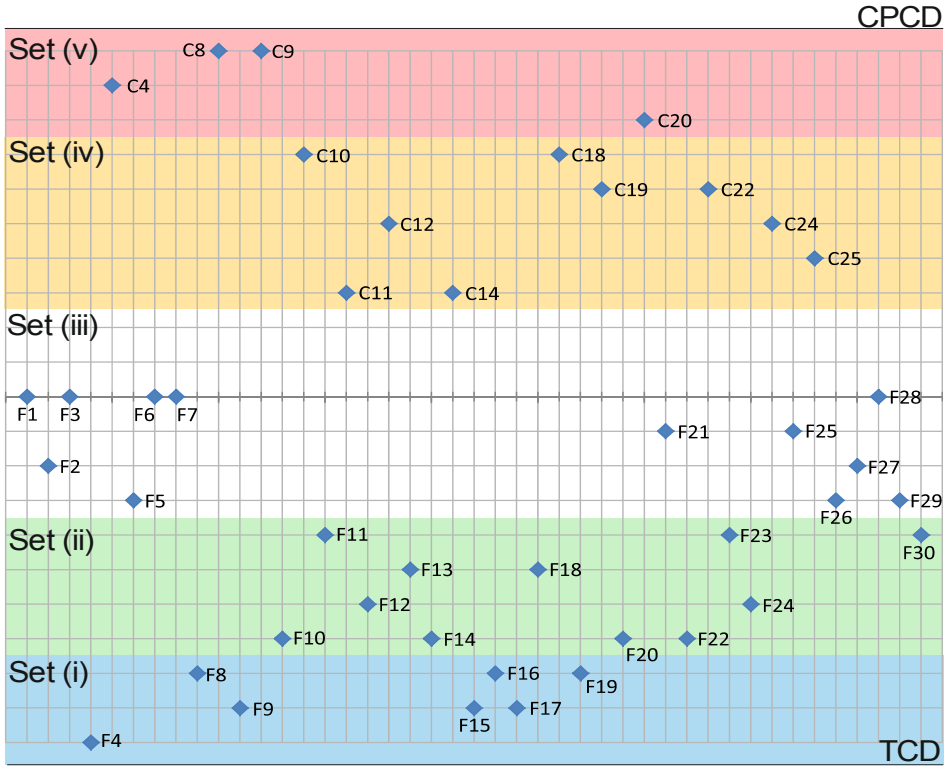


Figure 2-11 Congruency of features of TCDs and CPCDs

- Design principles associated with system features of Set (i) can be ignored because they are irrelevant for CPCDs.
- Design principles associated with system features of Set (ii) need some adaptation due to their partial relevance to CPCDs.
- Design principles associated with system features of Set (iii) can be directly transferred and used in EC of CPCDs, because these features are common for both TCDs and CPCDs.
- Design principles associated with features of Set (iv) and Set (v) are not available yet. Due to novelty of these features, new design principles for EC should be generated.

2.4.4 Discussions on the findings

It has been revealed that the reasons of mismatches of traditional design approaches to design of CPSs rooted from insufficiency of design principles for supporting features of CPSs. It implies the necessity of deriving design principles and approaches according to unique characteristics of families of CPSs including CPCDs. By generalizing the result, it has been proved that there are correlations between design principles and products' features. These correlations have been observed in embedded customization of TCDs. Consequently, the similar kind of correlation could be valid in embedded customization of CPCDs, or any other design approaches that are desired to be applied for CPSs. Similarities and differences of TCDs and CPCDs could be used as criteria for deriving EC design principles for CPCDs. This finding could be generalized as rules for other design approaches and groups of CPSs. Although it has been demonstrated that the design principles have correlation with the features of products and systems, the proposed method of deriving new design principles has not been practically tested. Therefore, it could be implemented as a general method for any design approaches after a validated practical test.

The investigation in the domain of *design principle* (i) has shed light on the reasons behind the mismatches of conventional design approaches and characteristics of CPSs, (ii) provided insight about the knowledge gap from the design principle perspective, (iii) proved that the design principles could be transferred systematically (e.g. to be exploited with design supporting tools), and (iv) argued that the design principles are varied and dependent to the chosen design approach, and the targeted product (or system) category. Since we are not interested in developing and deriving design principles for customization of CPCDs, we give up the continuation of the current line in order to focus on the assisting designers of CPSs to tackle the challenges appear in CPS pre-embodiment design. The mass customization approaches and design principles were mentioned as means of investigation and we don't feel any commitment to refer them again in the following chapters. From now on, we assume that the required design principles are provided for designing, and we try to focus on the essence of technical challenge.

2.5 CURRENT TOOLS TO SUPPORT DESIGN OF CPSs

2.5.1 General considerations

Nowadays technical systems are becoming more complex to be handled by traditional system engineering approaches [98]. It has been argued that to tackle this growing complexity of CPSs, we need to apply higher abstraction for formal modeling of these systems in the early design phases [99]. We embraced the idea of dealing with system design in the early design phases and opposed the idea of applying high abstraction in modeling CPSs. Our reasoning about these issues is presented in this sub-chapter. Additionally, the main part of the current sub-chapter belongs to discussion about the available design tools for supporting CPS designers. The main issue in our investigation was to explore the ways that the tools support design activities as well as the design phase they are applied. Among the available supporting tools for CPSs, no one specifically developed for customization of CPSs. However, there are some modeling tools that support some level of configuration and adaptation.

For instance, there has been an effort performed in Renault company for capturing customization issues in model-based system engineering (MBSE) [100]. They tried to introduce a variability modeling technique based on MBSE. Studying this technique, we understood that by variability modeling they mean modeling the compatibility constraints in assembly-by-company approach of MC. The ultimate goal of this technique is to guide users to make the customization decisions in order to increase possibility of components compatibility and system implementation. This modeling tool is implemented in SysML and called Co-OVM [101]. Other variability modeling solutions mentioned in the literature [102], [103], and [104] also focus on constraint modeling instead of system modeling.

Since there is not a specific tool for supporting embedded customization of CPCDs, we decided to broaden our investigation to explore modeling tools for CPS design. Using models for system design is a common practice nowadays. The terms model-driven development [105], model-driven engineering [106], or model-based design [107] are made to refer to the concepts addressing challenges of complex technical systems such as CPSs. Models can be used for (i) collecting requirements and specifications of systems, (ii) capturing the process of system development, (iii) simulating operations of systems, and (iv) analyzing the operations and architecture of systems[108].

Considering CPSs as integration of computation and physical processes, designing such systems requires understanding software and network processes in dynamic interaction with physical processes [108]. Different approaches have been employed for CPS design that could not be generalized as some of them are domain-specific approaches. Generally, design approaches can be divided into three categories of top-down for conceptual design, bottom-up for detail design, middle-out for disparate team work [109]. More specific approaches have been developed based on the mentioned general approaches.

Approach of model driven architecture (MDA) is known based on its three-stepped model transformation. In this approach, modeling starts with *computation independent model* (CIM), then, this model is transferred into *platform independent model* (PIM), and finally into *platform specific model* (PSM) [110]. This approach starts from functional aspect and

ends in architectural considerations of system design. Platform-based design (PBD) approach is originally employed for integrated circuit (IC) and microelectronic devices design. It considers cost as an important factor of implementation [111]. This approach integrates both top-down and bottom-up view in system design for connecting 'architecture space' and 'application space'. This approach exploits 'reuse' of platform components to provide different functionality and consequently reduce development risks, costs and time to market [112].

The term actor-oriented refers back to the concept introduced in mid of 1970s by [113]. Actor-oriented design can be considered in contrast with object-oriented [114]. In this view, actor-oriented modeling is able to formulate concurrency and sophisticated communication among components with the aid of ports. Classes in object-oriented can be defined by attributes and methods, while actors in actor-oriented approach can be defined by data (state), parameters, and ports. This approach is suitable for behavioral modeling.

2.5.2 Briefing on the studied systems

The approaches underpin creation and development of design frameworks and supporting tools. Since there are no well stablished framework for pre-embodiment design of CPSs, computational tools developed for conceptual design of complex technical systems are rare [115]. However, in engineering design and system evaluation context we can name several well-known tools. Some of these modeling tools are commercially available (e.g. LabVIEW, Simulink), many modeling tools are developed in academic and laboratory environments (Ptolemy, Metronomy), and numerous funded projects have focused on this issue (e.g. COMPASS [116], DANSE [117]). The following paragraphs brief these tools.

Simulink from Mathworks is initially developed for control system modeling [114]. It is one of the most used commercial model-based-design tools [118] that is able to handle continuous-time mixed dataflow simulation [119]. There are many add-on products for Simulink (e.g. SimEvents [120], Stateflow [121]) that can extend its application.

LabVIEW from National instruments, as a system-design platform, significantly improved functional design of embedded systems [122]. It is commonly used for data acquisition and instrument control modeling [123].

Modelica is an object oriented domain-specific modeling language [124] developed to support physical component-oriented modeling of complex systems (e.g. mechanical, electrical, hydraulic, thermal, control components). However, modeling of software components and the related issues are not supported by Modelica. The library of Modelica consists of generic model components and functions in various domains. Since Modelica is an open source modeling language, several commercial front-end software have been developed to support implementation of this language namely, AMESim [125], Dymola [126], Cymodelica, SystemModeler [127], MapleSim [128], CATIA Systems [129], and OpenModelica [130]. Modelica is an equational modeling tool [131]. It is object-oriented in contrast with Ptolemy [132]. In Modelica ports carry continuous-time signals which are connected to other ports through declarative relationships.

UML is originally created for software development. Therefore, concepts of time, concurrency and real-time design are missed in this language. Since UML is a language that can

be applied for a broad scope, UML profiles refine it for specific applications. The main problem of UML which is inherited to SysML is lack of consistency of the overall descriptions. Thus, it is not possible to check if two parts ppropriately fit together [133].

Ptolemy is based on actor-oriented modeling approach [134], where the actors communicate according to the rules defined by model of computation (MoC) [108]. Examples of MoC are discrete events (DEs) [135], continuous time (CT) [136], finite state machines (FSMs) [137], and process networks (PNs) [138]. Ptolemy II is specialized in handling timing, concurrency and heterogeneity challenges of CPSs [139].

Metro II supports platform-based design approach and handles components interaction by non-functional quantities [140]. It suffers from limitation of supporting continues time models and MoCs in function modeling [141].

MARTE is claimed to be able to support time, concurrency, and quantitative characteristics of software and hardware platforms [142]. It is developed as a profile of UML. Profiles in UML are standardized extension mechanisms to support domain-specific purposes [143]. MARTE is an effort towards linking SysML diagrams to semantics [144].

Metronomy integrates functional modeling from '*Ptolemy'* and architectural modeling from *'Metro II'* in order to provide a function-architecture co-simulation framework [99]. The work and results presented in [141] officially introduced Metronomy for timing verification of CPSs and after that cited by other researches, i.e., [145], [99], [146], and [147]. In Metronomy, timing contract is the sole linkage among architecture and function aspects [141].

BG-UMF (bond graph based unified meta-modeling framework) is an alternative of UML that uses a hybrid approach based on model unification and integration [109]. It originally developed with the focus on conceptual design of CPSs and works based on MDA approach. Similar to Ptolemy and Modelica, It uses MoC for supporting concurrency and timeliness. BG-UMF is claimed to be able to represent physical systems across multiple domains.

COMPASS project targeted three problematic issues in modeling system of system (SoS) namely [148]: (i) *Independence of constituents*: separate groups of stakeholders are involved in a system development process. They are working separately towards a same goal; therefore, changing in one component might affect the other components' design. (ii) *Verification of emergent behavior*: behaviors of a component in isolation are different from the behaviors of the same component composed in a system. (iii) Semantic heterogeneity: Multi-disciplinary approaches are required for designing and engineering of heterogeneous systems [149]. Therefore, conceptualization, designing, and analyzing of such systems are only possible by sharing common semantic framework [148]. COMPASS develops its models in SysML. The devolved models are translated systematically into CML, which is a language for semantic representation defined by this project [150].

Besides the mentioned tools, there are some languages and add-ones to support and extend functionality and applicability of the modeling tools. SystemC is a class library of C++ that is used for simulation [151]. SpecC is a super set of ANSI C [152]. They both support concurrency and a rich set of communicative primitives through special constructs

to represent hardware concepts [133]. Timing definition language (TDL) is developed to support modeling time and event-triggered components for distributed systems. TDL is used for programming dataflow models in Simulink and discrete event (DE) model in Ptolemy II [153]. Other remarkable efforts focusing on different aspects of system engineering and design namely, System-On-Chip Environment (SOCE) as a modeling framework based on SpecC language [154], Artemis as a tool for performance evaluation [155], PeaCE as an extension of Ptolemy for software-hardware co-design [156], Metropolis as an environment based on PBD paradigm [157], and ARTS as a simulation platform based on SystemC language [158].

2.5.3 Some reflection of the findings

Major challenges for the current system modeling tools are: (i) integration of heterogeneous components of CPSs, (ii) interoperation among this kind of components, and (iii) stimulation of modifying existing design [108]. Semantics of heterogeneous components in a system model have been largely neglected in the available platform-based design principles, and accordingly fails to support model-based design tools [111]. Following paragraphs elaborate more specific non-compliances of the mentioned tools to the requirements faced in CPS design and modeling.

Since UML lacks formal semantics, it cannot be used for simulation, and accordingly UML and SysML cannot support CPS design [109]. SysML focuses on syntax of diagrams instead of their semantics. Therefore, it is strictly limited in handling system design problems [108]. In other words, SysML diagrams could be interpreted differently as they are not connected to semantics. Several tools such as MARTE and COMPASS developed with the aim to solve the problems faced in SysML. However, in both of mentioned efforts the problem of neglecting morphological and physical properties inherited from SysML. Analytical capability of SysML is limited to the evaluation of simple parametric equations. Therefore, trade-offs, representing consequences of requirements changes, and demonstrating results of modifying system configuration are not viable in this modeling language [159]. PIDO framework could be considered as an effort towards bridging this gap [160]. However, morphological concerns are also completely neglected in this approach.

On the other hand, some tools (e.g. Modelica, LabVIEW, Simulink, and Metronomy) are developed for system engineering and analyses rather system-level design and conceptualization (by system-level we do not refer to SoC or embedded systems). For example, Metro II suffers from linear mapping between function and architecture. Though architecture is considered as a base for designing, the system components are considered as abstract semantic entities. Modelica can be considered as another example that is more suitable for system analyzing rather system-level design. It does not sufficiently support distributed, heterogeneous system modeling.

In Ptolemy II, heterogeneity is handled strictly hierarchical, which each node in hierarchy belongs to one domain and the components are dependent on the specific communication mechanism provided by their respective node [140]. Therefore, components are limited to communicate with only two levels of the hierarchy. In addition, there have been many efforts to integrate different modeling tools to support comprehensive modeling of heterogeneous systems [161], [162] and [163]. However, none of them sufficiently support

our desired functionality. Moreover, Ptolemy does not sufficiently support architectural aspects of the modeled systems [141].

Figure 2-12 summarizes our study about the available modeling tools. The aim is to demonstrate why the current tools are far from the desired one. The North-South direction represents design 'concern' as the original aim of the tools. In south we have analyzing and discipline-related concerns, and in the north, the design goal is to focus on interdisciplinary system-level considerations. The West-East direction represents the perspectives employed in the modeling tools. The tools, represented closer to the west side, deal with problems from logical viewpoint. However, going to the east side, the architectural and morphological perspectives receive more attention.

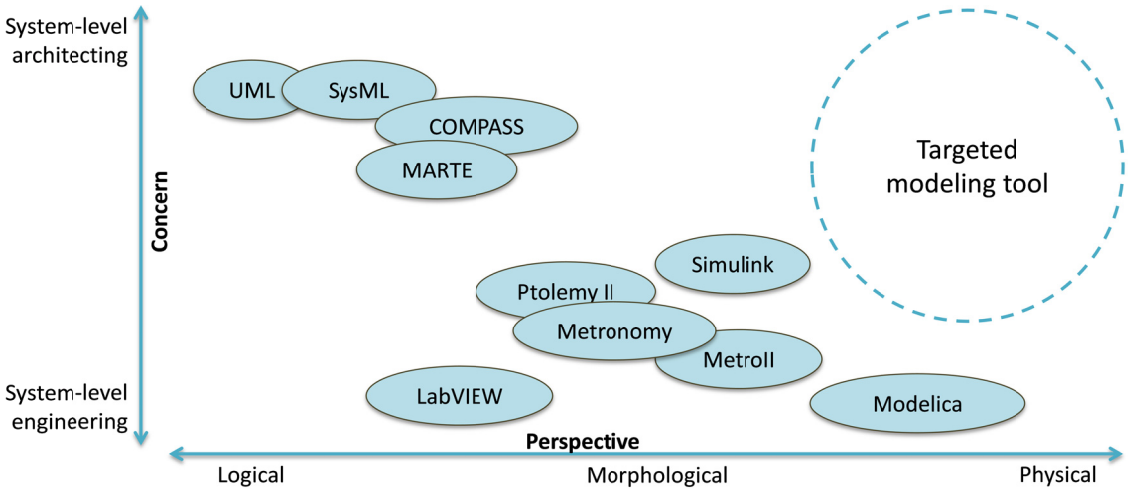


Figure 2-12 Position of the available (and desired) tools regarding design concerns and modeling perspective

2.5.4 Implications of the findings

The overall trend of system design is shifting from designing all system from scratch to design system by integration of predesigned or independently-designed components [133]. By growing complexity and heterogeneity of systems, a typical system producer is not able to design and fabricate all required components. For example Apple® orders the screen pack to Samsung, a big group of smartphones use Android operating system developed by Google, most of the processors of smartphones are designed and fabricated by Qualcomm, RAMs are developed by Texas instruments, and Apps developed by third parties. Accordingly outsourcing is becoming a dominant solution, and composability challenges are becoming the main issues in system design.

The complexity of handling system-level properties as well as diversity of stakeholders involved in system development are mentioned as challenges in CPS design rooted in decentralized control over constituents of these systems [148]. Accordingly, in system-level design of CPSs, system designers should concentrate on composability issues rather analytical evaluation of components. Admitting that the structural models are important in system design due to representing static information about construction of a system, this aspect is simply ignored in system modeling [108].

In system conceptualization and engineering design, morphological properties of components are as important as logical and operational properties, particularly in CPSs that are highly integrated in their surrounding physical world. This consideration is neglected in the most of the available modeling tools. The available tools usually have serious problems in formulating morphology, physical principles, and architectural attributes, because they do not capture physical information of their domains. That is the reason we targeted for a modeling tool that deals with system-level design as a composability issue and considers morphological aspects in system architecture besides the functional attributes and aspects. In the following paragraphs the other aspects of modeling CPSs are more elaborated as specific gaps we faced here.

- Applying higher abstraction is suggested in the literature [164] to tackle complexity in system-level design. The important issue is to understand that by system-design they usually mean multi-core processor system on chip and not CPSs. It is also argued that a model should be a "good abstraction" of the real system, with this meaning that it should only remove inessential details [108]. However, it is still unclear what is essential and what is not. Lower level of abstraction is the better choice as it is closer to final implementation, for the reason that higher-level abstraction hide unnecessary details which might be critical in CPSs [140]. The popularity of abstraction is rooted in heterogeneity and complexity of CPSs. Another way to tackle integration complexity is top-down decomposition [133]. To avoid abstraction, we stick to decomposition.
- Modeling languages significantly affect the design processes [165] [133]. For instance, some modeling languages are unable to handle concurrent functions. Therefore, it has been suggested that modeling languages should be developed specifically targeting the domains [166]. However, it should be considered that a single modeling language more likely is not able to handle all domains. Accordingly, a domain-specific language is not able to model CPSs as they are integration of heterogeneous domains. It could be concluded and proposed that for heterogeneous systems we should concentrate on developing a language-free modeling tool.
- As it mentioned before system design and engineering analysis can be considered as two dispersed concerns. However, they are traditionally complementing concerns. Sometimes, lack of relation between the system models (e.g. developed in SysML) and discipline-related models, (which consider detailed component-level properties and issues), results in a large gap between system design activities and design engineering analyses [159].
- Another problem with conventional system-level modeling and design tools is that they address either software or hardware but not both. Although for many years co-design was a hot topic in this context, the proposed methodologies usually considered SW and HW issues in a segregated way [133]. This problem rooted in the differences of timing and concurrency semantics as well as morphological and architectural attributes of HW and SW.
- Available modeling tools do not sufficiently support heterogeneous nature of CPSs [139]. This issue is critical in modeling connections between heterogeneous components. Intrinsically different (HW, SW, CW) components should be represented uniformly in order to be able to represent heterogeneous connections. The physical properties of components should be captured as an important part that interacts with functionality

of the modeled system. On the other hand, in a model, no matter how different the components are, they should be represented uniformly in order to capture their interconnection.

- The evidences show that morphological properties of a system cannot be studied in 2D modeling space, and suggest to capture 3D models of components in CPSs for better engineering analyses [145]. By reducing a system to a 2D model, not only one dimension is missed, but also all physical properties and architectural parameters are omitted in functional considerations. That is the reason the tools that cannot consider morphology of the system cannot go further than logical analysis.

We needed to separate what should be captured by our modeling tool from how they should be captured. The former refers to epistemological aspect and the later refers to methodological aspect of modeling. Accordingly, the epistemological issues were considered as our priority. By targeting system level design concern along with morphological perspective, we understood very soon that there is no practical or theoretical insight available to underpin our tool development. That was the reason that we decided to develop our own underpinning theoretical framework.

For developing the required underpinning theory we needed to find a well stablished modeling approach as a starting point. This approach should bridge the current modeling gap towards the desired modeling tool. We initially selected '*feature technology'* for this purpose due to its ability to handle morphological modeling. Our further study revealed that the flexibility of the *feature technology* makes modifications of the current situation possible. These modifications are needed for creating the desired version of feature which is named as system-level feature. Feature technology has some advantages that are useful for our purposes:

- Though features can be interpreted concerning logical and/or virtual things, they are typically considered to be parts of physical things.
- Features identify regions of a thing that have meaning and/or importance for one or multiple aspects and purposes.
- Features that can be recognized in multiple things indicate some sort of similarity in manifestation and/or functionality.
- Features are regarded as discrete but interacting modeling entities that can be captured, predefined, parameterized and represented in multiple forms.
- Feature entities can be grouped and catalogued according to multiple cognitive meanings and schemes.
- Features increase the conceptual level of analysis and synthesis of artefactual things.
- Depending on their nature, meaning, definition, and representation, features are adaptable, convertible and reusable entities.
- Features can be interpreted, recognized and/or defined on multiple compositional (structural) levels of things.
- Features can be standardized in both application-independent and application-dependent manners.

- Features can be captured in engineering ontologies which lend themselves to maintaining consistency and transferability, and reducing redundancy and overheads.
- Features increase efficiency and consistency of engineering modeling, analysis, while reducing efforts, cost, time and cognitive overload.
- Based on features, different product and service modeling methodologies can be developed and applied.

2.6 SYSTEM-LEVEL FEATURES AS A NEW CONCEPT FOR SYSTEM MODELING

2.6.1 The paradigm of features

For investigating system-level features, we needed to understand what does 'feature' mean traditionally, what is the origin of feature, and how its concept has been developed chronologically. Features are knowledge entities with engineering significance used in generating, analyzing, or evaluating designs of a part or assembly [167], [168]. Features can be considered as information sets regarding to form, design reasoning, performance, or manufacturing of parts or assemblies [169]. Although there are multiple views on feature technology in terms of application (e.g. feature recognition [170]), design-by-feature [171] [172] is the approach we need to focus in our investigation. This approach is also referred as feature-based modeling or feature-based design. The reason of development of this approach lies on its higher potential of supporting the design process better than traditional CAD systems [169].

From this perspective *'feature'* is defined as knowledge patterns concerning description of a part [167], a semantic entity describing a part or assembly by grouping respective information about design, manufacturing and functional manner [173], an entity that its formal and geometric presence is needed for design process [174], an information carrier that supports design and communication among engineering tasks towards manufacturing [175], a region of interest, which its respective information entity required in reasoning of design, engineering and manufacturing [176], and computer representable data associating with geometric design, manufacturing process and functional requirements [177].

In the classical theory of engineering product features, a feature is something that has significance in a given context [178]. For example, *form features* capture regions of both prismatic and freeform industrial shapes that have a meaning for humans or smart system agents from a semantic point of view (e.g. chamfer, hole, rounding, sharp edge, depression, protrusion) [179]. This idea has been extended to applications where geometry, structure, materialization, implementation, etc. imply some meaning in the context of a particular application (e.g. as is done by manufacturing features, assembly features, piping features) [180]. Feature-based design allows transferring many pieces of information available in the design phase to a database, which could be used by downstream applications [173]. Therefore, manufacturing features, assembly features, and process planning features could be derived from form features used in the design phase.

Chronological development of feature technology [181] has been briefed and modified as follow: (i) fundamental research about feature definition by Shah (1991) [167], (ii) definition

of application feature by Burgett *et al.* (1995) [182], (iii) parameterization of freeform feature by Horváth *et al.* (1996) [183] [184], (iv) machining process mapping by Chu *et al.* (1996) [185], (v) mapping product function for conceptual design by Brunetti *et al.* (2000) [186], (vi) definition of advanced, object oriented assembly feature by Ma *et al.* (2004) [187], (vii) generation of concept design at the assembly system level by Jin *et al.* (2007) [188], (viii) virtual reality, physics modeling via CAD by Igwe *et al.* (2008) [189], (ix) concept feature generation with a repository by Bohm *et al.* (2008) [190], (x) controlling the level of details via geometric feature classification by Chu *et al.* (2009) [191], and component-based system design by Lampka *et al.* (2013) [192].

Researches about feature technology started in the late 70s. In the early 90s, it was already a mature body of research and implementation work [193]. In the last two decades, attention was paid to ontology-based feature definition [194], information sharing [195], mapping [196], and feature conversion [197]. Feature-based design has had more influence on the detailed design and planning activities of product development processes, than on the conceptual or pre-embodiment design activities [198]. This comes from the dependence of meaning on the details of manifestation [199].

2.6.2 Feature technology in designing TCDs

Several issues are concerning the feature-based design e.g. feature classification, feature relations, feature validation, feature representation, and feature applications. Some of these issues were briefly visited in our investigation to clear out the route towards system-level features concept. Classification of features is useful for identifying the mechanisms they support in designing, providing common terminology, and developing data exchange standards [167]. Since initially features were mostly involved in form and geometric design of products, early classifications of features were about form features. They considered five categories: (i) form features, (ii) precision features, (iii) technological features, (iv) material features, and (v) assembly features [200], [201].

The classifications proposed later have added functional and application features to the former category of form features. Functional features are used for describing function of a component during conceptual design [202], form features refer to generic shape that conveys some engineering meanings [203], and application features are used during process planning and includes manufacturing, analysis and inspection features [169]. According to [204], features originate in reasoning process applied in multiple stages of product life cycles, and are intensely associated with particular application domains.

We proposed *'purpose'* as a criterion for classification of features. In our view, features are considered as products of human cognitive activities which are made according to particular ways of seeing towards specific engineering purposes. The *purposes* specify the semantic roles that the associating features are made for. These roles should be considered from both epistemology and methodology viewpoints. Epistemologically, the engineering purposes provide cognitive frames for making rational decisions. While, methodologically, the engineering purposes provide frameworks for decomposing complex objects into context-relevant formally-definable elements. Generally, '*purposes'* can be considered according to four categorical reasons: (i) the reason for which anything exists, (ii) the reason for which anything is thought, (iii) the reason for which anything is created, and (iv) the reason for which anything is done. Subsequently, the associating features are classified

respectively into *substance* features, *intellectual* features, *artefactual* features, and *procedural* features.

From the engineering point of view, this feature categorization could be used for classification of features in feature-based design. Accordingly, the *substance feature* includes all feature classes whose elements describe the characteristics of some existing or conceived substances, and define from which substances an object is established as in a real or virtual world (e.g. material features). The *Intellectual feature* category includes all feature classes whose elements make possible to reason about existing or conceived objects, and capture their paradigmatic, contextual, notional or implicative characteristics in the physical and cognitive world, respectively (e.g. conceptual features). The *Artefactual feature* category includes all feature classes whose elements are recognizable on existing or conceived objects in a real or imaginary world and make possible to create them and/or change their characteristics (e.g. form feature). The *procedural feature* category includes all feature classes whose elements contribute to capturing the process of existence and change, to conduct something in time, and to bring an object to an existence that is recognizable in a real or imaginary world (e.g. process feature).

In contrast with the features that describe components, system-level features cannot be nested in one of the mentioned categories. System-level features semantically are classified as very-high-level features. Consequently, as the highest level features, they cover all the mentioned purposes. System-level features capture existence of elements of the system, their architecture and attributes (substance feature); they capture paradigmatic system commonalities including contextual information for taxonomical reasons (conceptual feature). System-level features also capture morphological characteristics, parameters' relationships, and input/output transformations (artefactual feature), as well as temporal, and event-based state transitions, and procedural system operations (process feature).

2.6.3 Status of using high-level features in system modeling

In the beginning of the development of the feature technology, the feature-based design mostly was involved in parametric design. However, interests were drawn towards configuration, and conceptual design afterward [169], [205] and [186]. In the mid of 90s feature-based design started to evolve from detail (parametric) design towards conceptual and configuration product design [202], which could be considered as first attempts towards definition of higher level feature. According to granularity and application levels [181], features were divided into two categories of part and assembly features [187] [206]. Part features focus on geometric aspect. However, assembly features associate with mating relations among parts. In this viewpoint, system-level features are closer to assembly feature, as a higher level. Since the term *'high-level features'* is already reserved for other features, i.e., application and conceptual features, we consider system-level features as very-high-level features. System-level features associate with multiple pieces of information related to various traditional features, e.g., parts-, assemblies-, functional-, application-, and conceptual- features.

System-level features combine functional specifications with geometric ones, as intrinsically dependent parameters. Architecture and function aspects of systems considered as two complementing laminated layers [207] [208]. Traditionally, function aspect was used for design-by features. Among the first researches towards using feature technology for

designing by functions we can mention [209] [210]. There are also some cases that integrate function, structure and geometry in the feature-based product model (e.g. DICAD, GEKO) [211] [212]. However, these features have never been applied in system-level architecting and configuration. In many publications, system-level features are used with different meaning. The reason is that the system in various disciplines means differently. For example, system-level feature used as features of systems on a chip in [213]. The term system-level feature implies healthiness of complex mechanical systems in battlefields in [214], it refers to modeling online decision support system in [221]. By *system-level features*, we mean features corresponding with the systems (e.g. CPSs) that are composed of numerous off-the-shelf components, aggregated from hardware, software, and cyberware constituents, usually physically distributed, and open for other components to join or leave the system.

Theoretically, two categories of system-level features can be identified namely: *paradigmatic features* and *manifestation features*, which complement each other on different abstraction levels. We defined *paradigmatic system feature* (PSF) as a logically-based and physically-based abstraction of a system as whole that differentiates it from other comparable systems. These features are rooted in inherent characteristics of a system. A *system manifestation feature* (SMF) is related to a specific system in contrast to paradigmatic features that are about common characteristics of a group of systems that share some similarities. SMFs can be manipulated by designers and cannot violate overall paradigmatic system features.

The PSF category includes features that distinguish a particular type of system from other types of systems. They are generic and abstract, and do not have explicit relations with the engineering realization/implementation of systems. The advantage of introducing paradigmatic features is to avoid the often-cited 'blind man and the elephant' situation, i.e. to help avoid mistakes in identification of types of systems based on specific details and attributes [215]. Table 2-4 compares the paradigmatic system features CPSs with conventional technical systems.

The second category of system-level features includes features that exist only in a particular manifestation of a system. While PSFs are archetypal, SMFs are both phenotypic, and syntagmatic. Consequently, they form a part in larger syntactic units. A particular SMF may occur multiple times in a system in different instantiations (with different sets of values assigned to its parameters and annotations). SMFs may also appear on multiple aggregation levels. In the language of topology it means that various rougher and finer domain topologies can be defined over the overall domain of a system [215].

Considering the facts that PSFs and SMFs represent two different levels of abstractions and that they are conceptually independent from each other, a strong demarcation line have been drawn between paradigmatic features and manifestation features of systems. Notwithstanding the independence, paradigmatic features may be (strongly) reflected on and expressed by manifestation features [215]. SMFs can be represented by parameterized engineering models and uniquely characterized by spatiotemporal, architectural and operational attributes. Contrasting abstract paradigmatic features, SMFs are concrete and self-contained entities. Just as traditional e.g. form-, assembly-, manufacturing-, features,

SMFs can be represented by computational constructs containing a structured set of interrelated parameters, constraints, values and semantic annotations [215].

Table 2-4 Comparison of paradigmatic features in different systems

Paradigmatic features	Ordinary systems	Low-end CPSs	High-end CPSs
Sophistication of overall manifestation appearance	Conventional manifestation	Low-end manifestation	High-end manifestation
System complexity	Ordinarily compound	Linearly complex	Non-linearly complex
Functional objectives	Providing means	Providing utilities	Providing services
Overall architecture	Closed system boundary	Extendable system boundary	Open system boundary
Construction topology	Predefined fix architecture	Predefined modifiable architecture	Runtime dynamically architected
System design principle	Hierarchically composable	Heterogeneously composable	Heterachically compositional
Functional organization	Integrated structure	Distributed structure	Decentralized structure
Functional connectivity	Stand alone	Static networked	Dynamically networked
Operating principles	Execution and control	Sensing, computing and activating	Observing, reasoning, activating and motivating
Functional intelligence	Functionally determined	Functionally smart with limited awareness	Functionally autonomous and conscious
Readiness for operation	Condition activated	Alert in operation	Proactive in operation
Intelligent components	None or dedicated problem solvers	Individual agent-based reasoning	Cooperating agent swarm reasoning
System control	Manually controlled, Feedback controlled	Adaptive and self-adaptive control	Intelligent control according to goal driven operational scenarios
Level of composition of scales	Mono-scale composition	Dual-scale composition,	Multi-scale composition
Placement in context	Largely independent of context	Context sensitive (situated) operation	Context driven operation
Adaptability	Partial external modifiability	Self-adapting capabilities	Self-evolving capabilities
System relations	Unrelated or physical of relations	Semantic relations	Pragmatic relations
Heterogeneity	Mono-disciplinary system	Multi-disciplinary system	Trans-disciplinary system

2.6.4 Issues concerning computational handling of system-level features

From the perspective of computer technology, features are semantic representation of a system and its components that can describe, visualize and analyze different engineering aspects within whole product lifecycle [181]. In the context of feature-based design, *purpose* of features could be refined based on (i) the design approach which the features are employed for, (ii) the product that the features are used for modeling them, (iii) the modeling tool that supposed to handle the features, (iv) the users that operate the modeling tool, to design the product through the design approach. Figure 2-13 shows the differences between conventional feature-based design (CFBD) and system-level feature-based design (SLFBD) cases. The mentioned quadruple aspects have significance in computational perspective which is discussed in the following paragraphs.

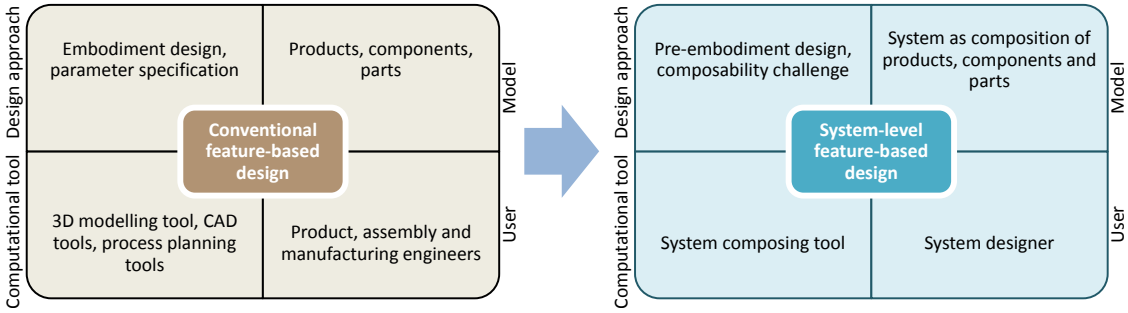


Figure 2-13 Comparison of conventional feature-based design and system-level feature-based design

Design approach determines (i) the objectives of designing action, (ii) the design phase that the modeling actions should be taken place, (iii) the kinds of challenges should be dealt with, and (iv) the kinds of parameters that should be specified. Feature technology from its beginning has been used for many design approaches such as design for assembly (DFA) [216], design for manufacturing (DFM) [217], and cost prediction [218]. In the most of the conventional design approaches, modeling actions take place in embodiment and detail design phases. However, in the system design approach, modeling is required in the pre-embodiment design phase. Since in system design approach, the components are off-the-shelf and designers should cope with composability challenges, the modeling should happen in the pre-embodiment (conceptualization and configuration) design phase, which the system designers should specify how the components connect to each other functionally and architecturally towards delivering the desired overall system output.

The subject of design in CFBD is products, components, and parts. However, in SLFBD, technical systems (i.e. CPS) are subjects of design and modeling. These technical systems are higher level compositions of products, components and parts. Particularly, the modeling subjects of CFBD (i.e. system on chip) are the components of the systems that are subjects of modeling in SLFBD. In other words, for computationally modeling of CPSs, models of components should be used as elements of the greater model. It is noteworthy that models of the components cannot be used directly, since they are not reflecting system-level knowledge specifications.

The most common tools used in CFBD are 3D modeling tools, optimization and analysis tools, and process planning tools, which are developed for specific tasks and involved in detailed level modifications of components. These tools exploit features developed for a particular *purpose* in a specific context. However, SLFBD and accordingly system-level tools work with system-level features (e.g. SMFs) which satisfy compound *purposes*. Therefore, SMF-based modeling tool should have a knowledge-intensive environment and working principles, which shares no similarity with other conventional modeling tools.

The users in CFBD tools are product engineers, assembly and manufacturing experts, and the experts who are involved in embodiment design of the products. These people are more likely experts in their discipline and fully aware of the functional and architectural aspects of the modeled components. However, system engineers who are the users of SLFBD tools are not fully aware of every functional and architectural aspects of all components used for composing the model. They use the components as black-boxes whose interfaces are known for them. Towards solving composability challenges, system design-

ers connect the interfaces of the components (most probably) without being aware of their working details. In other words, though working mechanisms, attributes and parameters should be known and analyzed computationally, CPS designers should not be overwhelmed with extra information (except about interfaces and constraints).

From a computational point of view, a specific domain of the system together with the operations performed by it creates an SMF. From a practical point of view, SMFs are supposed to be a kind of prefabricated 'components' that can be used in both architecture and operation design of systems. For this reason, both aspects should be captured in the computational information structures and models of SMFs. Moreover *multi-purpose* relationship among SMFs should be provided. Ma et al. in [219] reviewed the researches about relations in feature based models. They recognized two types of *geometric* and *non-geometric* relations. Geometric relations associate with topological and geometrical constraints [220], [221]. While non-geometric relations associate with e.g. sequential constraints [222], procedural constraints [223], function flow constraints [224] [198], machining constraints [225], etc., which all of them are valid in SMFs interfacing.

2.6.5 Conclusion: System-level features-based approach can solve the problem

As a main conclusion, it can be conceived that though there are indications that on the basis of system-level features, architecting of CPSs can be more effective, literature does not offer effective system-level solutions for CPS design. The closest research works [226] are focused on the embedded system design. Normally, in this context, system means system on a single chip which is integration of software and hardware constituents. It is truly mentioned that the main issue for embedded system design is lack of unified formal representation framework for software and hardware co-design [226]. The similar challenges are addressed in several researches (e.g. Ptolemy [227] [228], and Metropolis [157]), and are also valid for CPS design.

Other findings could be briefed as following: In knowledge-based systems, features could be stored as knowledge frames consisting of a data structure that represents an entity type, and collection of named 'slots' [229]. The slots can be filled by information and linked to other frames. Consequently, developing knowledge frames for SMFs is pointed as one of the objective of our research in the following research cycles. However, towards that end, we had to develop an underpinning theory that supports system-level feature-based design. System-level features have significate differences with traditional features. For system design, (i) form features are not important for modeling unless values of morphological parameters are needed for calculating values of other parameters, (ii) operation context should be considered in the modeling system due to its undeniable role in the real life, and (iii) HW, SW, and CW should be represented uniformly in the system-level in order to capture their relations. This solution solves the main obstacle of modeling of hybrid systems, which traditionally have been improperly solved through abstractions.

The term "abstraction levels" is frequently assigned to the levels that entail aggregation of components in a system [230]. We oppose this view by proposing a physical view. In particular, by connecting and aggregating components together we could have a system, and not by abstraction. In our view, in a system modeling, abstraction could be only used for visualizations and naming conventions. Figure 2-14 illustrates the point in which abstraction could be applied in the system modeling. As it can be seen, three semantic

conversions are applied in modeling of a system. In the first conversion the real world physical and computational object (e.g. CPS) is converted to modeling entities. These modeling entities are related to each other for representing real world relations semantically. The generated model is converted to visual entities for representing architecture and operations of the real world system. The last conversion is the cognitive perception of the users of the modeling tool. Applying abstraction in modeling is inevitable in order to simplification of the modeled complex system. If the required abstraction is applied in the first conversion, plenty of useful pieces of information will be lost. Our proposal is to apply physically-based conversion in the first conversion, and postpone the abstraction to the second conversion.

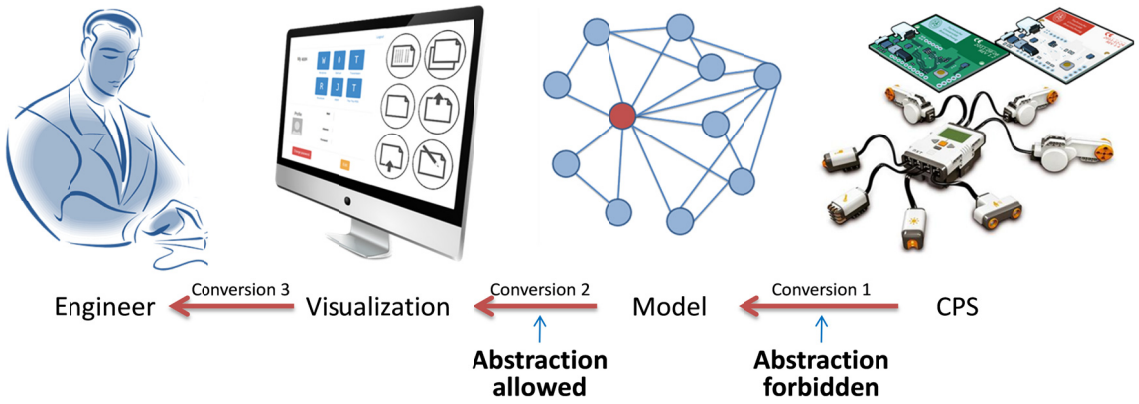


Figure 2-14 Information conversions in a system modeling

Besides the issues mentioned above, there are functional requirements of the feature-based design collected by Case and Gao [193] as summarization of research done by [231] and [232]: (i) database should capture sufficient information for all determined applications (general applicability), (ii) the probable needs of the designers for defining features should be supported (flexibility), (iii) the modeling environment should provide possibility for modification, manipulation, and creation of features and their relationship (modifiability), and (iv) interfacing and integration with other application software should be provided (integrity and extendibility). The above implications of the findings should be considered in an integral manner. The next subsection is focused on what should be considered in the required theoretical framework, computational methodologies, and tool development.

2.7 IMPLICATIONS OF THE FINDINGS AND CONCLUDING REMARKS

The first aim of the first research cycle was to explore the knowledge gap in *'designing CPSs'*. For this reason the phenomenon and the knowledge gap has been explored from different perspective (domains of study: *CPSs, design approaches, design principles, design supporting tools, and feature-based design*). The results could be briefed as following.

- Unique characteristics of CPSs create a new kind of challenges in their pre-embodiment design. The most important characteristics are: heterogeneity, distributed integrity, situation dependability, and complexity. They also have intensive relationship with their environment which implies a demand for adaptation or customization.

- The mentioned characteristics also create difficulty in adopting the conventional MC approaches for designing CPCDs. Our comparative study approved these mismatches, and showed that none of the traditional MC approaches sufficiently supports customization of CPCDs. However, our investigation revealed that EC is the most appropriate approach to adapt for CPCDs.
- Considering correlation between design principles of EC and features of TCDs, It can be concluded that new design principles should be developed based on the new features of CPCDs. The congruency of features of CPCDs and TCDs is useful for determining how the new design principles could be derived. Assuming that all required principles can be derived through the proposed mechanism, we decided to shift our focus to a more challenging issue implied by our explorative studies which is to support CPS designers in their design activities.
- In facing the emerging challenges of CPS design, the available supporting tools are not sufficiently effective due to these issues: firstly, since these tools are too much discipline related, they are more suitable for system analysis and engineering rather system-level conceptualization and design. Secondly, although they consider both functional and architectural attributes of CPSs, they mostly neglect physical and morphological properties in system modeling.
- We proposed concept of system-level feature to address both mentioned problems. Although the traditional feature technology lacks in many aspects such as appropriate functional formulation, it has the potential to support the requirements for system-level design and modeling. Accordingly, a heavy modification was predicted to adapt the traditional feature-based design for system-level feature-based design.

The five domains of study conducted us to a narrowed down topic of research which is "system-level- feature-based pre-embodiment design of CPSs". And our aim has established as "developing a theoretical and conceptual framework for supporting CPS modeling tool". Our survey to uncover the state of art about CPS modeling showed that many modeling environment that are currently used for modeling and simulation of CPSs, originally were created for system on chip development or embedded system design, which have been enhanced and adapted for CPS design. However in CPS design, design concerns are totally different. Accordingly, the approaches such as PBD and MDA are not useful in the new design context.

In contrast with embedded systems which designers tackle the compositionality challenges, in CPSs the main concern of design is composability issues. The composability challenge implies that the components are available, and the designers should find best components matches towards composing a system that delivers the desired functionality, performance, attributes and properties. We consider this challenge as system-level design, since the designers do not deal with component design. Accordingly, our modeling tool should support system-level design and address composability challenges.

There is an interesting trade-off issue concerning the use of computer-based modeling tools in modeling complex heterogeneous systems [133]: On one hand they need a domain-independent system-level modeling solution to be generally applicable. On the other hand, they need a domain-specific model having an underpinning mathematical model with strong properties (e.g. finite-state machines, data flow) to make formal analysis and component level consideration much easier. Our idea for solving this contradiction is

to nest the domain-specific properties (supported by ontological semantics) into features of components, and to push the generality issues to system-level model composition.

As it is discussed before and suggested by the literature, the early stages of design are the best for system-level design. We use the term 'pre-embodiment' design to refer these phase. It implies configuration and conceptual design of a system mostly by using off-the-shelf components. Accordingly, catalogues of components play an important role in system-level decision making. For instance, some components might be required which could not be found in catalogues. These components should be ordered to be designed, prototyped and fabricated by specific experts. In this situation, the required specifications are sent to the related engineers and experts, and the components are created, tested and sent back accordingly.

Another important issue in modeling CPSs is to incorporate physical and morphological properties of CPSs as they have tight relation with their embedding environment. Logical and analytical formulation of systems does not solely support the system-level design of CPSs. Since They are two sides of a coin, considering both functional and architectural aspects are equally important in CPS design [140]. The functional aspects are well-formulated in the current modeling tools; however the architecture aspects have barely taken into consideration.

Feature technology has the potential to serve us for tackling the mentioned problem. The chronological development of feature technology demonstrates a great extendibility towards different application. That is the reason we believe that it could be well extended to system-level design application by capturing both operational and architectural aspects of CPSs. The main challenge towards this end is lack of theoretical foundations to support our purpose. Subsequently, our research work was directed towards development of theoretical framework for supporting system-level feature-based CPS design. We decided to start our work with empirical study of the functional and architectural relations among components. The empirical study has supported our inductive reasoning towards theory formulation. The next chapter elaborates on our research towards development of a theoretical framework for system-level feature-based modeling.

2.8 REFERENCES

[1] Pourtalebi, S., Horváth, I., and Opiyo, E., (2013), "Multi-aspect study of mass customization in the context of cyber-physical consumer durables", Proceedings of the ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. V004T005A006-V004T005A006.

[2] Edwards, S., Lavagno, L., Lee, E.A., and Sangiovanni-Vincentelli, A., (1997), "Design of embedded systems: formal models, validation, and synthesis", Proceedings of the IEEE, Vol. 85 (3), pp. 366-389.

[3] Wolf, W.H., (1994), "Hardware-software co-design of embedded systems", Proceedings of the IEEE, Vol. 82 (7), pp. 967-989.

[4] Want, R., Fishkin, K.P., Gujar, A., and Harrison, B.L., (1999), "Bridging physical and virtual worlds with electronic tags", Conference on Human Factors in Computing Systems - Proceedings, pp. 370-377.

[5] Prekop, P., and Burnett, M., (2003), "Activities, context and ubiquitous computing", Computer Communications, Vol. 26 (11), pp. 1168-1176.

[6] Atzori, L., Iera, A., and Morabito, G., (2010), "The Internet of Things: A survey", Computer Networks, Vol. 54 (15), pp. 2787-2805.

[7] Foundation, N.S., (2010), "Cyber-Physical Systems (CPS)", Vol. 2013, NSF, NSF official web site, 2010.

[8] Foundation, N.S., (2013), "Cyber-Physical Systems (CPS)", Vol. 2013, NSF, NSF official web site, 2013.

[9] Poovendran, R., (2010), "Cyber–physical systems: Close encounters between two parallel worlds [point of view]", Proceedings of the IEEE, Vol. 98 (8), pp. 1363-1366.

[10] Tan, Y., Goddard, S., and Perez, L.C., (2008), "A prototype architecture for cyber-physical systems", ACM SIGBED Review, Vol. 5 (1), pp. 1-2.

[11] Sha, L., Gopalakrishnan, S., Liu, X., and Wang, Q., (2008), "Cyber-physical systems: A new frontier", Proceedings of the, Taichung, pp. 1-9.

[12] Anonymous, (2006), "Cyber-Physical Systems", (NSF), official web site of National Science Foundation, 2006.

[13] Miclea, L., and Sanislav, T., (2011), "About dependability in cyber-physical systems", Proceedings of the, Sevastopol, pp. 17-21.

[14] Shafi, Q., (2012), "Cyber physical systems security: A brief survey", Proceedings of the, Salvador, Bahia, pp. 146-150.

[15] Lee, E.A., (2008), "Cyber physical systems: Design challenges", Proceedings of the, Orlando, FL, pp. 363-369.

[16] Rajkumar, R., Lee, I., Sha, L., and Stankovic, J., (2010), "Cyber-physical systems: The next computing revolution", Proceedings of the, Anaheim, CA, pp. 731-736.

[17] Kim, K.D., and Kumar, P.R., (2012), "Cyber-physical systems: A perspective at the centennial", Proceedings of the IEEE, Vol. 100 (SPL CONTENT), pp. 1287-1308.

[18] Mindell, D.A., (2002), Between human and machine: feedback, control, and computing before cybernetics, Johns Hopkins University Press.

[19] Bennett, S., (2004), "Control and the digital computer: The early years", Measurement and Control, Vol. 37 (10), pp. 307-311.

[20] Lukasik, S.J., (2011), "Why the arpanet was built", IEEE Annals of the History of Computing, Vol. 33 (3), pp. 4-21.

[21] Wicks, A.L., and Kemerling, J.R., (2003), "A brief early history of wireless technology", Experimental Techniques, Vol. 27 (6), pp. 57-58.

[22] Gupta, S.K.S., Mukherjee, T., Varsamopoulos, G., and Banerjee, A., (2011), "Research directions in energy-sustainable cyberphysical systems", Sustainable Computing: Informatics and Systems, Vol. 1 (1), pp. 57-74.

[23] Gamage, T.T., McMillin, B.M., and Roth, T.P., (2010), "Enforcing information flow security properties in Cyber-Physical Systems: A generalized framework based on compensation", Proceedings of the, Seoul, pp. 158-163.

[24] Shi, J., Wan, J., Yan, H., and Suo, H., (2011), "A survey of Cyber-Physical Systems", Proceedings of the, Nanjing.

[25] Horváth, I., and Gerritsen, B.H., "Outlining nine major design challenges of open, decentralized, adaptive cuber-physical systems".

[26] Horváth, I., and Pourtalebi, S., (2015), "Fundamentals of a mereo-operandi theory to support transdisciplinary modelling and co-design of cyber-physical systems", Proceedings of the ASME 2015 International Design Engineering Technical Conferences, Boston, Massachusetts, USA.

[27] Lee, E.A., and Seshia, S.A., (2011), Introduction to embedded systems: A cyber-physical systems approach, Lee & Seshia.

[28] Shi, J., Wan, J., Yan, H., and Suo, H., (2011), "A survey of cyber-physical systems", Proceedings of the Wireless Communications and Signal Processing (WCSP), 2011 International Conference on, IEEE, Nanjing, pp. 1-6.

[29] Wan, J.F., Yan, H.H., Suo, H., and Li, F., (2011), "Advances in Cyber-Physical Systems Research", Ksii Transactions on Internet and Information Systems, Vol. 5 (11), pp. 1891-1908.

[30] Horváth, I., and Gerritsen, B.H.M., (2012), "Cyber-Physical Systems: concepts, technologies and implementation principles", Proceedings of the TMCE, Horváth, I., Rusák, Z., Albers, A., Behrendt, M. (Eds.), Organizing Committee of TMCE 2012, Germany, pp. 19-36.

[31] Marwedel, P., (2011), Embedded system design: Embedded systems foundations of cyber-physical systems, Springer Science+ Business Media.

[32] Maas, R., Maehle, E., and Großpietsch, K.E., (2012), "Applying the organic robot control architecture ORCA to cyber-physical systems", Proceedings of the, Cesme, Izmir, pp. 250-257.

[33] Bergé, J.-M., Levia, O., and Roulliard, J., (1995), High-level system modeling: specification languages, Kluwer Academic Publishers.

[34] Wang, X.L., Huang, H.B., Deng, S., and Chen, L.N., (2012), "A service-oriented architecture framework for cyber-physical systems", Vol. 126 LNEE, Changchun, 2012, pp. 671-676.

[35] Horváth, I., (2012), "Beyond advanced mechatronics: new design challenges of Social-Cyber-Physical systems".

[36] Kobetski, A., and Axelsson, J., (2012), " Federated Embedded Systems – a review of the literature in related fields, " Technical Report, SWEDISH INSTITUTE OF COMPUTER SCIENCE, Software and Systems Engineering Laboratory, 2012, pp. 1-22.

[37] Correll, N., Arechiga, N., Bolger, A., Bollini, M., Charrow, B., Clayton, A., Dominguez, F., Donahue, K., Dyar, S., and Johnson, L., (2009), "Building a distributed robot garden", Proceedings of the, Institute of Electrical and Electronics Engineers.

[38] Thiagarajan, A., Ravindranath, L., LaCurts, K., Madden, S., Balakrishnan, H., Toledo, S., and Eriksson, J., (2009), "VTrack: Accurate, energy-aware road traffic delay estimation using mobile phones", Proceedings of the, Berkeley, CA, pp. 85-98.

[39] Horváth, I., (2014), "What the Design Theory of Social-Cyber-Physical Systems Must Describe, Explain and Predict?", in: An Anthology of Theories and Models of Design, Springer, pp. 99-120.

[40] Swaminathan, J.M., (2001), "Enabling customization using standardized operations", California Management Review (3), pp. 125-135.

[41] Amine, A., and Cadenat, S., (2003), "Efficient retailer assortment: a consumer choice evaluation perspective", International Journal of Retail & Distribution Management, Vol. 31 (10), pp. 486-497.

[42] Ahmad, S., and Schroeder, R.G., (2002), "Refining the product-process matrix", International Journal of Operations & Production Management, Vol. 22 (1), pp. 103-124.

[43] Kumar, A., (2004), "Mass customization: metrics and modularity", International Journal of Flexible Manufacturing Systems, Vol. 16 (4), pp. 287-311.

[44] Li, Y., (2010), "Multi-platform strategy and product family design".

[45] Jiao, J.R., Simpson, T.W., and Siddique, Z., (2007), "Product family design and platform-based product development: a state-of-the-art review", Journal of intelligent Manufacturing, Vol. 18 (1), pp. 5-29.

[46] Davis, S., (1987), Future Perfect, Addison - Wesley Publishing, Reading MA.

[47] Hart, C.W.L., (1995), "Mass customization: Conceptual underpinnings, opportunities and limits", International Journal of Service Industry Management, Vol. 6 (2), pp. 36-45.

[48] Pine, B.J., (1993), Mass Customization: The New Frontier in Business Competition, Harvard Business School Press, Boston.

[49] Tseng, M.M., and Jiao, J., (1996), "Design for mass customization", CIRP Annals - Manufacturing Technology, Vol. 45 (1), pp. 153-156.

[50] Worren, N., Moore, K., and Cardona, P., (2002), "Modularity, strategic flexibility, and firm performance: A study of the home appliance industry", Strategic Management Journal, Vol. 23 (12), pp. 1123-1140.

[51] Hawkins, B., (2003), "An automotive perspective on mass customisation - Can we virtually build to order?", IEE Colloquium (Digest), Vol. 3-10031, pp. 9-12.

[52] Fiore, A.M., Lee, S.E., and Kunz, G., (2004), "Individual differences, motivations, and willingness to use a mass customization option for fashion products", European Journal of Marketing, Vol. 38 (7), pp. 835-849.

[53] Lo, K.P.Y., (2012), "Service design strategies for customization implications of conflicting emotions and concerns", Proceedings of the, London.

[54] Boysen, N., Fliedner, M., and Scholl, A., (2007), "A classification of assembly line balancing problems", European Journal of Operational Research, Vol. 183 (2), pp. 674-693.

[55] Petersen, T.D., Nielsen, K., and Jørgensen, K.A., (2010), "Classification of customisation means for product configuration", Proceedings of the, Goteborg, pp. 267-274.

[56] Park, Y., and Nahm, A.Y., (2011), "Classification of mass customisation: A socio-technical system perspective", International Journal of Services and Operations Management, Vol. 8 (3), pp. 322-334.

[57] Bayou, M.E., and Reinstein, A., (2003), "A management accounting taxonomy for the mass customization approach", Vol. 11, 2003, pp. 169-189.

[58] Piller, F., (2006), "Mass Customization Success factors and challenges to co-create value with your customers", Proceedings of the International Conference of Mass Customization, ICMC.

[59] Duray, R., Ward, P.T., Milligan, G.W., and Berry, W.L., (2000), "Approaches to mass customization: Configurations and empirical validation", Journal of Operations Management, Vol. 18 (6), pp. 605-625.

[60] Spring, M., and Dalrymple, J.F., (2000), "Product customisation and manufacturing strategy", International Journal of Operations & Production Management, Vol. 20 (4), pp. 441-467.

[61] MacCarthy, B., Brabazon, P.G., and Bramham, J., (2003), "Fundamental modes of operation for mass customization", International Journal of Production Economics, Vol. 85 (3), pp. 289-304.

[62] Da Silveira, G., Borenstein, D., and Fogliatto, F.S., (2001), "Mass customization: Literature review and research directions", International Journal of Production Economics, Vol. 72 (1), pp. 1-13.

[63] Tien, J.M., Krishnamurthy, A., and Yasar, A., (2004), "A taxonomic approach to real-time mass customization", Proceedings of the, Vol. 5, The Hague, pp. 4238-4243.

[64] Tien, J.M., (2012), "The next industrial revolution: Integrated services and goods", Journal of Systems Science and Systems Engineering, Vol. 21 (3), pp. 257-296.

[65] Sharma, D., (1987), "Manufacturing strategy: an empirical analysis", Ohio State University.

[66] Pine, J., (1993), "Mass customizing products and services", Planning Review, Vol. 21 (4), p. 8.

[67] Fisher, M., Jain, A., and MacDuffie, J.P., (1995), Strategies for product variety: lessons from the auto industry, Oxford University Press, New York.

[68] Lampel, J., and Mintzberg, H., (1996), "Customizing customization", Sloan Management Review, Vol. 38 (1), pp. 21-&.

[69] Ross, A., (1996), "Selling uniqueness", Manufacturing Engineer, Vol. 75 (6), p. 260.

[70] Spira, J., (1996), "Mass customization through training at Lutron Electronics", Computers in Industry, Vol. 30 (3), p. 3.

[71] Gilmore, J.H., and Pine 2nd, B.J., (1997), "The four faces of mass customization", Harvard business review, Vol. 75 (1), pp. 91-101.

[72] Tien, J., Krishnamurthy, A., and Yasar, A., (2004), "Towards real-time customized management of supply and demand chains", Journal of Systems Science and Systems Engineering, Vol. 13 (3), pp. 257-278.

[73] Joneja, A., and Lee, N.K.S., (1998), "Automated configuration of parametric feeding tools for mass customization", Computers and Industrial Engineering, Vol. 35 (3-4), pp. 463-466.

[74] Tseng, M.M., and Jiao, J.X., (1997), "Case-based evolutionary design for mass customization", Computers & Industrial Engineering, Vol. 33 (1-2), pp. 319-323.

[75] Domeshek, E.A., and Kolodner, J.L., (1991), "Toward a case-based aid for conceptual design", International Journal of Expert Systems, Vol. 4 (2), pp. 201-220.

[76] Suh, N.P., (1998), "Axiomatic Design Theory for Systems", Research in Engineering Design, Vol. 10 (4), pp. 189-209.

[77] Bi, Z.M., and Zhang, W.J., (2001), "Modularity technology in manufacturing: Taxonomy and issues", International Journal of Advanced Manufacturing Technology, Vol. 18 (5), pp. 381-390.

[78] Jiao, J., Tseng, M.M., Duffy, V.G., and Lin, F., (1998), "Product family modeling for mass customization", Computers and Industrial Engineering, Vol. 35 (3-4), pp. 495-498.

[79] Koomsap, P., (2013), "Design by customer: concept and applications", Journal of intelligent Manufacturing, Vol. 24 (2), pp. 295-311.

[80] Tseng, M.M., and Jiao, J., (1998), "Concurrent design for mass customization", Business Process Management Journal, Vol. 4 (1), pp. 10-24.

[81] Tseng, M.M., Jiao, R.J., and Wang, C., (2010), "Design for mass personalization", CIRP Annals - Manufacturing Technology, Vol. 59 (1), pp. 175-178.

[82] Tseng, M.M., Lei, M., and Su, C., (1997), "Collaborative control system for mass customization manufacturing", CIRP Annals - Manufacturing Technology, Vol. 46 (1), pp. 373-376.

[83] Lee, H.L., (1996), "Effective inventory and service management through product and process redesign", Operations Research, Vol. 44 (1), pp. 151-159.

[84] Cross, N., (2007), "From a design science to a design discipline: Understanding designerly ways of knowing and thinking", Design research now, pp. 41-54.

[85] Piller, F., Schubert, P., Koch, M., and Möslein, K., (2005), "Overcoming Mass Confusion: Collaborative Customer Co Design in Online Communities", Journal of Computer Mediated Communication, Vol. 10 (4), pp. 00-00.

[86] Huffman, C., and Kahn, B.E., (1998), "Variety for sale: Mass customization or mass confusion?", Journal of Retailing, Vol. 74 (4), pp. 491-513.

[87] Matzler, K., Stieger, D., and Füller, J., (2011), "Consumer Confusion in Internet-Based Mass Customization: Testing a Network of Antecedents and Consequences", Journal of Consumer Policy, Vol. 34 (2), pp. 231-247.

[88] Chen, Z., and Wang, L., (2010), "Personalized product configuration rules with dual formulations: A method to proactively leverage mass confusion", Expert Systems with Applications, Vol. 37 (1), pp. 383-392.

[89] Feitzinger, E., and Lee, H.L., (1997), "Mass customization at Hewlett-Packard: the power of postponement", Harvard business review, Vol. 75, pp. 116-123.

[90] Boynton, A.C., Victor, B., and Pine Ii, B.J., (1993), "New competitive strategies: challenges to organizations and information technology", IBM Systems Journal, Vol. 32 (1), pp. 40-64.

[91] Alford, D., Sackett, P., and Nelder, G., (2000), "Mass customization - an automotive perspective", International Journal of Production Economics, Vol. 65 (1), pp. 99-110.

[92] Sigala, M., (2006), "A Framework for developing and evaluating mass customisation strategies for online travel companies", Information and Communication Technologies in Tourism 2006, pp. 112-124.

[93] Pourtalebi, S., Horváth, I., and Opiyo, E., (2014), "New features imply new principles? deriving design principles for mass customization of cyber-physical consumer durables", Proceedings of the TMCE, Budapest, Hungary, pp. 95-108.

[94] Singh, V.K., and Jain, R., (2009), "Situation based control for cyber-physical environments", Proceedings of the Military Communications Conference, 2009. MILCOM 2009. IEEE, pp. 1-7.

[95] Wu, F.-J., Kao, Y.-F., and Tseng, Y.-C., (2011), "From wireless sensor networks towards cyber physical systems", Pervasive and Mobile Computing, Vol. 7 (4), pp. 397-413.

[96] Horváth, I., and Gerritsen, B.H., (2013), "Outlining nine major design challenges of open, decentralized, adaptive cuber-physical systems", Proceedings of the.

[97] Miorandi, D., Sicari, S., De Pellegrini, F., and Chlamtac, I., (2012), "Internet of things: Vision, applications and research challenges", Ad Hoc Networks, Vol. 10 (7), pp. 1497-1516.

[98] Asan, E., Albrecht, O., and Bilgen, S., (2014), "Handling Complexity in System of Systems Projects – Lessons Learned from MBSE Efforts in Border Security Projects", in: Complex Systems Design & Management, Aiguier, M., Boulanger, F., Krob, D., Marchal, C. (Eds.), Springer International Publishing, pp. 281-299.

[99] Guo, L., (2015), "From MetroII to Metronomy, Designing Contract-based Function-Architecture Co-simulation Framework for Timing Verification of Cyber-Physical Systems", Electrical Engineering and Computer Sciences University of California at Berkeley, 2015.

[100] Dumitrescu, C., Tessier, P., Salinesi, C., Gérard, S., Dauron, A., and Mazo, R., (2014), "Capturing Variability in Model Based Systems Engineering", in: Complex Systems Design & Management, Aiguier, M., Boulanger, F., Krob, D., Marchal, C. (Eds.), Springer International Publishing, pp. 125-139.

[101] Tessier, P., Servat, D., and Gérard, S., (2008), "Variability Management on Behavioral Models", Proceedings of the VaMoS, pp. 121-130.

[102] Lee, J., and Muthig, D., (2006), "Feature-oriented variability management in product line engineering", Communications of the ACM, Vol. 49 (12), pp. 55-59.

[103] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., and Peterson, A.S., (1990), "Feature-oriented domain analysis (FODA) feasibility study", DTIC Document, 1990.

[104] Sinnema, M., and Deelstra, S., (2007), "Classifying variability modeling techniques", Information and Software Technology, Vol. 49 (7), pp. 717-739.

[105] Selic, B., (2003), "The pragmatics of model-driven development", IEEE software (5), pp. 19-25.

[106] Schmidt, D.C., (2006), "Guest editor's introduction: Model-driven engineering", Computer, Vol. 39 (2), pp. 0025-0031.

[107] Paterno, F., (2012), Model-based design and evaluation of interactive applications, Springer Science & Business Media.

[108] Derler, P., Lee, E.A., and Vincentelli, A.S., (2012), "Modeling Cyber-Physical Systems", Proceedings of the IEEE, Vol. 100 (1), pp. 13-28.

[109] Ragavan, S.V., Shanmugavel, M., Ganapathy, V., and Shirinzadeh, B., (2015), "Unified meta-modeling framework using bond graph grammars for conceptual modeling", Robotics and Autonomous Systems.

[110] Kleppe, A.G., Warmer, J.B., and Bast, W., (2003), MDA explained: the model driven architecture: practice and promise, Addison-Wesley Professional.

[111] Keutzer, K., Rabaey, J.M., and Sangiovanni-Vincentelli, A., (2000), "System-level design: orthogonalization of concerns and platform-based design", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Vol. 19 (12), pp. 1523-1543.

[112] Sangiovanni-Vincentelli, A., (2002), "Defining platform-based design", EEDesign of EETimes.

[113] Hewitt, C., (1977), "Viewing control structures as patterns of passing messages", Artificial intelligence, Vol. 8 (3), pp. 323-364.

[114] Lee, E.A., (2003), "Model-driven development-from object-oriented design to actor-oriented design", Proceedings of the Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago, Citeseer.

[115] Vargas-Hernandez, N., Shah, J.J., and Lacroix, Z., (2003), "Development of a computer aided conceptual design tool for complex electromechanical systems", Proceedings of the Proceedings of 2003 AAAI Spring Symposium on Computational Synthesis.

[116] Woodcock, J., Bryans, J., Canham, S., and Foster, S., (2014), "The COMPASS Modelling Language: Timed Semantics in UTP", Communicating Process Architectures.

[117] Arnold, A., Boyer, B., and Legay, A., (2013), "Contracts and Behavioral Patterns for SoS: The EU IP DANSE approach", arXiv preprint arXiv:1311.3631.

[118] Sztipanovits, J., and Karsai, G., (1997), "Model-integrated computing", Computer, Vol. 30 (4), pp. 110-111.

[119] Ong, C.-M., (1998), Dynamic simulation of electric machinery: using MATLAB/SIMULINK, Prentice hall PTR Upper Saddle River, NJ.

[120] Gray, M.A., (2007), "Discrete event simulation: A review of SimEvents", Computing in Science & Engineering, Vol. 9 (6), pp. 62-66.

[121] Stürmer, I., Kreuz, I., Schäfer, W., and Schürr, A., (2007), "The MATE Approach: Enhanced Simulink® and Stateflow® Model Transformation", Proceedings of the Proceedings of MathWorks Automotive Conference, Dearborn (MI), USA (June 2007).

[122] Bishop, R.H., (2007), LabVIEW 8, Pearson.

[123] Kalkman, C.J., (1995), "LabVIEW: a software system for data acquisition, data analysis, and instrument control", Journal of clinical monitoring, Vol. 11 (1), pp. 51-58.

[124] Elmqvist, H., and Mattsson, S.E., (1997), "An introduction to the physical modeling language Modelica", Proceedings of the Proceedings of the 9th European Simulation Symposium, ESS, Vol. 97, pp. 19-23.

[125] Marquis-Favre, W., Bideaux, E., and Scavarda, S., (2006), "A planar mechanical library in the AMESim simulation software. Part II: Library composition and illustrative example", Simulation Modelling Practice and Theory, Vol. 14 (2), pp. 95-111.

[126] Brück, D., Elmqvist, H., Mattsson, S.E., and Olsson, H., (2002), "Dymola for multi-engineering modeling and simulation", Proceedings of the Proceedings of Modelica, Citeseer.

[127] Bogodorova, T., Sabate, M., Leon, G., Vanfretti, L., Halat, M., Heyberger, J.B., and Panciatici, P., (2013), "A modelica power system library for phasor time-domain simulation", Proceedings of the Innovative Smart Grid Technologies Europe (ISGT EUROPE), 2013 4th IEEE/PES, IEEE, pp. 1-5.

[128] Hřebíček, J., and Řezáč, M., (2008), "Modelling with Maple and MapleSim", Proceedings of the 22nd European Conference on Modelling nad Simulation ECMS 2008 Proceedings, pp. 60-66.

[129] Bhattacharya, P., Welakwe, N.S., Makanaboyina, R., and Chimalakonda, A., (2006), "Integration of CATIA with Modelica", Proceedings of the proceedings of Modelica conference.

[130] Fritzson, P., Aronsson, P., Pop, A., Lundvall, H., Nystrom, K., Saldamli, L., Broman, D., and Sandholm, A., (2006), "OpenModelica-A free open-source environment for system modeling, simulation, and teaching", Proceedings of the IEEE International Symposium on Computer-Aided Control Systems Design, pp. 1588-1595.

[131] Tiller, M., (2012), Introduction to physical modeling with Modelica, Springer Science & Business Media.

[132] Fritzson, P., and Engelson, V., (1998), "Modelica—A unified object-oriented language for system modeling and simulation", in: ECOOP'98—Object-Oriented Programming, Springer, pp. 67-90.

[133] Sangiovanni-Vincentelli, A., (2007), "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design", Proceedings of the IEEE, Vol. 95 (3), pp. 467-506.

[134] Lee, E.A., Neuendorffer, S., and Wirthlin, M.J., (2003), "Actor-oriented design of embedded hardware and software systems", Journal of circuits, systems, and computers, Vol. 12 (03), pp. 231-260.

[135] Lee, E.A., (1999), "Modeling concurrent real-time processes using discrete events", Annals of Software Engineering, Vol. 7 (1-4), pp. 25-45.

[136] Liu, J., (1998), "Continuous time and mixed-signal simulation in Ptolemy II", Proceedings of the Dept. of EECS, University of California, Berkeley, CA, Citeseer.

[137] Lee, E.A., and Tripakis, S., (2010), "Modal Models in Ptolemy", Proceedings of the EOOLT, pp. 11-21.

[138] Lee, E., and Parks, T.M., (1995), "Dataflow process networks", Proceedings of the IEEE, Vol. 83 (5), pp. 773-801.

[139] Akkaya, I., Liu, Y., and Lee, E., (2015), "Modeling and Simulation of Network Aspects for Distributed Cyber-Physical Energy Systems", in: Cyber Physical Systems Approach to Smart Electric Power Grid, Khaitan, S.K., McCalley, J.D., Liu, C.C. (Eds.), Springer Berlin Heidelberg, pp. 1-23.

[140] Davare, A., Densmore, D., Guo, L., Passerone, R., Sangiovanni-Vincentelli, A.L., Simalatsar, A., and Zhu, Q., (2013), "metroll: A design environment for cyber-physical systems", ACM Trans. Embed. Comput. Syst., Vol. 12 (1s), pp. 1-31.

[141] Liangpeng, G., Qi, Z., Nuzzo, P., Passerone, R., Sangiovanni-Vincentelli, A., and Lee, E.A., (2014), "Metronomy: A function-architecture co-simulation framework for timing verification of cyber-physical systems", Proceedings of the Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on, pp. 1-10.

[142] Gérard, S., Espinoza, H., Terrier, F., and Selic, B., (2010), "6 Modeling Languages for Real-Time and Embedded Systems", in: Model-Based Engineering of Embedded Real-Time Systems, Giese, H., Karsai, G., Lee, E., Rumpe, B., Schätz, B. (Eds.), Vol. 6100, Springer Berlin Heidelberg, pp. 129-154.

[143] Fontoura, M., Pree, W., and Rumpe, B., (2000), The UML profile for framework architectures, Addison-Wesley Longman Publishing Co., Inc.

[144] Mallet, F., and André, C., (2009), "On the semantics of UML/MARTE clock constraints", Proceedings of the Object/Component/Service-Oriented Real-Time Distributed Computing, 2009. ISORC'09. IEEE International Symposium on, IEEE, pp. 305-312.

[145] Cancila, D., Zaatiti, H., and Passerone, R., (2015), "Cyber-Physical System and Contract-Based Design: A Three Dimensional View", Proceedings of the Proceedings of the WESE'15: Workshop on Embedded and Cyber-Physical Systems Education, ACM, Amsterdam, Netherlands, pp. 1-4.

[146] Nuzzo, P., Sangiovanni-Vincentelli, A.L., and Murray, R.M., (2015), "Methodology and Tools for Next Generation Cyber-Physical Systems: The iCyPhy Approach", INCOSE International Symposium, Vol. 25 (1), pp. 235-249.

[147] Nuzzo, P., Sangiovanni-Vincentelli, A.L., Bresolin, D., Geretti, L., and Villa, T., (2015), "A Platform-Based Design Methodology With Contracts and Related Tools for the Design of Cyber-Physical Systems", Proceedings of the IEEE, Vol. 103 (11), pp. 2104-2132.

[148] Fitzgerald, J., Larsen, P., and Woodcock, J., (2014), "Foundations for Model-Based Engineering of Systems of Systems", in: Complex Systems Design & Management, Aiguier, M., Boulanger, F., Krob, D., Marchal, C. (Eds.), Springer International Publishing, pp. 1-19.

[149] DeLaurentis, D., (2005), "A taxonomy-based perspective for systems of systems design methods", Proceedings of the Systems, Man and Cybernetics, 2005 IEEE International Conference on, Vol. 1, IEEE, pp. 86-91.

[150] Miyazawa, A., Lima, L., and Cavalcanti, A., (2013), "Formal models of sysml blocks", in: Formal Methods and Software Engineering, Springer, pp. 249-264.

[151] Panda, P.R., (2001), "SystemC-a modeling platform supporting multiple design abstractions", Proceedings of the System Synthesis, 2001. Proceedings. The 14th International Symposium on, IEEE, pp. 75-80.

[152] Gajski, D.D., Zhu, J., Dömer, R., Gerstlauer, A., and Zhao, S., (2012), SPECC: Specification Language and Methodology, Springer Science & Business Media.

[153] Resmerita, S., Derler, P., Pree, W., and Naderlinger, A., (2010), "5 Modeling and Simulation of TDL Applications", in: Model-Based Engineering of Embedded Real-Time Systems, Giese, H., Karsai, G., Lee, E., Rumpe, B., Schätz, B. (Eds.), Vol. 6100, Springer Berlin Heidelberg, pp. 107-128.

[154] Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Abdi, S., and Gajski, D.D., (2008), "System-on-chip environment: a SpecC-based framework for heterogeneous MPSoC design", EURASIP Journal on Embedded Systems, Vol. 2008, p. 5.

[155] Pimentel, A.D., (2008), "The artemis workbench for system-level performance evaluation of embedded systems", International journal of embedded systems, Vol. 3 (3), pp. 181-196.

[156] Ha, S., Kim, S., Lee, C., Yi, Y., Kwon, S., and Joo, Y.-P., (2007), "PeaCE: A hardware-software codesign environment for multimedia embedded systems", ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 12 (3), p. 24.

[157] Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., and Sangiovanni-Vincentelli, A., (2003), "Metropolis: An integrated electronic system design environment", Computer, Vol. 36 (4), pp. 45-52.

[158] Mahadevan, S., Virk, K., and Madsen, J., (2007), "ARTS: A SystemC-based framework for multiprocessor Systems-on-Chip modelling", Design Automation for Embedded Systems, Vol. 11 (4), pp. 285-311.

[159] Albarello, N., and Kim, H., (2014), "Application of an MBSE Approach for the Integration of Multidisciplinary Design Processes", in: Complex Systems Design & Management, Aiguier, M., Boulanger, F., Krob, D., Marchal, C. (Eds.), Springer International Publishing, pp. 85-96.

[160] Kim, H., Fried, D., and Menegay, P., (2012), "Connecting SysML Models with Engineering Analyses to Support Multidisciplinary System Development", American Institute of Aeronautics and Astronautics.

[161] Fritzson, P., (2010), Principles of object-oriented modeling and simulation with Modelica 2.1, John Wiley & Sons.

[162] Liu, J., Wu, B., Liu, X., and Lee, E.A., (1999), "Interoperation of heterogeneous CAD tools in Ptolemy II", Proceedings of the Design, Test, and Microfabrication of MEMS/MOEMS, International Society for Optics and Photonics, pp. 249-258.

[163] Szykman, S., Fenves, S.J., Keirouz, W., and Shooter, S.B., (2001), "A foundation for interoperability in next-generation product development systems", Computer-Aided Design, Vol. 33 (7), pp. 545-559.

[164] Ando, Y., (2014), "Efficient System-Level Design Space Exploration for Large-Scale Embedded Systems", 2014.

[165] Hudak, P., (1997), "The Promise of Domain-Specific Languages", Proceedings of the keynote address from Usenix DSL Conf., Yale Univ., New Haven, Conn.

[166] Karsai, G., Sztipanovits, J., Ledeczi, A., and Bapty, T., (2003), "Model-integrated development of embedded software", Proceedings of the IEEE, Vol. 91 (1), pp. 145-164.

[167] Shah, J.J., (1991), "Assessment of features technology", Computer-Aided Design, Vol. 23 (5), pp. 331-343.

[168] Turner, G.P., and Anderson, D., (1988), An object-oriented approach to interactive, feature-based design for quick turnaround manufacturing, Purdue University, Schools of Engineering, Engineering Research Center for Intelligent Manufacturing Systems.

[169] Salomons, O.W., van Houten, F.J.A.M., and Kals, H.J.J., (1993), "Review of research in feature-based design", Journal of Manufacturing Systems, Vol. 12 (2), pp. 113-132.

[170] Joshi, S., and Chang, T.-C., (1990), "Feature extraction and feature based design approaches in the development of design interface for process planning", Journal of intelligent Manufacturing, Vol. 1 (1), pp. 1-15.

[171] Shah, J.J., and Mäntylä, M., (1995), Parametric and feature-based CAD/CAM: concepts, techniques, and applications, John Wiley & Sons.

[172] Gindy, N., (1989), "A hierarchical structure for form features", The International Journal of Production Research, Vol. 27 (12), pp. 2089-2103.

[173] Giacometti, F., and Chang, T.-C., (1990), "A model for parts, assemblies and tolerances", Proceedings of the Preprints of the first IFIP WG 5.2 Workshop on Design for Manufacturing.

[174] LUBY, S., DIXON, J., and SIMMONS, M., (1986), "Creating and Using a Feature Database, Computers in Mech", Computers in Mechanical Engineering, pp. 25-33.

[175] Shah, J.J., (1990), "Philosophical development of form feature concept", CAM-I report P-90-PM-02, pp. 55-70.

[176] de Gruijter, J.J., and ter Braak, C.J.F., (1990), "Model-free estimation from spatial samples: A reappraisal of classical sampling theory", Mathematical Geology, Vol. 22 (4), pp. 407-415.

[177] Allada, V., (1995), "Feature-based modelling approaches for integrated manufacturing: State-of-the-art survey and future research directions", International Journal of Computer Integrated Manufacturing, Vol. 8 (6), pp. 411-440.

[178] Xie, Y., and Ma, Y., (2015), "Design of a multi-disciplinary and feature-based collaborative environment for chemical process projects", Expert Systems with Applications, Vol. 42 (8), pp. 4149-4166.

[179] Bidarra, R., and Bronsvoort, W.F., (2000), "Semantic feature modelling", Computer-Aided Design, Vol. 32 (3), pp. 201-225.

[180] Syaimak, A., and Axinte, D., (2009), "An approach of using primitive feature analysis in manufacturability analysis systems for micro-milling/drilling", International Journal of Computer Integrated Manufacturing, Vol. 22 (8), pp. 727-744.

[181] Yin, C.G., and Ma, Y.S., (2012), "Parametric feature constraint modeling and mapping in product development", Advanced Engineering Informatics, Vol. 26 (3), pp. 539-552.

[182] Burgett, S.R., Bush, R.T., Sastry, S.S., and Sequin, C.H., (1995), "Mechanical design synthesis from sparse feature-based input", Proceedings of the Smart Structures & Materials' 95, International Society for Optics and Photonics, pp. 280-291.

[183] Poldermann, B., and Horváth, I., (1996), "Surface design based on parametrized surface features", Proceedings of the Proc. Int. Symposium on Tools and Methods for Concurrent Engineering, Institute of Machine Design, Budapest, pp. 432-446.

[184] Vergeest, J.S., Horváth, I., and Spanjaard, S., (2001), "Parameterization of freeform features", Proceedings of the Shape Modeling and Applications, SMI 2001 International Conference on., IEEE, pp. 20-29.

[185] Chu, C.-C.P., and Gadh, R., (1996), "Feature-based approach for set-up minimization of process design from product design", Computer-Aided Design, Vol. 28 (5), pp. 321-332.

[186] Brunetti, G., and Golob, B., (2000), "A feature-based approach towards an integrated product model including conceptual design information", Computer-Aided Design, Vol. 32 (14), pp. 877-887.

[187] Ma, Y.-S., Britton, G.A., Tor, S., Jin, L.-Y., Chen, G., and Tang, S.-H., (2004), "Design of a feature-object-based mechanical assembly library", Computer-Aided Design and Applications, Vol. 1 (1-4), pp. 397-403.

[188] Jin, Y., and Li, W., (2007), "Design concept generation: a hierarchical coevolutionary approach", Journal of Mechanical Design, Vol. 129 (10), pp. 1012-1022.

[189] Igwe, P.C., Knopf, G.K., and Canas, R., (2008), "Developing alternative design concepts in VR environments using volumetric self-organizing feature maps", Journal of intelligent Manufacturing, Vol. 19 (6), pp. 661-675.

[190] Bohm, M.R., Vucovich, J.P., and Stone, R.B., (2008), "Using a design repository to drive concept generation", Journal of Computing and Information Science in Engineering, Vol. 8 (1), p. 014502.

[191] Chu, C.-H., Wu, P.-H., and Hsu, Y.-C., (2009), "Multi-agent collaborative 3D design with geometric model at different levels of detail", Robotics and Computer-Integrated Manufacturing, Vol. 25 (2), pp. 334-347.

[192] Lampka, K., Perathoner, S., and Thiele, L., (2013), "Component-based system design: analytic real-time interfaces for state-based component implementations", International Journal on Software Tools for Technology Transfer, Vol. 15 (3), pp. 155-170.

[193] Case, K., and Gao, J., (1993), "Feature technology: an overview", International Journal of Computer Integrated Manufacturing, Vol. 6 (1-2), pp. 2-12.

[194] Horváth, I., Pulles, J., Bremer, A., and Vergeest, J., (1998), "Towards an ontology-based definition of design features", Proceedings of the Proceeding of the Workshop on mathematical foundations for features in computer aided design, engineering, and manufacturing, SIAM, pp. 1-12.

[195] Kim, K.-Y., Manley, D.G., and Yang, H., (2006), "Ontology-based assembly design and information sharing for collaborative product development", Computer-Aided Design, Vol. 38 (12), pp. 1233-1250.

[196] Tessier, S., and Wang, Y., (2013), "Ontology-based feature mapping and verification between CAD systems", Advanced Engineering Informatics, Vol. 27 (1), pp. 76-92.

[197] Kim, O., Jayaram, U., Jayaram, S., and Zhu, L., (2009), "An ontology mapping application using a shared ontology approach and a bridge ontology", Proceedings of the ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. 431-441.

[198] Brunetti, G., and Grimm, S., (2005), "Feature ontologies for the explicit representation of shape semantics", International journal of computer applications in technology, Vol. 23 (2-4), pp. 192-202.

[199] Chen, G., Ma, Y.S., Thimm, G., and Tang, S.H., (2006), "Associations in a Unified Feature Modeling Scheme", Journal of Computing and Information Science in Engineering, Vol. 6 (2), pp. 114-126.

[200] Butterfield, W., Green, M., Scott, D., and Stoker, W., (1986), "Part features for process planning", CAM-I report R-85-PPP-03.

[201] Shah, J.J., (1988), "Feature transformations between application-specific feature spaces", Computer-Aided Engineering Journal, Vol. 5 (6), pp. 247-255.

[202] Maropoulos, P.G., (1995), "Review of research in tooling technology, process modelling and process planning part II: Process planning", Computer Integrated Manufacturing Systems, Vol. 8 (1), pp. 13-20.

[203] Wingård, L., (1991), "Introducing form features in product models: a step towards CADCAM with engineering terminology", Computer System for Design and Development.

[204] Cunningham, J., and Dixon, J., (1988), "Designing with features: the origin of features", Proceedings of the Proceedings of the 1988 ASME International Computers in Engineering Conference and Exhibition, Vol. 1, pp. 237-243.

[205] Shah, J., (1991), "Conceptual development of form features and feature modelers", Research in Engineering Design, Vol. 2 (2), pp. 93-108.

[206] Li, G.-d., Zhou, L.-s., An, L.-l., Ji, J.-f., Tan, C.-b., and Wang, Z.-g., (2010), "A system for supporting rapid assembly modeling of mechanical products via components with typical assembly features", The International Journal of Advanced Manufacturing Technology, Vol. 46 (5-8), pp. 785-800.

[207] Borgo, S., Carrara, M., Vermaas, P.E., and Garbacz, P., (2006), "Behavior of a technical artifact: an ontological perspective in engineering", Proceedings of the Formal Ontology in Information Systems: Proceedings of the Fourth International Conference (FOIS 2006), Bennett, B., Fellbaum, C. (Eds.), Vol. 150, IOS Press, p. 214.

[208] Suh, N.P., (1990), The principles of design, Oxford University Press, New York [etc.].

[209] Johnson, A.L., (1991), "Designing by functions", Design Studies, Vol. 12 (1), pp. 51-57.

[210] Johnson, A.L., and Thornton, A.C., (1991), "Towards real CAD", Design Studies, Vol. 12 (4), pp. 232-236.

[211] Grabowski, H., Anderl, R., Holland-Letz, V., Patzold, B., and Suhm, A., (1989), "An Integrated CAD/CAM-System for Product and Process Modelling", Proceedings of the International GI-IFIP Symposium, Vol. 89, pp. 8-10.

[212] Bauert, F., Beitz, W., Weise, E., and Salem, N., (1990), "Modeling methods for a flexible computer-aided embodiment design system", Research in Engineering Design, Vol. 2 (1), pp. 15-34.

[213] Vacca, M., Graziano, M., Demarchi, D., and Piccinini, G., (2012), "TAMTAMS: A flexible and open tool for UDSM process-to-system design space exploration", Proceedings of the Ultimate Integration on Silicon (ULIS), 2012 13th International Conference on, pp. 141-144.

[214] Kacprzynski, G.J., Roemer, M.J., Modgil, G., Palladino, A., and Maynard, K., (2002), "Enhancement of physics-of-failure prognostic models with system level features", Proceedings of the Aerospace Conference Proceedings, 2002. IEEE, Vol. 6, pp. 6-2919-2916-2925 vol.2916.

[215] Pourtalebi, S., and Horváth, I., (2016), "Towards a methodology of system manifestation features-based pre-embodiment design", Journal of Engineering Design, Vol. 27 (4-6), pp. 232-268.

[216] Cutkosky, M.R., Tenenbaum, J.M., and Muller, D., (1988), "Features in process-based design", Proceedings of the ASME computers in Engineering (CIE) conference, San Francisco, pp. 557-562.

[217] Shah, J., Hsiao, D., and Robinson, R., (1990), "A framework for manufacturability evaluation in a feature based CAD system", Proceedings of the Proceedings of the 1990 NSF design & manufacturing conference.

[218] Wierda, L.S., (1991), "Linking design, process planning and cost information by feature-based modelling", Journal of Engineering Design, Vol. 2 (1), pp. 3-19.

[219] Ma, Y.S., Chen, G., and Thimm, G., (2008), "Paradigm shift: unified and associative feature-based concurrent and collaborative engineering", Journal of intelligent Manufacturing, Vol. 19 (6), pp. 625-641.

[220] da Hounsell, M.S., and Case, K., (1999), "Feature-based interaction: an identification and classification methodology", Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, Vol. 213 (4), pp. 369-380.

[221] Van Holland, W., and Bronsvoort, W.F., (2000), "Assembly features in modeling and planning", Robotics and Computer-Integrated Manufacturing, Vol. 16 (4), pp. 277-294.

[222] Faheem, W., Hayes, C.C., Castano, J., and Gaines, D., (1998), "What is a manufacturing interaction", Proceedings of the Proceedings of the 1998 ASME Design Engineering Technical Conferences, Citeseer, pp. 1-6.

[223] Regli, W.C., and Pratt, M.J., (1996), "What are feature interactions", Proceedings of the ASME Design Engineering Technical Conferences, Design for Manufacturability Symposium, New York, NY, August, pp. 18-22.

[224] Feng, C.-X.J., Huang, C.-C., Kusiak, A., and Li, P.-G., (1996), "Representation of functions and features in detail design", Computer-Aided Design, Vol. 28 (12), pp. 961-971.

[225] Sormaz, D.N., and Khoshnevis, B., (2003), "Generation of alternative process plans in integrated manufacturing systems", Journal of intelligent Manufacturing, Vol. 14 (6), pp. 509-526.

[226] Zha, X.F., and Sriram, R.D., (2006), "Feature technology and ontology for embedded system design and development", Proceedings of the Proceedings of the ASME Design Engineering Technical Conference, Vol. 2006.

[227] Hylands, C., Lee, E., Liu, J., Liu, X., Neuendorffer, S., Xiong, Y., Zhao, Y., and Zheng, H., (2003), "Overview of the Ptolemy Project, TechReport UCB/ERL M03/25, Department of Electrical Engineering and Computer Science", University of California, Berkeley.

[228] Lee, E.A., and John, I., (1999), "Overview of the ptolemy project", Electronics Research Laboratory, College of Engineering, University of California, 1999.

[229] Frost, R., (1988), "Introduction to knowledge based systems", IEE REVIEW.

[230] Zha, X.F., Fenves, S.J., and Sriram, R.D., (2005), "A feature-based approach to embedded system hardware and software co-design", Proceedings of the ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. 609-620.

[231] Pratt, M., and Wilson, P., (1988), Requirements for support of form features in a solid modelling system, Computer Aided Manufacturing-International.

[232] Shah, J.J., and Rogers, M.T., (1988), "Functional requirements and conceptual design of the feature-based modelling system", Computer-Aided Engineering Journal, Vol. 5 (1), pp. 9-15.

Chapter 3

MEREO-OPERANDI THEORY
as a theoretical framework

In the previous chapter we discussed the necessity and proved the importance of applying a physical view in modeling of complex systems, and specifically in a feature-based modeling supported design of CPSs. As much as the latter is concerned, an approach following the lines of traditional feature technology could obviously not be considered since it suffers from limitations in many aspects, such as providing system-level semantic reasoning about heterogeneous system components and their functional relationships, and combining architectural modeling with modeling of operations on multiple levels. Coping with heterogeneity, complexity, and integrity of CPSs was the major issue that had to be addressed at forming a concept for system-level feature-based modeling. This needed a coherent and comprehensive theoretical (conceptual) framework. The overall objective of the second research cycle was to introduce the notional elements of this conceptual framework proposed to underpin system-level feature-based modeling of CPSs. We refer to this novel conceptual framework as mereo-operandi theory (MOT). This theory was used to facilitate the methodological and computational implementation of our system-level feature-based modeling and simulation methodology and the related computational tool.

In order to get a sufficiently deep insight in the open and challenging issues of CPS modeling, we first conducted an empirical study, which is presented in Sub-chapter 3.2. The results of the empirical study entailed the need for an additional survey with regards to the available theories. Based on the conducted analysis we could identify what needed to be described, explained and predicted by the sought for theory. Lacking a theory for such a framework, we filled the gap by developing the concept of MOT, which is presented in Sub-chapter 3.4 in detail. The elements of this theory have been developed to a level, where they not only serve as building blocks of a logical framework, but also guide the computational implementation (e.g. the specification of the overall data structure). Sub-chapter 3.5 provides further information about the elements of the conceptual framework. Validation of the conceptual framework with regards to the practical representation power and compliance is presented in Sub-chapter 3.6. Finally, Sub-chapter 3.7 reflects on the findings, and consolidates them with regards to the objectives of the follow-up research cycles.

3 MEREO-OPERANDI THEORY AS A THEORETICAL FRAMEWORK

3.1 ON THE ACTIVITIES COMPLETED IN THE SECOND RESEARCH CYCLE

In the previous research cycle, the phenomenon of architecting (pre-embodiment design) of cyber-physical systems has been studied from five different perspectives (i.e. considering five domains of study). It has been revealed that system-level feature-based modeling of CPSs should be considered as a potential solution, which could fill in the knowledge gap of pre-embodiment design that was elaborated on in Chapter 2. It was also discussed that for realization of the proposed concept, we needed to develop an underpinning theory that supported system-level feature-based modeling of CPSs. This chapter reports on the theory developed for this purpose, which is called mereo-operandi theory (MOT). The word 'mereo' expresses that MOT has one of its roots in spatiotemporal mereotopology [1] [2], while the word 'operandi' refers to the 'modus operandi' theory of engineering systems.

As discussed before, we have chosen the phase of pre-embodiment design as one of our concerns. In this phase, CPS designers should tackle many interrelated challenges of system conceptualization and configuration. They are more interested in the functionality of the system as a whole and the composability of the parts, than in the very technical issues of detailing the parts and their connections. In other words, the main challenge of CPS designer in the pre-embodiment design phase is 'composability'. Composability is a measure of the degree to which parts can be assembled in various combinations to satisfy specific user requirements [3]. Composition typically relies on off-the-shelf parts, however, when they are not available or not appropriate from the point of view of the overall operation of the system, the required parts should be designed and integrated.

This chapter presents the major assumptions and considerations that supported the composition of the proposed theoretical (conceptual) framework that count upon all known characteristics of CPSs and facilitates a design supporting computational tool. Towards this end, MOT was developed to offer mechanisms for (i) formally definition of architectural relations among parts, (ii) formally definition of relations among operations provided by parts, (iii) aggregating models of parts architecturally and operationally, (iv) capturing information about both roles of parts and their interdependences / interoperations, (v) formulating uniform information sets to capture different kinds of parts, e.g., software, hardware, cyberware, and aggregated-ware. The theory of cyber-physical systems was used as a guide to cope with system compositionality and component composability issues. These viewpoints are complementary and therefore a combination of them was applied in our work [4].

At designing the second research cycle, we decided to also conduct empirical investigations related to modeling of CPSs first, and then to utilize the obtained insights in formulating the target theory by inductive reasoning. In addition, we also considered and combined some classic theories in the framework. The outcome of this theory synthesis was

projected onto a practical case with the objective of confirmation and consolidation. For the explorative empirical study, the case of a smart watch was selected based on the reasoning that, contrary to its appearing simplicity, it resembles many characteristics of more complicated CPSs, and it could even be considered as a functional component of a CPS. It was also important consideration that the complexity of a smartwatch was manageable for conducting the explorative studies. The level of heterogeneity of the components of a smartwatch was also considerable, but comprehendible for our empirical study. In the case of the smartwatch, the analogue hardware was relative simple and permanent, but this cannot be claimed for the digital hardware. Likewise, the software and cyberware constituents showed a relative large technical complexity. For this reason, we restricted our focus and investigation on the operating system. The control of the chosen smartwatch (Moto 360) was managed by an Android-variant operating system, which is open source and thus required pieces of information were accessible.

3.1.1 Fundamental assumptions

Four fundamental assumptions have been extracted from the previous research cycle as the basis for theory development:

Assumption 1: *Cyber-physical systems are aggregates of system components, which are coarse grained building blocks with inside cohesion among the included entities (constituents) and with minimal interfaces to the outside.*

Components of a system can be homogeneous, when they include only hardware, software, or cyberware constituents, or heterogeneous (aggregated), when they are formed by intertwining HW, SW, and CW constituents. Components are typically self-contained entities from an operation view point, while constituents are not. Systems are regarded as aggregates of components, and components as aggregates of components or constituents. This reflects a pure physical view, rather than a logical (set-theoretic) or abstract (functional) view. The advantage of the pure physical view is that identification of systems components and the distinction of HW, SW and CW constituents are straightforward in the physical realm.

Assumption 2: *In order to provide a proper pre-embodiment model of cyber-physical systems, it is necessary and sufficient to describe and characterize them from architectural and operational perspectives.*

The architectural aspect of such a system model makes it possible to describe what components the system consists of, while the operational aspect captures what the system as whole and its components do. However, the two perspectives are mutually dependent on each other. Therefore, they should be combined no matter if a comprehensive system model or a specific component model is considered.

Assumption 3: *Instead of an exhaustive description of all operations, which would consider each and every primary operation (e.g. mechanical transformation by a gearbox), secondary operation (e.g. thermal waste due to friction), and tertiary operation (e.g. dislocation of atoms towards a crack), a non-exhaustive operational description is sufficient for system or component development.*

By using the term 'operation' we refer to those transformations and/or changes of a system, which happen in the physical (spatiotemporal) space, and that can be recognized by scientific observations, measurements, or any other means. We hypothesized that CPSs can be sufficiently characterized by their primary operations in the pre-embodiment phase of development. Secondary and tertiary phenomena, which are not related to, or have only an infinitesimal influence on the primary operations, can usually be neglected in this phase.

Assumption 4: *All systems, including CPSs, can be decomposed into a finite number of semantically meaningful units that capture both architectural and operational aspects. The units are called system manifestation features.*

By definition, features are regions of an artifact or process that have semantic meaning in a specific context, as well as significance from a particular aspect. As it is elaborated in chapter 2, the well-known theory and methodology of shape-induced mechanical part features cannot be adapted straightforwardly to system features. However, in the case of CPSs, system-level features can be interpreted on multiple de-aggregation levels. Therefore, system-level features need to be defined and implemented on multiple levels of the structural hierarchy. Moreover, the principles of feature-based design should be extended to be able to manage with heterogeneity of constituents and manifestation levels.

3.2 FORERUNNING EMPIRICAL STUDIES

It has been recognized that the critical system thinking is necessary, but is not solely sufficient for devising the underpinning theory. As mentioned above, the theory should reflect and explain real-life realities and relations. Therefore, it should also be grounded on the observation/investigation of the architectural structuring and operational behaviors of many relevant real-life cases. The theory is supposed to describe them in a generic (abstract) manner as the systems theory does it [5] [6]. Consequently, specific empirical studies have been planned and performed on the smartwatch case [7]. As a product, this wearable computing system is shown in Figure 3-1. The basic functionality of the smartwatch includes functions such as (i) chronograph, (ii) cellular phone, (iii) multifunctional calculator, (iv) GPS navigator, (v) compass, (vi) media-player, (vii) translator, (viii) accelerometer, (ix) thermometer, (x) altimeter, (xi) barometer, and (xii) heart rate monitor. Many more complementing functions are provided by flexibly downloadable (mobile) Apps. The software part includes 'Android-Wear' as operating system, and a pool of dedicated Apps [8]. The cyberware includes collected and downloaded data structures and various media materials (e.g., data, audio, and video files).

Figure 3-1 Moto360, a smart consumer durable

It has to be mentioned that the smartwatch can be connected to an Android smartphone for notifications and control over various features. The connection is realized through

Bluetooth low-energy connection technology (Figure 3-2). The watch can: (i) receive notifications from several apps (i.e., e-mailing apps, social network apps, text massaging apps, and Google Now), (ii) monitor health conditions of users, (iii) show weather forecasts, (iv) navigate to a destination, and (v) hear the voice commanding of users by the Google Now engine. However, the majority of (required) information is retrieved from the connected smartphone. Therefore, without an Android smartphone (e.g., when there is no internet connection) the smartwatch is limited in its performance.

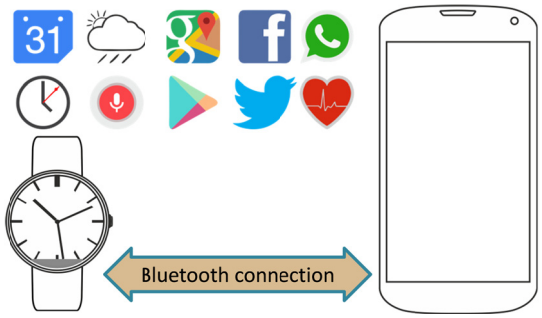


Figure 3-2 Smartwatch-smartphone pairing

3.2.1 De-aggregation of physical architecture of HW, SW, CW

Moto 360 is the first smartwatch released in September 2014 by Motorola. The operating system, Android Wear, is a light version of 'Android OS', designed for smartwatches and other wearable gadgets. Android Wear integrates 'Google Now' technology and mobile notifications into a smartwatch form factor. Google Now uses a natural language user interface to answer user-initiated queries, make recommendations, and proactively deliver information to the user based on his/her search habits, location histories and messaging activities. The empirical study of the physical architecture of Moto 360 started with de-aggregation of its parts, i.e., HW, SW, CW, and AW. The information obtained in the course of the architectural de-aggregation was collected from an online repairing wiki named IFIXIT [7]. The images collected from this website are collected in Figure 3-3 to show the layout of the assembly. In the first level of de-aggregation, there could be three components identified, namely 'body', 'screen', and 'innards'. In the next level, each of these components was de-aggregated into several lower-level components.

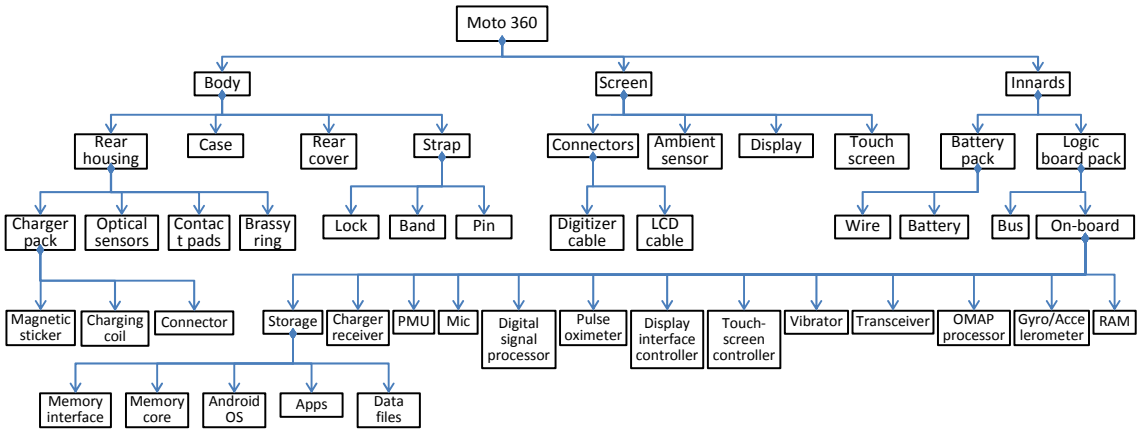


Figure 3-3 Hierarchy tree of Moto360

The lowest level of de-aggregation in Figure 3-3 represents the first level software and cyberware constituents as sub-domains of storage. In order to explore the architectural relations in software, the de-aggregation was continued with regards to the Android OS. Architectural relations in hardware entities were relatively easy to observe. However, keeping the physical view for software and cyberware entities was not as simple as for

hardware entities. The reason was twofold: (i) software codes are dynamic since they are in many places at the same time (e.g., in the storage, in the RAM, in the processor) with different architectural relations, and (ii) they are converted into fundamentally different forms (e.g., language code, machine code, binary code) during their life cycle. These issues are further elaborated in the next sub-chapters.

Figure 3-4 shows an architectural de-aggregation of Android OS and the related apps (the architectural hierarchy tree) on five levels. These are: (i) the kernel and low level tools, (ii) native libraries, (iii) the Android Runtime, (iv) the application framework, and on top of all (v) the applications. In this figure, nesting the entities inside each other is done due to the limitation of space. It is worth mentioning that this de-aggregation process can be continued further. As an example, the 'storage' was de-aggregated into HW, SW, and CW constituents, one of which is the Android operating system.

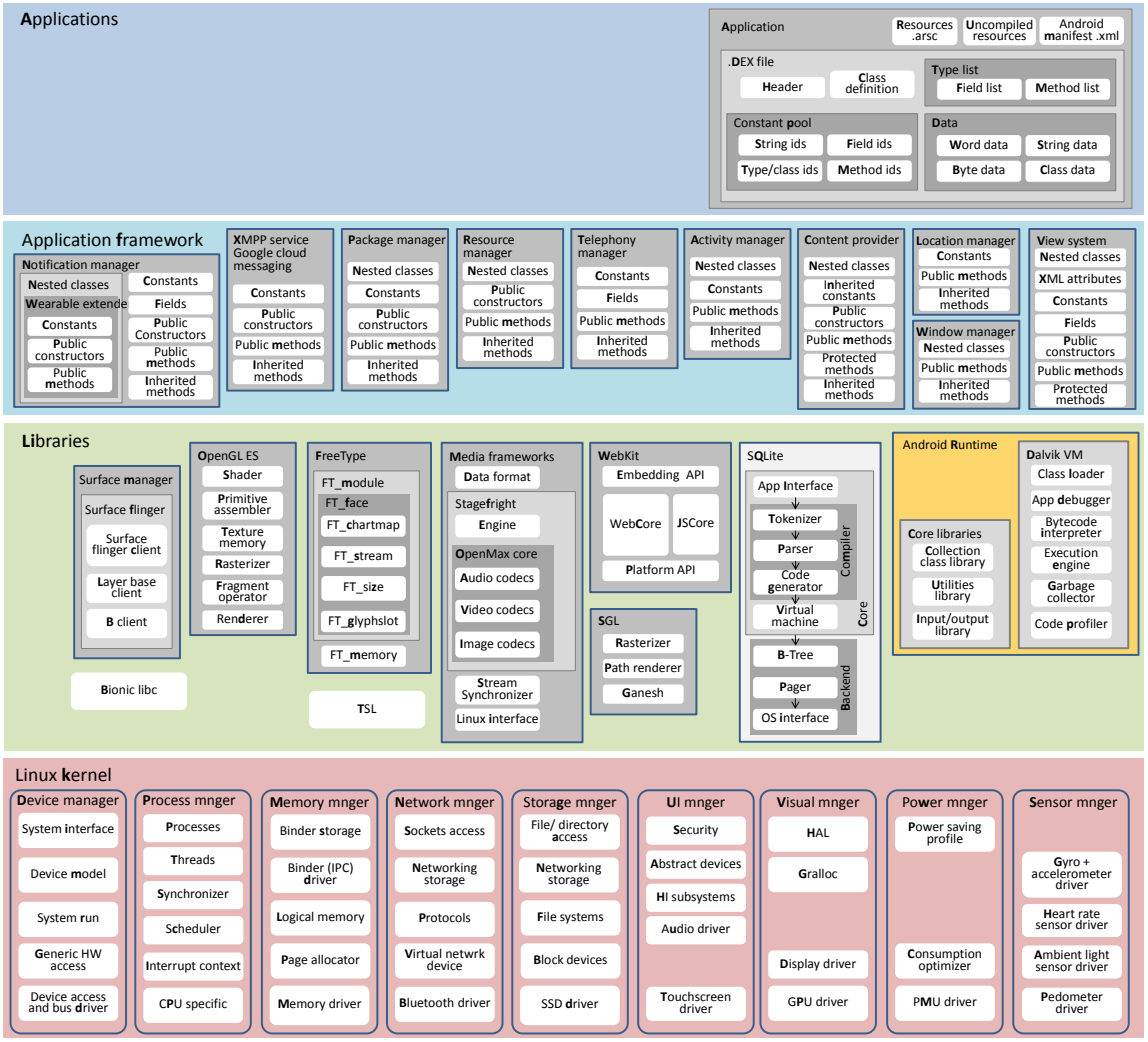


Figure 3-4 Architectural de-aggregation of Android OS

In the lowest level, Android OS befits from the modified version of *Linux kernel* for special needs in power management, memory management, and the runtime environment. The second layer consists of libraries, which are written in C/C++. The *Android Runtime* consists of the *Dalvik virtual machine* and the *Java core libraries*. The *core libraries* are collections of

classes, utilities, tools, inputs, and outputs. The applications and the framework, as well as the runtime core libraries are written in Java and run in the Dalvik virtual machine. *Frameworks* provide abstractions of the underlying native libraries and Dalvik capabilities to applications. In the framework layer, the *activity manager* is designed to manage the life cycle of apps. The *package manager* is there for tracking the apps installed on the device. In the library layer, the *window manager* is on top of the *surface manager*. It controls composing of different processes and windows. *Telephony manager* is responsible for phone calls, and *content providers* let Apps share data with other Apps. *View system* contains all building blocks of user interface and handles event dispatching, layout and drawing tasks. *XMPP service* allows Apps to send device to device data massages.

The *libraries* for CPU and GPU intensive tasks are compiled to device optimized native code. Basic libraries like the *libc* or *libm* were developed especially for low memory consumption. In the libraries layer, *surface manager* handles screen access for the *window manager* from the framework layer. Media framework is placed in the library layer due to the need for heavily optimization of audio and video codecs. *OpenGL* is in charge of 3D graphic handling. However, *SGL* is there for 2D graphic tasks. *FreeType* is used for rendering text onto bitmaps. *SSL* stands for security socket layer. *SQLite* is a relational database management system contained in a *C programming library*. The *SQLite* library is linked to Apps through *Dalvik* and thus becomes an integral part of the apps. *WebKit*, which is an open source browser engine, is most probably removed from the libraries of Android wear. These additional relationships have been not taken into consideration in the architectural investigations, but well in the operational investigations.

To formally synthesize the outcome of the empirical study, we applied a symbolic formalism to describe the architectural relations identified in Android. This way, we intended to capture the aggregation from an architectural point of view. The Linux kernel (K) is an aggregate of: (i) device manager (KD), (ii) process manager (KP), (iii) memory manager (KM), (iv) network manager (KN), (v) storage manager (KG), (vi) UI manager (KU), (vii) visual manager (KV), (viii) power manager (KW), and (ix) sensors manager (KS). This can be formally defined as:

$$K := \{KD , KP , KM , KN , KG , KU , KV , KW , KS\}$$

With the same logic (and considering bold letters in Figure 3-4), we defined libraries (L), application frameworks (F), Android runtime (R), and applications (A) as:

$$L := \{LM , LO , LF , LM , LW , LS , LQ , LB , LT\}$$

$$F := \{FN , FX , FP , FR , FT , FA , FC , FL , FW , FV\}$$

$$R := \{RD , RC\}$$

$$A := \{AR , AU , AM , AD\}$$

We applied the same formalism for the second level of de-aggregation of the Android software, but it can also be applied to the next levels. This empirical study provided practical evidence that even rather complex software systems can be de-aggregated to a level where we can identify system level features, but can also derive their architectural relationships. It was also observed that the hierarchical de-aggregation of architectural entities of

all types of constituents/components (i.e. HW, SW, CW, and AW) can be interlinked with a functional decomposition. The literature advised us that this applies to linear systems, unconditionally. This also gives floor to imposing a physical view in modeling, since a composition of lower level architectural entities is the manifestation of a higher level entity, but with the constraint that one particular instance of a lower entity can be a part of only one entity of a higher level aggregation. At the same time, an entity can be connected to several other entities on the same level of aggregation. It hinted at the fact that architectural relations can be captured as connectivity and containment relations.

These findings played an important role in considering the formalisms of mereotopology to describe these architectural relations. The architectural relationship among the SW, HW, CW, and AW entities can be captured by 'part-of' (P) and 'connected with' (C) relations. Figure **3-5** demonstrates all possible 'part-of' relations within a system. In this figure (and from now on in the whole of the thesis) we use specific symbols to identify entities. For example, we use a rectangle for HW, a triangle for SW, a circle for CW, and a pentagon for AW. The 'part-of relations' are written as *a***P***b*, and this symbolic expression can be read as '*a*' is a part of '*b*', where '*a*' and '*b*' are constituents (i.e. software, hardware, or cyberware) and components (aggregatedware) of a system. The symbolic examples of the 'part-of' architectural relations identified in the case of the Moto 360 smartwatch are shown in Figure 3-5:

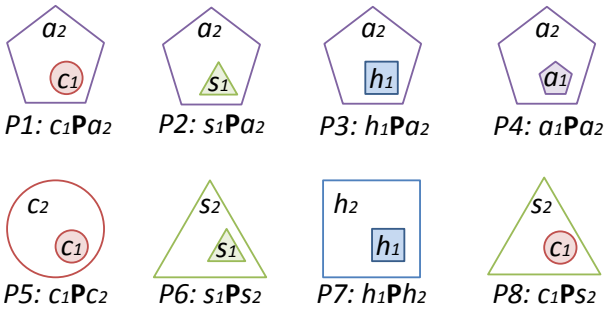


Figure 3-5 All possible 'part-of' relations in a technical system

P1: Gesture pattern library is a part of the touch screen controller

P2: Android is a part of the storage

P3: Bus is a part of the logic board

P4: The transceiver is a part of the logic board

P5: A swipe pattern is a part of the gesture pattern library

P6: The application frameworks is a part of the Android OS

P7: Magnetic sticker is a part of charger pack

P8: Media framework library is a part of Android OS

There are 'connected with' relations among many different kinds of entities. The possible connectivity relations are represented in Figure 3-6. The examples of these relations in the case of the Moto 360 smartwatch are:

C1: The rear cover is connected with the strap

C2: The health data in the relational database of the user profile

C3: The 'window manager' is connected with 'surface manager' in Android OS

C4: The 'storage is connected with the 'RAM'

C5: The 'network manager' of Android is connected with 'transceiver' module

C6: The heart rate sensor is connected with the generated signal

C7: The health profile is connected with the health app

C8: The health App is connected with the pedometer sensor

C9: The battery pack is connected with the logic board

C10: The acceleration conversion algorithm is connected with the gyro / accelerometer

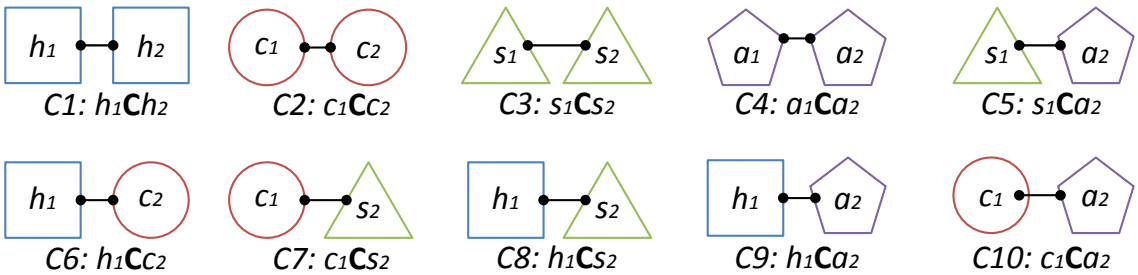


Figure 3-6 Possible 'connected with' relations in a technical system

The mentioned types of architectural relationships were used as the starting point for defining the mereotopological relations used in the specification of system-level features.

3.2.2 De-aggregation of physical operation of HW, SW, CW

In order to explore the full range of operations of Moto 360, first we identified the general (highest level) and then de-aggregated these to the lower level operations in a physicality-conscious manner. In the highest level, there were four operations found, namely (i) receiving user requests for information, (ii) retrieving data from the connected smartphone, (iii) processing the information, and (iv) displaying the required information. These four operations together support all services provided by the smartwatch (i.e. navigation, notification, voice commanding, displaying weather forecast, monitoring health status, etc.).

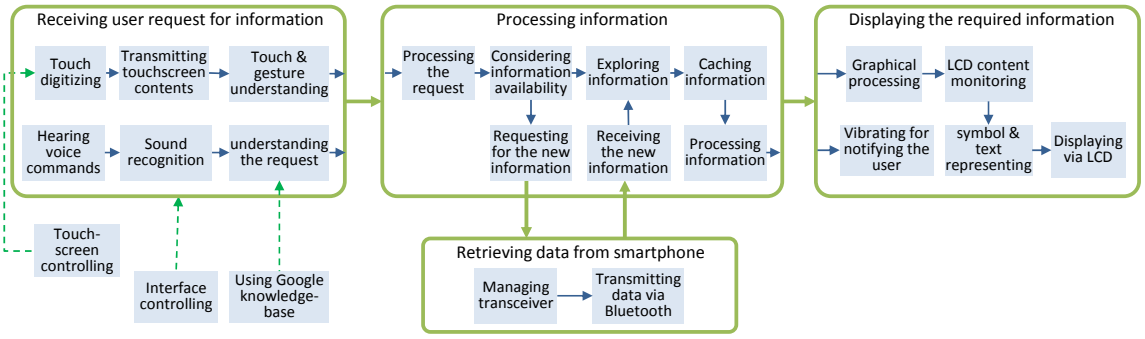


Figure 3-7 First and second level operations of Moto 360

Figure 3-7 shows the first and the second level operations. Due to space limitation, presenting the results of de-aggregation of all levels of operations of Moto 360 is not possible. The complexity is illustrated by Figure 3-8, which shows the results of de-aggregation of touch digitizing, one of the second level operations. The dashed arrows represent con-

tainment relations in this figure. It means that each column presents a de-aggregation level whose number is specified by the circles.

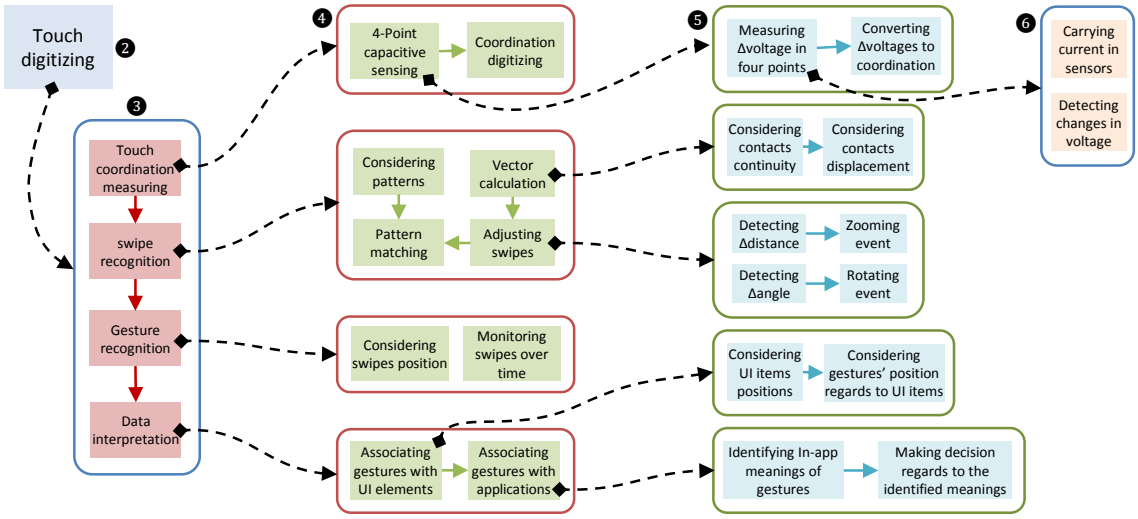


Figure 3-8 Results of de-aggregation of a second level operation (touch digitizing)

Containment and connectivity relations identified in the smartwatch have disclosed important information in the context of operation. In comparison with architectural containment and connectivity relations, there are some major differences with regards to these relations. In an operation perspective containment implies existence of lower level operation that is physically necessary to realize a higher level operation; therefore they have to be aggregated with other lower level operations. When two operations are interconnected, it is very probable that one of them is needed (at least partially) for the realization of the other one. This aggregation may be sequential, parallel, or any combination of these. The ordering of lower level operations in aggregation needs the consideration of their dependencies as well as their timing. This is the reason that 'dependencies' and 'time' are considered as important factors in modeling complex operational relations.

From a physical viewpoint, operations were seen as transformation (i.e. a component transforms its inputs into outputs by its operation). For instance, there is a transformation when a digitizer converts the user touch into a command as its operation. From a physical viewpoint, operations can also be seen as state transition. When there is a change in the state of an architectural domain, an operation should definitely be performed. Therefore, operations should be seen as both input/output transformations and state transitions. Although there are significant differences between operations of HW, SW, and CW entities, all of them can be uniformly formulated as I/O transformations and state transitions. Operations of a HW constituent are (usually) defined by (observable) operational parameters, such as pressure, acceleration, deformation, and so forth. However, operations of a SW constituent, e.g. information processing, instructing, and control of other components, are not physically tangible. It is worth mentioning that 'existence' is a basic operation that could be assigned to any HW, SW and CW entity, which is the main operation of CW constituents. The reason is that CW constituents are normally handled by other components (for instance, software is needed to operationalize them).

3.2.3 Concept of multi-granularity from operational and architectural viewpoints

A system can be de-aggregated multiple times both architecturally and operationally. Architectural de-aggregation more likely results in smaller and more homogenous components in lower levels. Operational de-aggregation more likely results in less complex and shorter duration operations in lower levels. A smartwatch can be considered as a component of a CPS, but it can also be considered as a system composed of several components. The 'OMAP' processor on the logic board of Moto360 is a system (on a chip) and can be de-aggregated into a myriad of other components architecturally and operationally. In the case of a CPS, multiple systems can play the role of a component, and an aggregation of numerous components forms these systems on the next level of de-aggregation, and so forth. We assumed that the sought for system-level modeling tool should support multi-granularity. It means that system designers should be able to choose the level of aggregation/de-aggregation they want to work on. The composed systems are supposed to be usable (if needed) as components for a higher level systems (i.e. to realize the concept of systems of systems in modeling).

The realization of such a modeling tool entails an explicit handling of both architectural and operational multi-granularity. However, it has to be taken into account that there is a fundamental difference in architectural and operational containment relations. As evidenced by the practical examples, architectural containment relations can be captured as hierarchical 'part-of' relations. It means that a lower level entity is only a part of one higher level entity in a given situation. In other case, the imposed strictly physical view is violated (i.e. an object can be at multiple place at the same time). However, this is not the case with operation containment relations, which are not necessarily hierarchical. What it means is that a lower level operation might be included as a part of several higher level operations concurrently. For instance, in the smartwatch case, the 'processing' operation in a same time is a part of 'navigation', 'voice commanding', and 'notification' operations in a higher level. This is called 'web-type' containment relation (Figure 3-9). Formulating the operation with this in mind helps representation interwoven operations (e.g. simulation of a CPU overload in parallel operations).

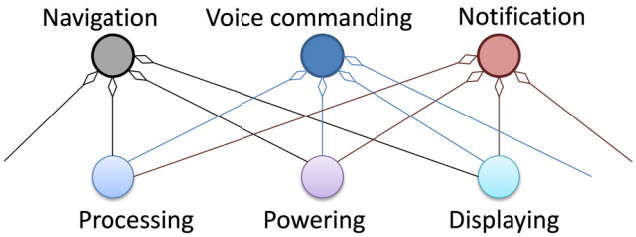


Figure 3-9 A simple example of a web type operation

3.2.4 The lessons learned from the empirical studies

The empirical studies proved to be useful because it casted light on various issues such as (i) software dynamicity (metamorphosis in its different states), (ii) integrity of HW, SW, and CW constituents, (iii) relationships (interactions) among constituents, (iv) relationships of system components and the embedding environment and users), and (v) inseparability of architecture and operation. Below we further operate on the lessons learned in these contexts.

According to our observation, software entities behave differently in various phases of their life cycle. Software can exist in the form of a language code, a compiled code, a runtime code, an instruction in the processor, etc. and consequently it may change its size,

form, and physical relations. Subsequently, it is important to consider the morphological and physical attributes of SW (as well as of CW) and capture the different states of their operations separately in the respective forms. It should also be taken into consideration that when a software entity is stored as binary code on storage passively, it is nothing but a cyberware entity.

The modeling approach should be able to capture all architectural containment and connectivity relations among SW, HW, CW, and AW, even when the modeled system is not operating. Concurrent handling the connectivity and containment relations from both architectural and operational perspective facilitates the integrity and aggregation of constituents. Operational connectivity relations specify dependencies of the interrelated operations. In addition, the interactions among connected operations can also be defined by operational connectivity. The orientation of the operational relations plays a role in a pattern of operational relationships. The observations hinted at the fact that connectivity relations among operations can be formulated through streams. As defined by Pahl and Beitz, but also many others, in every real system material, energy, and information streams can be identified [9] [10]. We concluded that the new modeling approach may considered these streams, and the combination of them, as the subjects of transformations, and the whole of the transformations of the streams in a system can capture the operation of its various constituents, as well as of the whole of a system.

Since interactions among components should be considered from both architectural and operational aspects, the same applies to the interactions between a system and its embedding environment and users. Let us recall that a multitude of architectural and operational interactions with the surrounding environment and the user were found in the case of Moto 360. Some of the architectural relations with user are rather simple, such as the interaction between the strap and wrist of the user. Other architectural relationships of it, such as specific architectural interaction with the environment, are much more complicated (e.g. consider the design solutions for making the watch 'water resistance'). The cases of 'voice commanding' and 'touching commands' are examples of complicated operations and interactions. At the same time sensing 'ambient light' to adjust the screen brightness are less complicated component-environment operational interactions.

The results of our studies showed that while the operational connectivity relations can be characterized by the parameters of the input/output streams, the architectural connectivity relations need to be characterized by information assigned to structural and morphological parameters. We argue that the idea of formulating architectural and operational interaction among components could be used respectively by utilizing 'morphology' and 'streams' in order to capture interactions between system components and user/ environment. This was exploited in the case of specification of system-level features.

It has been our conclusion that architecture and operation should be formulated, represented, modeled, and designed in synergy, rather than separately. If they are separated we miss important pieces of information in the system modeling that could be derived from the complementing constituents and their operations. In turn this opportunity can positively affect consistency, comprehensiveness, and accuracy of the model. A simple example of the real life need for architectural and operational synergy is when the user of smartwatch raises and turns his wrist to read the screen. The screen automatically turns on

by detecting the displacement of the watch, which is an operation initiation based on changing in architectural position.

An important observation was that operations might change architectural relations in almost all specific cases. For instance, transferring a file from the smartphone to the smartwatch creates new architectural containment and connectivity relations. In other words, we needed to capture the architectural states that were changed by operations causing state transitions. Moreover, morphological attributes and parameters which were defined as architectural aspects in our modeling exercise played an important role with regards to the operation of components. For example, the size of the analyzed semiconductor units affects the heat generated by the processor.

3.3 EXPLORATION OF POSSIBLE UNDERPINNING THEORIES

In order to develop a unified theory that allows similar representation and simulation for all heterogeneous components of a system and their constituents, we took on a somewhat challenging undertaking. The latest literature advised us on (i) where to look for the roots and elements of a new theory, (ii) how to formulate it to be reasonably comprehensive, and (iii) how to operationalize it in a computer aided practical methodology. The starting point was the theory of *mereotopology* and the recent efforts to extend it from the spatial domain to the spatiotemporal domain and beyond [11] [12] [13] [14]. We started out from the conclusion that trans-disciplinary system modeling raises the need for concurrent handling of architectural and operational aspects, relations, and parameters. At the same time, capturing and representation of architectural and operational aspects of a CPS need different means. The architectural representation is supposed to identify all physical entities and their mutual relationships. The operational description should capture: (i) the enablers (causes) of operations, (ii) the actors of operations, (iii) the working space of operations, (iv) the changes caused by the effects of the enablers, (v) the changes caused by the interaction(s) of the enablers, (vi) the effects of the working space, and (vii) the logical and chronological flow of operations. Several efforts have been made towards an application context-independent description and integration of the architecture and the operation of electromechanical and electronics systems. Among others, publications, such as [15] [16] [17] [18], reported on results that are relevant in the context of our research. However, the overwhelming majority of the published approaches did not specifically consider unification of the representation and handling of HW, SW and CW constituents [19] [20] [21].

3.3.1 Mereotopological fundamentals

Following the work of some pioneers, we considered mereotopology as a possible theoretical basis for architectural modeling. The name *mereology* came from the formal theory of parts and associated concepts developed by Leśniewski [22] [23]. It literally means "science or theory of parts" [24]. Mereology captures how physical and conceptual parts relate and what it means for a part to be related to the whole and other parts. Addressed by classical mereology, part-whole relations essentially involve a notion of mereological fusion, or sum [1]. Classical mereology is a formal theory that describes part-whole relation [25]. Because of its neutrality, the concept of mereology is widely applicable [26]. Consequently, this concept has been adopted for specification and representation of architectural containment relations in our conceptual framework. Examples of using mereology in hardware

domain are numerous [27] [28] [29]. Mereology is also used to study composite entities and to address the compositionality problem of software constituents [30]. They raised the issue of compositionality of: (i) simple entities, (ii) operations, (iii) events, and (iv) behaviors in their work.

In contrast with mereology, topology is dealing with continuous spaces that can be generated in a combinatorial manner. In its most abstract form, topology implies ontological laws pertaining to the boundaries and interiors of wholes to relations of contact and connectedness, and so on [31]. In its concrete form, topology handles structural relationships composed by linking discrete entities of continuous spaces. By capturing the neighborhoods of a point in continuous spaces, topology is able to formally define concepts such as boundary, interior, and exterior [32]. Consequently, it can provide notations required for formal definition of architectural connectivity relations in our conceptual framework.

Extending mereology-based specification with topological concepts leads us to *mereotopology* (MT) that provides an ontological specification of a conceptualization of the spatiotemporal entity relations of a system [33]. The theory of mereotopology integrates mereology for part-whole relation and topology for component connectedness [34] [35]. In our theoretical framework, MT could play an important role in describing architecture of CPSs [36]. In this sense, components-system relations and component-component relations can be formalized together and defined by mereotopological interpretations [4]. MT offers multiple theories that differ from each other in terms of the underpinning concepts (for instance, they use either points to reason about spatial relations of physical entities, or regions). Most often, regions are used as entities of MT [37]. Regions are proper parts of the whole, and have various (not necessarily permanent) relations with each other in space. MT can provide a manifestation-independent logical specification for computational processing of architectures of artifacts [38] [39]. The mereotopological abstraction makes it possible to describe domains without considering their actual metrics (sizes), morphology (shape), and materialization (attributes).

Regions and their relationships may change in time (which is a challenge in architecture modeling), for instance, in the case of assembly modeling of non-steady spatial structures [40]. Specification of temporal characteristics of regions and reasoning about their relationships in time need temporal logic, not only predicate logic. This has been included in the theory of *spatiotemporal mereotopology* (ST-MT) [41]. In addition, various theories are available to handle multi-dimensional mereotopology [42] [43]. Our investigation has shown that further *'physicalization'* of the entity relationships is possible. It has been argued that ST-MT can capture complex operational relations in the physical domain [44].

Another relevant and useful knowledge domain is that of the *domain theory*, also referred to as *organ theory*. Domain theory is developed based on the theory of technical systems proposed by Hubka, V. and Eder, W.E. [45], and has been further developed by Andreassen, M.M. [46]. According to the domain theory, three views can be applied in product design namely 'part', 'organ' and 'transformation' [47]. The term transformation refers to sequence of activities needed for utilization of the product [46]. It implies the technical activities required for using a system in the application which it is designed for. The term organ refers to architectural entities regarding to their functionality. As functional elements, organs are supposed to operate in a system according to their design intents. The term

part in the domain theory refers to structural properties, materialization and assembly. For briefing, part, organ, and transformation explain architecture, function, and application of a system, respectively. According to the view of domain theory [45], architectural characteristics of a system should be considered as (i) 'form' that is also named as morphology, (ii) 'material' that implies architectural attributes, e.g., elasticity, strength, conductivity, (iii) 'surface quality' that refers to properties e.g. smoothness, reflection, and (iv) 'dimension' that refers to size and tolerances.

3.3.2 Morphological fundamentals

The theory of morphology is dealing with the descriptive analysis of spatial forms (shapes) and their characteristic (distinguishing) features [48]. Morphologists also study the possible shapes-implied relation of three dimensional objects and the principles and laws of regularized shape morphing [49]. Mathematical morphology (MM) is a new mathematical theory which can be used to process and analyze both shapes and images [50] [51]. In the MM theory, shapes and images are treated as coherent point sets [52], and morphological transformations are defined to extract their features and attributes, typically based on Minkowski addition and subtraction [53]. The basic MM operators are dilation and erosion and the other morphological operations are synthesized from these two basic operations [54]. In computational modeling of products morphology has an important role in specifying compatible spatial interfaces.

Our conceived modeling approach intends to handle the explicit shape, e.g. no structural or morphological abstractions have been considered. Striving for an aggregation-driven (abstraction-free) representation of the architectures of CPSs in the physical space, we use semantic abstraction only in the form of *naming conventions* for the components aggregated at various aggregation levels. HW, SW and CW constituents, and the components aggregated thereof, have different spatiality and occupy the space differently.

Being three-dimensional (3D) artifacts, HW constituents can be seen from both external and internal morphological views. In an external view, they are finite regions of the 3D metric space with closed boundaries (e.g. real subspaces of the $\mathbf{E^3}$ space). In an internal view, they are finite point-sets of the $\mathbf{R^3}$ attribute space with specific internal continuity properties and attribute distributions. The position, orientation, shape, and motion of an artifact change in the $\mathbf{E^3}$ space, while other physical properties change in the $\mathbf{R^3}$ space.

SW constituents and system components are quasi-morphed, as they do not have a specific geometric form. However, they have physical extent, which plays a role in their relations to hardware and/or cyberware. This '*dimensionality*' can be understood as an implicit spatiality of software and considered from both external and internal views. In an external view, SW is a code structure that needs a part of the 3D metric space to be stored and processed (e.g., as a sequence of dipoles on a magnetic disk). In an internal view, SW is a set of instructions for a processor that is seen as part of a one-dimensional attribute space. Cyberware is of similar nature and reflects both the external and the internal aspects of SW constituents. A cyberware (e.g., a set of data in a file or record) may exist as a physically stored signal or code sequence, but also as digital values for a processor.

Like in the case of HW components, spatial position of SW and CW components can be identified by reference points in the geometric and the attribute spaces. In our interpretation, atomic entities of software and cyberware are not separately distinguishable parts.

This allows considering HW, SW and CW components, as well as compound HW-SW, SW-CW, HW-CW, and HW-SW-CW components (which are together named as aggregated-ware) as operational system domains, with specific physical properties and operation capabilities (affordances). A domain capturing the architectural manifestation (spatiality) of a component can include subdomains of HW, SW and CW, and the relationships among them.

3.3.3 Exploitation of the engineering modus operandi theory

The general theory of technical system science is a meta-theoretical control for design, development and modeling CPSs [45]. It comprises of (i) the property theory, (ii) general transformation and process theory, and (iii) the general conformational theory. Property theory has a central role in the theory of technical system and implies the specification of the architectural domains that have significance from the operational perspective. General transformation theory considers operations as converting input into output by operands. The general conformational theory defines the operation of a system as an aggregation of operation of its components including the synergetic effects of their relationships [45].

Engineering modus operandi is used to describe the physical and social ways of operations and behaviors of systems. By this term we refer to three things together: physical operation, functionality and behavior. If we consider "operations" as tangible outcomes of a system and the ways of delivering that output by sub-systems, we can describe its function at a higher level of generality and abstraction. In fact, operation explains how a function is accomplished. However, behavior defines the expected function of a system in context of its environment. Then, we need to consider operation of a system and its components in order to understand how the desired function is delivered, and the system behaves in its environment.

In a technical system, operations are defined based on four concepts: (i) the physical and computational phenomena that enable the operation, (ii) the morphological attributes of components, (iii) the physical operation of components, and (iv) the operation flows of the system. Take the example of writing with a pen: (i) ink should be carried to the paper by gravity and surface tension (as physical phenomena), (ii) the shape and size of ball and tube should be purposefully defined (as morphology), (iii) conducting the ink, rolling the ball, carrying the ink, placing the ink on the paper should be conceived (as physical operations of components), and (iv) writing should be completed (as flow of operation that is the result of procedural aggregations of the units of operations). These four concepts should be specified and harmonized at defining the operation of a system in order to ultimately provide the desired operation.

The essence and most important feature of engineering systems is the set of transformations what they implement. Operations were regarded as transformations in several functional modeling methodologies, which all use somewhat similar approaches [55]. In our approach too, operations are considered as transformations between input flows and output flows. These flows connect operations together in order to make a higher level operation. A flow might be energy, material, signal or combination of them [9]. In the context of our work we named them as (energy, material and information) streams, with a view to their role to act as operational interfaces. Operations also transform start states, values, and events into end states, values and events respectively. We refer to this issue as

state transition. In fact, states of the architectural domains are transited during operations. Combining both views facilitate uniformly modeling of all kind of constituents and operations [56].

As mentioned above, the term 'modus operandi' refers to the theory of engineering systems that claims that a particular operation of a system can be realized in multiple alternative ways depending on the first principles and the manifestation of the architecture [1]. Figure 3-10 gives an overview of the involved concepts. Methodologically, the engineering modus operandi (EMO) principle addresses two aspects of operations, namely the procedures and the methods. In our framework, a method deals with parameters and their relations, and explains how a unit of operation (UoO) is realized as a result of act of lows of nature on the physical and computational parameters of the architectural components. On the other hand, a procedure determines the composition of UoOs to provide a flow of operation (FoO). Procedures can be defined through operational containment and connectivity relations; however methods should be defined by mathematical and algorithmic notations. In EMO, while the procedure represents relations among UoOs, the methods explain how UoOs are manifested.

The complex operation of a CPS can be captured either by a top-down approach, or by a bottom-up approach. In the first case, the intended overall operation of the whole system is systematically de-aggregated (decomposed) to a set of interlinked operations of system components. This needs particular solutions for the intrinsic compositionality challenge. In the second case, the overall operation of the system is aggregated (composed) of the specific operations of off-the-shelf or custom-made components. This raises the need for addressing the composability challenge, which boils down to operations and interface matching issues.

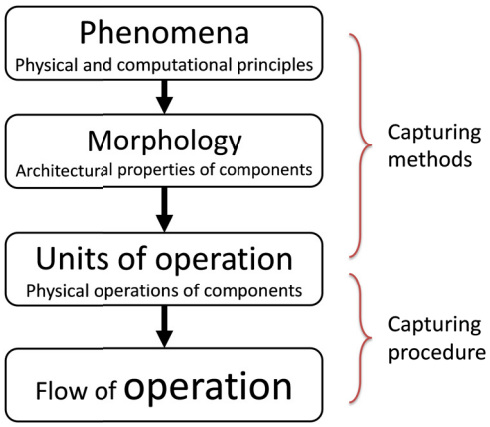


Figure 3-10 Computational concepts for capturing operations

3.3.4 Handling multi-physical processes

A FoO is a logical and temporal (timed) arrangement of transformative actions (UoOs), whose digital processing is enabled by some associated computational methods. A domain may be involved in multiple different physical operations at a given moment in time, or over a time period [57]. As an example, the main operation of a processor is digital instruction processing, but its operation also consumes energy, generates heat, and so forth [58]. Concept of multi-physics is developed for handling this kind of issues in several simulation and modeling tools [59] [60] [61]. Although the mathematical representations required for handling multi-physical processes are needed in simulation, the mathematical equations should be specified in the modeling phase. Multi-physics captures models of operations based on the relations of several constraints about the variables in different modes of operations and multiple simultaneous physical phenomena [62]. Multi-physics comprises of three concepts of multi-field, multi-domain and multi-scale [63]. These three concepts might be combined together sometimes to create more complex modeling and simulation situations.

Multi-field implies considering multiple physical fields in simulation of operations of a system. For instance, operation of the processor in Moto360 creates heat, and the generated heat may affect computing performance of the processor. The examples of publications about multi-field are [64] [65] [66] [67]. Multi-domain implies the changes of boundaries of different properties for example in solidifying a melted metal. There are several documents that reports this issue including [68] [69] [70] [71]. Multi-scale implies the multitude of scopes ranging from molecular operation level to a planet scale operations such as considering effect of gravity on GPS satellites positioning. The examples of studies in this domain are [72] [73] [74] [75]. Sometime all three concepts should be considered in handling multi-physics processes [63]. For example, charging of Moto360 should be done by placing it on an inductive charger dock. The electromagnetic field created by the charger not only charges the battery, it might also affect performance of the processor.

In the context of our work we considered three kinds of issues that can be handled by multi-physics, namely, (i) software manifestation issue which implies the dynamic form of software constituents, (ii) concurrent operations issue which refers to multi-domain and multi-field concepts, and (iii) subordinate operations issue which is the same as multi-scale concept that explained before.

3.4 CONCEPTUAL FRAMING OF THE MEREO-OPERANDI THEORY

The mereo-operandi theory supports trans-disciplinary modeling of cyber-physical systems by combining the principles of mereotopology and modus operandi, and simultaneously addressing architecture and operation modeling issues. Our hypothesis was that MOT can be operationalized in system architecting and modeling by using SMFs. In this Sub-chapter we concentrate on the formal definition of SMFs and the relations among them. The MOT is supposed to provide a robust conceptual framework for formal definition of SMFs, and to support concurrent architectural and operational SMFs-based modeling of CPSs. A graphical abstraction of the conceived constructs to be captured by the theory is given in Figure 3-11. We are aware of the fact that operationalization of MOT in practical design projects cannot be done effectively without having multiple (e.g. entity, relations, phenomenon, etc.) *ontologies*. However, due to the needed development efforts and time we did not include and address it in our research. Actually, it turned out that intellectualization of MOT can be done without defining and having these ontologies.

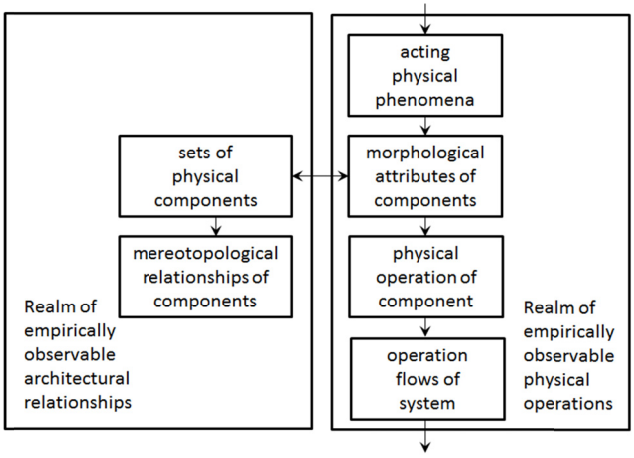


Figure 3-11 The architectural and operational aspects of system modeling

3.4.1 Issues related to formal definition of SMFs

While the previous efforts documented in the literature were made to capture internal and external object-change relationships from a spatiotemporal perspective, our work tried to capture and represent architecture-operation relationships by means of a physical exten-

sion of the mereotopological theory [25] [76] [77]. The starting point of the theory synthesis was the four assumptions introduced earlier in Sub-chapter 3.1. The first assumption implies that a CPS should be considered as a heterogeneous composition of HW, SW and CW constituents. HW constituents could be analogue hardware, digital hardware, and analogue physical constituents. SW constituents might be system software, application software, and development software. CW constituents can be categorized as (i) system enabling knowledge, (ii) repository data structures, and (iii) digital media ware [1].

The second assumption entails the need for a solution for representing a system by integrating its operation and architecture. In order to provide a shared (both physically and logically interpretable) entity, we introduced the notion of architectural domain. A domain is generated by a decomposition or composition of the whole or part of the architecture of a CPS. It is a finite and bounded part within the whole physical extent of a CPS, which lends itself to a particular set of operations. It can be represented by one single system component of an aggregate of operationally related components. Let's identify a particular CPS by the symbol Σ, denote the whole physical extent of Σ by D_Σ, and a given architectural domain of it by D_i. Likewise, let us denote the overall operation of Σ by O_Σ, and the set of operations occurring on, or related to, a particular D_i by $O_{i,j}$. In addition, let us introduce the symbol $\otimes$ to denote the mutual relations between the domains D_i and the related operations $O_{i,j}$ of a Σ. With these, a CPS can be defined by the following symbolic formula: $\Sigma := D_\Sigma \otimes O_\Sigma$, where $D_\Sigma = \cup D_i$, and $O_\Sigma = \cup O_{i,j}$. In this formula, the symbol $\cup$ represents the union operation, which is seen as a logical equivalent of the aggregation of D_i and a proper combination of $O_{i,j}$. Note that the notions of domain and operation are considered inseparable in the logical space **L** and in the physical space $\mathbf{R^3}$, as discussed above. Each of the architectural domains has at least spatial relations with each other.

The third assumption suggests that the working of a system can be captured by a necessary and sufficient subset of operations. Let's call it *characteristic operation* and denote it by CO_Σ. Let us denote the pertinent sub-set of primary operations by O_Σ' , the secondary operations by O_Σ'' , and the tertiary operations by O_Σ'''. With these: $CO_\Sigma := O_\Sigma' \cup O_\Sigma'' \cup O_\Sigma'''$, where: $CO_\Sigma \subseteq O_\Sigma$, $O_{i,p}' \subseteq O_\Sigma'$, $O_{i,q}'' \subseteq O_\Sigma''$, $O_{i,r}''' \subseteq O_\Sigma'''$. With regard to performing these operations, it is assumed that operations are enabled by: (i) one or more interacting physical, computational, informing, or synergic phenomena, $\varphi_{i,k}$, appearing on a D_i, which has got: (ii) a particular spatial arrangement and/or extent of a D_i, namely μ_i, and (iii) performs particular transformative actions, $\alpha_{i,l}$, or changes, $\zeta_{i,m}$, and (iv) $\alpha_{i,l}$ or $\zeta_{i,m}$ happen in a logically proper and physically feasible manner as observable operations, O_Σ', or O_Σ'', or O_Σ'''.

Finally, the fourth assumption postulates that CPSs can be de-aggregated into a finite number of distinct (semantically identifiable) entities depending on their actual physical manifestations. The result of the subsequent de-aggregation steps is a set of interrelated system components. The opportunity of composing hierarchical structures from a set of components has already been utilized in various approaches of component-based design (CBD) [78] [79].

3.4.2 Defining architectural relations by mereotopology

According to IEEE 1471, *architecture* is: (i) the fundamental organization of a system embodied in its components, (ii) their relations to each other and to the environment, and (iii) the principles guiding its design and evolution. Consequently, an architectural model

of a CPS is supposed to define the high level configuration of all system components, as well as the connections that coordinate the operations of the components. Derived from the empirical study, for architectural modeling, we introduced specific instances of '*part-of*' and '*connected-to/with*' relations. These relations are used in system-part, part-part, and part-element contexts (Figure 3-12). The internal relations imply homogeneity and external relation imply heterogeneity of the relationships. It has to be noted that these relations lend themselves to a kind of circularity. Namely, an entity can be a component of (a whole) systems, a component of another (compound) component, and a constituent of a component. Considering this, internal and external relations have been distinguished. The 'part-of' relations between the system and the domains representing its components, and within a component (with a view to included components and constituents), are regarded as internal relations. They have been specified as:

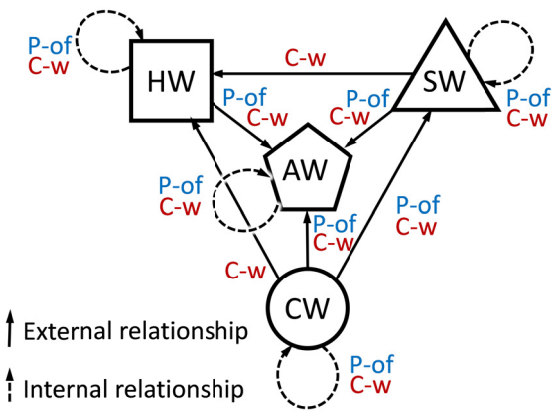


Figure 3-12 Internal and external relations within and between components and constituents

- part-of-system-as-component (PSysCom)
- part-of-component-as-component (PComCom)
- part-of-component-as-constituent (PComCon)
- part-of-constituent-as-constituent (PConCon)

In order to become a component of a system, a constituent should be in PComCon relation. The nature of components and constituents can be indicated by using indices h, s, c and a, standing for HW, SW, CW or AW. They are applied after the abbreviations denoting components or constituents, (e.g., $PCom_aCon_h$). The 'connected-to' relations are among the components of a system and among the constituents of a component, respectively, and are regarded as external relations. 'Connected-to' expresses one-directional relation, while 'connected-with' expresses bi-directional relation. They have been formalized as:

- component-connected-to-component (CComtCom)
- component-connected-with-component CComwCom)
- constituent-connected-to-constituent (CContCon)
- constituent-connected-with-constituent (CConwCon)

The relation 'connected' is further articulated as, e.g., *adjacent connected* ('a.connected'), *remotely connected* ('r.connected'), and *indirectly connected* ('i.connected'). This articulation is helpful from the aspects of specifying operational relationships. They are textually and symbolically represented as exemplified below:

- component-x.connected-with-component (x.CComwCom)
- constituent-x.connected-with-constituent (x.CConwCon)

where: x can define type of relations among components and constituents, this indicators (Connected-x relationships) are the architectural basis (or carriers) of operational relationships.

In our conceptualization, a SW or a CW domain cannot be embedded in HW domain, and HW, SW and CW parts cannot intersect, but can be adjacent in the physical space. Therefore, SW and CW domains can have only 'connected-with' as architectural relations with HW domains. It has to be noted that atomic entities of SW and CW are considered to be spatially undistinguishable parts. Consequently, no architectural or operational relations are supposed to exist among them. Since every domain may have relations with multiple other domains, and the relations can also be multiple, it is reasonable to include the concept of layering, which is logical, physical, and representational layering in one. This leads us to a *stratified mereotopology* (SMT) [80]. A layer includes the set of domains and the set of pertinent relations that belong together in a particular architectural/operational situation [81]. SMT lends itself to the realization of situated analysis and constraint management. Consider a software tool, which offers multiple alternative interfacing, e.g., command line interpreter, graphical user interface, menu driven interface, form-based interface, touch panel manager, and natural language interface. In this case the relations among the domains of the concerned HW, SW and CW constituents, and their architectural and operational relations are placed on different layers. As a bottom line, in the pre-embodiment design phase, the operational relations should be checked when architectural relations are changed, and vice versa. In case of dynamically operating open systems, existing domains may become disconnected from the system architecture, while other domains may become connected to it. Therefore, *existence* of a domain with respect to the system is the most basic operation of architectural components.

3.4.3 Defining operations and operational relations

According to engineering modus operandi theory, operational modeling should cover and provide parameterized representation of: (i) the physical and computational effects underlying the operations happening on domains, (ii) the morphological characteristics of the domains, (iii) the conduct of operations and their outcomes, and (iv) the flow of operations as blending of the particular operations [1]. These four elements were named respectively as phenomena, morphology, action, and process in Figure 3-13. MOT implies complex data and relation structures, which should extend to architectural, geometric, morphological, material, physical, temporal, and behavioral aspects. Methods capture relations among the parameters associating with operation and architecture. The phenomena that affect operations of a system can be captured through methods.

The morphology of domains is a dual-constrained overall variable. On the one hand, morphology determines how the physical effect can manifest in operation, and, on the other hand, operation assumes specific (proper) morphology-effect relationships. In the case of hardware components, morphology captures the physical extent, form, shape, dimensions, texture, and other geometric attributes. Considering that morphological modeling should effectively support pre-embodiment design, a minimal morphological representation (MMR) has been stated as adequate target. While MMR varies for different objects, it should carry both geometric and topological information needed to model the operations of physical components.

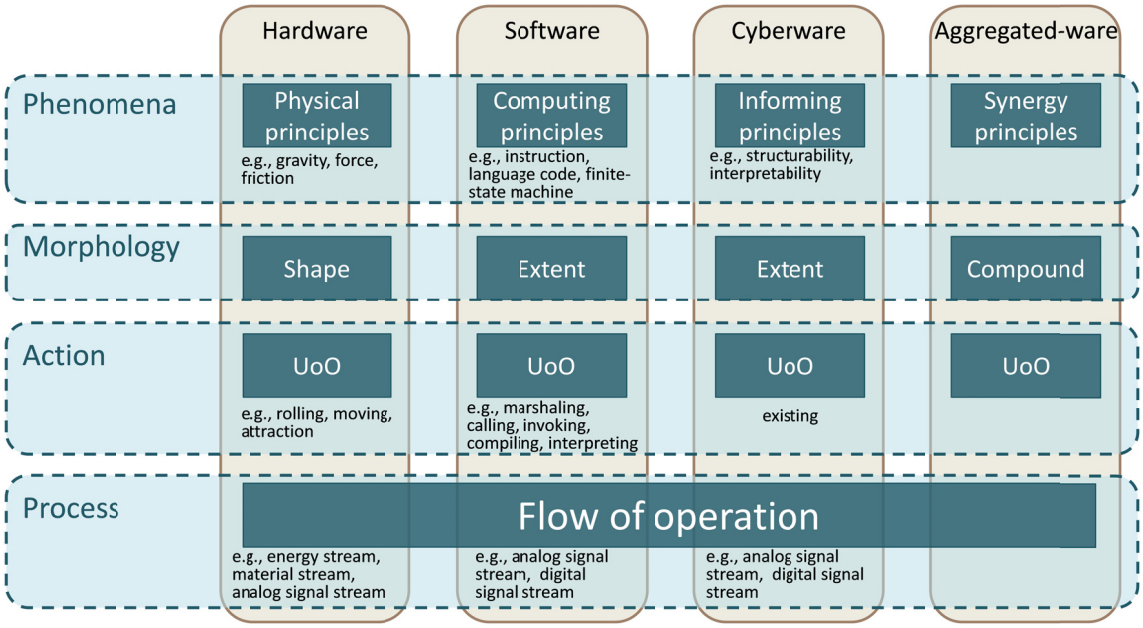


Figure 3-13 Overview of the unified aspects of capturing the operation of HW, SW, and CW

Capturing the morphology of software constituents is much less straightforward. Difficulties emerge because software concurrently exists as a thing and as a process. It is a thing when it exists in the form of pseudo-code of algorithms, a computer language code, or compiled instructions stored on any storage device. It exists as a process when it is a set of executable processor instructions and is executed by the processor. The architecture can be precompiled or run-time compiled. The thing-like existence can be characterized by the measure 'extent', which expresses the physical size of the software. The extent makes software physical thing and manageable in the physical space. Representation of the morphology of cyberware is handled by the theory based on the same considerations.

The concept of unit of operation has been introduced as generalization of the physical phenomenon and digital computing induced aggregate operations of interrelated HW, SW, and CW constituents. Methodologically, a UoO is defined as a quadruple of: (i) methods (M) of operation which include phenomena and related parameters, (ii) morphological properties (mp) of the associating domain, (iii) the transition (δ) over the domain including start state and end state, (iv) the transformation (τ) performed by the UoO including input and output streams and events:

$$u \equiv (M, mp, \delta, \tau)$$

If we consider UoOs as procedural elements in a domain, *flow of operation* (f) is considered as aggregation of the UoOs of that domain by means of procedure (γ) of operation. Accordingly, it can be noted as: $f \equiv (U, \gamma)$ where (U) is a set of included UoOs. This set can be represented as: $\mathrm{U} = \{\mathrm{u}_1, \mathrm{u}_2, \ldots, \mathrm{u}_\mathrm{n}\}$. Procedure is a double: $\gamma \equiv (CO, PO)$, where $CO = \{co_1, co_2, \ldots, co_n\}$ and $PO = \{po_1, po_2, \ldots, po_n\}$ are sets of operational containment relations (PO), and operational connectivity relations (CO).

Multiple FoOs can be assumed for each particular domain. Methodologically, it makes sense to specify a FoO if there exist a domain that is proper as carrier of the concerned

compound and elementary operations, which change the attributes of states and transformations the states of the domain. This allows applying various state-transition representations as the basis of computer representation and digital simulation of operations of domains [82] [83]. From a computational viewpoint, the logic of both the physical transformations and the computational transformations is specified as a sequence, a hierarchy, or a web of procedural steps, as discussed among others in [84]. The procedural steps are computed as state-transition algorithms [85] [86]. The algorithms implement individual methods or a composition of them. The concept of methods facilitates the uniform treatment of physical and computational operations. As von der Beeck showed, the state-chart formalism has become quite successful due to its hierarchical representation of state information and its expressive power in modeling concurrency by simultaneously active states and parallel transition execution [87].

3.5 ELABORATION OF THE MOT-BASED FRAMEWORK

3.5.1 Construction of modeling building blocks

As one form of MMR, skeleton modeling of artifacts has been widely studied in the literature, in particular, unique and complete-able skeleton representations [88] [89]. Uniqueness means that a particular skeleton model represents only one and exactly one 3D artifact, while completeness means that the information structure associated with a particular skeleton model carries all pieces of information that are necessary and sufficient for a proper 3D reconstruction (representation) of the morphology of components in both $\mathbf{E}^3$ and $\mathbf{R}^3$. Skeleton modeling uses two basic modeling entities, namely bones and ports (Figure 3-14.) Bones specify the position and orientation of ports in the modeling space, arrange and interconnect a set of ports in space, provide reference points for ports, and transfer and transform physical effects between ports [90]. There is a surface patch assigned to each port which serves as the seed of the boundary surface to define its shape, as well as to act as external interfaces for physically coupling with the surface patch of another port [1]. Bones are the connectors of ports, according to the logic of the presumed energy transfer and morphological changes. They are brought together to define the possible streams of energy within or upon system domains. A particular configuration of ports defines the kinematic degrees of freedom of a skeleton. Ports can describe active places or interfaces of HW, SW and CW constituents, while skeleton elements can represent their 'body' (including physical objects, software algorithms, or data structures) [1].

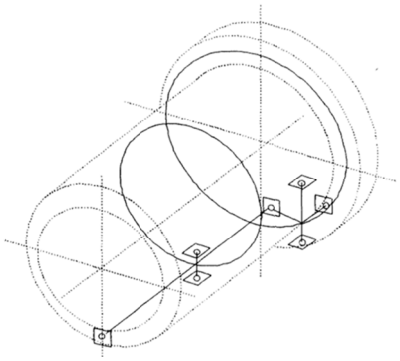


Figure 3-14 Example of a skeleton model

FoO is the fundamental concept introduced for the purpose of specification and computation of operation processes of systems (Figure 3-15). FoO arranges the operations of UoOs in a logically, physically, and temporally feasible arrangement (a work flow) [91]. Primary, secondary, or tertiary operations can be blended in a FoO. However, it is required that UoOs of the physical constituents of a domain to be integrated in time with those of computational constituents. This can be achieved by including *timed computing* in FoOs. This is complicated in many cases, because complex system operations may depend on

instantaneous circumstances and conditions. Real-time handling of the change in the objectives of operation, the available control information, and emergent environmental effects seems to be a partially solved issue. Nevertheless, FoO should represent dynamic operation in time (DOT) and synthesize time-continuous physical operation (that may include singularities and discontinuities) with time-discrete (event-triggered) operation of computational elements. FoO brings these operation modes into a common procedural framework, by imposing explicit time management. It extends to three aspects of timing course of the operations in real time by: (i) considering time as explicit parameter in the representation of physical properties, (ii) guaranteeing the execution of the operations by the time at which the results are needed, and (iii) management of time-critical networking concurrent/parallel computing of data.

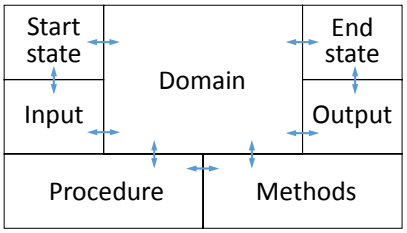


Figure 3-15 logical model of FoO

To achieve these goals, operation timing should be included in UoOs from which FoOs are built up. The overall operation of a domain can be modeled as a discrete-event process, which consists of an event record with the associated time-stamp. For a time-aware (real-time) FoO, *task-oriented scheduling* (TOS) and *event-oriented programming* (EOP) need to be combined and procedurally integrated. A challenge for EOP is to extend control to emergent events and unscheduled event interactions in FoOs. A two-phase event-oriented control is to be implemented, with a first phase dealing with event detection, and the second phase with event handling. The time-driven synchronous state machines can provide this functionality. *State-transition models* can also be used for specification of human–computer interaction [92].

3.5.2 Issues concerning the kernel and interfaces of SMFs

The conceptual framework provided by MOT should advise us concerning the information structure of SMFs as modeling building blocks. These building blocks should capture the aforementioned sets of information in a way that makes creating a model by combining

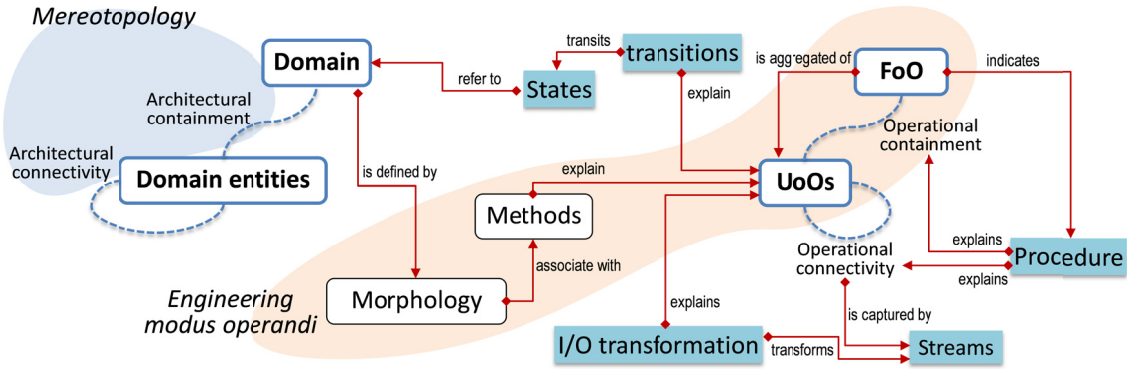


Figure 3-16 The information sets and their relations defined by MOT

SMFs possible. For this purpose we needed to consider modularity issues such as uniformity and connectivity. Consequently, SMFs should consist of two parts: the kernel and the interface. The interface part captures information needed for connecting SMFs together, and the kernel captures all other information sets that require for defining a SMF. The

information sets needed in order to be able to specify SMFs have been defined as shown in Figure 3-16.

We must recollect here that the concept of domain was introduced for defining architectural aspects of SMFs. A domain is created by aggregation of its entities and defined by respective morphological attributive parameters. The containment and connectivity relations in the architecture side are formulated by mereotopological notations. In the operation side, FoO is aggregated of UoOs through the operational containment and connectivity relations explained by procedure. The operational connectivity relations among UoOs are captured by streams (e.g. energy, information, and material streams). UoOs are explained by state transition, input/output transformation and methods of operation. Transitions change states of the domain, and transformation turns the input streams into the input ones. By interconnecting SMFs, a higher level SMF is created. Consequently, FoOs and domains of the connected SMFs are considered respectively as UoOs and domain entities in the newly created SMF. Multi-granularity of SMFs is supported accordingly. In this way, streams are the main operational interfaces which are supported by events and timestamps in order to including timing issues. Thus, morphological and mereotopological connectivity relations were formulated as contracts (input assumption and output guarantee) to support architectural interfacing.

3.6 DEMONSTRATION OF VALIDITY THROUGH PRACTICAL CASES

The conceptual framework introduced as MOT is derived and formulated based on the empirical studies performed formerly. The theoretical results can be examined by applying on the empirical cases. Accordingly, validity of the theoretical conceptualization can be tested. Although the comprehensive confirmation of MOT is not possible without elaboration of the SMF information structure, partially applying the theory on the practical cases can be used as demonstration to show how it works practically. Therefore, we avoided going into an exhaustive practice, and instead we have used an example for each case.

3.6.1 Mereotopological definition of architecture

Mereotopological definition of architecture of the studied smartwatch needed all 'part-of' relations that have been introduced by MOT. For instance:

- ***Part-of-system-as-component:*** the innards pack (in), as a combination of compound components, is a part of the smartwatch (sm) system: $PSys_{sm}Com_{in}$
- ***Part-of-component-as-component:*** motherboard (mo), as a component, is a part of the innards pack (in), which is higher-level aggregative component: $PCom_{in}Com_{mo}$
- ***Part-of-component-as-constituent:*** the runtime environment (re) is a component, which can be de-aggregated into SW and CW constituents in a lower architectural level. Being purely software, the Dalvik virtual machine (dv) can be considered as a constituent of it: $PCom_{re}Con_{dv}$
- ***Part-of-constituent-as-constituent:*** the Bytecode interpreter (bi) is a SW constituent of the Dalvik virtual machine (dv), which is a software constituent in itself: $PCon_{dv}Con_{bi}$

The connectivity relations have also been applied to check their necessity and sufficiency. In the context of the smartwatch, the below connectivity relations were used:

- ***Component-connected-to-component***: the gyro/accelerometer (ga) component is connected to the motherboard (mo): $CCom_{ga}tCom_{mo}$
- ***Component-connected-with-component:*** the touch panel pack (tp) has a dual connection with the motherboard (mo): $CCom_{tp}wCom_{mo}$
- ***Constituent-connected-to-constituent***: the core library (cl), as SW constituent, is connected to the Dalvik virtual machine (dv): $CCon_{cl}tCon_{dv}$
- ***Constituent-connected-with-constituent***: the un-compiled SW resources of the App (ap) are two-ways connected with the resource manager (rm): $CCon_{ap}wCon_{rm}$
- ***Component-adjacent-connected-to-constituent***: the innards pack (in) is connected to the body (bo) of the smartwatch: $a.CCom_{in}tCon_{bo}$
- ***Component-remotely-connected-to-component***: the Bluetooth module (bw) of the smartwatch is dual connected with the Bluetooth module of the smartphone (bp) of the user: $r.CCom_{bw}wCom_{bp}$
- ***Component-indirectly-connected-to-component***: the App (ap) is connected to the processor (pr) indirectly (through the virtual machine and the kernel layer): $i.CCom_{ap}tCom_{pr}$

In general, it was assumed that connectivity relations can be established between domains (components and constituents) that physically exist. The fulfilment of this condition creates a basis for complementing these relationships with physical and computational relationships. In combination, they specify operational transformations over a given domain.

3.6.2 Formal definition of operations

From a process perspective, a UoO, as well as a FoO, has a start state and an end state. In the case of operations of physical nature, there are input streams and output streams of material and energy, as well as of analogue signal, while in the case of computational operations, there are input and output streams of information. For both physical and computational operations, the transformation is described as a (algorithmic) procedure and processed by relevant (alternative) computational methods. Below, we exemplify the description of the physical and the computational operations of different architecture and operational domains of the smartwatch. We focus on the procedural elements of the transformations.

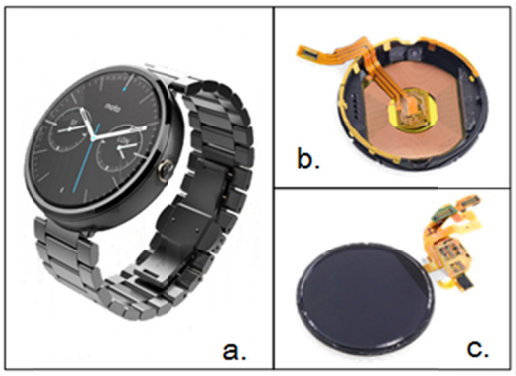


Figure 3-17 Subject of case study: a. the smartwatch, b. the inductive charging domain, c. the touch detection domain

A domain with typical physical operation is the domain of the inductive charger pack (ICP) (Figure 3-17). Placed inside the rear housing of the smartwatch, ICP is in charge of receiving energy from the electromagnetic field of the inductive charger and converting it into electric

current. The start state of operation is in which the smartwatch is at the moment when placed onto the inductive charger in a specified position. The end state is in which the smartwatch is at the moment when picked up from the charger. In these states, both the input and the output are energy streams. The input stream is magnetic energy field, and the output stream is electric current picked up by the battery of the smartwatch. The physical transformation can be modeled by the following procedure:

Begin

1. *Generate electromagnetic field by charger*
2. *Bring smartwatch into physical contact with the charger*
3. *Connect inductively to the electromagnetic field*
3. *Take power from the electromagnetic field*
4. *Convert power into electric current*
5. *Conduct the current to the battery*

End

These together represent one FoO, which composed of several UoOs. There are specific computational and/or mathematical methods assigned to each element of the procedure (UoO), which cannot be discussed here due to space limitations. In addition to the characteristic variables of the physical effects (e.g. the strength of magnetic field, and the duration of charging) and the quantitative formulas that describe their relationships, the methods also include the morphological variables of the lower level components of the ICP such as charging coil, magnetic sticker, and the connector, as well as spatial position relationships, for example, the distances between two coils, the length, diameter and number of turns of the charging coil, etc.

The domain of the touch recognition / localization software is a domain with typical computational operation. This domain plays an important role in the smartwatch since touch and gestures are the main input modes (Figure 3-17 - c). Actually, the touch recognition / localization domain is a compound domain, i.e., the shared touch panel, the capacitive sensors, the connector to the motherboard, the chipset of touch panel controller, and other software constituents. From an operational perspective, the UoO of touch recognition and the UoO of swipe (sweep vector) and gesture identification are intertwined. We analyzed the UoO of touch recognition that involves the descriptive architectural and morphological variables of the whole digitizer unit and the embedded software constituents that is in charge of calculating touch positions.

The recognition of touch is based on the multiple capacitive sensors, which sense and measure Δ voltages in different points. These sensors are at the boundary of the operational domain. The software constituent computes and recognizes the location of (subsequent) elementary touches based on the differences in the detected Δ voltages. Considering the physical domain of the touch panel, the elementary touches can be combined into a sweep vector. As mentioned above, this is seen however as a different computational UoO of the same touch recognition/localization domain. The reason is that the changes in the Δ voltage values should be processed in time in order to be able to compute and identify swipes and gestures. The start state is the untouched state of the domain and the end state is again the untouched state of the domain. The start state and the end state are characterized by various time and event variables. The input consists of material, energy

and analogue signal streams, and output comprises the digital information stream generated by the domain. The touching human fingers create distortion of the electrostatic field on the touch panel, which generates currents in the capacitive sensors according to the analog touch signal produced by the user. The computed touch positions are the digital information output.

The transformation procedure between start state and the end state is as follows:

Begin

1. *Activate if there is a touch*
2. *Capture the Δ voltage data in all sensors*
3. *Send the data to the touch panel controller*
4. *Calculate the position of the point of touch*
5. *Record the data of the points of touch*
6. *Repeat procedure until touch is detected*

End

The steps of the above procedure are done by dedicated parts of the software constituent. Capturing of Δ voltage data can be done by alternative methods, depending on the number and relative position of sensors.

3.7 CONCLUDING REMARKS

The issue of transdisciplinary pre-embodiment design of CPSs onto an application, system, constituent, and method-neutral foundation have been addressed in this chapter by introducing a theoretical framework named as MOT. It was intended to give an overview of the elements of this foundation, but due to the concomitant complexity it could not deal with all (technical details). Extension of spatiotemporal mereotopology into the physical (and computational) realms is a rational step, which can be justified by the growing number of publications reporting on research in this direction.

3.7.1 Reflection on the findings

A comprehensive study of the current literature informed us about the fact that synergetic modeling and co-design of cyber-physical systems, which contain countless analog and digital hardware components, control and application software components, and knowledge, data and media contents as cyberware components, are yet not solved [93] [94] [95]. The reasons can be found partly in the differences in the design and implementation concerns, and partly in the cultural (methodological) differences between the above-mentioned areas of development [4]. The current disciplinary separation is also caused by the lack of a unified theoretical framework and a comprehensive system conceptualization methodology. Instead of a unified theory, disconnected partial theories are typically used for a multi-disciplinary ideation and conceptual modeling of cyber-physical systems. CPSs are not only heterogeneous, but also complex systems that may show various emergent characteristics and behaviors [96] [97]. They typically have many interactions with their surrounding environments and elicit the information controlling their operation from real life processes [98]. A concurrent, trans-disciplinary modeling and co-design of their hardware, software and cyberware parts in the early phase of development can lead not only to process advantages, but also to quality enhancements and market benefits.

MOT applies the principles of spatiotemporal mereotopology for identifying and representing the architectural elements of system and their mutual relationships [99] [24]. The physical entities are interpreted as proper parts of the whole, domains, that lend themselves to 'associatable' operations [100]. Though domains are defined as spatial entities, mereotopological notation makes it possible to describe them without considering their actual metrics, morphology and materialization [101]. Thereupon, MOT is able to describe all HW, SW and CW constituents notwithstanding their inherent differences. By imposing a purely physical view, MOT assumes that existence is an intrinsic (an indispensable) operation of all entities included in a system. When existent, each domain performs either physical or computational transformations, or concurrently both. Since HW, SW and CW have different morphologies and operate according to different principles, the concept of flow of operation has been introduced in MOT. It provides a uniform scheme for representation of operations of HW, SW and CW constituents, and AW components. In fact, FoO creates consistency with respect to the material, energy and information streams transformation, and the state transitions of the corresponding architectural domains.

3.7.2 Enhancements of the concept of MOT

In order to extend and implement MOT, the next research cycles should focus on (i) refinement and improvement of MOT as a whole and its elements, (ii) extending MOT with the theory of system manifestation features, and (iii) transferring the extended theory into a SMFs-based pre-embodiment design methodology for CPSs. As far as the second and third objectives are concerned, some tasks have been identified: (i) development of a conceptual and procedural (methodological) framework for SMFs and designing with them, (ii) elaboration of a computational representation of SMFs, and (iii) elaboration of a SMFs-based synthesis methodology for CPSs.

3.8 REFERENCES

[1] Horváth, I., and Pourtalebi, S., (2015), "Fundamentals of a mereo-operandi theory to support transdisciplinary modelling and co-design of cyber-physical systems", Proceedings of the ASME 2015 International Design Engineering Technical Conferences, Boston, Massachusetts, USA.

[2] Muller, P., (2002), "Topological spatio–temporal reasoning and representation", Computational Intelligence, Vol. 18 (3), pp. 420-450.

[3] Doboli, A., Umbarkar, A., Subramanian, V., and Doboli, S., (2014), "Two experimental studies on creative concept combinations in modular design of electronic embedded systems", Design Studies, Vol. 35 (1), pp. 80-109.

[4] Pourtalebi, S., Horváth, I., and Opiyo, E.Z., (2014), "First steps towards a mereo-operandi theory for a system feature-based architecting of cyber-physical systems", Proceedings of the INFORMATIK 2014, Big Data – Komplexität meistern, E. Plödereder, L.G., E. Schneider, D. Ull (Hrsg.) (Ed.), GI, Stuttgart, Germany.

[5] Hubka, V., and Eder, W.E., (2012), Theory of technical systems: a total concept theory for engineering design, Springer Science & Business Media.

[6] Hubka, V., and Eder, W.E., (2002), "Theory of technical systems and engineering design synthesis", in: Engineering design synthesis, Springer, pp. 49-66.

[7] Motorola, (2014), "Moto 360 Teardown smartwatch, https://www.ifixit.com/Teardown/Motorola+Moto+360+Teardown/28891, Access date: Dec. 20, 2014", 2014.

[8] Android, (2015), "Developers web-page, http://developer.android.com/reference/packages.html, access date: Jan. 15, 2015", 2015.

[9] Pahl, G., and Beitz, W., (2013), Engineering design: a systematic approach, Springer Science & Business Media.

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

[11] Bennett, B., Cohn, A.G., Torrini, P., and Hazarika, S., (2000), "Describing rigid body motions in a qualitative theory of spatial regions", In Proceedings of the National Conference on Artificial Intelligence, pp. 503-509.

[12] Del Mondo, G., Stell, J.G., Claramunt, C., and Thibaud, R., (2010), "A graph model for spatiotemporal evolution", Journal of Universal Computer Science, Vol. 16 (11), pp. 1452-1477.

[13] Demoly, F., Matsokis, A., and Kiritsis, D., (2012), "A mereotopological product relationship description approach for assembly oriented design", Robotics and ComputerIntegrated Manufacturing, Vol. 28 (6), pp. 681-693.

[14] Demoly, F., Yan, X.-T., Eynard, B., Rivest, L., and Gomes, S., (2011), "An assembly oriented design framework for product structure engineering and assembly sequence planning", Robotics and Computer-Integrated Manufacturing, Vol. 27 (1), pp. 33-46.

[15] Duntsch, I., Wang, H., and McCloskey, S., (2001), "A relation-algebraic approach to the region connection calculus", Theoretical Computer Science, Vol. 255, pp. 63-83.

[16] Polkowski, L., (2014), "Mereology in engineering and computer science", in: In Mereology and the Sciences, Springer International Publishing, pp. 217-291.

[17] Salustri, F.A., (2002), "Mereotopology for product modeling: A new framework for product modeling based on logic", Journal of Design Research, Vol. 2, pp. 1-12.

[18] Simons, P.M., and Dement, C.W., (1996), "Aspects of the Mereology of Artifacts", in: Formal ontology, Springer, pp. 255-276.

[19] Liu, J., Lajolo, M., and Sangiovanni-Vincentelli, A., (1998), "Software timing analysis using HW/SW cosimulation and instruction set simulator", Proceedings of the In Proceedings of the 6th International Workshop on Hardware/Software Codesign, IEEE Computer Society, pp. 65-69.

[20] Mantripragada, R., and Whitney, D.E., (1999), "Modeling and controlling variation propagation in mechanical assemblies using state transition models", IEEE Transactions on Robotics and Automation, Vol. 15 (1), pp. 124-140.

[21] Randell, D.A., and Cohn, A.G., (1989), "Exploring naive topology: Modelling the force pump", In Proceedings of the Third International Qualitative Physics Workshop, pp. 1-14.

[22] Leśniewski, S., (1930), O podstawach matematyki [On the Foundations of Mathematics], Przegląd filozoficzny.

[23] Lejewski, C., (1983), "A note on Leśniewski's axiom system for the mereological notion of ingredient or element", Topoi, Vol. 2 (1), pp. 63-71.

[24] Kim, K.-Y., Yang, H., and Kim, D.-W., (2008), "Mereotopological assembly joint information representation for collaborative product design", Robotics and Computer-Integrated Manufacturing, Vol. 24 (6), pp. 744-754.

[25] Hovda, P., (2009), "What is classical mereology?", Journal of Philosophical Logic, Vol. 38 (1), pp. 55-82.

[26] Eschenbach, C., and Heydrich, W., (1995), "Classical mereology and restricted domains", International Journal of Human-Computer Studies, Vol. 43 (5), pp. 723-740.

[27] Polkowski, L., and Skowron, A., (1996), "Rough mereology: A new paradigm for approximate reasoning", International Journal of Approximate Reasoning, Vol. 15 (4), pp. 333-365.

[28] Zeng, Y., (2002), "Axiomatic theory of design modeling", Journal of Integrated Design and Process Science, Vol. 6 (3), pp. 1-28.

[29] Johnson, J., (1998), "Rough mereology for industrial design", Proceedings of the Rough Sets and Current Trends in Computing, Springer, pp. 553-556.

[30] Bjørner, D., and Eir, A., (2010), "Compositionality: Ontology and mereology of domains", in: Concurrency, compositionality, and correctness, Springer, pp. 22-59.

[31] Smith, B., (1996), "Mereotopology: A theory of parts and boundaries", Data and Knowledge Engineering, Vol. 20 (3), pp. 287-303.

[32] Galton, A., (1999), "The mereotopology of discrete space", in: Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science, Springer, pp. 251-266.

[33] Smith, B., and Welty, C., (2001), "Ontology: Towards a new synthesis", Proceedings of the Formal Ontology in Information Systems, ACM Press, USA, pp. iii-x, pp. 3-9.

[34] Borst, P., Akkermans, H., and Top, J., (1997), "Engineering ontologies", International Journal of Human-Computer Studies, Vol. 46 (2), pp. 365-406.

[35] Varzi, A.C., (1994), "On the boundary between mereology and topology", Proceedings of the 16th International Wittgenstein Symposium, Hölder-Pichler-Tempsky, Vienna, Austria, pp. 423–442.

[36] Cohn, A.G., and Varzi, A.C., "Mereotopological Connection", Journal of Philosophical Logic, Vol. 32 (4), pp. 357-390.

[37] Varzi, A.C., (1996), "Parts, wholes, and part-whole relations: The prospects of mereotopology", Data and Knowledge Engineering, Vol. 20 (3), pp. 259-286.

[38] Cohn, A., and Hazarika, S.M., (2001), "Continuous transitions in mereotopology", In Proceedings of the 5th Symposium on Logical Formalizations of Commonsense Reasoning, pp. 1-10.

[39] Lemon, O., and Pratt, I., (1998), "Complete logics for QSR: A guide to plane mereotopology", Journal of Visual Languages and Computing, Vol. 9, pp. 5-21.

[40] Allen, J.F., (1983), "Maintaining knowledge about temporal intervals", Communications of the ACM, Vol. 26 (11), pp. 832-843.

[41] Muller, P., (1998), "A qualitative theory of motion based on spatio-temporal primitives", Proceedings of the In Principles of Knowledge Representation and Reasoning-International Conference, pp. 131-143.

[42] Stell, J.G., and West, M., (2004), "A four-dimensionalist mereotopology", In Formal Ontology in Information Systems, pp. 261-272.

[43] Galton, A., (2004), "Multidimensional Mereotopology", KR, Vol. 4, pp. 45-54.

[44] Salustri, F.A., and Lockledge, J.C., (1999), "Towards a formal theory of products including mereology", Proceedings of the Proceeding of 12th International Conference on Engineering Design, pp. 1125-1130.

[45] Hubka, V., and Ernst Eder, W., (1987), "A scientific approach to engineering design", Design Studies, Vol. 8 (3), pp. 123-137.

[46] Andreasen, M.M., Howard, T.J., and Bruun, H.P.L., (2014), "Domain Theory, its models and concepts", in: An Anthology of Theories and Models of Design, Springer, pp. 173-195.

[47] Hansen, C.T., and Andreasen, M.M., (2002), "Two approaches to synthesis based on the domain theory", Engineering Design Synthesis: Understanding, Approaches and Tools, pp. 93-108.

[48] Zwicky, A.M., (1992), "Some choices in the theory of morphology", Formal grammar: Theory and implementation, pp. 327-371.

[49] Leyton, M., (1987), "A limitation theorem for the differentiable prototypification of shape", Journal of Mathematical Psychology, Vol. 31 (4), pp. 307-320.

[50] Vincent, L., (1989), "Graphs and mathematical morphology", Signal Processing, Vol. 16 (4), pp. 365-388.

[51] Serra, J., and Soille, P., (2012), Mathematical morphology and its applications to image processing, Springer Science & Business Media.

[52] Sutton, M.A., Orteu, J.J., and Schreier, H., (2009), Image correlation for shape, motion and deformation measurements: basic concepts, theory and applications, Springer Science & Business Media.

[53] Ghosh, P.K., (1993), "A unified computational framework for Minkowski operations", Computers & Graphics, Vol. 17 (4), pp. 357-378.

[54] Loncaric, S., (1998), "A survey of shape analysis techniques", Pattern recognition, Vol. 31 (8), pp. 983-1001.

[55] Kurfman, M., Rajan, J., Stone, R., and Wood, K., (2001), "Functional modeling experimental studies", Proceedings of the Proceedings of DETC2001, DETC2001/DTM-21709, Pittsburgh, PA.

[56] Szykman, S., Racz, J.W., and Sriram, R.D., (1999), "The representation of function in computer-based design", Proceedings of the Proceedings of the 1999 ASME design engineering technical conferences (11th international conference on design theory and methodology), Citeseer.

[57] Gavrilescu, M., Magureanu, G., Pescaru, D., and Doboli, A., (2010), "Accurate modeling of physical time in asynchronous embedded sensing networks", Proceedings of the Intelligent Systems and Informatics (SISY), 2010 8th International Symposium on, IEEE, pp. 477-482.

[58] Pourtalebi, S., and Horváth, I., (2016), "Towards a methodology of system manifestation features-based pre-embodiment design", Journal of Engineering Design.

[59] Pryor, R.W., (2009), Multiphysics modeling using COMSOL: a first principles approach, Jones & Bartlett Publishers.

[60] Ellis, M., Gaston, D., Forget, B., and Smith, K., (2011), "Preliminary Coupling of the Monte Carlo Code OpenMC and the Multiphysics Object-Oriented Simulation Environment (MOOSE) For Analyzing Doppler Feedback in Monte Carlo Simulations", Idaho National Laboratory (INL), 2011.

[61] Ribes, A., and Caremoli, C., (2007), "Salome platform component model for numerical simulation", Proceedings of the Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, Vol. 2, IEEE, pp. 553-564.

[62] Sigmund, O., (2001), "Design of multiphysics actuators using topology optimization – Part I: One-material structures", Computer Methods in Applied Mechanics and Engineering, Vol. 190 (49–50), pp. 6577-6604.

[63] Michopoulos, J.G., Farhat, C., and Fish, J., (2005), "Modeling and Simulation of Multiphysics Systems", Journal of computing and information science in engineering, Vol. 5 (3), pp. 198-213.

[64] Nowacki, W., (1975), Dynamic problems of thermoelasticity, Springer Science & Business Media.

[65] Tiersten, H., (1971), "On the nonlinear equations of thermo-electroelasticity", International Journal of Engineering Science, Vol. 9 (7), pp. 587-604.

[66] Michopoulos, J., and Sih, G., (1984), "Coupled theory of temperature moisture deformation and electromagnetic fields", Institute of Fracture and Solid Mechanics report IFSM-84-123, Lehigh University.

[67] Coleman, B.D., and Noll, W., (1963), "The thermodynamics of elastic materials with heat conduction and viscosity", Archive for Rational Mechanics and Analysis, Vol. 13 (1), pp. 167-178.

[68] Livne, E., (2003), "Future of airplane aeroelasticity", Journal of Aircraft, Vol. 40 (6), pp. 1066-1092.

[69] Farhat, C., (2004), "CFD-based nonlinear computational aeroelasticity", Encyclopedia of computational mechanics, Vol. 3, pp. 459-480.

[70] Smith, M.J., Cesnik, C.E., and Hodges, D.H., (2000), "Evaluation of some data transfer algorithms for noncontiguous meshes", Journal of Aerospace Engineering, Vol. 13 (2), pp. 52-58.

[71] Geuzaine, P., Grandmont, C., and Farhat, C., (2003), "Design and analysis of ALE schemes with provable second-order time-accuracy for inviscid and viscous flow simulations", Journal of Computational Physics, Vol. 191 (1), pp. 206-227.

[72] Zohdi, T.I., and Wriggers, P., (2008), An introduction to computational micromechanics, Springer Science & Business Media.

[73] Chen, W., and Fish, J., (2006), "A generalized space–time mathematical homogenization theory for bridging atomistic and continuum scales", International Journal for Numerical Methods in Engineering, Vol. 67 (2), pp. 253-271.

[74] Chung, P.W., (2004), "Computational method for atomistic homogenization of nanopatterned point defect structures", International Journal for Numerical Methods in Engineering, Vol. 60 (4), pp. 833-859.

[75] Fish, J., and Yuan, Z., (2005), "Multiscale enrichment based on partition of unity", International Journal for Numerical Methods in Engineering, Vol. 62 (10), pp. 1341-1359.

[76] Varzi, A.C., (1998), "Basic problems of mereotopology", In Formal Ontology in Information Systems, pp. 29-38.

[77] Gruhier, E., Demoly, F., Kim, K., and Gomes, S., (2014), "Mereotopology and product design", Proceedings of the Proceedings of TMCE 2014, Budapest. Hungary.

[78] Cesário, W., Baghdadi, A., Gauthier, L., Lyonnard, D., Nicolescu, G., Paviot, Y., Yoo, S., Jerraya, A.A., and Diaz-Nava, M., (2002), "Component-based design approach for multicore SoCs", Proceedings of the Proceedings of the 39th annual Design Automation Conference, ACM, pp. 789-794.

[79] Leavens, G.T., and Sitaraman, M., (2000), Foundations of component-based systems, Cambridge University Press.

[80] Purao, S., and Storey, V.C., (2005), "A multi-layered ontology for comparing relationship semantics in conceptual models of databases", Applied Ontology, pp. 117-139.

[81] Donnelly, M., (2003), "Layered mereotopology", Proceedings of the In IJCAI, pp. 1269-1274.

[82] Broy, M., (1997), "The Specification of System Components by State Transition Diagrams", Proceedings of the TUM-I 9630, Technische Universitat Munchen, Citeseer.

[83] Desharnais, J., Frappier, M., and Mili, A., (1998), "State transition diagrams", In Handbook on Architectures of Information Systems Springer Berlin pp, pp. 147-166.

[84] Lee, B., and Lee, B., (1998), "Hierarchical concurrent finite state machines in ptolemy", Proceedings of the Application of Concurrency to System Design, 1998. Proceedings., 1998 International Conference on, IEEE, pp. 34-40.

[85] Drusinsky, D., and Harel, D., (1989), "Using statecharts for hardware description and synthesis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 8 (7), pp. 798-807.

[86] Harel, D., (1987), "Statecharts: A visual formalism for complex systems", Science of Computer Programming, Vol. 8, pp. 231-274.

[87] Von der Beeck, M., (1994), A comparison of statecharts variants, Springer, Berlin.

[88] Demoly, F., Toussaint, L., Eynard, B., Kiritsis, D., and Gomes, S., (2011), "Geometric skeleton computation enabling concurrent product engineering and assembly sequence planning", Computer Aided Design, Vol. 43 (12), pp. 1654-1673.

[89] Horváth, I., and Thernesz, V., (1996), "Morphology-inclusive conceptual modelling with feature-objects", Proceedings of the in Proceedings of SPIE, Vol. 2644, pp. 563-572.

[90] Horváth, I., and van Der Vegte, W., (2003), "Nucleus-Based Product Conceptualization-Part 1: Principles and Formalization", Proceedings of the International Conference on Engineering Design, Stockholm, Sweden, pp. 1-10.

[91] Deelman, E., Gannon, D., Shields, M., and Taylor, I., (2008), "Workflows and e-Science: An overview of workflow system features and capabilities", Future Generation Computer Systems, Vol. 25 (5), pp. 528-540.

[92] Wasserman, A.I., (1985), "Extending state transition diagrams for the specification of human-computer interaction", IEEE Transactions on Software Engineering, Vol. 8, pp. 699-671.

[93] E.A., L., (2010), " CPS foundations", Proceedings of the in Proceedings of the 47th Design Automation Conference, ACM, pp. 737-742.

[94] Horváth, I., and Gerritsen, B.H., (2012), "Cyber-physical systems: Concepts, technologies and implementation principles", Proceedings of the Proceedings of the International Tools and Methods of Competitive Engineering Symposium, Vol. Vol. 1, Delft University of Technology, Karlsruhe, Germany, pp. 19-36.

[95] Rajkumar, R., Lee, I., Sha, L., Stankovic, J., and Conference, A.C.M., (2010), "Cyber-physical systems: The next computing revolution", Proceedings of the in Proceedings of the 47th Design Automation New York, ACM, New York, pp. 731-736.

[96] Bellomo, N., Bianca, C., and Mongiovì, M.S., (2010), "On the modeling of nonlinear interactions in large complex systems", Applied Mathematics Letters, Vol. 23, pp. 1372-1377.

[97] Horváth, I., and Gerritsen, B., (2013), "Outlining nine major design challenges of open, decentralized, adaptive cyber-physical systems", Proceedings of the ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. Volume 2B, Portland, Oregon, USA.

[98] Horváth, I., (2014), "What the design theory of social-cyber-physical systems must describe, explain and predict?", in: An Anthology of Theories and Models of Design, Springer London, pp. 99-120.

[99] Asher, N., and Vieu, L., (1995), "Toward a geometry of common sense: A semantics and a complete axiomatization of mereotopology", Proceedings of the IJCAI (1), Citeseer, pp. 846-852.

[100] Kim, K.-Y., and Yang, H., (2008), "The Role of Mereotopology and SWRL Rules to Represent Joint Topology Information for Design Collaboration", Proceedings of the ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. 131-139.

[101] Borgo, S., Guarino, N., and Masolo, C., (1996), "A pointless theory of space based on strong connection and congruence", KR, Vol. 96, pp. 220-229.

Chapter 4

SPECIFICATION OF SMFs

structuring and processing information

The theoretical framework developed in the previous research cycle was used as the basis for theorizing and specification of system manifestation features (SMFs). Our assumption was that a SMF-based CPS modeling theory, methodology, and computational tool can be developed based on the analogy of the conventional part-feature technology, and considering the implications of MOT. As far as the theory of system manifestation features is considered, it was seen as a complement of MOT. From a methodological point of view, SMFs are transdisciplinary modeling building blocks of high semantics, which are predefined in parametric forms and instantiated in particular application context. The generic information structure and processing methodology of SMFs have been defined based on the proposed SMF theory. Parameterization of SMFs extends to structure, morphology and attributes. SMFs were introduced to address several composability and compositionality issues of multi-disciplinary CPS architecting and design.

The main objective of the empirical and rational investigations was to transfer the SMFs theory into a comprehensive information structure that captures all required pieces of information from both architectural and operational aspects. The objective of this chapter is to present our findings concerning the required detailed pieces of information, as well as their structuring, as needed for shaping the architecture and operation knowledge frames (AKFs and OKFs). The concepts supporting the SMFs theory are presented in Sub-chapter 4.2. The concepts captured in the architecture knowledge frames and in the operation knowledge frames are introduced and explained through practical cases in Sub-chapters 4.3 and 4.4, respectively. The issues of interlacing of the knowledge frames on the same aggregation level, or among different aggregation levels are explained in Sub-chapter 4.5. The last Sub-chapter discusses the main findings and analyzes their implications. The results presented in this Chapter were used for the specification of the computational constructs for implementation of SMFs-based modeling tool that are discussed in Chapter 5.

4 SPECIFICATION OF SMFS: Structuring and processing information

4.1 INTRODUCTION

4.1.1 Objectives of the work and the chapter

This Chapter reports on the work done and results achieved in the third research cycle of the promotion research. It was discussed in the preceding Chapters that an efficient conceptualization of CPSs needs multi-disciplinary tools and unified approaches [1]. As a novel approach, the idea of system-level features (SLFs) has been introduced for modeling these intricate systems [2]. As it can be seen in the literature, though claimed to be promising, elaboration of the concept and implementation of system-level feature-based support of structural configuration and behavioral simulation of CPSs is still in an early stage, actually it is proliferating as a research issue in the time of writing this thesis. Primarily, the major challenges for CPSs developers are posed by the heterogeneity of the constituents and the intricacy of their interactions, in combination with their broad functional spectrum [3] [4]. Designers may come across with some of these challenges already in the pre-embodiment design phase, where off-the-self or custom-made heterogeneous constituents should be built together and their interoperability should be achieved [5].

Though it is often named differently, e.g. as configuration design [6], the challenges and specific issues of pre-embodiment design are well acknowledged in the literature [7]. The major novelty of the work reported in this Chapter is that we imposed a massively physical view on pre-embodiment design and modeling. The necessity of this was explained in Chapters 2 and 3. It has been realized that by doing so we can develop a tool that may go well beyond the affordances of traditional logic-based modeling tools or languages. In the previous Chapter it was explained that the mereo-operandi theory has been ideated and developed to underpin transdisciplinary modeling and pre-embodiment design of intricate heterogeneous systems. It includes numerous concepts to (i) uniformly represent various parts of these systems, (ii) facilitate integration of their inherently diverse hardware, software, and cyberware constituents [8], and (iii) represent not only the architectural relations and composition of these constituents as aggregated-ware (AW), but also their operations and interactions as a synergetic whole.

4.1.2 Research methods applied in this research cycle

In the third research cycle, we concentrated on a complementing theory that supports formal definition and specification of information sets and knowledge frames for system-level features-based modeling. The research mainly concentrated on the *theoretical fundamentals* of SMFs and the *computational framework*. Towards these ends, dominantly critical system thinking was used as a research method, combined with empirical case studies done with concrete consumer durables. This combination of the methods allowed us to conduct systematic exploration as well as rigorous validation of the theoretical

concepts. Throughout this chapter, an explanatory practical case is used to explain the concepts and to demonstrate the feasibility of the proposed approach.

Figure 4-1 Side lift with the Pablo rehabilitation/ training system

The chosen case of study is a commercialized product, marketed under the fantasy name Pablo®[1] (Figure 4-1). Developed for motor rehabilitation of shoulder, lower arm, and hand functions, and for repetitive training, this basic system (device) can be connected to a PC via a USB interface. Including a hand loop, an element of the device is a sensor grip that makes it possible to perform exercises and contains strength and movement sensors to measure the forces of various (cylinder and pinch) grip patterns, and stretching and bending of the human hand. The device is able to measure the mobility range of the arm and the different characteristics of movements. The handle can determine which position the hand is in and when the exercise is finished. This simple system will be used to clarify the concepts introduced in association with both architecture and operation modeling.

4.1.3 SMF theory as a bridge between MOT and implementation

As a starting point for our research we presumed that: (i) the requirements engineering phase of pre-embodiment design has been completed, (ii) the concepts concerning the overall architecture and operations of the system have been devised, and (iii) sufficient number of pre-designed SMFs (as modeling building blocks that resembles architectures and operations of the real world components of CPSs) are available in a pre-embodiment design phase. The latter obviously minimizes the need for creative or innovative design actions [9]. [10]. Other technical assumptions were that intricate technical systems, such as CPSs: (i) consist of large number of off-the-shelf or only pre-designed components [11], (ii) can be modeled by using SMFs, (iii) both the architecture and the operations of the systems can be described by an SMFs-based system model [12], and (iv) can be adapted by a systematic configuration process, or by using the embedded customization options by the end-users [13]. Consequently, our work aimed at modeling the architecture and operations of CPSs on system-level as well as on multiple component levels in their physicality. As an ultimate objective, realization of a support tool was set. The target tool was supposed to support both the development of SMF entities (resembling components of a system) and their composition into a system model. The necessity of relying on novel architectural and operational modeling principles in the development process was obviously recognized.

[1] See:http://tyromotion.com/en/products/pablo

MOT offers a rather comprehensive theoretical platform for concurrent modeling of architectural elements and their operations. Other high light of MOT is that it can be straightforwardly operationalized in the practice as it is exemplified in the rest of the Chapter. The theoretical framework entailed by MOT was used to guide the conceptualization of SMFs and to transfer the theoretical concepts into a computational approach [14]. By this transformation a big step is made towards computational implementation of system-level feature-based modeling of CPSs. Epistemologically, the theory of SMFs is seen as a complement of MOT towards a practical methodology. This complementing theory: (i) fosters the development of specific procedures, methods and algorithms for a unified handling of system-level features, (ii) interlinks their architectural and operational aspects, (iii) enables the development of SMFs ontologies and libraries, and (iv) facilitates the management of interaction of complex systems with the stakeholders (users) and the surrounding environment [15].

The theory of SMFs simply depicts: (i) the pieces of information that should be captured for defining uniform modeling entities named as SMFs, (ii) the interrelations among these pieces of information inside the SMFs, and (iii) the interlacing of SMFs in a same and different aggregation levels in order to model a heterogeneous system. These pieces of information are captured by information structures offered by the introduced knowledge frames. They enable specifying SMFs on various aggregation levels and are flexible enough to allow both architecture and operation aggregation and interfacing even if the architectural and operational connections of SMFs are not standardized, or do not obey set contracts.

4.2 CONCEPTUALIZATION OF SYSTEM-LEVEL FEATURES

4.2.1 Revisiting the concept and technology of features

One of our first research questions concerned the opportunity of extending traditional feature technology to system-level features. A *feature* is a notable property, as well as a distinctive (prominent) characteristics of things (artefacts, systems, processes and phenomena) that sets them apart from similar items for humans or smart systems [16]. In the classical theory of engineering product features, a feature is something that has significance in a given context [17] and from a semantic point of view [18]. Feature technology has been extended to applications where geometry, structure, materialization, implementation, etc. imply some meaning in the context of a particular application (e.g. as is done by manufacturing features, assembly features, piping features, etc.) [19]. They are often called *part features* or *application features*. Though it was addressed by many researchers, transformation between feature spaces and conversion of feature definitions between various applications remained a partially solvable problem due to the need for handling intrinsic meanings [20]. In the last two decades, attention was paid to ontology-based feature definition [21], information sharing [22], mapping [23], and feature conversion [24]. Feature-based design had more influence on the detailed design and planning activities of product development processes, than on the conceptual or pre-embodiment design activities [25]. This comes from the dependence of meaning on the details of manifestation [26]. We hypothesized that the concept of features and the idea of feature-based design can be generalized, adapted and reused as a methodological analogy in the context of multi-disciplinary systems. Accordingly, the core principles of the classical feature theory

have been imported into a specific system-level feature theory and have been extended with new principles that are entailed by MOT and the contexts of CPSs.

System manifestation features are concurrently genotypic and phenotypic in nature. In our context, 'genotypic' means that SMFs can be used to determine the overall composition or makeup of a system, or a component thereof. Phenotypic means that they can also be used to determine specific physical and visual traits of a system/component. Technically, these are made possible by multi-level and multi-aspect parameterization of SMFs. The system-level features developed by using the information structures proposed the SMFs theory give the opportunity for system designers to focus their attention on those aspects of the system that are the most relevant and important at a given moment.

4.2.2 Need for multi-aspect information structuring and integral management

SMFs are regarded as architectural domains of systems having significance from operational viewpoint. As system modeling entities, SMFs represent both physical and computing transformations of domains. A particular SMF may occur multiple times in a system in different instantiations (with different sets of values assigned to its parameters and annotations). SMFs may also appear on multiple aggregation levels. In the language of topology it means that various rougher and finer domain topologies can be interpreted over the overall domain of a system. For instance, a digital processor simultaneously represents an architectural domain and an operation performer. In a higher resolution, the domain of the processor is an aggregate of a number of interrelated digital elements, and the operation of the processor is the sum of the specific operations of these elements. This issue is called 'multi-granularity of SMFs' and will be revisited in the next subchapters from different perspectives. SMFs can be uniquely characterized by spatiotemporal, architectural, and operational attributes. Just like in the case of traditional part features, SMFs can be represented by computational constructs containing a structured set of interrelated parameters, constraints, values, and semantic annotations.

A basic requirement is that all information sets required for modeling and simulation of a CPS should be incorporated in a SMF. To a large extent, MOT clarified these information sets, i.e. morphology, flow of operation, methods, procedure, state transition, Input/output transformation, streams, containment relations, and connectivity relations. The multi-aspect structuring of SMFs implies the need for: (i) description of domains of different physical extents and aggregation levels, (ii) capturing and representing operations happening on different aggregation levels, and (iii) interfacing between different architecture and operation levels. SMFs are seen by MOT as prefabricated 'components' that can be used in designing both the architecture and the operation of a system. In other words, instead of working with two separate models for architecture and operations, the proposed computational approach uses SMFs as dual-aspect building blocks. From a computational point of view, an SMF is a virtual entity parametrically representing a specific domain and the related operations.

Using SMFs supports fast configuration and adaptation of the architecture and operations of systems. It also facilitates the reuse of components in pre-embodiment design of different system variants. Nevertheless, aggregation of SMFs brings the challenge of *feature interfacing*, which is a well-known issue for the traditional feature theory, into the forefront [27]. In addition, SMFs should be made interoperable in order to realize the functional

objectives of the designed systems [28]. As mentioned in the literature, features trigger ideas concerning the way of organizing knowledge that facilitates computing and inferences [7] [29]. Below we utilize this potential from both architectural and operational aspects.

4.3 ARCHITECTURE KNOWLEDGE FRAME FOR SYSTEM MANIFESTATION

In our research, we followed a 'from-concrete-to-abstract' reasoning in explaining how architectural entities and relationships of SMFs could be derived and represented. We (i) started out from a concrete existing system of moderate complexity, (ii) identified its system manifestation features, (iii) took one particular SMF, (iv) identified its components and constituents, (v) described their spatiotemporal mereotopological relations, (vi) specified the operational procedures and computational methods, and (vii) defined the computational knowledge frame of the concerned SMF. As interpreted by MOT, an architectural/operational domain of a CPS is the spatiotemporal manifestation of an AW component, which may include HW, SW and CW constituents. Accordingly, we described the domains of SMFs, the parts of the domains, and the relationships among them by spatiotemporal mereotopological operations introduced in [12]. As architectural relationships within a domain and between domains, both *part-of* and *connected-to/with* relationships have been applied. It is a technical matter, but is worth mentioning, that specific graphical and symbolic representations have been used for the *part-of* relationship of physical aggregates. For instance, the graphical relation symbol A◇⟶B means that *B* is a part of *A*. A connected-to relation is represented by an A•⟶B arrow, and a connected-with relation is represented by an A•—•B arrow. We will use these graphical symbols below.

4.3.1 Defining architecture aggregation levels

According to SMF theory, the aggregate of all first level components of a system is the system itself. The aggregates of second level components are the first level components, and so forth - down to the levels of constituents. Various graphical entity symbols were used in our research to indicate the type of architectural entities on the respective levels of aggregation. The hierarchical relations between them are captured and indicated only as logical relationships, while the physical relationships are captured by the mereotopological expressions.

Description of the hardware constituents of an SMF is more straightforward than that of the software constituents. The reason is that a SW constituent exists in two alternative forms: (i) in source (language) code (when it is a kind of white box, whose content is directly readable and changeable by programmers), and (ii) in compiled or binary code (when it is a kind of black box, whose machine instruction content is to be processed by processors). These two forms are discrete and need different treatment from both architectural and operational points of view. Notwithstanding that both forms should be considered in design, below we deal with only the run-time form of SW constituents. The possible containment relationships of components and constituents can be formally described by the following mereotopological expressions (relations):

$PSysCom^{1}_{Ab}$: describes that component AW_b is a level 1 component and as such is directly *part-of* the system.

PCom$^{1}_{Aa}$Com$^{2}_{Ad}$: represents that component AW_d is a level 2 component and is a *part-of* AW_a, which is a component of aggregation level 1.

PCom$^{3}_{Af}$Con$^{4}_{Hg}$: means constituent HW_g is a level 4 constituent and is a *part-of* component AW_f, which is a component of aggregation level 3.

PCon$^{4}_{Sh}$Con$^{5}_{Sj}$: means constituent SW_j is a level 5 constituent and is a *part-of* constituent SW_h, which is a constituent of aggregation level 4.

The other architectural relationships between entities were described by either *connected-to* or *connected-with* relations. These types of relations played a dual role as: (i) spatial neighborhood relations, and as (ii) carrier architectural relations of some operations. In the latter role, they can be: (a) one-directional relations (expressing dependence of B on A), or (b) bi-directional relation (expressing mutual operational interdependence of A and B).

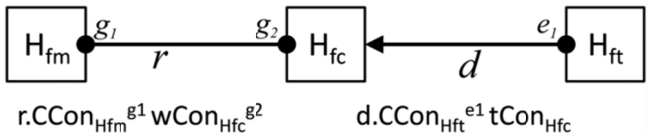


Figure 4-2 Various part-of relations on multiple levels of aggregation

The graphical scheme, shown in Figure 4-2, represents two plates (H_{fm} and H_{fc}) of a capacitor that are remotely *connected-with* each other, and there is one connector (H_{ft}), which is in a direct connection with one of the plates, H_{fc}. The *connected-to/with* type relations are indicated by letters placed below the connection line (i.e. *r* stands for 'remote', and *d* is for 'direct' connection). The symbols above the arrows (at the ends of the connection line) indicate the enabler of the remote or direct connection of the components (i.e. g_1 and g_2 stand for electromagnetic fields, and e_1 for electric current). The mereotopological relations of the entities H_{fm}, H_{fc} and H_{ft} are described symbolically as: r.CCon$_{Hfm}^{g1}$wCon$_{Hfc}^{g2}$, and d.CCon$_{Hft}^{e1}$tCon$_{Hfc}$. Note that the codes can indicate various different kinds of dependencies and connectivity (i.e. existence, processing, heat producing, energy consumption, information transmission).

4.3.2 Specification of containment relationships

Let us take the case of the Pablo system now. From an architectural point of view, this system as a whole has a physical part and a computational part. Both can be modeled as an aggregate of SMFs. Among others, the physical part includes an accelerometer sensor, which consists of a MEMS (micro-electro-mechanical system) detector, H_f, and an ASIC (application-specific integrated circuit) converter, A_s. These two parts will be used as representative SMFs in our further investigations. The lower-level architectural components and constituents of the MEMS detector and the ASIC converter, as well as their relationships, are shown in the lower part of Figure 4-4.

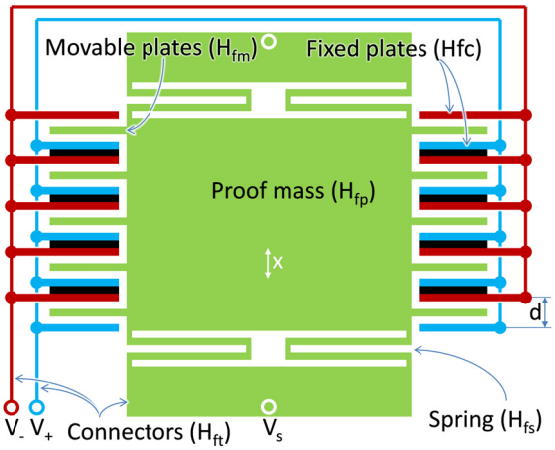


Figure 4-3 Schematic architecture of the MEMS detector (H_f)

On a lower level (of de-aggregation), the elements of the accelerometer capture the acceleration of movements, and convert it into a capacitance change. As shown by its

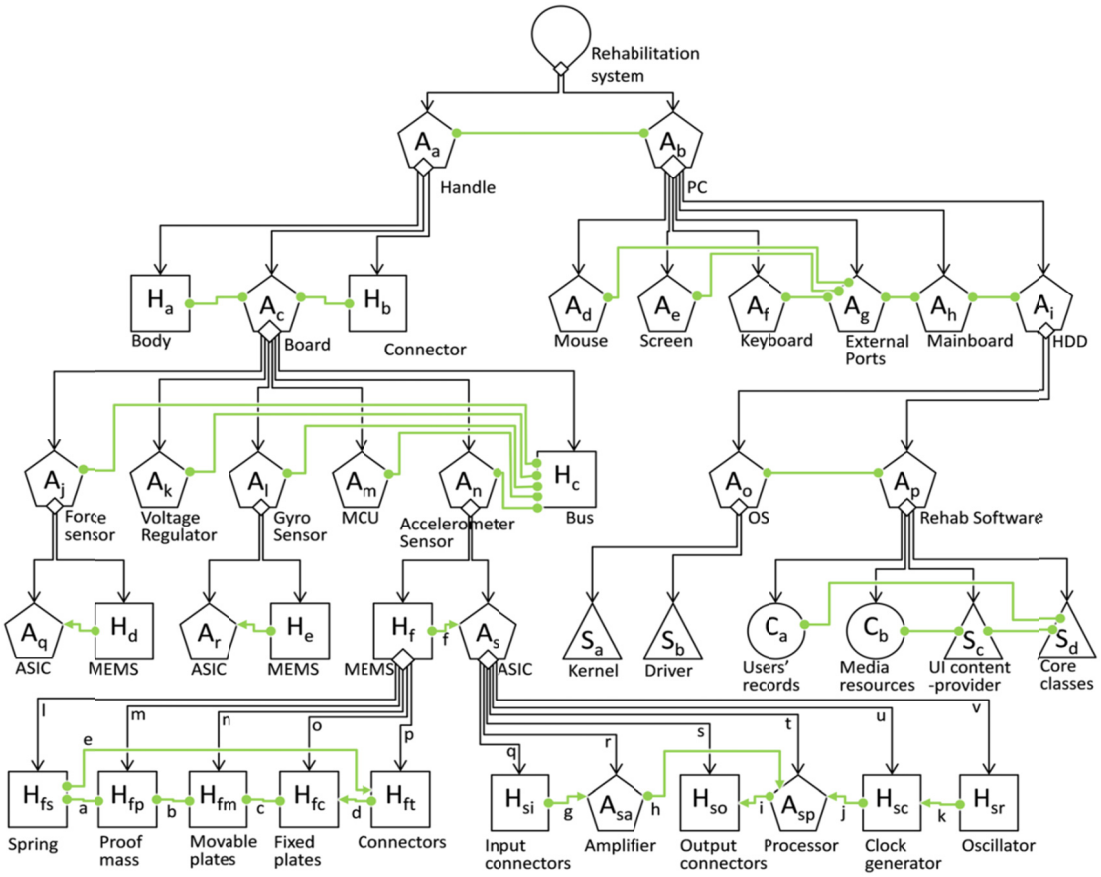


Figure 4-4 Examples of connected-to/connected-with relations

schematic architecture in Figure 4-3, the MEMS detector (H_f) implements this functionality through a number of hardware constituents. The *proof mass* (H_{fp}) moves up and down due to acceleration. The *springs* (H_{fs}) at two ends of H_{fp} brake and reinstate it. There are two kinds of *fixed plates* (H_{fc}) connected to *V+* and *V-*. Together with the *movable plates* (H_{fm}), which are connected to and move with H_{fp}, these plates create capacitors. The exerted acceleration changes the distance between each H_{fm} and H_{fc} plates, hence changes the electric current. The change of the electric current is proportional with the acceleration. These *part-of* (containment) relations can be formally specified as given below:

a: $\mathbf{PCon^4_{Hf}Con^5_{Hfs}}$

b: $\mathbf{PCon^4_{Hf}Con^5_{Hfp}}$

c: $\mathbf{PCon^4_{Hf}Con^5_{Hfm}}$

d: $\mathbf{PCon^4_{Hf}Con^5_{Hfc}}$

f: $\mathbf{PCon^4_{Hf}Con^5_{Hft}}$

These symbolic expressions can be read as declarative statements. For instance, the meaning of expression (a) is: 'The spring (H_{fs}), which is a constituent on the fifth aggregation level, is a part of the MEMS (H_f), which is a constituent on the fourth aggregation level'.

4.3.3 Specification of connectivity relationships

Figure 4-5 specifies and graphically visualizes the connectivity and containment relations between the constituents of the MEMS accelerometer. The constituents are all hardware constituents, working based on different physical phenomena and effects. The following symbols were used to indicate the characteristic attributes of the included constituents: (i) *m*: movement, (ii) *r*: resistance against deformation, (iii) *e*: electric current, and (iv) *g*: electromagnetic field. The connectivity relationships of the constituents can be described by the connected-to and connected-with relations given below.

a: $\mathbf{d.CCon_{Hfs}^{r1}\,wCon_{Hfp}^{m2,e3}}$

b: $\mathbf{d.CCon_{Hfp}^{m1}\,wCon_{Hfm}^{e2}}$

c: $\mathbf{r.CCon_{Hfm}^{g1}\,wCon_{Hfc}^{g2}}$

d: $\mathbf{d.CCon_{Hfs}^{e4}\,tCon_{Hft}}$

e: $\mathbf{d.CCon_{Hft}^{e1}\,tCon_{Hfc}}$

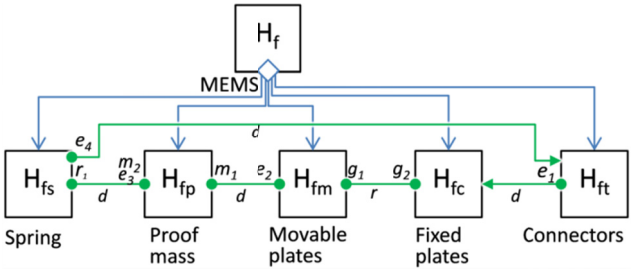


Figure 4-5 Connectivity and containment relations among the constituents of H_f

For instance, the symbolic expression (a) means: 'The proof mass ($\underline{H_{fp}}$) as a constituent is directly connected with the spring ($\underline{H_{fs}}$) constituent'. The enabler on the spring-side of the connection is resistance against deformation (r_1), while the enablers on the proof mass-side of the connection are movement ($\underline{m_2}$) and electric current ($\underline{e_3}$). The symbols indicating the enablers clarify what are shared in the mutual connections. For instance, the proof mass deforms the spring, and the spring exerts a reinstating force. Moreover, $\underline{H_{fp}}$ conducts electric current to $\underline{H_{fs}}$. Using this form of symbolization of the containment and connectivity relationships supports their computational processing.

4.3.4 Formal specification of architecture knowledge frames

To capture the architectural entities and relationships of SMFs for computation, the concept of *architecture knowledge frames* (AKFs) were introduced. Every AKF represents one particular domain of manifestation and, by doing so, forms the spatiotemporal basis of exactly one SMF. As a structured spatiotemporal abstraction, this domain may represent a whole system, or any one of its first-, second-, third-level, or so forth de-aggregates, which are treated uniformly as SMFs. The abstracted architectural domains of SMFs reflect one particular level of component aggregation. The architected system, as a top-level formation, manifests itself as the aggregate of the first-level compound components (AW), while the lowest-level domains of components are formed by aggregates of HW, SW and/or CW constituents. The domains on the intermittent levels are formed in the same aggregative manner. The specific contents of AKFs are the basis of the digital processing of the architectural entities and their relationships. It is to be noted that in the case of systems of higher complexity, additional levels of aggregation/de-aggregation can be considered, and thus AKFs can in principle be extended to any number of aggregation levels.

An architecture knowledge frame consists of the following sections: (i) header, (ii) domain metadata, (iii) domain entities, (iv) entity attributes, (v) entity morphologies, (vi) spatial positions, (vii) containment relations, (viii) connectivity relations, (ix) input assumptions, (x) output guarantees, and (xi) auxiliary data (Figure 4-6). The header section contains the

identifiers of all entities that can be referenced in containment and connectivity relations, in addition to the overall attributes and states of the domain. Altogether these form a part of the various computational information structures describing SMFs. During computation of operations, references are made to these fields of AKFs from *operation knowledge frames* (OKFs). These external structural connections of SMFs are included in the 'domain metadata' section of the AKFs. This way, any web-type arrangement of domains of SMFs can be captured. The names and purposes of the represented SMFs are also included in meta-data field, to support informing designers and keyword-based retrieval.

AKF // H_f //

header	*entity identifiers*		*domain attributes*		*domain states*	
	MEMS accelerometer: H_f connectors: H_{ft} fixed plates: H_{fc} movable plates: H_{fm} proof mass: H_{fp} spring: H_{fs}		digital hardware: DHW analogue hardware: AHW silicon: Si spring deformation limit: X_S		acceleration: a supply energy: se output voltage: ov working temperature: TA	
domain metadata	*description:* *purpose:* *structure:*	MEMS detector hardware component capturing acceleration of movements and converting it into capacity change: UoO_{CM} $A_a \diamond\!\!\rightarrow A_c \diamond\!\!\rightarrow A_n \diamond\!\!\rightarrow H_f$				
domain entities	*list of DEs:*	H_{ft}	H_{fc}	H_{fm}	H_{fp}	H_{fs}
entity types	*groups of ETs:*	AHW	AHW	AHW	AHW	AHW
architectural attributes	*list of AAs:*	Si	2 x (5 finger) Si	2 x (5 finger) Si	Si	X_S: 5 µm Si
entity morphologies	*size of closure box:* *model files of EMs:*	(520 x 80) µm -	(680 x 180) µm -	(500 x 170) µm -	(250 x 170) µm -	(250 x 85) µm -
spatial positions	*records of SPs (centre):*	(260 x 40) µm	(340 x 300) µm	(340 x 300) µm	(340 x 300) µm	(340 x 110) µm
containment relations	*list of COs:*	$PCon^4_{Hf}Con^5_{Hft}$	$PCon^4_{Hf}Con^5_{Hfc}$	$PCon^4_{Hf}Con^5_{Hfm}$	$PCon^4_{Hf}Con^5_{Hfp}$	$PCon^4_{Hf}Con^5_{Hfs}$
connectivity relations	*list of CRs:*	$d.CCon_{Hfs}{}^{r1}wCon_{Hfp}{}^{m2,e3}$ $d.CCon_{Hfp}{}^{m1}wCon_{Hfm}{}^{e2}$ $r.CCon_{Hfm}{}^{g1}wCon_{Hfc}{}^{g2}$ $d.CCon_{Hfs}{}^{e4}tCon_{Hft}$ $CCon_{Hft}{}^{e1}tCon_{Hfc}$				
input assumptions	*list of IAs:*	a: ± 50g se: 3.6V, 500 µA				
output guarantees	*list of OGs:*	ov: 800~0 mV				
auxiliary data	*list of ADs:*	–40°C < TA < 105°C				

Figure 4-6 Instantiation of AKF in the case of the MEMS detector, H_f

As examples, we present two instantiations of AKF for two different SMFs. The first AKF is instantiated for the MEMS accelerometer (H_f), which is an electromechanical component. The AKF shown in Figure 4-6 captures the pieces of information needed to describe the architectural domains of this electromechanical component, including the physical activating constituents, whose names and relationships are shown in Figure 4-5.

Figure 4-7 shows an instance of another AKF, which is the knowledge frame of the ASIC converter of the accelerometer (A_s). This is a computational component. The AKFs of these SMFs include the specification of the lower level entities making up their domains, assign reference-able identifiers to each of them, and specify their types, architectural attributes, convex spatial closures and reference points relative to the reference system of coordinates of the domain. The containment and connectivity relations are specified for all included entities. In general, the morphology of the physical entities can be specified by *skeleton models* (if the entity geometries are still in the development process), or by computer aided design (*CAD) geometries* (if they are standard or commercialized components).

AKF // A_s //

header	*entity identifiers*		*domain attributes*		*domain states*		
	Detector: A_{sa} in-connectors: H_{si} out-connectors: H_{so} processor: A_{sp} clock generator: H_{sc} oscillator: H_{sr}		digital hardware: DHW analogue hardware: AHW silicon: Si copper: Cu		acceleration: a supply energy: se output voltage: ov output signal: os		
domain metadata	*description:* *purpose:* *structure:*	ASIC converter aggregate-ware component converting input voltage into acceleration, velocity and displacement values: UoO_{CA} $A_a \diamond\!\!\rightarrow A_c \diamond\!\!\rightarrow A_n \diamond\!\!\rightarrow A_s$					
domain entities	*list of DEs:*	H_{si}	A_{sa}	H_{so}	A_{sp}	H_{sc}	H_{sr}
entity types	*groups of ETs:*	AHW	DHW	AHW	DHW	DHW	DHW
architectural attributes	*list of AAs:*	Cu	Si	Cu	Si	Si	Si
entity morphologies	*size of closure box:* *model files of EMs:*	- -	- -	- -	- -	- -	- -
spatial positions	*records of SPs (centre):*	-	-	-	-	-	-
containment relations	*list of COs:*	$PCom^4_{As}Con^5_{Hsi}$	$PCom^4_{As}Com^5_{Asa}$	$PCom^4_{As}Con^5_{Hso}$	$PCom^4_{As}Com^5_{Asp}$	$PCom^4_{As}Con^5_{Hsc}$	$PCom^4_{As}Con^5_{Hsr}$
connectivity relations	*list of CRs:*	$CCon_{Hsi}\ tCom_{Asa}$ $CCom_{Asa}\ tCom_{Asp}$ $CCom_{Asp}\ tCon_{Hso}$ $CCon_{Hsc}\ tCom_{Asp}$ $CCon_{Hsr}\ tCon_{Hsc}$ $CCon_{Hsi}\ fCon_{Hft}$					
input assumptions	*list of IAs:*	ov: 800~0 mV se: 3.6V, 500 μA					
output guarantees	*list of OGs:*	os: I^2C protocol					
auxiliary data	*list of ADs:*	-					

Figure 4-7 Instantiation of AKF in the case of the ASIC converter, A_s

4.4 OPERATION KNOWLEDGE FRAME OF SYSTEM MANIFESTATION FEATURES

Theoretically, operation of an SMF can be only existence or (existence and) transformation. Every domain that is identified from an architectural point of view is supposed to be

existent, but not necessarily conducting transformation. In the case of an existence operation, the attributes of the start state and the end state of a domain are identical, i.e. a 'no-change situation' is idealized. In the case of a transformation operation, the start state of an existent architectural domain is converted into a different end state under the effects of various physical phenomena. That is, manifestation changes are assumed, which also concern the conversion of the input events, quantities and qualities into the output events, quantities and qualities. The transformation can be physical or computing.

4.4.1 Operation aggregation levels

A reductionist stance was taken in our work and based on this we argue that flows and units of operations can be reasoned out from the overall operation of a system, and that the overall system operation can be aggregated from a finite set of discrete, but interlaced flows and units of operations. However, it does not entitle us to deal with non-linear, stochastic, or random systems. In line with these, one of the foundational assumptions of MOT is that, likewise architectural domains, operations of SMFs can be generated by aggregation. It has also been assumed that the flows and units of operations are (i) discrete (individually identifiable), (ii) finite (in terms of cardinality and occurrence), (iii) definitive (implied by physical and computational laws), and (iv) orientated (i.e. not reversible in the operation process of a system). In the following paragraphs, we present the concepts and principles that help consideration of these assumptions in the computational specification of the operations of SMFs.

The overall operation of a domain is described by a *flow of operation* (FoO), while the operation of the entities is described by a *unit of operation* (UoO). Since a domain is an architectural equivalent of a particular SMF, the overall operations of SMF are captured by FoOs that are aggregates of UoOs. Since domains may represent architectural manifestation of SMFs of various aggregation levels, a FoO plays the role of a UoO on a higher aggregation level, while a UoO plays the role of a FoO on a lower aggregation level. A practical implication is that specification of a higher-level SMF requires the specification of the operations of all incorporated lower-level SMFs. The involvement of SMFs in performing an operation can be conditional and temporal. These issues need attention at specifying the conduct of transformations performed by SMFs.

As discussed in Chapter 3, a specific logical model was introduced to facilitate a consistent (formal) representation of FoOs of SMFs. According to this model, the operations related to a domain of any architectural level can be defined by seven conceptual elements, namely, by: (i) the description of the domain itself, D, (ii) start state of the domain, SS, (iii) end state of the domain, ES, (iv) input events and values, I, (v) output events and values, O, (vi) procedure of transformation, P, and (vii) methods that are associated with procedural elements, M. Symbolically, $\mathrm{FoO} = \{\mathrm{D}, \mathrm{SS}, \mathrm{I}, \mathrm{P}, \mathrm{M}, \mathrm{O}, \mathrm{ES}\}$. With respect to digital processing, transformations are described by duals of a *procedure* (logic of implementing the transformation) and *methods* (computational algorithms of implementing the transformation). As mentioned earlier, a transformation can be of physical or computational nature. A procedure determines: (i) the UoOs, which eventually transform the concerned material, energy and information streams, (ii) the timed logical sequences of these UoOs, and (iii) the permanent or conditional constraints imposed on UoOs. Further explanation on these will be given in the following subsections. The specification of the procedures depends on the aggregation level the domain of an SMF represents.

Due to the bijective relationship between a domain and the operations performed by it, whenever the architectural attributes of the domain are changed, the performed operation will also change, and vice versa. Interaction between domains and operations should be considered across the aggregation levels. The reason is that a lower-level UoO can contribute to multiple higher level FoOs and/or UoOs as procedural element, or may have direct or indirect effects on other operations. Therefore, while a system can logically be de-aggregated into a containment hierarchy, its overall operation cannot be de-aggregated purely hierarchically. As a trivial example, a processor as an architectural domain is a unique part of the logic board and a unique aggregate of several subdomains, but it can contribute to the realization of operations on multiple operational levels of the system.

4.4.2 Streams of energy, information and material

The transformations made by SMFs extend to material (M), energy (E) and information (I) streams. In Figure 4-8, an example is given to show how higher level operations are aggregated from lower level operations. This figure also shows how the procedural relationships of UoOs are established on different de-aggregation levels. Like domains and parts thereof in the architectural realm, operations are in containment relationship with each other (that is, an operation can be specified as part of another operation). The architectural 'connected to/connected with' relations are carriers of operational relations of FoOs that specify the transformations of M-E-I streams. These streams are represented by arrows in Figure 4-8. These show the logical ordering (dependencies) of the related UoOs and FoOs, but temporal ordering or time sequences are not represented. As shown, each UoO represents a FoO in a lower level.

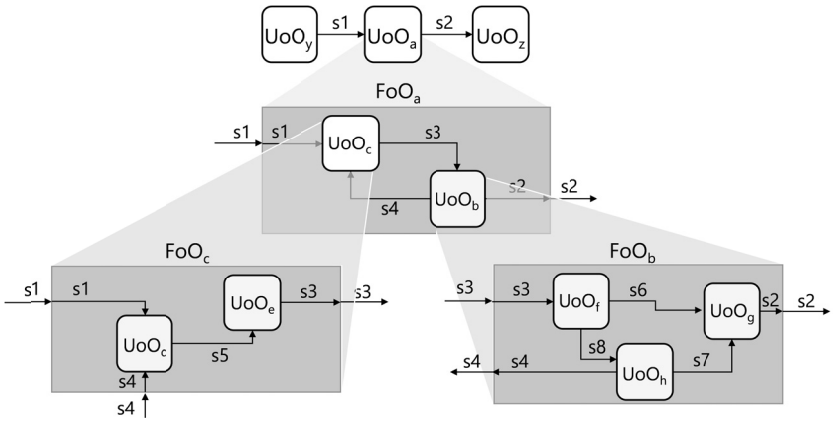


Figure 4-8 Containment and connectivity relations of FoOs

There are two types of M-E-I streams associated with each FoO, namely: (i) internal streams (between the UoOs), and (ii) external streams (that connect the UoOs of a FoO to other UoOs outside that FoO). External M-E-I streams are actually inputs and outputs for operations of a SMF. The external streams appearing in a lower level FoO may be internal streams between UoOs on a higher level, or external streams of some UoOs. For instance, the two FoOs (FoO_b and FoO_c) together form a higher level FoO (FoO_a). In a graphical representation, their connectivity can be indicated by connecting the associated external streams that have the same identifiers. Figure 4-9 shows the aggregation of the mentioned FoOs.

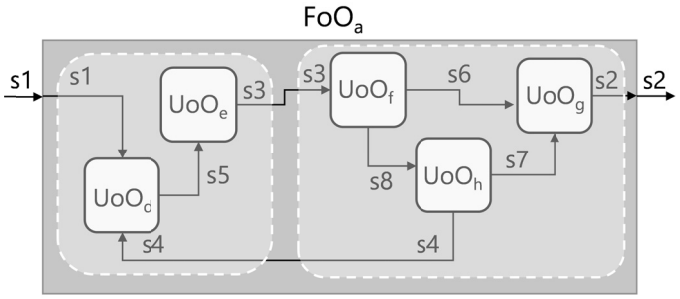


Figure 4-9 Aggregation of two FoOs

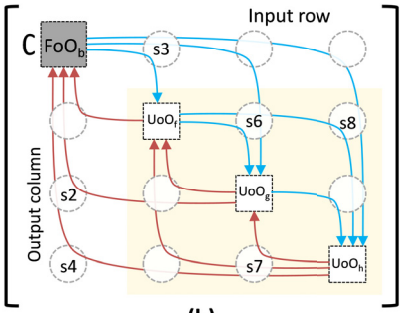
External streams are considered as operational interfaces of SMFs. For modeling operations, not only the concerned internal attributes of FoOs, but also the interfaces between these FoOs should be determined. Interfaces should be able to connect SMFs according to both containment and connectivity relations. Specification of the interfaces is an important issue for three reasons. Firstly, they should allow an easy replacement of SMFs without paying attention to their internal operations and parameters. That is, they should support flexibility and interchangeability. Secondly, the interfaces make it possible to define SMFs as independent modeling building-blocks. For example, if an off-the-shelf component is used, we are not interested in the details of its internal operation, but the interface should capture all pieces of information that are needed at building the component into a system as an SMF. Thirdly, the interfaces should capture that amount of information that is exactly needed for a correct architectural and operational embedding of an SMF. In fact, over-defined or incompletely defined interfaces make the matching of SMFs complicated, time consuming, or even impossible.

4.4.3 Matrix of streams

From a computational point of view, aggregation of the operations of SMFs is facilitated by an indexing convention, which is implemented in, and applied by the modeling system. As shown in Figure 4-9, the containment relationships can be rendered by comparing the indexes of UoOs and FoOs. It means that UoOs with the same index as a given FoO should be regarded as its de-aggregation on a lower level. The identifiers of the M-E-I streams also help establish patterns of operational connectivity. Repeating the same stream identifiers in two FoOs indicates that these FoOs are operationally connected. Based on the above formal specification, the operational interfaces can determine what kinds of information need to be captured for the computational modeling of SMFs. These are: (i) identifier of the FoO, (ii) identifiers of the UoOs making up that FoO, (iii) identifiers of the internal streams, (iv) the UoOs that are connected by the respective internal streams, (v) identifiers of the external (out-going and in-coming) streams, and (vi) the UoOs that receive/send external streams. In order to capture all these pieces of information in a compact form, the concept of *matrix of streams* (MoS) has been introduced. A MoS provides a uniform computational representation of all transformations made by UoOs, or by any individual combination of them, on M-E-I streams.

$$\begin{bmatrix} \complement FoO_b & s3 & - & - \\ - & UoO_f & s6 & s8 \\ s2 & - & UoO_g & - \\ s4 & - & s7 & UoO_h \end{bmatrix}$$

(a)



(b)

Figure 4-10 MoS of FoO_b: (a) graphical representation of operations and streams, (b) the matrix of operations and streams

Two groups of operations are associated with a FoO: (i) the operations of the UoOs making up the FoO, and (ii) the operations complementing the operations of the FoO. As such, the latter operations are externally related to processing M-E-I streams by the FoO). This second group of system-related complementary operations is indicated by the symbol ∁ (*complement*). The complementing operations may be operations of the embedding environment, a coupled system, a system user, etc.). All of these relationships and the other contents are represented in the MoS, as shown graphically in Figure 4-10. This figure

depicts the arrangement of FoO_b and the included UoOs, exemplified in Figure 4-8, together with the related internal and external M-E-I streams. In the matrix representation, the operations complementing FoO_b are placed into the first position of the descending main diagonal of the matrix. It symbolizes a kind of 'gateway' for both the input and output streams. In the rest of the positions in the main diagonal, the respective UoOs of FoO_b are placed. In this context, the complementing operation represents all operations of the embedding system, while the UoOs de-aggregated in the main diagonal represent the operations (FoOs) of the customizing SMFs.

The arrows in Figure 4-10.b. show the orientation, while the circles contain the identifiers of the different M-E-I streams that are processed by the respective UoOs. The places of the identifiers will be the same in the matrix representation as in the graphical representation of a FoO. The size of MoS matrix equals with the number of UoOs in the FoO added by 1. As shown in Figure 4-10.b, all incoming streams of the concerned FoO are included in the top first row of MoS, which is called *input row*. All outgoing streams of the FoO are included in the left first column of MoS, which is called *output column.* That is, the first row and column of MoS always represent external streams, while the rest of the matrix describes the internal streams of a FoO. MoS can be scaled up to capture the aggregation of arbitrary number of FoOs.

To demonstrate the applicability of MoS formalism to a real-life case, we use the example of the Pablo rehabilitation device again. Through this example, we can clarify how application oriented meaning can be assigned to the formalism and symbols used in Figure 4-11. We can also demonstrate how information about operations can be extracted from practical processes. Mentioned was earlier that a typical operation mode of the device is side lift. The device captures the range of the movements when side lift is exercised. The two components of the device that are involved in this flow of operation, namely, the MEMS detector and the ASIC converter, were discussed from an architectural perspective before. They transform acceleration into capacitance change (UoO_{CM}), and convert the value of capacitance change into value of displacement (UoO_{CA}), respectively. Since they have specific operational domain and distinct operations, we regard them as two SMFs.

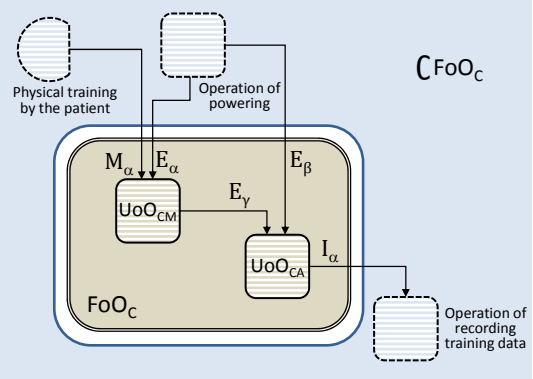


Figure 4-11 Incoming, internal and outgoing M-E-I streams in the FoO of the accelerometer

$$\begin{bmatrix} \complement FoO_C & M_\alpha, E_\alpha & E_\beta \\ - & UoO_{CM} & E_\gamma \\ I_\alpha & - & UoO_{CA} \end{bmatrix}$$

(a)

Descriptors:

M_α : Moving of UoO_{CM} (± 5 g)
E_α : Electrical energy (3.6V, 500 μA)
E_β : Electrical energy (3.6V, 500 μA)
E_γ : Electrical energy (Max. $\pm$860mV)
I_δ : Info (*a, v, d*)

(b)

Figure 4-12 Specification of the FoO of the accelerometer (UoO_C): (a) the contents of MoS, and (b) descriptors of the streams

Let us denote the flow of operation by FoO_C and all complementary operations of FoO_C by $\complement FoO_C$. With these, we can construct the MoS of the two SMFs of the rehabilitation device, as shown in Figure 4-12.a. The descriptors of the M-E-I streams are shown in Figure 4-12.b. The units of

operations of the two discussed SMFs of the device are included in the main diagonal of the instantiated MoS. The MoS describes that UoO_{CM} transforms an incoming material stream, M_{α} (acceleration produced by the patient during physical training) and an energy stream, E_{α} (input powering of the MEMS accelerometer SMF), without producing any outgoing stream. It also shows that UoO_{CA} transforms two incoming energy streams, E_{β} (input powering of the ASIC converter SMF) and E_{γ} (the voltage resulted from capacitance change), respectively, into an outgoing information stream, I_{α}, (pieces of information about acceleration, velocity, and displacement).

Having discussed the computational representation of UoOs and the streams related to them (on different levels of granularity), we elaborate on some issues of specifying time dependencies of the operations of SMFs below. First, we discuss the concepts of timing and handling of operational conditions in the specification and computation of operations of the SMF represented by the ASIC converter. Then, we use the example of the MEMS detector to discuss the computational methods. Finally, we cast light on the contents of the knowledge frame describing its complex operations, and discuss the interlacing of AKF and OKF for computational modeling and simulation of SMFs on multiple levels of aggregation. This approach of information structuring and integral handling is suitable for conjoint handling of AKFs and OKFs of arbitrary number of SMFs.

4.4.4 Timing and conditional operations

Obviously, operations of SMFs should happen in a controlled manner. Therefore, logical, temporal and conditional constraints should be considered in the computation of physical and computing operations. Toward this end, first of all, we need to operationalize the assumptions and to create opportunity for a purposeful logical arrangement, time scheduling and constraint management of FoOs and UoOs. In the literature, *time stamping* and *temporal logics* are widely used to capture temporal aspects of existence and transformative processes. While time stamping is typically used to assign date and time data to state changes (transformations), temporal logics is used to support procedural scheduling and synchronization. For temporal management of operations, both event and chronology oriented approaches have been proposed [30].

Consideration of time in the formal representation of SMFs has a dual perspective: (i) incorporation of time variables in operational processes, and (ii) time-oriented programming and execution of computation and control of operational processes. Systems having a large number of physical (analogue) and cyber (computing) components pose a challenge for real time processing and control. In the case of large complexities, there can be a dissonance between the time required for digital computation and the time elapsed by physical operations. The specification of SMFs is supposed to support the resolution of this dissonance and to create a robust platform for time-sensitive computation. Towards this end, every time-dependent operational process needs be represented as a *timed operation sequence* (TOS) by introducing and evaluation of time variables. The activation points in time and the durations of the units of operations of the concerned material, energy and information streams should be included in quantitative forms in a TOS. In the context of the Pablo rehabilitation device, there are several UoOs that should be described as timed operation sequences. Consider, for instance, the unit of operation UoO_{CA}, which converts value of the current change into value of acceleration, velocity and displacement. The procedure of this UoO includes five lower-level time-dependent units of operation:

- UoO_{CAA}: amplifying and measuring value of the input voltage (domain: in-connector H_{si}, detector A_{sa})
- UoO_{CAT}: measuring time variable (domain: oscillator H_{sr}, clock generator H_{sc})
- UoO_{CAC}: calculating and sending acceleration value (domain: processor A_{sp}, out connector H_{so})
- UoO_{CAV}: calculating and sending velocity value (domain: processor A_{sp}, out connector H_{so})
- UoO_{CAD}: calculating and sending displacement value (domain: processor A_{sp}, out connector H_{so})

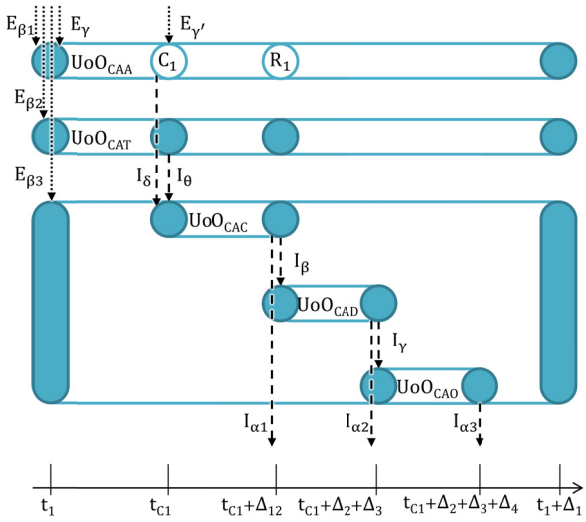


Figure 4-13 Time-dependent parameterization of operation flow

The above mentioned five lower-level units of operation need to be described as TOSs. This can be done by including the operation's start point in time, halt points in time, resume points in time, and end point in time. This time-dependent parameterization of operation flow (FoO_{CA}) is shown in Figure 4-13. The start point in time and the end point in time define the whole duration of this FoO. The change of a domain due to physical or computational operation is considered as an event [31]. Every event has its own time duration - this explains why halt points in time and resume points in time are made explicit in the time-oriented parameterization of the descriptors. For example, the stream E_γ starts at t_1, is halted at t_{C1}, is resumed at $t_{C1}+\Delta_2$, and stops at $t_1+\Delta_1$. These time dependences are specified for each stream in the table of descriptors, as shown in Figure 4-14. According to the applied convention, the first item in the descriptors block defines the start time and the last item defines the end time of an operation. As an example, the time specification for stream E_γ (in the fourth row of Figure 4-14) is as follows: t_1, .t_{C1}, $\Lambda t_{C1}+\Delta_2$, $t_1+\Delta_1$, where the first item is the start time, the item after '.' is the halt time, the item after 'Λ' is the resume time, and the last item is the end time. An operation descriptor with one timing item only means that it does not have duration.

It can be seen in Figure 4-13 that, as input of UoO_{CAA}, $E_{\beta1}$ stands for the supply of energy, while E_γ and $E_{\gamma'}$ are the inputs generated by the MEMS that detects acceleration. The input energy streams $E_{\beta1}$ and $E_{\beta2}$ are operated on only one time. However, the stream $E_{\beta3}$ is operated on three times by three different UoOs. Performed on the same architectural domain, these three UoOs form a group of operations with respect to $E_{\beta3}$ and captured by a timed operation sequence. The operation multiplicity of stream $E_{\beta3}$ is reflected by the content of the top row of the MoS shown in

Streams:

$E_{\beta1}$; $(t_1, t_1+\Delta_1)$:energy supply(3.6V 500 μA)
$E_{\beta2}$; $(t_1, t_1+\Delta_1)$:energy supply(3.6V 500 μA)
$E_{\beta3}$; $(t_1, t_1+\Delta_1)$:energy supply(3.6V 500 μA)
E_γ ; $(t_1, .\ t_{C1}\Lambda t_{C1}+\Delta_2, t_1+\Delta_1-t_{C1}+\Delta_2)$:regular *V-in*(1.2*V*)
$E_{\gamma'}$; $(t_{C1}, t_{C1}+\Delta_2)$:varied *V-in* (800~0 *mV*)
I_δ ; $(t_{C1}, t_{C1}+\Delta_2)$:Info (value of the *V-in*)
I_θ ; $(t_{C1}, t_{C1}+\Delta_2)$:Info (time)
$I_\beta = I_{\alpha1}$; $(t_{C1}+\Delta_2)$:Info (*acceleration*)
$I_\gamma = I_{\alpha2}$; $(t_{C1}+\Delta_2+\Delta_3)$:Info (*Velocity*)
$I_{\alpha3}$; $(t_{C1}+\Delta_2+\Delta_3+\Delta_4)$:Info (*displacement*)

Events:

t_1 : Turn on the ASIC
$t_1+\Delta_1$: Turn off the ASIC
t_{C1} : Apply acceleration

Conditions:

C_1 : (t_{C1}) : if($E_{\gamma'} \geq E_\gamma$. 3/2), Then($I_\theta \Lambda I_\delta$)
R_1 : $(t_{C1}+\Delta_2)$: repeat(C_1)
$t_{C1} \geq t_1$
$t_1+\Delta_1 \geq t_{C1}+\Delta_2+\Delta_3+\Delta_4$

Constraints:

–

Figure 4-14 Descriptors of the procedure of FoO_{CA}

Figure 4-15. In this sample case, the time constraints on the operation of the concerned SMF are the point of time of the input and the point of time of the output of the respective streams.

$$\begin{bmatrix} CFoO_{CA} & E_{\gamma}, E_{\gamma'}, E_{\beta 1} & E_{\beta 2} & E_{\beta 3} & E_{\beta 3} & E_{\beta 3} \\ - & UoO_{CAA} & - & I_{\theta} & - & - \\ - & - & UoO_{CAT} & I_{\delta} & - & - \\ I_{\alpha 1} & - & - & UoO_{CAC} & I_{\beta} & - \\ I_{\alpha 2} & - & - & - & UoO_{CAD} & I_{\gamma} \\ I_{\alpha 3} & - & - & - & - & UoO_{CAO} \end{bmatrix}$$

Figure 4-15 MoS of the procedure of FoO_{CA}

Many of the physical and computational operations of SMFs are typically executed under (i) space, (ii) time, (iii) attribute, (iv) logical, and (v) recurrence constraints. These constraints should also be included in the computational specification of the operations of SMFs. Usually, space, time and attribute constraints specify a threshold (minimum) value, or a ceiling (maximum) value, or a variation interval (min/max values) for the respected variables. The logical constraints express either propositional or production rules. Production rule-type conditional constraints are declared as: *if* (conditions) *then* (consequence) *else* (alternative). The consequences are the states of the M-E-I streams that are influenced by the conditional events. In order to express temporal sequences, time variables are assigned to these events in our approach.

Since the techniques of using value constraints are well-known in the literature [32] and rather frequently used in engineering computations [33], we do not address this issue here. We only touch upon the use and utility of logical (if-then) and recurrence constraints, which we refer to as *conditional constraints*. These are introduced at a given point in time, and either remain active in the rest of the operation, or are deactivated at a given point in time or when a given time duration is elapsed. For instance, the empty circles in the upper part of the diagram in Figure 4-13 introduce two conditional constraints, C_1 (which specifies a logical condition), and R_1 (which specifies a recurrence condition) in the context of UoO_{CA}, These conditions are included in the condition field of the table of descriptors, which specifies the working variables associated with the streams and the timed events too.

4.4.5 State transitions

State transitions are considered as alternative ways of interpreting operations. In fact, an operation changes the state of the respective architectural domain. In other words, state of a domain can be changed (transited) by an operation, and therefore should be considered as a concept that links the architectures realm to the operation realm. Furthermore, states are linked to events, which trigger operations and stream transformations. Therefore, there are implicit links between streams and domain states. Each AKF captures chunks of information related to a domain in a specific state. However, state transitions may change the pieces of information that is considered in the AKF. For instance, overloading a helical spring may change its architecture as well as morphology. In general, one domain may provide several operations, and one operation may affect (change states of) several domains. These kinds of associations become even more complicated if the computational processing of information units is considered. Consequently, the association of operation aspect and architecture aspect

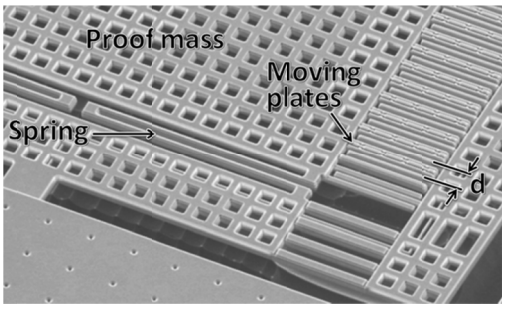


Figure 4-16 MEMS accelerometer [35]

was not considered as a one-to-one relation.

4.4.6 Methods as computational equivalents of transformations

Important resources for computation of operation of SMFs are *methods* that are represented by quantitative formulas based on which the transformation of the input streams of UoOs into output streams can be computed. An arrangement of methods in needed to transform the starting state of an SMF into the end state. The methods should be compliant with physical and computational laws and principles, as well as the constraints of operation. Methods are specified by mathematical expressions, systems variables, and various constraints. The physical and computational laws (e.g. gravity, electromagnetic field, friction, etc.) and principles are represented by equations or set of equations. System variables are groups formed by individual variables and values associated with an operation parameter. System variables quantify input parameters (e.g. electrical DC current of 12 V, 2 mA, for 25 min), intermittent parameters, and output parameters (e.g. vertical displacement of 173 mm). The methods have a one-to-one relation with the procedural elements of a UoO, but a particular method can be applied to multiple procedural elements. Below we give a practical example for deriving the methods from real life operation. The use and usefulness of methods in computation of operations can be demonstrated through the example of a MEMS accelerometer.

A MEMS accelerometer is an analogue electromechanical component that detects acceleration based on the movement of capacitor plates attached to a spring. As illustrated in Figure 4-16, it converts the detected acceleration into capacitance change, ΔC. The input acceleration produces a dynamic force in the proof mass ($F = m\,a$), which moves the plates against the spring. In turn, the spring deforms and causes a displacement x on the capacitive plates ($x = \frac{F}{k_s}$, where k_s is the spring constant (Young's modulus)). Knowing that $x = d\frac{\Delta C}{C_0}$ (where: d is the distance between the constant plates, C_0 is the initial capacitance, and ΔC is the change of capacitance proportional with displacement x), the change in capacitance is proportional to acceleration [34]. These transformative steps can be described by one equation:

$$\Delta C = \frac{mC_0}{k_s d} a$$

Many parameters of the MEMS accelerometer, such as Young modulus of elasticity of the spring material, the proof mass, the distances between capacitor plates, and the initial capacitance, are constant. Taking this into consideration, we can define a so-called accelerometer constant (k_a) that can be expressed as: $k_a = \frac{k_s d}{m C_0}$. Using this, the relationship between the input and output quantities of the operational domain representing the MEMS accelerometer is: $a = k_a \Delta C$. These equations are the basis of the computational method and algorithm.

The ASIC converter performs computational operations. It is a chipset that converts the value of capacitance change, ΔC, into a displacement value. As an operational domain, it includes an: (i) oscillator, (ii) a clock generator, (iii) computing units, (iv) detectors, and (v) filters. The UoOs of the converter is described before. Knowing that ($a = k_a \Delta C$), and ($d = t v_0 + \frac{1}{2} t^2 a$), an equation describing the operation of the converter (i.e. converting

the value of the input voltage to the distance) can be derived. The ASIC converter applies a Monte Carlo method when the acceleration is varying. Assuming that the value of acceleration in the second equation is constant, the displacement can be calculated by considering the time (t), accelerometer constant (k_a), value of current change (ΔC), and initial speed (v_0). Thus, the computational method can be described by the following equations:

$$a = k_a \Delta C$$

$$\overline{v} = t k_a \Delta C$$

$$d = t v_0 + \frac{1}{2} t^2 k_a \Delta C$$

It should be taken into consideration at applying this method that it is accurate only in the case of low acceleration and short distances.

4.4.7 Handling operational layers

An important element of the OKF formalism is the use of layers for processing coinciding physical and computational operations. Actually, layering allows separating (computational) concerns, but also avoiding possible conflicts of computing concurrent operations (Figure 4-17). Introduction of layers not only rationalizes the computational specification of operations, but also provides flexibility in processing concurrent elements of operation. For instance, layering makes the various states of software (language code, compiled executable, processor instruction) describable. Similarly, two or more different concurrent operations of components, whose computation needs different methods, can be captured (e.g. dynamic loading, stress calculation, and thermal dilatation calculation can be combined into a multi-physics operation). In addition, layering also support the integral treatment of primary, secondary and tertiary physical effects (e.g. rotational motion of a rubber wheel, the wear due to friction, and the material behavior on molecular level). Each layer captures those alternative operations that are related to the same sub-domain. Layers are not considered to be independent of each other. On the contrary, they are defined to share architecture and operation identifiers and variables. In this way, they can connect individual operations into a composite operation, which is needed in computation of multi-physics operations. For instance, an energy consumption variable used in a layer assigned to energy-change can be specified as dependent on the temperature variable used in another layer assigned to heat-change. The status of the individual variables and constraints described in the various layers can indicate if execution of an operation is possible, or not.

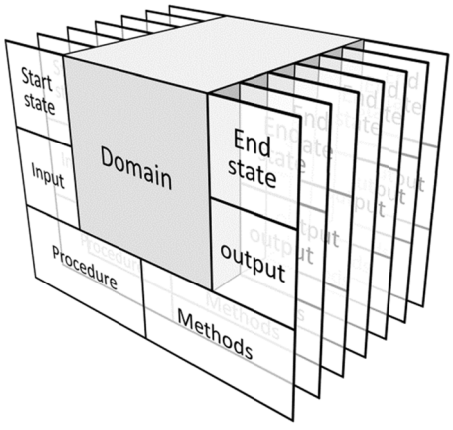


Figure 4-17 Multi-layer view of a UoO

A typical example of using layers in computing the simultaneous changes due to multiple concurrently present physical changes is consideration of effects of temperature variations in a wide temperature range on the MEMS accelerometer, which need to be minimized in the process of optimization [36]. Because of the temperature variations thermal deformation occurs, which has a negative effect on the output performance of the accelerome-

ter [37]. The main operation (i.e. detecting the amount of acceleration) and the accompanying operation (i.e. deformation under varying temperature) can be described using two layers. The two layers contain the data needed for the computation of both respective operations, and the results of the computations on the layers can be blended when they are available. The assignment of computational methods to the layers of the springs of the detector, which connect the proof mass to the frame, can be seen in the OKF shown in Figure 4-18. Calculating the changes for both layers in the case of a sequence of sufficiently small Δt, we can find the deviations of the output capacitance and resonance frequency caused by fluctuations of temperature. The influence of temperature on the Young's modulus of elasticity the spring material is approximated by the following method:

$$k(T) = k(T_o) + k(T_o)\, \zeta_E\, \Delta T,$$

where: $k(T)$ is the Young's modulus of elasticity spring material at a temperature T, $k(T_o)$ is the Young's modulus at the ambient temperature T_o, ζ_E is the temperature coefficient of Young's modulus, and ΔT is the relative temperature change. The deflection of the beam parts of the spring can be calculated by the following method:

$$d = 2\left(\frac{F}{4}\right)\left(\frac{l^3}{k(T)I}\right),$$

where: d is deflection, F is acceleration force, k(T) is the actual Young's modulus of the spring material, and I is moment of inertia of the spring beam. It should be noted that layers may inherent data not only from the general specifications that can be found in the domain states, domain transformations, operation constraints, and units of operation sections of the concerned OKF, but also from the descriptive fields of the corresponding AKF.

4.4.8 Formal specification of operation knowledge frames

Resembling the formalism of AKFs, operation knowledge frames have been defined and used as means of structuring the information needed to describe physical and computational operations of SMFs. An AKF captures domain-specific containment (architectural aggregation) and connectivity (architectural dependence) information for computing, whereas an associated OKF describes a flow of operation and multiple units of physical and computational operations on some aggregation level. OKFs have a complex and sophisticated architecture since they have been designed to capture all information necessary for computation. The information is used for: (i) creating a pre-embodiment artefact model, (ii) integration of the computational methods/algorithms, and (iii) simulation of the behavior of an SMF in various contexts. In addition to time sequenced operations, the information structure of OKFs can describe concurrent multi-physics operations and can be used as the basis of controlling parallel computations of operations. From a computational viewpoint, every OKF is associated (exchanges data) with the AKF of a particular SMF. An OKF consists of the following sections: (i) header, (ii) operation metadata, (iii) domain states, (iv) domain transformations, (v) operation constraints, (vi) units of operation, (vii) operation layers, and (viii) auxiliary data (Figure 4-18). The header section contains the identifiers of all entities that can be referenced by internal and external connectivity (operation) relations. It extends to the symbols (variables) used for the procedural elements (UoOs), M-E-I streams transformed by the domain, and the states and events of the domain. The meta-data field of the OKF contains information about the FoO and the corresponding overall domain.

OKF // FoO_{CM} //

	procedural elements (UoOs)	*domain streams*	*domain states and events*
header	UoO_{CMC}: conducting electricity into the capacitors' plates UoO_{CMA}: receiving acceleration and converting it to force UoO_{CMP}: pressing the spring by the force UoO_{CMD}: changing distance between plates of capacitors UoO_{CMO}: conducting electricity from capacitors to the MEMS output	E_α: energy supply M_α: applied acceleration M_β: applied force to H_{fs} M_γ: applied movement M_δ: displacement of capacitors' plates E_δ: potential electricity in capacitors E_γ: regular *V-out* $E_{\gamma\prime}$: varied *V-out*	e1: turn on the MEMS e2: receive acceleration e3: turn off the MEMS e4: send out regular *V-out* e5: send varied *V-out* S1: spring is not pressed S2: distance between H_{fm} and H_{fc} is maximum S3: *V-out* is 1.2V

operation metadata	*FoO descriptor: domain*	FoO_{CM}: H_f
start state of the domain	*list of SS variables:*	S1, S2, S3
end state of the domain	*list of ES variables:*	S1, S2, S3
list of streams	*list of streams and their starting time and duration:*	E_α (t_1,Δ_1); M_α (t_2,Δ_2); M_β (t_2,Δ_2); M_γ (t_2,Δ_2); M_δ (t_2,Δ_2); E_δ (t_1,Δ_1); E_γ $(t_1, .t_2, \wedge t_2+\Delta_2, t_1+\Delta_1)$; $E_{\gamma\prime}$ (t_2,Δ_2);
input events	*list of IE variables:*	e1: (E_α) e2: (M_α) = *a*
output events	*list of OE variables:*	e4: (E_γ) e5: ($E_{\gamma\prime}$) = ΔC
transformative procedure:	*list of physical and or computing units of operation:*	$\begin{bmatrix} FoO_{CM} & E_\alpha & - & M_\alpha & - & - \\ - & UoO_{CMC} & - & - & - & - \\ - & - & UoO_{CMA} & M_\beta & - & E_\delta \\ - & - & - & UcO_{CMP} & M_\gamma & - \\ - & - & - & - & UoO_{CMD} & M_\delta \\ E_\gamma, E_{\gamma\prime} & - & - & - & - & UoO_{CMO} \end{bmatrix}$
numerical processing of FoO	*order of numerical methods*	$\Delta C_{(E\gamma)} = \frac{m_{(Hfp)} C_{0(E\alpha)}}{k_{(Hfs)} d_{(Hfc)}} a_{(M\alpha)}$
algorithmic processing of FoO	*order of algorithmic methods*	-
time stamps	*start time, suspension times, duration, commencing times, end time variables:*	t_1 = t(e1) ≈ t(e4) $t_1+\Delta_1$ = t(e2) ≈ t(e5) t_2 = t(e3)
scheduling conditions	*list of conditions*	$t_2 \geq t_1$ $t_1 + \Delta_1 \geq t_2+\Delta_2$
operation constraints	*list of constraints*	–40°C < TA < 105°C ; 2.2 V < E_α < 3.6 V ; -5g < M_α < +5g ; E_γ= 1.2 V ; 0 mV < $E_{\gamma\prime}$ < 800 mV

operation domain entities	*list of operation domain entities*	$H_{ft} \oplus H_{fs} \oplus H_{fp} \oplus H_{fm} \oplus H_{fc}$	H_{fp}	$H_{fp} \oplus H_{fs}$		$H_{fm} \oplus H_{fc}$	$H_{fm} \oplus H_{fc} \oplus H_{ft}$
UoOs	*list of units of operations:*	UoO_{CMC}	UoO_{CMA}	UcO_{CMP}		UoO_{CMD}	UoO_{CMO}
numerical methods of UoOs	*list of numerical methods*	Conducting E_α	$F_{(M\beta)} = a_{(M\alpha)} m_{(Hfp)}$	$x_{(Hfs)} = \frac{F_{(M\beta)}}{k_{(Hfs)}}$		$x_{(Hfs)} = x_{(Hfm)}$	$\Delta C_{(E\gamma)} = \frac{x_{(Hfs)} C_{0(E\alpha)}}{d_{(Hfc)}} a_{(M\alpha)}$
algorithmic methods of UoOs	*list of algorithmic methods*	-	-	-		-	-
operation layers of UoOs	*identifier and order of layers*	-		$L_{CMC,1}$	$L_{CMC,2}$		
	software manifestation	-	-	-	-	-	-
	concurrent operations	-	-	$x = \frac{F}{k_s}$	$k(T) = k(To) + k(To)\,\zeta k\,\Delta T$ $d = 2\left(\frac{F}{4}\right)\left(\frac{l^3}{k(T)l}\right)$	-	-
	subordinate operations	-	-	-	-	-	-
auxiliary data		-	-	-	-	-	-

Figure 4-18 Instantiation of OKF in the case of FoO_{CM}

The description of the operational state of the domain includes the start state and the end state, the specification of the timing of the streams, and the input and output events. The transformation section includes the contents of the MoS, and the information about the numerical and algorithmic processing of the concerned flow of operation. The operation constraints section specifies the overall time stamps, the scheduling constraints for the FoO, and the operation constraints with regards to the changes in the M-E-I streams. The section of units of operation includes the identifiers of the sub-domains and the UoOs performed on them. This organization of the content of the OKFs makes it possible to refer to one or more UoOs of an OKF from a higher level OKF. Introducing the operation layer concept, and the related section, in the OKF makes it possible to handle multi-state, concurrent and hierarchical operations.

The indexes in the OKF shown in Figure 4-18, refer to domains and streams. Due to the fact that a UoO can be referred to from multiple different aggregation levels, attention is to be paid to parameterization. The issue is that parameters used on different levels are not independent from each other. Certain parameters and parameter relationships may be shared by the different levels, but some parameters may be relevant just on a particular level of aggregation. In order to be able to handle the variables on different levels, the shared parameter variables are specified as 2-tuples so as $V = \{v, n\}$, where: v is a valuated variable used in a SMF, and n is the aggregation level. However, in computing a mathematical equation or evaluating a logical relationship only those parameters are considered, which belong to the same aggregation level.

4.5 INTERLACING OF AKF AND OKF

In order to support computationally information processing, five kinds of relationship should be derived from the knowledge frames, namely (i) connections among AKFs on the same aggregation level, (ii) connections among AKFs on two different aggregation levels, (iii) connections among OKF on the same aggregation level, (iv) connections among OKFs on two different aggregation levels, (v) connection between AKFs and OKFs. The first kind of relations can be derived from the architectural connectivity relations formulated in AKFs. The second kind of relations can be derived from the architectural containment relations formulated in AKFs. The third and the fourth kinds of relations are captured by the matrix of streams represented in OKFs. The fifth kind of relations can be derived from metadata fields of both AKF and OKFs. The following paragraphs provide more specific information about these kinds of relations among knowledge frames.

4.5.1 Connections among AKFs and OKFs

We focused on the specification of the descriptive information structures in the preceding subchapters. Therefore, we separated the architectural and operational aspects and the architectural and the operation knowledge frames. However, they are integral elements of SMFs and should be processed in a conjoint manner when SMFs are used for artefactual modeling of a system, or when the operation of systems built up by SMFs is computationally simulated. We refer to the link between the architecture and operation knowledge frames as interlacing. From a data processing point of view, interlacing is enabled by a set of references (field pointers) among the OKFs and the AKFs associated with them. This creates a composite view whose advantage is that it involves both the architectural and

the operational aspects when systems are developed by using SMFs as modeling entities. On the one hand, the introduced AKF and OKF specification formalism rationalizes and simplifies the development and modification of SMFs, and, on the other hand, it supports a dynamic artefactual modeling and behavioral simulation of heterogeneous systems based on pre-programmed SMFs and structures thereof. A significant advantage of the described approach is its flexibility with regards to aggregation and de-aggregation.

There are data fields included in the AKFs and OKFs for the sake of representing lower level entities. In AKFs, this data field is named *domain entities*. In OKFs, this data field is called *units of operation* (Figure 4-19). The lines show the mutual relations between the metadata fields of two complementary knowledge frames, as well as those among the domain entities and the associated units of operations. The reference identifiers used consistently in both types of knowledge frame helps to simplify the search in, and link run-time associated frames. Considering that one architectural domain may be linked to several UoOs and vice versa, multiple reference identifiers are typically included in the metadata fields. The simultaneous reference to the domains entities and to the units of operation in the OKFs offers an opportunity for a direct checking of the correctness of the specification of the contents of the frames before computation.

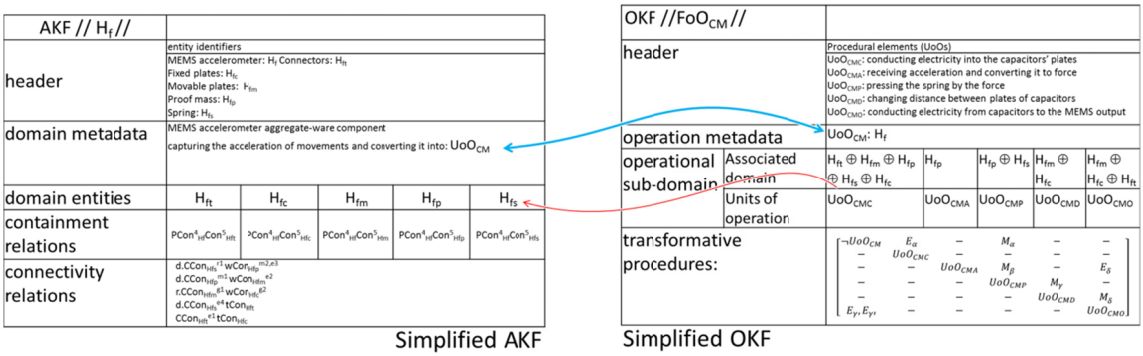


Figure 4-19 Interlacing the metadata of AKFs and OKFs

4.5.2 Multi-level handling of knowledge frames

Interlacing happens not only horizontally, that is between the fields of AKF and those of the corresponding fields of the OKF, but also vertically, that is among the specific fields of AKFs and OKFs representing different aggregation levels. Figure 4-20 shows the relationships among AKFs describing unique architectural components of an aggregate SMF. The information about containment can be retrieved from the metadata fields of the predefined SMF. Connectivity relations are specified among the subordinate components within an AKF, and make it possible to establish operational relations among them. The so-called *permanent internal* containment and connectivity relations are coded when the contents of a SMF are specified. Therefore, these relations can be changed only by redefining the whole SMF. Other parts of the relations, called *dynamic external* containment and connectivity relations, are established when SMFs are combined into structures in the design process. The data concerning these relations are included in the respective aggregate level AKF and OKF in runtime. The connectivity data can refer from the fields of the aggregate level frames to the fields of any lower level frames, as needed by the intended operation.

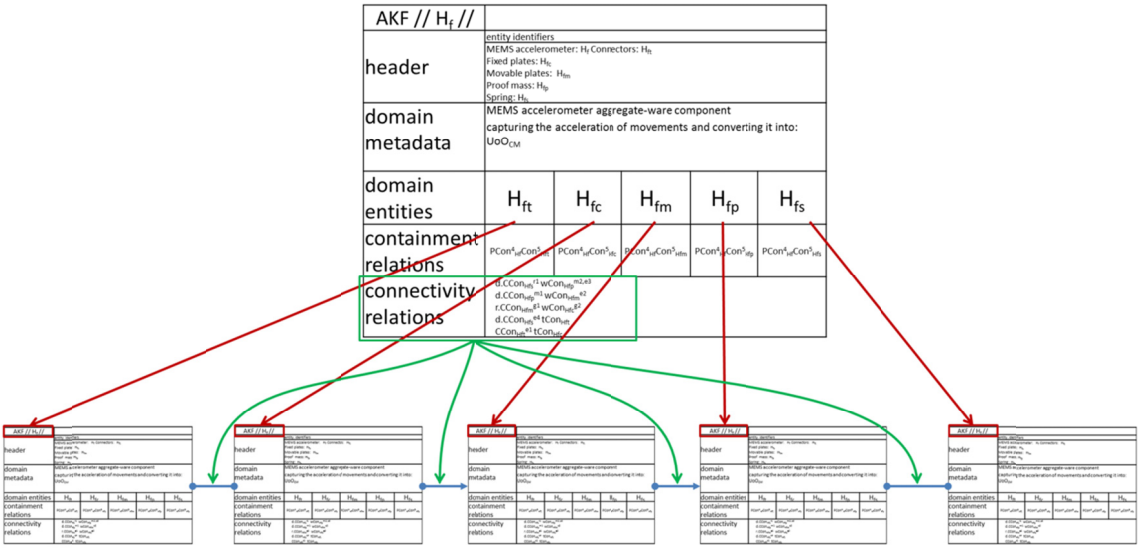


Figure 4-20 Containment and connectivity relations among AKFs of various levels

The *dynamic external* relations give the basis for describing time and constraint dependent operations, i.e. controlled transformations of M-E-I streams. An explanatory example is given in Figure 4-21. In this case, two aggregation levels of operations are considered. The top matrix (MoS_C) represents the streams of the accelerometer sensor (FoO_C), which is an aggregation of the MEMS acceleration detector (UoO_{CM}) and the ASIC converter (UoO_{CA}). The streams of the MEMS detector (MoS_{CM}) are shown on the left side of the figure and the streams related to the ASIC converter (MoS_{CA}) are demonstrated on the right side of this figure. The two aggregation levels are interrelated by the transformed streams.

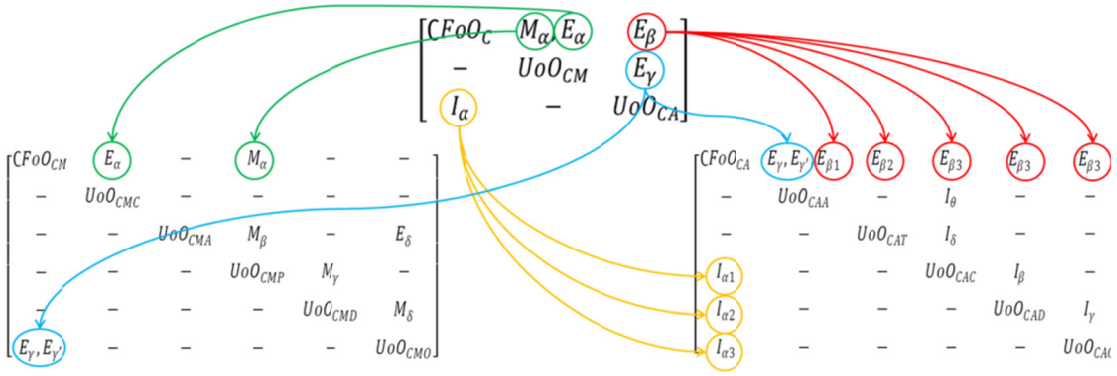


Figure 4-21 Streams establishing operational relations on two levels

The internal and the external operational relations are shown in the respective matrices of streams (Figure 4-21). The higher level UoO concerns one material stream, one information stream, and three energy streams. These streams variously appear in the lower level UoOs. For example, after being transferred into FoO_C, the material stream, M_α, is transformed by UoO_{CMP}. In MoS_C, the *external energy supply* stream, E_β, is received by UoO_{CA}. MoS_{CA}, provides power for all lower level UoOs. The appearance of the indices of a stream in a lower level indicates that the concerned stream is de-aggregated. As an example, by de-aggregating the information stream (I_α) we get to the three lower level streams. In the

MoS_C, there is an internal energy stream (E_γ) from UoO_{CM} to UoO_{CA}. As explained before, the MEMS detector detects the acceleration and generates a proportional *output voltage*, E_γ, which is converted into information about the *acceleration*, $I_{\alpha 1}$, *velocity*, $I_{\alpha 2}$, and *displacement*, $I_{\alpha 3}$, by the ASIC converter. In the lower aggregation level, E_γ is an external stream from FoO_{CM} to FoO_{CA} that connects the two corresponding MoOs. These transformations can be modeled and processed through the multi-level handling of operational relations, as described above.

4.6 CONCLUDING REMARKS

In this chapter, the constructs proposed by MOT have been used for architecture and operation modeling of SMFs as building blocks in pre-embodiment design of heterogeneous systems. SMFs naturally implement what is called HW, SW and CW co-design [38]. One of the main issues considered in this research cycle was information structuring and parameterization. Like in the 'classical' part feature theory, various issues of information structuring and parameterization have been addressed. Parameterization has been considered not only from geometric and structural aspects in the presented work, but also from a semantic aspect.

A strictly physical view is enforced, which applies the principle of aggregation in the generation of the architecture and the operations of a system, rather than that of abstraction. AKFs and OKFs have been introduced towards a consistent specification of the architectural and operational aspects of SMFs. Their information structures are the basis of the artefactual modeling of the domains of SMFs, and the simulation of the conceivable operations in the pre-embodiment phase of system development. The proposed SMF theory complements the mereo-operandi theory and facilitates the realization of a computer-supported system configuration and adaptation methodology [39].

During our research we obtained some new insights. For instance, due to aggregate complexity, the information structures needed to describe the architectural and operational characteristics of SMFs cannot be anything else but complicated. For this reason, it was inevitable to introduce some level of modularity with respect to entity specification and software programming. The modularization of the architecture and operation information structures led to a large number of information constructs, and to a multitude of relationships among them. At studying practical cases, it was recognized that composability of SMFs was strongly influenced by the number and nature of relationships. In our approach, various constraints (e.g. temporal, logical) are applied to the relationships, the combination of which has an important role in fulfilling specific composability conditions.

Composability is the degree to which SMFs can be assembled in various combinations to satisfy specific system and user requirements. A composition of SMFs should meet at least the major composability conditions in order to form an architecturally and operationally correct and feasible system. To increase composability of SMFs, sufficient functional and interfacing information is to be included in the specification of the related information constructs. The currently proposed solution is based on formalized input assumptions and output guaranties.

The next step of the research is considered as the computational level detailing of the SMFs-based methodology in order to feasibility test of the theoretical conceptualization. Furthermore, it was also our goal to make the specification, operationalization, and exploitation of a large number of SMF possible. Even though we have recognized the advantages of a parallel specification or augmentation of the proposed computational approach with a SMFs-based, user-orientated pre-embodiment design methodology, we could not accomplish it yet because of the necessary experimental work. This has been done in the next phase of our research, together with the practical investigation of utility and usability issues.

As system modeling entities, SMFs may represent both physical and computing transformations of the domains. A system model can be formed by aggregating the domains into a feasible structure and combining their UoOs into FoOs. Parameterization of SMFs comprising HW, SW and CW constituents represent a challenge because of the associated heterogeneity and complexity issues [40]. Nevertheless, a straightforward and comprehensive constraints management and satisfaction is needed. To the best of our knowledge, no underlying mathematical theory has been developed for parametrized handling of system manifestation features. Being aware of the difficulties, we could strive after only a partial theory supporting parameterized computational processing of SMFs. We believe that the main advantages of the SMF-based approach are as follows:

1. connecting previously separated areas of system design such as designing analogue hardware, digital hardware, system and application software, knowledge structures, concept ontologies, information/data models, and multimedia contents.
2. providing a uniform system development strategy and methodology for the pre-embodiment design phase of system development, by only incrementally deviating from the currently applied technologies, methodologies, and best practices.
3. making possible for systems designers to focus on individual elements without losing sight of the overall architecture and operation framework, and without drowning designers in cognitive burdens.
4. allowing dynamic modeling and easy modification of the system concept by various aggregates of domains and flow of operations on a relatively high level.
5. making system modeling transparent for each stakeholder representing different professional fields, balancing between different views, and reaching faster to agreement this way.

4.7 REFERENCES

[1] Sztipanovits, J., (2012), "Cyber Physical Systems—Convergence of Physical and Information Sciences", it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik, Vol. 54 (6), pp. 257-265.

[2] Liang, H., Nannan, X., Zhejun, K., and Kuo, Z., (2012), "Review of Cyber-Physical System Architecture", Proceedings of the 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), pp. 25-30.

[3] Bar Yam, Y., (2004), "Multiscale variety in complex systems", Complexity, Vol. 9 (4), pp. 37-45.

[4] De Micheli, G., (1996), "Hardware/Software Co-Design: Application Domains and Design Technologies", in: Hardware/Software Co-Design, De Micheli, G., Sami, M. (Eds.), Vol. 310, Springer Netherlands, pp. 1-28.

[5] Abdul-Ghafour, S., Ghodous, P., Shariat, B., Perna, E., and Khosrowshahi, F., (2014), "Semantic interoperability of knowledge in feature-based CAD models", Computer-Aided Design, Vol. 56, pp. 45-57.

[6] Tiihonen, J., Lehtonen, T., Soininen, T., Puikkinen, A., Sulonen, R., and Riitahuhta, A., (1998), "Modeling configurable product families", Modularity in use-experiences from five companies.

[7] Hotz, L., Felfernig, A., Günter, A., and Tiihonen, J., (2014), "A Short History of Configuration Technologies", in: Knowledge-based Configuration - From Research to Business Cases, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, pp. 9-19.

[8] Gerritsen, B.H., and Horváth, I., (2015), "Advancements in advanced modelling of complex products and systems", Engineering Computations: International Journal for Computer-Aided Engineering and Software, Vol. 32 (1).

[9] Baldwin, C., and Clark, K., (2006), "Modularity in the Design of Complex Engineering Systems", in: Complex Engineered Systems, Braha, D., Minai, A.A., Bar-Yam, Y. (Eds.), Springer Berlin Heidelberg, pp. 175-205.

[10] Oreizy, P., Gorlick, M., Taylor, R., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., and Wolf, A., L. , (1999), "An Architecture-Based Approach to Self-Adaptive Software", IEEE Intelligent systems, Vol. 14, 1999, pp. 54-62.

[11] Tiihonen, J., Soininen, T., and Sulonen, R., (1996), "State of the practice in product configuration–a survey of 10 cases in the finnish industry", Knowledge intensive CAD, Vol. 1, pp. 95-114.

[12] Horváth, I., and Pourtalebi, S., (2015), "Fundamentals of a mereo-operandi theory to support transdisciplinary modelling and co-design of cyber-physical systems", Proceedings of the ASME 2015 International Design Engineering Technical Conferences, Boston, Massachusetts, USA.

[13] Da Silveira, G., Borenstein, D., and Fogliatto, F.S., (2001), "Mass customization: Literature review and research directions", International journal of production economics, Vol. 72 (1), pp. 1-13.

[14] Pourtalebi, S., Horváth, I., and Opiyo, E.Z., (2014), "First steps towards a mereo-operandi theory for a system feature-based architecting of cyber-physical systems", Proceedings of the INFORMATIK 2014, Big Data – Komplexität meistern, E. Plödereder, L.G., E. Schneider, D. Ull (Hrsg.) (Ed.), GI, Stuttgart, Germany.

[15] McGrenere, J., and Ho, W., (2000), "Affordances: Clarifying and evolving a concept", Proceedings of the Graphics Interface, Montreal, Canada.

[16] Shah, J.J., (1991), "Assessment of features technology", Computer-Aided Design, Vol. 23 (5), pp. 331-343.

[17] Xie, Y., and Ma, Y., (2015), "Design of a multi-disciplinary and feature-based collaborative environment for chemical process projects", Expert Systems with Applications, Vol. 42 (8), pp. 4149-4166.

[18] Bidarra, R., and Bronsvoort, W.F., (2000), "Semantic feature modelling", Computer-Aided Design, Vol. 32 (3), pp. 201-225.

[19] Syaimak, A., and Axinte, D., (2009), "An approach of using primitive feature analysis in manufacturability analysis systems for micro-milling/drilling", International Journal of Computer Integrated Manufacturing, Vol. 22 (8), pp. 727-744.

[20] Altidor, J., Wileden, J., Wang, Y., Hanayneh, L., and Wang, Y., (2009), "Analyzing and implementing a feature mapping approach to CAD system interoperability", Proceedings of the ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. 695-707.

[21] Horváth, I., Pulles, J., Bremer, A., and Vergeest, J., (1998), "Towards an ontology-based definition of design features", Proceedings of the Proceeding of the Workshop on

mathematical foundations for features in computer aided design, engineering, and manufacturing, SIAM, pp. 1-12.

[22] Kim, K.-Y., Manley, D.G., and Yang, H., (2006), "Ontology-based assembly design and information sharing for collaborative product development", Computer-Aided Design, Vol. 38 (12), pp. 1233-1250.

[23] Tessier, S., and Wang, Y., (2013), "Ontology-based feature mapping and verification between CAD systems", Advanced Engineering Informatics, Vol. 27 (1), pp. 76-92.

[24] Kim, O., Jayaram, U., Jayaram, S., and Zhu, L., (2009), "An ontology mapping application using a shared ontology approach and a bridge ontology", Proceedings of the ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. 431-441.

[25] Brunetti, G., and Grimm, S., (2005), "Feature ontologies for the explicit representation of shape semantics", International journal of computer applications in technology, Vol. 23 (2-4), pp. 192-202.

[26] Chen, G., Ma, Y.S., Thimm, G., and Tang, S.H., (2006), "Associations in a Unified Feature Modeling Scheme", Journal of computing and information science in engineering, Vol. 6 (2), pp. 114-126.

[27] Chen, G., Ma, Y.-S., Thimm, G., and Tang, S.-H., (2004), "Unified feature modeling scheme for the integration of CAD and CAx", Computer-Aided Design and Applications, Vol. 1 (1-4), pp. 595-601.

[28] Demoly, F., Yan, X.-T., Eynard, B., Rivest, L., and Gomes, S., (2011), "An assembly oriented design framework for product structure engineering and assembly sequence planning", Robotics and Computer-Integrated Manufacturing, Vol. 27 (1), pp. 33-46.

[29] Ostrosi, E., and Ferney, M., (2007), "Fuzzy Product Configuration in Advanced CAD Systems", in: Digital Enterprise Technology, Cunha, P., Maropoulos, P. (Eds.), Springer US, pp. 225-232.

[30] Eidson, J.C., Lee, E., Matic, S., Seshia, S., and Zou, J., (2012), "Distributed real-time software for cyber–physical systems", Proceedings of the IEEE, Vol. 100 (1), pp. 45-59.

[31] Tan, Y., Vuran, M.C., and Goddard, S., (2009), "Spatio-temporal event model for cyber-physical systems", Proceedings of the Distributed Computing Systems Workshops, 2009. ICDCS Workshops' 09. 29th IEEE International Conference on, IEEE, pp. 44-50.

[32] Goldratt, E.M., (1990), Theory of constraints, North River Croton-on-Hudson, NY.

[33] Benhamou, F., Jussien, N., and O'Sullivan, B.A., (2013), Trends in constraint programming, John Wiley & Sons.

[34] Lyshevski, S.E., (2002), MEMS and NEMS: systems, devices, and structures, CRC Press.

[35] Fedder, G.K., Howe, R.T., Liu, T.-J.K., and Quevy, E.P., (2008), "Technologies for cofabricating MEMS and electronics", Proceedings of the IEEE, Vol. 96 (2), pp. 306-322.

[36] Liu, G., Yang, F., Bao, X., and Jiang, T., (2015), "Robust Optimization of a MEMS Accelerometer Considering Temperature Variations", Sensors, Vol. 15 (3), pp. 6342-6359.

[37] Wang, K., Li, Y., and Rizos, C., (2009), "The effect of the temperature-correlated error of inertial MEMS sensors on the integration of GPS/INS", Proceedings of the Proceedings of the Symposium of the International Global Navigation Satellite Systems Society, IGNSS, Australia.

[38] Wolf, W.H., (1994), "Hardware-software co-design of embedded systems [and prolog]", Proceedings of the IEEE, Vol. 82 (7), pp. 967-989.

[39] Elgh, F., (2014), "Automated engineer-to-order systems–a task-oriented approach to enable traceability of design rationale", International Journal of Agile Systems and Management 21, Vol. 7 (3-4), pp. 324-347.

[40] Merali, Y., and McKelvey, B., (2006), "Using Complexity Science to effect a paradigm shift in Information Systems for the 21st century", Journal of Information Technology, Vol. 21 (4), pp. 211-215.

Chapter 5

COMPUTATIONAL CONSTRUCTS

for implementation of SMFs-based modeling toolbox

The third research cycle concentrated on a purposeful systematic structuring and processing of descriptive information and procedural knowledge for specification of SMFs. The information structures resulted from the fourth research cycle were the input for this research cycle. In this Chapter it is discussed how they were converted into information schema constructs (ISCs) for defining warehouse databases of genotypes and phenotypes of system manifestation features, and to computational schema constructs (CSCs) for the computational procedures of the SMFs-based modeling toolbox. The contents of the ISCs and CSCs are also discussed in this Chapter, together with the feasibility test conducted in order to validate the theoretical concepts. The main challenge in the fourth research cycle was the development of all database schemata that are needed to record the modeling entities and the various relations among them according to the information structures conveyed by the AKF and OKF.

Another major objective of this research cycle was to (i) elaborate the overall architecture of the system manifestation features-based modeling toolbox (SMF-TB), (ii) specify the interactions among the various modules of this toolbox, as well as (iii) define the methodological procedure of SMFs creation. The architecture and the main functions of the modeling toolbox are explained in Sub-chapter 5.1. The process of SMF creation is defined as a three-step procedure in Sub-chapter 5.2. These steps are named as: (i) genotype creation (discussed in Sub-chapter 5.3), (ii) deriving phenotypes (discussed in Sub-chapter 5.4), and (iii) deriving instances (discussed in Sub-chapter 5.5). The mentioned sub-chapters present both database architecture and the overall workflow, as well as the specific steps of genotype, phenotype, and instance creation. The generated SMF instances are used for architecting and pre-embodiment modeling CPSs, but also for customization of CPCDs. The issues related to processing, composing, modification and simulation of CPSs are discussed in Sub-chapter 5.6. The chapter closes with concluding remarks in Sub-chapter 5.7.

5 COMPUTATIONAL CONSTRUCTS FOR IMPLEMENTATION OF SMFS-BASED MODELING TOOLBOX

5.1 INTRODUCTION: A bird's eye view on the proposed modeling tool

5.1.1 Overall concept of the system-level manifestation features-based modeling toolbox

This research cycle was concerned with (i) elaboration of the overall architecture of the proposed system manifestation features-based modeling toolbox (SMF-TB), (ii) specification of the interactions among the various modules of this tool, and (iii) definition of the methodological procedures of creation of and modeling with SMFs. The first step towards the implementation of the system-level manifestation features-based modeling toolbox was converting the conceptual knowledge frames (AKF & OKF) and the specified information structures into information engineering entities that could be implemented, stored, and processed, respectively, digitally. The completed process involved, on one hand, the specification of the information and computational constructs, and on the other hand, the construction of the data processing workflows for SMF creation and modeling.

We hypothesized that the targeted SMF-TB provides functions for warehousing and managing SMFs, and (ii) realization of the CPS modeling process. Consequently, we conceived two fundamental parts, namely: (i) a SMFs creation and warehousing *platform*, and (ii) an SMF instantiation and model composition *workbench* (Figure 5-1). The *platform* helps knowledge engineers to create SMFs and store them as individual modeling building blocks. The *workbench* has to general functionality (i) helps system designers to generate CPS models through instantiation and composition of SMFs, and (ii) supports performing visualization and simulation of the modeled CPSs (which is not in the focus of this research cycle). The four major computational issues related to SMF creation and CPS modeling were (i) information structuring and database architecture definition, (ii) software architecting and specification of component interactions, (iii), defining SMF creation process and processing workflow, and (iv) defining the SMF instantiation and model composition process. Each of these issues considers the SMF-TB from different aspects and viewpoints.

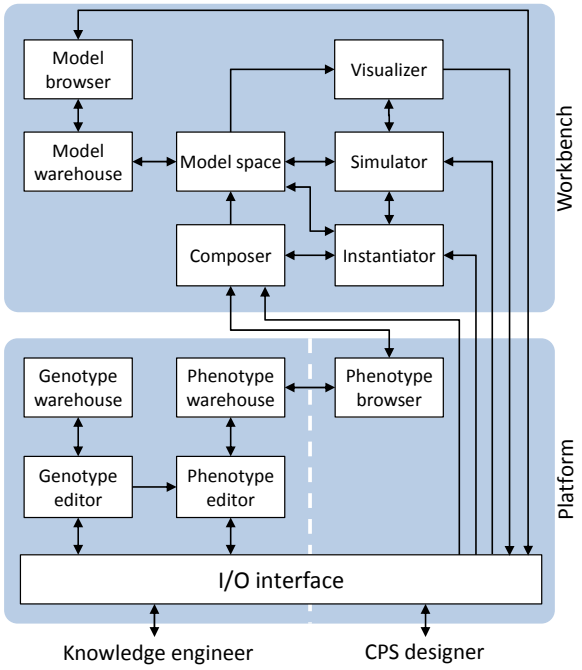


Figure 5-1 Overall architecture of SMF-TB

5.1.2 Architecture of the modeling toolbox

The MOT and SMF theories have been used as a guiding framework for information technological and computational implementation of the SMFs-based pre-embodiment design methodology and toolbox. The overall architecture of SMF-TB and the specific software tools included in the *Platform* part and in the *Workbench* part are shown in Figure 5-1. The *platform* includes those software modules (seen as tools) that are for specification, warehousing, and retrieval of SMF implements. It provides user interfaces for both knowledge engineers and CPS designers. For realizing the computational functions of warehousing of SMFs, relational databases have been considered. The pieces of information specified in AKF and OKF, and the relationship among them were the starting point at developing database architectures. Thus, at the back-end of the platform, there are two specific warehouse databases for managing SMFs - one for SMF genotypes (GTs) and the other for SMF phenotypes (PTs).

The *Workbench* includes the tools for composition, instantiation, visualization, and simulation of SMF-based models of CPSs. As integral element of the *Workbench*, the *Composer* tool checks for the constraints and the interface contracts of SMF phenotypes, as well as the architectural and operational relations between them. The *Instantiator* tool assigns values to the parameters of phenotypes. The system model is built in the *Model space* by coupling and instantiating selected phenotypes. The model data and relations are saved and stored in the database of the *Model warehouse*. There is a *Model browser* included that can be used to browse the saved models, to retrieve relevant ones, and to invoke the data belonging to the chosen model or SMFs thereof. The *Visualizer* module, like a CAD system, produces visual images of architecture views, operation views, and simulation views. The *Simulator* tool simulates the physical and computational operations of the system model according to predefined scenarios. Its functionality facilitates time-dependent analyses and simulations of the system model.

5.1.3 Concerned stakeholders and interaction with the modeling tool

Following the conventional but useful principle of part feature-based modeling, we assumed that system-level features should be programmed as parameterized entities that can be actualized in various application contexts by assigning values to their parameters. However, this task is more complicated in the case of system-level features due to their larger complexity and descriptive/prescriptive information content. It was also assumed that managing system-level features needs a 'library', more sophisticated than typically used in the case of traditional part features, to support generation, retrieval and operationalization of the feature entities. As mentioned above, SMFs are regarded as modeling building blocks in the SMF-TB, which should be devised according to the actual design task and context prior to composing them into a CPS model. In our vocabulary, this task is referred to as *'SMF creation'* and assigned to '*knowledge engineers'*. The execution and control of the CPS design and modeling procedures are in the hand of '*system designers'*. They are supposed to provide the required pieces of information for a computational processing of the SMFs and composition of the system model. Figure 5-2 is included here to indicate the human involvement as a use case diagram.

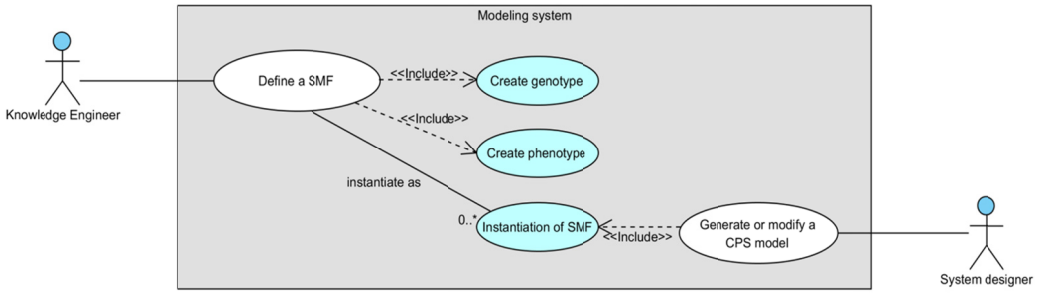


Figure 5-2 Use case diagram of user concerns in the SMF-based modeling system

Knowledge engineers, who create SMFs specifications on demand, are in charge of, e.g. converting electromechanical, electronics, software, cyberware, etc. components specified in digital catalogues and repositories into information entities for SMFs according to the knowledge frames. Their job extends to extracting pieces of information (e.g. related to operations, streams, events, conditions, constraints, temporal specifications, states, morphological properties, attributes, contracts, connections, and so forth) from relevant sources and process them for the SMF-TB. CPS architects, designers and analysts can retrieve these preprocessed SMFs from various warehouses and can use them to generate a new CPS models by instantiation and composition. SMFs can be created as redesigns of existing SMFs or from scratch, even for non-existing but feasible components. The wide variety of warehoused multidisciplinary SMF entities and their architectural and operational specification and parameterization are important contributions to making the work of CPS designers efficient. They can use various search methods and can start out of SMF samples of different specification levels depending on the design requirements. It has to be mentioned that these kinds of design methodological issues are not addressed in our research. The attention is given to the information engineering and computational processing of SMFs.

5.2 THE PROCESS OF CREATING SMFs

5.2.1 Principle of process decomposition

The term 'process' is defined as an overall course of actions that should be completed to achieve a particular objective. From a structural point of view, a process is an arrangement of composite 'sub-processes' until they keep their generality [1] [2]. The sub-processes can be further decomposed into 'procedures'. In general, a procedure is regarded as a logical and temporal way of doing something. According to our definition, procedures are purposefully arranged sets of cycles of information conversion actions. While a process is about a plan of achieving an objective, procedures define the steps and actions of how that objective can be achieved [3] [4]. It is entailed by our definition that a procedure can be decomposed into several cycles of interrelated 'activities' [5]. Within a cycle, the activities can be arranged according to some logical 'relations' and/or 'patterns', such as sequential, parallel, or circular. Activities can be decomposed to elementary actions, which form terminal (procedural) entities.

Figure 5-3 presents the mentioned process elements as well as their hierarchical relationships, and puts them into the context of our research work. Two sub-processes of SMFs-based modeling have been defined: (i) creation and warehousing SMFs as parameterized

modeling entities, and (ii) modeling CPSs as compositions of SMFs instances. Creation of SMFs needs to specify their structural, morphological, attributes and relational parameters. Methodologically, this multi-faceted specification could not be squeezed into one procedures due to computational and system compositional considerations. Therefore, two procedures have been defined, with multiple activities involved in each of them. The two procedures focus on two discrete in-process manifestations of SMFs, which have been called as genotypes and phenotypes. Together with the functional architecture of the modeling tool and the process of creating SMFs, the above technical terms will be explained in detail in the rest of the thesis.

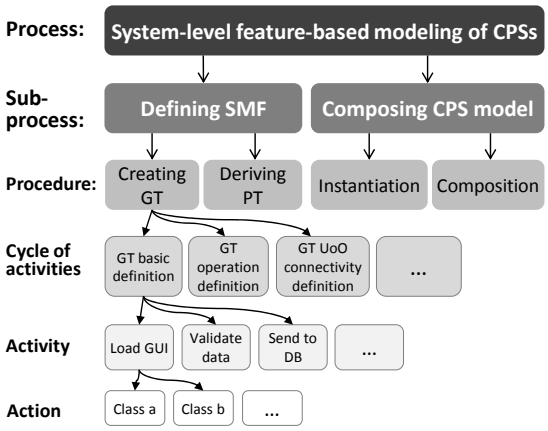


Figure 5-3 Decomposition of a process to intermittent and terminal elements

5.2.2 Stages of creating SMFs

The sub-process of modeling CPSs is decomposed to (i) the SMF instantiation and the SMF composition procedures. SMFs are specific knowledge constructs in which descriptive data sets, specification of relations, declarative statements, qualitative equations, and computational methods are all included. As mentioned in the fourth chapter and in [6], the proposed information structures are arranged in the AKF and OKF. While defining SMF genotypes and phenotypes may be done with the involvement of knowledge engineers, the instantiation and composition are tasks for system architecture and operation designers. The abovementioned procedures are implemented by using various components of the SMFs-based modeling toolbox such as the I/O interface, the GTs and PTs editor, the GTs and PTs warehouses, and the ontology manager [7].

From an application point of view, SMFs encapsulate all chunks of information that describe what the components of the modeled system are (architecture) and what they do (operations). The chunks of information concerning the architecture and the operations are tightly interrelated. This is important because in the physical world any architecture change may affect the operations, and vice versa. We had to consider some pragmatic issues concerning the SMF creation and specification procedure. One of them is, for example, the issue of entering a relatively large amount of data during the process [8]. It is a challenge for knowledge engineers, who are in charge of converting the structural, operational and attributes data of components into SMF as information entities. In addition, consistency of the symbols, notations, and relations should be maintained throughout the phases of creating and specifying SMFs and at archiving various SMFs implements in the warehouses. Having the dual objectives of (i) processing the related information in a consistent way and (ii) reducing the cognitive workload as much as possible, we have developed specific procedures for defining SMFs. As shown in Figure 5-4, the overall sub-process of SMF creation decomposes to the sub-process of ontology development and the sub-process of SMF creation. From a procedural perspective, the SMF creation process is preceded by the specification of feature concept ontology [9].

5.2.3 Consider but neglect – Our assumption about using smart ontologies for creating SMFs

The ontology development sub-process is regarded as an important preparatory part, but also as a complementary sub-process of SMF creation [10]. Its preparatory and complementary nature comes from the assumption that initial definition of SMFs is based on various concept and entity ontologies. In other words, it is supposed that the formal specification of concepts and entities related to the specification of the architecture and operations of an SMF are obtained from dedicated ontologies. It is also assumed that the SMF ontologies are comprehensive enough to capture all concepts and entities necessary for defining hardware, software, cyberware, and aggregated-ware types of SMFs. However, no issues and tasks concerning the development of these ontologies have been addressed in the completed research. The reason is that this would have gone beyond its initially defined scope and extent. On the other hand, we have made a number of assumptions on how ontology contents should be imported into the SMF creation process.

Nevertheless, for the sake of clarity, we must clarify terminological issues related to foreseen use of ontology contents. In our diagramming exercises, we used suffix '_concept' for naming the ontology tables. These relational tables capture all categorizations of a concept. The kinds defined in the concept tables might have ontological relations to other kinds from other concept tables. We need this level of 'semantic smartness' in order to facilitate the knowledge engineering tasks associated with SMF creation and prevent errors. The relations between ontological entities are not presented in this work. Furthermore, contrary to the fact that we did explicitly not deal with ontology development in the current work, we identified the main requirements for ontological classifications. The assumed ontological tables have been included in our schematic diagrams, and the relations between SMF types' tables and ontological tables have also been elaborated on. On the other hand, these tables were not investigated with sufficient depth as ontological inputs, and they might be subject to change in the future as a result of some follow up investigations and scrutinizing.

5.2.4 Overview of the SMFs creation process

As shown in Figure 5-4, creation and parameterization of SMFs happen in three interdependent stages, assuming that the 'concepts' needed for commencing with creation are obtained from background concept/entity ontologies. The definition sub-process is transitive in the sense that each step of SMF creation builds on the previous one, and inherits the information and specifications from the preceding ones. In the first step, a composition of ontological concepts is used to define alternative physically coherent and feasible structures for SMF genotypes. The genotype creates multiple structural relations among semantically compatible ontological concepts in order to form indeed physically coherent structures [10]. The second step of the sub-process introduces structural parameteriza-

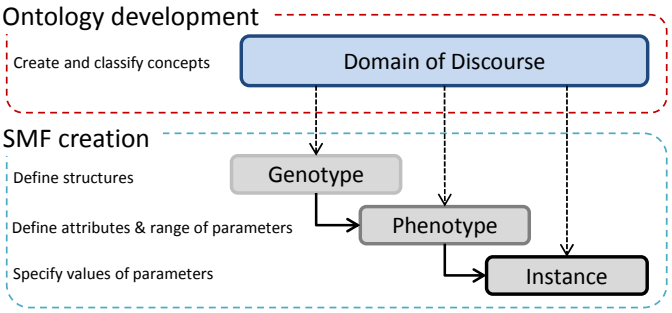


Figure 5-4 The overall sub-process of creating SMFs

tion over the structures captured within a genotype. The structural attributes become specified for each phenotype by inheriting from the chosen genotype. This allows selecting and assigning morphological and attributive parameters to them. In the third step, SMF instances are derived from a morphologically and physically parameterized phenotype by assigning values to its morphological and attributive parameters. In the process of modeling, deriving SMF instances starts with a kind of 'coupling' of selected phenotypes through their interfaces in order to make them architecturally and operationally interconnected, and follows with assigning shared values to their morphological and attributive parameters (variables).

Creation of SMF genotypes

Our starting point is that, when system designers take part in a brainstorming session to generate ideas and formally conceptualize a CPS, they think in terms of compositionality of the system as a whole, as well as in terms of composability of possible components (e.g. a processor, web browser, inductive charger, accelerometer, and so forth). They exchange ideas and share concepts with each other either by uttering components as semantic entities, or representing (sketching) them as physical entities. For example, system designers are normally aware of what an 'inductive charger' means, how does it typically look like, what it does, what the principles of operation are, what kinds of components it has, where it can be used, and much more, without even referring to a particular inductive charger.

According to the assumption of our theoretical framework, all of these chunks of information can be exchanged among system designers and can be built into a computational model of the system through a complex semantic modeling entity named SMF genotype (GT). As stated above, a GT defines and actualizes a unique set of internal structural relationships among concepts which are offered in ontological classes (as genes). Since genotypes define type-level components, they can refer to more general or more specific categories. For example '*electromotor*', '*stepper motor*', and '*hybrid synchronous stepper*' belong to same genotype and various templates. It is noteworthy that more specific genotypes lead to entering more pieces of information into the database, and vice versa. In addition, generic templates of a genotype can provide default values for defining more specific templates of that genotype. As an example, knowledge frames of electromotor (as a generic template) can be inherited and extended for developing knowledge frames of stepper motor (as a more specific template), and so forth. The mechanism of the inheritance is referred to as genotype template and is further elaborated on in Section 5.3.5.

Deriving SMF phenotypes

When system designers consider a specific component, they usually refer to a particular SMF phenotype (PT). The arrangement of the structural entities of PTs and the values of the variables representing their structural parameters are either inherited from the parent genotype, or interactively defined by knowledge engineers. Thus, a PT is based on a specific structure derived from the alternative structures incorporated in a parent GT. Multiple child phenotypes can be derived from a parent genotype. In addition, morphological and attributive parameters are to be specified for each derived phenotype. Concrete phenotypes can be generated by assigning specific values to the variables representing their parameters. Therefore, the procedure of defining phenotypes includes both data input and data conversion actions.

From a computational viewpoint, two types of parameters are used within the specification of phenotypes. There are variables, which can take range values and nominal values, and constants, which take default scalar or symbol values. Exemplifying a PT of a specific electromotor (e.g. RF370CA15 DC Motor) [11], variable parameters are the input current and voltage, and the output torque. These play a role in characterizing the energy stream and the material stream of the SMF phenotypes. A variable parameter is defined as a range of output value (such as max 21.1 g.cm), and a constant is a shaft diameter (such as 12 mm). The interrelationships of the chunks of information needed for creating genotypes and phenotypes are summarized in Figure 5-5. In this figure, the boxes with underlined texts contain ontological chunks of information, and the boxes whose names start with GT and PT contain chunks of information describing the genotype and phenotype, respectively.

Deriving SMF instances

SMF instances are derived from PTs in a given architectural and operational context of

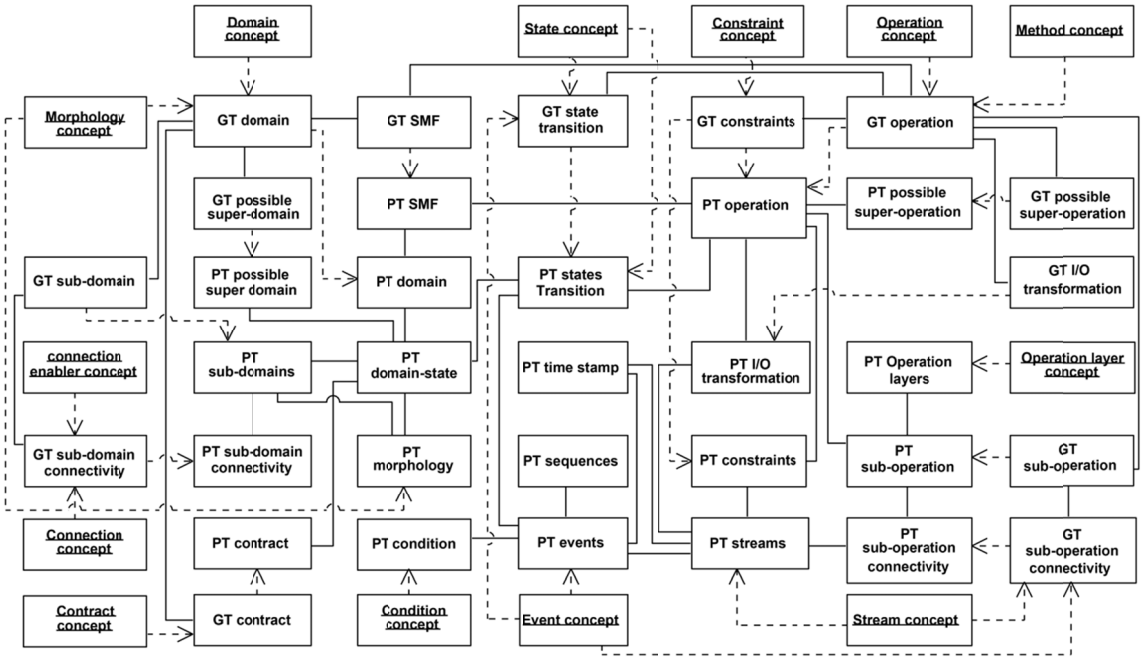


Figure 5-5 Interrelationships of the chunks of information required for creation of genotypes and phenotypes

system modeling. The context is determined by the architectural and operational characteristics of the other 'coupled' instantiated of phenotypes (SMF instances). The major difference between a phenotype of a SMF and an instance of a SMF are that a phenotype (i) is independent of (isolated from) other phenotypes, (ii) has the potential to operate, but is not yet included in the system model, (iii) neither its input and output streams, not its events are specified, and (iv) only possible ranges of values of its variable parameters are defined, while an instance (i) reflects a particular interaction of multiple SMF phenotypes, (ii) operates as element of the model in a defined context, (iii) both its input and output streams and the potential events are specified, and (iv) concrete values are assigned to its parameters and variables.

The parameterized representation of PTs allows selecting multiple value sets for the parameters. For example, if the PT of a stepper motor is composed in two places of a model, we have two instances of a same PT. Accordingly, two different sets of values are assigned to the parameters of the instances which are subjected to different workload, energy input, and environment temperature, as well as different behaviors (e.g. battery usage, output speed, and heat generation). Procedurally, when a model is built, instantiation of PTs starts with connecting their interfaces to those of other PTs. This is called composition. Then, specific values are assigned to the variables of the related architectural and operational parameters. The constraints of parameters of the coupled instances should also be considered. Though, deriving instances of phenotypes is not part of the SMF creation procedure, specification of the morphological and attributive parameters of the phenotype should consider the demands of compromised instantiation. Among others, the reason is that certain values may be default values as a result of structural specification of phenotypes.

5.2.5 Generic workflow of SMF creation

The software modules of SMF-TB involved in the process of SMF creation are: the I/O interface, GT editor, GT warehouse, PT editor, and PT warehouse. Figure 5-6 provides an overview of the generic workflow and the procedural interactions between these modules in the SMF definition sub-process. I/O interface serves as the front-end of SMF-TB. The contents of the information input forms are provided by GT editor and PT editor. These editors execute: (i) linking of the data fields of the forms to the entities and relations of the database, (ii) checking and converting the entered data to information entities and relations among them, (iii) calling the programming procedures of the database management system, and (iv) making queries to obtain the required data from the warehouses.

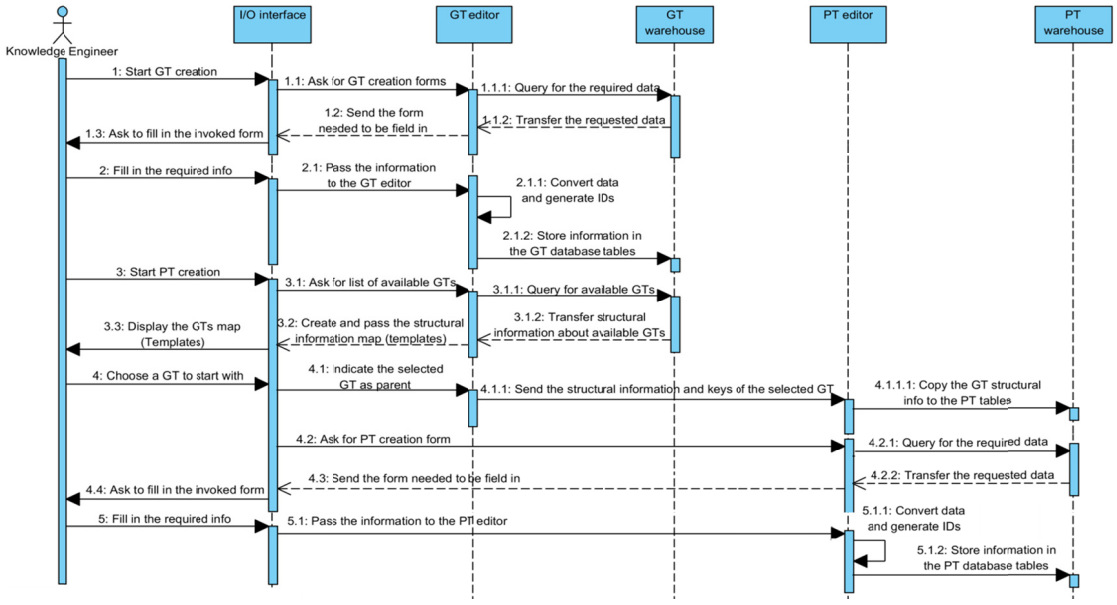


Figure 5-6 Generic workflow and procedural interactions

Because of the different information contents and procedural/semantic implications, the content information of GTs and PTs are stored in two different warehouses. Though their

organization on low level is different, the warehouses show similarity in terms of their internal organization. They are built up from three functionally interrelated components: (i) relational data tables (database), (ii) database management component, and (iii) meta-level knowledge base. The inheritance of data from a genotype to phenotypes is jointly handled by GT editor and PT editor.

The GT and PT warehouses are used in two different steps of SMF definition. The workflow of creating a PT starts with searching in the GT warehouse to find an appropriate parent GT among the stored ones. If this search is unsuccessful, then a GT can be interactively created based on the form offered by the GT editor. In both cases, the GT warehouse is responsible for sending the various chunks of information to the PT editor, which arranges them according to the PT input form. When specification of a PT is completed, the result is stored in the PT warehouse. In the course of instantiation and model composition, the required PTs are retrieved from the PT warehouse.

5.3 INFORMATION SCHEMA CONSTRUCTS FOR CREATING OF GENOTYPES

5.3.1 Introducing information schema constructs as implementation entities

In the philosophy of science, a *construct* is an object that is created in the mind to capture information about an existent or a conceived thing. Cognitively, a construct is a *specific mind model*, which includes and relates parameters that are necessary and sufficient to provide a simplified representation of a thing. According to the related literature, the most general interpretation of constructs is that they are abstraction-based structural formations that are used to describe, represent, and examine phenomenon of theoretical or procedural interest [12]. In research, constructs are typically used to describe a concept formed about an observable phenomenon (e.g. driving a car) or an unobservable phenomenon (e.g. attitude of driver).

In the context of digital computation, constructs encapsulate variables and their relationships to implement particular computational concepts [13]. They may capture input, outcome, structural, transformational, logical, knowledge, storage, interaction, interoperation, etc. concepts. These information constructs can be both static and dynamic, and determine the structure - (de)composition - of a computational or informational model [14]. Formative constructs have been used to enable programing of data-intensive computational procedures [15]. In the context of SMFs-based modeling of CPSs, constructs have been employed: (i) to support the implementation of complex database schemata, and (ii) to assist organization of complex information structures [6]. That is why we named them *information schema constructs* (ISCs). They are neither objects, nor concepts – but they can be a model of both. ISCs have been defined based on a semantic framework, which specifies the chunks of information needed to represent modeling concepts and their 'functional 'relationships in a formal manner.

We used ISCs in our research as cognitive enablers of designing the external schemata of the warehouse databases and the data input/transformation procedures. The basis of specifying the necessary ISCs for processing genotypes, phenotypes and instances of SMFs were the conceived design workflow and the theoretically defined information structures. The advantage of using constructs for complex data scheme specification and computa-

tion organization tasks is that they can be derived easily from the specified unit functionalities, and that they enable handling of information in a logically structured and consistent-in-time manner [16]. In addition, they: (i) are reusable in multiple contexts, (ii) provide upward compatibility, and (iii) make validation of large-scale semantic information structures more transparent. We can differentiate construct transformations performed: (i) within the context of a single constructs (i.e. construct versioning), (ii) on two or more semantically matching constructs (i.e. construct integration), and (iii) on two or more semantically mismatched constructs (i.e. construct adaptation).

As semantically arranged structure of identifiers, parameters, keys, values, descriptors and functional relations, ISCs are based on one or more physical/logical concepts (theoretical definitions). They also specify the data elements to be processed, their relationships, and the pertaining processing commands and procedures. A construct can be implemented in various programming languages. Eventually, an ISC enables the implementation of tailor-made elementary functions by defining executable code-level components or selecting off-the-self components that realize a function.

5.3.2 Chunks of information needed for creating genotypes of SMFs

The information structures that underpin the definition of genotypes and phenotypes are derived based on the chunks of information contained by the AKF and OKF [6]. All required chunks of information have been explored, correlated, and the relational tables of information have been designed accordingly. Since in genotype creation step knowledge engineers specify structure of relations between ontological concepts, pieces of information embedded in AKF and OKF have rationally scrutinized and refined subsequently. The ISCs of a genotype are supposed to use the following ontological tables: (i) domain concepts, (ii) morphology concepts, (iii) connection concepts, (iv) connection enabler concepts, (v) contract concepts, (vi) stream concepts, (vii) state concepts, (viii) event concepts, and (xi) operation concepts.

The information sets taken over from an AKF are as follows:

Metadata:	*< kind_of_possible_super_domain >; < kind_of_domain >; < description_of_domain >; < associated_operation >; < morphological_kind >; < representative_state >.*
Domain entities:	*< GT_entity_name >; < GT_entity_description >; < kind_of_entity >.*
Entity morphologies:	*< kind_of_entity_morphology >.*
Containment relations:	*< list_of_GT_entities >.*
Connectivity relations:	*< kind_of_connection >; < connected_entities >; < kind_of_connection_enabler >.*
Contracts:	*< kind_of_contract >.*

The information sets taken over from an OKF are as follows:

Metadata:	*< kind_of_possible_super_operation >; < kind_of_operation >; < description_of_GT_operation >; < related_GT_domain >.*
States of domain:	*< kind_of_start_state >; < kind_of_end_state >; < start_event >; < end_event >.*
Streams:	*< kind_of_external_stream >; < kind_of_internal_stream >.*
Events:	*< kind_of_even >.*
Transformative procedure:	*< associated_stream >; < stream_direction >.*
Methods of FoO:	*< nil >.*

Time stamps: *< nil >.*

Scheduling conditions: *< nil >.*

Operational constraints: *< nil >.*

UoOs: *< name_of_GT_UoO >; < description_of_GT_UoO >; < kind_of_UoO >.*

Domain entities of UoOs: *< UoO_associated_entity >.*

Methods of UoOs: *< nil >.*

Operational layers: *< nil >.*

5.3.3 Information schema constructs of the genotypes database

AKF and OKF accommodate the pieces of information without determining functional semantic relations among them. However, these relations have significance from computational constructs point of view. Therefore, they were subject of further examination. The first step towards recognizing these relations was to figure out the existence of functional dependency or semantical association among corresponding information entities. The second step was the investigation of the kind of relationships and the direction of dependency. Moreover, all of the explored relationships were further investigated considering the objective of creating SMFs according to the real world conceptual associations. From the viewpoint of organizing data, genotypes are structures formed by *entity-relationship tables* (ERTs). The structures of ERTs have been depicted by graphical ERT diagrams (such as the one shown in Figure 5-7). Each rounded rectangle represents an entity table in the database of the warehouse. Contents of the rows of the tables in these diagrams serve as names of columns of tables in the database that can accommodate records as their rows. There are relations between the entity tables, as well as between the information entities. A relation needs to be specified if dependency coupling or semantic association exists between two (chunks of) information entities.

There are primary keys (🔑) and foreign keys (🔑) included in the relational tables. The primary keys (PKs) are unique identifiers of each row, and they also identify the records in the respective tables. It is possible to connect two tables by repeating the PKs of one table in another one as foreign keys (FKs). Based on PKs, contents of the rows of tables can be related to the records of other tables. PKs can be repeated once or multiple times in another table. Repeating PKs (which become FKs in that table) implies that multiple relations have been established between the tables. The established relations do not indicate the direction of dependence per se – it needs semantic interpretation.

ISC for primary GT specifications

Each genotype could be specified by its unique name and a short description. The first table named as 'GT_SMF' is allocated to this purpose (Figure 5-7). Additional specifications are included in the other tables. Referring to [6], a SMF should be defined as an architectural domain performing particular operations. A SMF possesses a domain that may

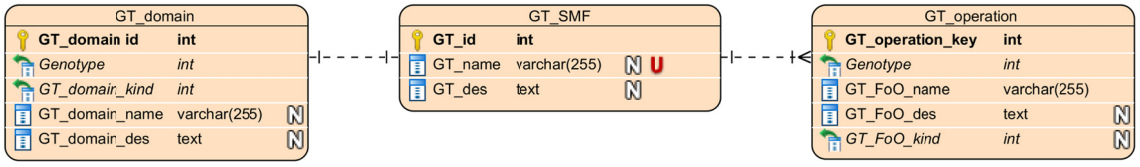


Figure 5-7 Construct for arranging the primary relational tables for GTs

perform one or many operation(s). Therefore, there is a one-to-one relation between GT_SMF and GT_domain, and a one-to-many relation between GT_SMF and GT_operation. A genotype is identified by its unique name and a short description. The PK of the "GT_SMF" table is a unique identifier "GT_id." In order to facilitate a proper categorization of genotypes and prevent misinterpretation, the "GT_name" parameter does not allow entering similar names for multiple genotypes (i.e. it accepts only unique varchar values). As FK, the "GT_domain_kind" is imported from the "domain concept" table. The "GT_FoO_kind" is imported from the "operation concept" table of the concept ontology. A SMF should have at least one domain and one operation. Figure 5-8 shows the tables filled with an example of the studied smartphone. The architectural domain of this smartphone is its body including all components (e.g. screen, battery, and housing). This compound domain performs many operations, e.g. making call, taking photo, providing navigation, and playing music.

GT_domain table

PK	GT	name	descriptions	kind
118	65	Smart-phone	Architectural composition of ...	Aggregated -ware

GT_SMF table

PK	name	descriptions
65	Smart-phone	A cellular phone that ...

GT_operation table

PK	GT	name	descriptions	kind
232	65	Making call	Stablishing ...	Remote communication
234	65	Taking photo	Capturing ...	Capturing cyberware
235	65	navigation	Positioning ...	Information delivering

Figure 5-8 Primary tables of genotype (i.e. GT_SMF, GT_domain, and GT_operation tables)

ISC for specification of operational containment

As mentioned before, an operation can be de-aggregated into several UoOs, and can be considered as a UoO for several higher level FoOs. These relations are captured by the ISC shown in Figure 5-9. Three levels of operations are defined as: (i) super-operation, (ii) flow of operation (FoO), and (iii) unit of operation (UoO). FoO specifies the overall operations of the genotype, while UoOs describe the specific operations of components included in the genotype. The term 'super-operation' is used to refer to a possible higher-level operation, of which a FoO of the genotype is a part. The information specifying these levels is included in the "GT_possible_super_operation," "GT_operation," and "GT_UoO" tables. They capture all chunks of information associated with the three operation levels.

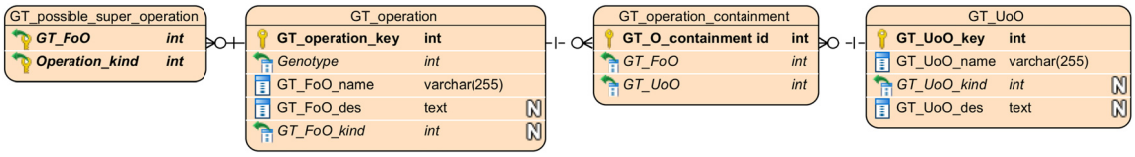


Figure 5-9 Construct for specification of operation containment by GTs

UoOs can be aggregated in different compositions to form FoOs. What it means from a database management point of view is that it should support linking many UoOs to one particular FoO, and vice versa. UoOs are represented in the "GT_UoO" table, while FoOs are included in the "GT_operation" table. In the case of a many-to-many relation between them, an association table, called "GT_operation_ containment," establishes mutual relationships. The FKs indicated by the "-kind" suffix are imported from the "operation concept" table. In the corresponding relational data tables (RDTs) of the FoO and UoO tables, the "name" and "descriptions" specify the respective columns. Referring back to the smartphone example, *processing* as a UoO is needed for several FoOs e.g. navigation,

making calls, and taking photos. Consequently, we need to stablish many-to-many relations between tables concerning FoO and UoO. It means, providing possibility of linking one FoO to many UoOs, and linking one UoO to many FoOs. Table 'GT_operation_containment' establishes a many-to-many relation between 'GT_operation' and 'GT_UoO'. The example shown in Figure 5-10 makes this relation more clear. It should be noted that the FKs (that represent kinds of operations) are integer (shown by 'int' sign). However, in Figure 5-10, they are textually presented for the purpose of clarification.

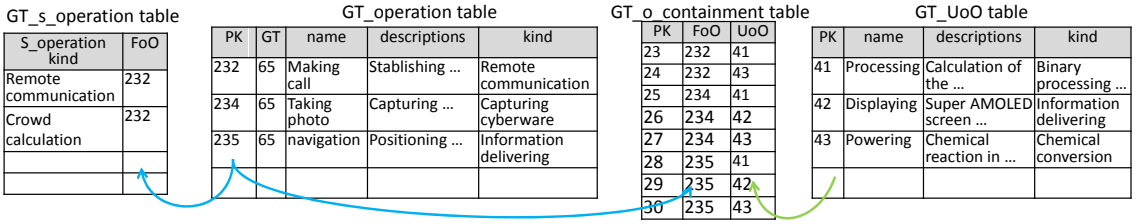


Figure 5-10 Operation containment tables of genotype (i.e. GT_s_operation, GT_operation, GT_o_containment, and GT_UoO tables)

ISC for specification of operation connectivity

Representation of operational connectivity is an important aspect of organizing the database schemata. The reason is that the "procedure" of an FoO is established by many procedural elements (UoOs) and the operational connections among them. The table "GT_UoO_connectivity" (shown in the middle of Figure 5-11) is the basis of the related schemata. This table accommodates the corresponding FKs. In the physical world, connections between UoOs are made through energy, information, and material streams. Therefore, the FK "GT_stream_kind" is used to refer to the stream connecting two UoOs. The FK "GT_event_kind" refers to the kind of event that triggers the connecting stream.

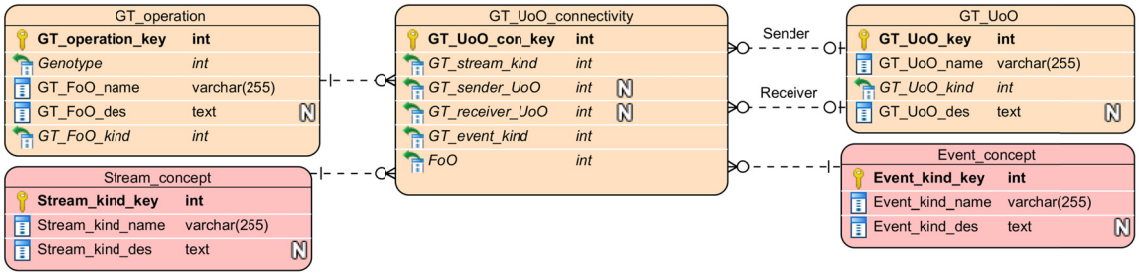


Figure 5-11 Construct for arranging relational tables for capturing operation connectivity by GTs

This scheme identifies the sender UoO, as well as the receiver UoO, in order to be able to capture both the source and the sink of the stream. Correspondingly, the FKs are imported from the "GT_UoO" table and included in the associated RDTs. These FKs are "nullable," because of the possibility of having either incoming streams or outgoing streams. The FK "FoO" refers to a higher level operation in which the captured connection is a part of the procedure. We note that the FKs "GT_event_kind" and "GT_stream_kind" are supposed to be imported from the ontological tables "event_concept" and "stream_concept," respectively. Figure 5-12 presents an example of the filled tables that capture operation connectivity.

GT_operation table

PK	GT	name	descriptions	kind
232	65	Making call	Stablishing ...	Remote communication
234	65	Taking photo	Capturing ...	Capturing cyberware
235	65	navigation	Positioning ...	Information delivering

GT_UoO_connectivity table

PK	sender UoO	receiver UoO	Stream kind	Event kind	FoO
31	43	41	energy2	EW2	234
32	43	42	energy3	ED5	234
33	41	42	information6	ZQ3	234
34	43	41	energy2	EJ6	232

GT_UoO table

PK	name	descriptions	kind
41	Processing	Calculation of the ...	Binary processing ...
42	Displaying	Super AMOLED screen ...	Information delivering
43	Powering	Chemical reaction in ...	Chemical conversion

Figure 5-12 Operation connectivity tables of genotype (i.e. GT_operation, GT_UoO_connectivity, and GT_UoO tables)

ISC for specification of I/O transformation

Each operation performed by an SMF is formally defined as a transformation. A physical transformation can be a state transition and an input-to-output transformation. Transformations operate on input and output streams. The construct that specifies the transformations is called I/O_transformation, and it creates relations with the genotypes' database tables. These relations are graphically shown in Figure 5-13. The "GT_IO_transformation" table establishes one-to-many relations with the "GT_input_stream" and "GT_output_stream" tables. These tables assign kinds of streams as inputs and outputs to the transformation. The "GT_stream_kind" represents the FKs imported from the "stream_concept" table. The one-to-many relations are needed here because, from a physical viewpoint, one single operation may transform (convert) multiple input streams into multiple output streams.

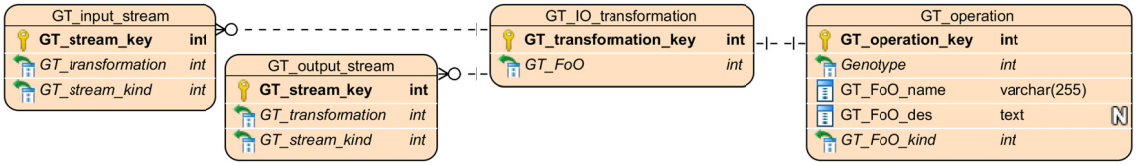


Figure 5-13 Construct for arranging relational tables for capturing I/O transformation by GTs

ISC for specification of states of domain

The states of operational domains are changed when physical or computational transformations are taking place. Physically, a start state of a domain is transformed to an end state as a result of completion of the FoOs associated with the domain. A genotype describes a state change by considering two events as the start and the end of the state transition. For the computational realization the "state_kind" and "event_kind" concepts are imported as FKs from the "state concept" and the "event concept" tables. The construct specifying the relations between the domain and the operations (through state transition) is shown in

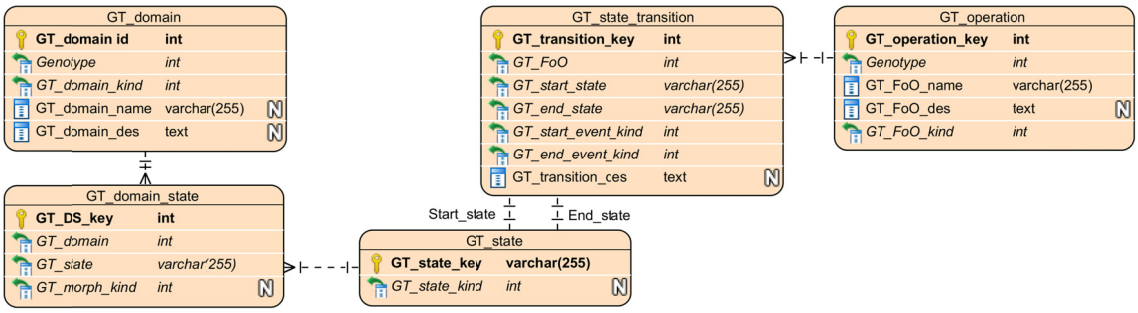


Figure 5-14 Construct for arranging relational tables for capturing state transitions by GTs

Figure 5-14. There is a many-to-many relation between the "GT_domain" and the "GT_state" tables, which is described by the "GT_domain_state" table. This allows a domain to be in multiple states, and a state to associate with characteristics of several domains. This way, a state is linked to a domain through the "GT_domain_state," which is a kind of virtual entity.

All tables associated with the (architectural) domain are connected to the "GT_domain_state" table. The advantage is that this solution makes possible to trace back the changes in the domain during simulation. The column named "morph_kind" in the "GT_domain_state" table is connected to the "morphology concept" table of the concept ontology, which assigns PKs from concepts included in morphological taxonomies (e.g., solid, gas, liquid) to "GT_morph_kind." As "GT_morph_kind" can be different in two states, it is also connected to the "GT_domain_state" table. Figure 5-15 presents an example of information records in the discussed relational tables in the case of the smartphone.

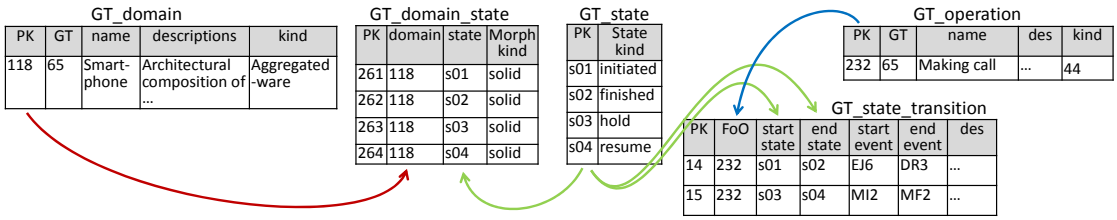


Figure 5-15 Tables regarding states of domains (i.e. GT_operation, GT_state_transition, GT_state, GT_domain, and GT domain-state)

ISC for specification of architectural containment

The specification of architecture also requires containment relations, but slightly different ones from the specification of operations. The two reasons of this are that subdomains (entities) may not belong to more than one domain in a particular state, and that the state of the domain influences the possibility of a containment relationship. This can be clarified by an example concerning the smartphone. In the case of initiating the *taking photo* operation of the smartphone, there is no image file (entity) in the memory in the start state yet. However, there is an image file in the end state and it is a part of the smartphone system. The dependence of the containment relations on the state (and consequently on the operation) is considered by linking the "GT_entity" and "GT_possible_super_domain" tables to the "GT_domain-state" table. This enables monitoring the addition and removal of domain entities and super-domains due to operations on the domain. The relations associated with the "GT_domain_state" are shown in Figure 5-16.

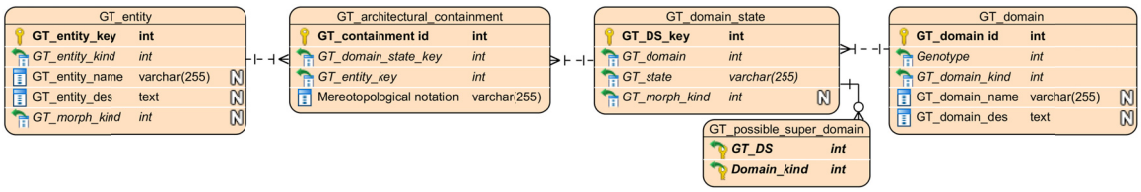


Figure 5-16 Construct for arranging relational tables for capturing architectural containment by GTs

The containment relation between domain entity and domain-state is defined as a many-to-many relation. Each domain-state may concern several domain entities, and each

domain entity may be included as part of many domain-states. For example, the battery is a part of the smart phone in all of its operation states. The *'GT_entity'* table accommodates almost the same information pieces as the '*GT_domain*' table. Figure 5-17 presents an example of the filled tables. In this example, we applied filter (simplification) to be able to show the important issue clearly.

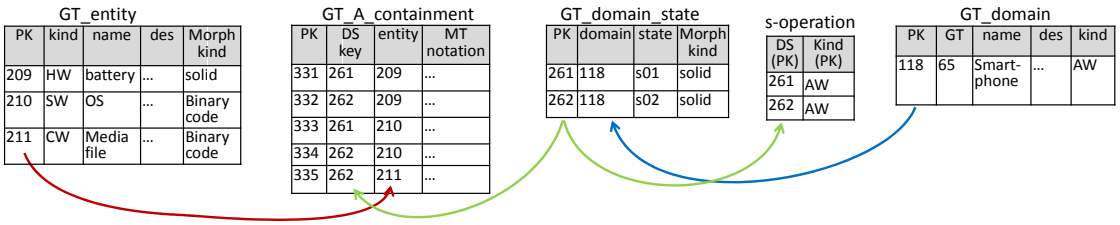


Figure 5-17 Tables regarding architectural containment (i.e. GT_domain, GT domain-state, GT_entity, and GT_A_containment tables)

ISC for specification of architecture connectivity

Architectural connectivity is interpreted on both entity level (connection between entities of a domain) and domain (possible connection with other domains) level. The possible connections are declared as input assumptions and output guarantees. The construct shown in Figure 5-18 presents four database tables, from which the "GT_entity" and "GT_domain_state" tables have been discussed above. The "GT_entity_connection" table accommodates several FKs that establish a self-many-to-many relation of the "GT_entity" table, which captures connections among domain entities, as well as a one-to-many relation between the "GT_domain_state" and the "GT_entity_connection" tables assuming a relevant state. Since connections among domain entities change in various states, the "GT_DS" key specifies in which state a captured entity connection is valid.

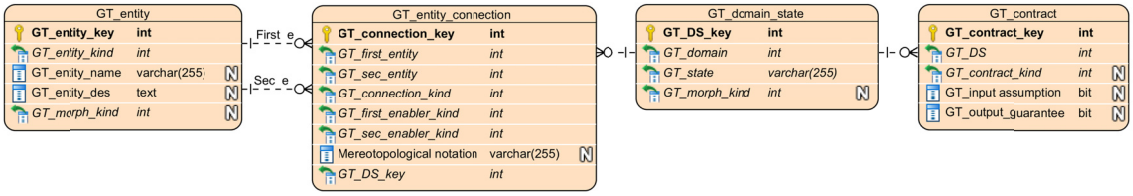


Figure 5-18 Construct for arranging relational tables for capturing architectural connectivity by GTs

There are two FKs (one for each) for connected entities, and two for connection enablers [6]. In addition, an FK is included there for specifying the kind of connection, by referring to the "connection concept" table of the concept ontology. The column storing mereotopological notation captures all chunks of information needed to describe architectural connections. There is a one-to-many relation between the "GT_domain_state" and

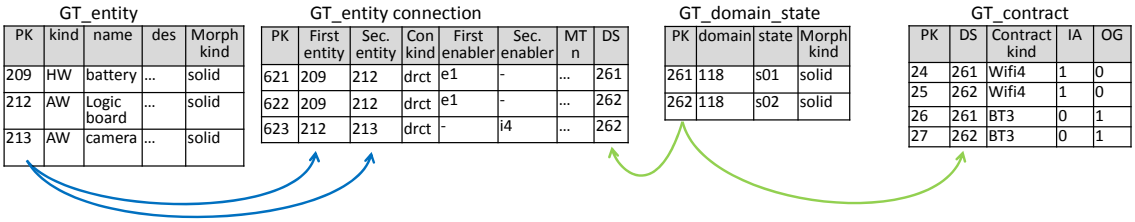


Figure 5-19 Tables regarding architectural connectivity (i.e. GT domain-state, GT_entity, GT_entity connection, and GT_contract)

"GT_contract" tables because every domain may have several possible connections to other domains. The two columns included in the "GT_contract" table specify whether the contract is an input assumption or an output guarantee. The "GT_contract_kind" key refers to the "contract concept" table of the concept ontology. Figure 5-19 shows an example for the tables discussed in this section. In this sample case, the battery is connected to the logic board with the possibility of sending electricity (e1 kind) in both states. However, the camera is connected to the logic board with possibility of sending information (i4 kind) just in the second state.

ISC for specification of entity - UoO relations

Operations of a domain are described in terms of FoOs, while operations of domain entities are described by the UoOs included in the FoOs. This later entails that UoOs refer to the concerned domain entities. With regards to the corresponding construct, it establishes a many-to-many relation between the "GT_architectural_containment" and "GT_UoO" tables (Figure 5-20). Instead of the "GT_entity" table, the "GT_architectural_containment" table is used in the construct because it includes the "GT_domain_state" key, which is needed to specify the validity of the relation in different states of operation.

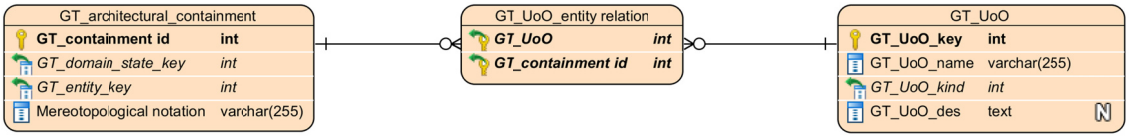


Figure 5-20 Construct for arranging relational tables for capturing relations between domain entities and UoOs by GTs

5.3.4 Template-based definition of a genotype

Some practical considerations

There are numerous pieces of information to be defined and filled in for creating compound SMFs. This implies the need for paying attention to the workload and to opportunities for reducing and simplifying the activities necessitated by an initial definition of a SMF. With the intention of reducing the conceptual and procedural complexity, we devised a step-wise creation procedure for genotypes. Another opportunity is a 'copy and change' type of adaptation of genotypes of SMFs. This can be considered in those cases, where a required GT is in the same category as some other GTs (e.g. multiple classes of cars engines). In this case there is a chance that many of the GT database entities may be reused. It means that certain available GTs can be reused as templates for creating resembling GTs in the same category. Template creation can reduce both the size of database and the workload of knowledge engineers. It can also help importing functional relations of entities (e.g. relations among a cylinder and a piston are more likely identical in all genotypes of the classes of engines).

In the rest of this section, the ideated mechanism of template creation is explained through a simple example, a ball pen. This example is relevant since many resembling manifestations of a ball pen may exist. Considering these manifestations and their combinations as genotypes allows us to cast light on the issue of component reuse and to make the idea of genotype templates tangible. It is presumed that the same principles that are found in the case of manifestations of ball pens can be applied at creating genotype

templates of more compound systems comprising hardware, software, and cyberware constituents, or aggregate-ware components.

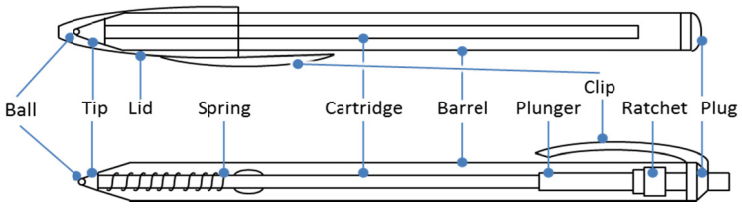


Figure 5-21 Two examples of pens and their domain entities

Two manifestations of a ball pen are shown In Figure 5-21. Their structural parts are identified by the given names and connected with architectural relations as shown in Figure 5-22. It can be seen in these figures that many of the structural entities are shared by the two manifestations but there are several morphologically different entities too. At the same time, important differentiating attributes such as sizes, materials, and colors are yet not determined). There is an opportunity to create a generic model of the ball pens that contains both the shared entities and the differentiating ones by aggregation (without capturing any other distinguishing attributive parameters). This generic model is a genotype of both of these ball pens, and allows deriving structurally and morphologically differing, parametrically not evaluated variants.

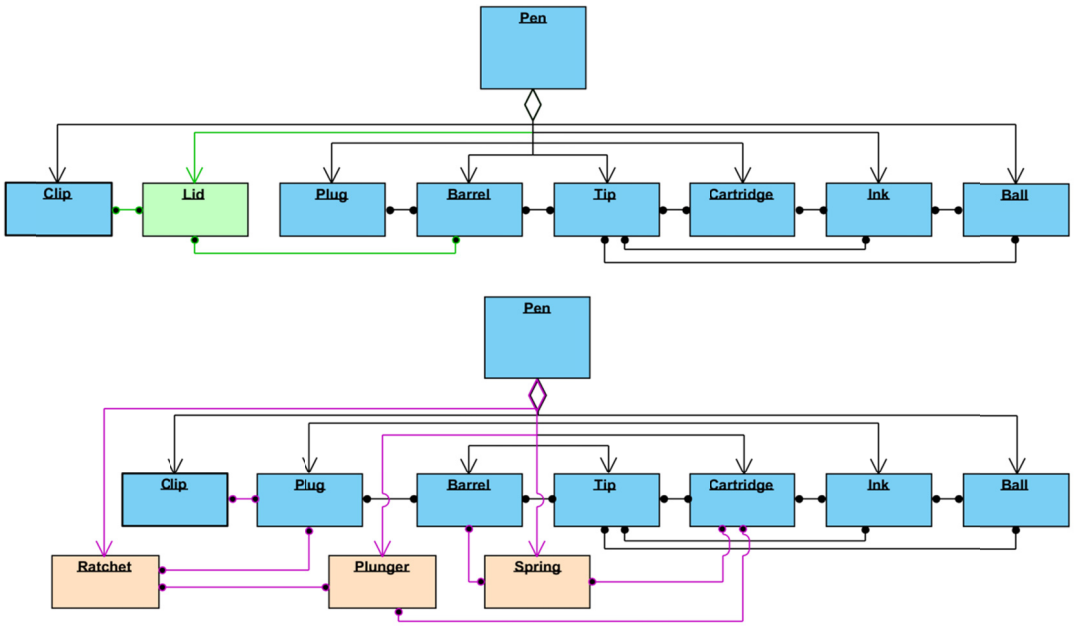


Figure 5-22 The genotype of the pen is generated by combining these different architectural entities and relationships

In our example, the body of the pens is the (main) domain of a SMF. The genotype includes all other shared or unique structural parts as possible domain entities and represents them with structural, morphological, and physical parameterization. They are supposed to be in a 'part-of' relationship (A◇⟶B: B is a part of A) with the domain of the pen. Some of the original 'connected-with' relations among the domain entities (indicated with the A•—•B symbol in Figure 5-22) are kept, but new ones are also established. This way we get to a genotype template based on which a large number of genotypes can be derived from the aggregate genotype represented in this manner. What is more, practically unlimited number of phenotypes and arbitrary instance manifestations can be derived from these, by assigning values to their attributive parameters.

Computational approach of creating genotype templates

The creation of genotype templates needs a dedicated computational approach. We proposed a seven-step procedure, which includes: (i) creation an initial set of GTs (task of a knowledge engineer), (ii) choosing the most comprehensive one as the template kernel (task of a knowledge engineer), (iii) aggregation of the complementing elements of the GTs of the initial set (task of a knowledge engineer), (iv) inclusion of the parameters, relations, and data of the concerned GTs as default values (task of a knowledge engineer and for the modeling tool), (v) changing (adding, removing, and/or modifying) the fields which are variant (task of a knowledge engineer), (v) connecting the primary and foreign keys of the same database entities in the template (task for the modeling tool), and (vi) connecting the changed database entities to the template (task of a knowledge engineer and for the modeling tool). Since SMFs encapsulate both architecture and operation aspects, templates have to be able to capture both of them. Consequently, genotype templates should accommodate architecture connectivity and containment, as well as operation connectivity and containment relations. These were included in four specific tables, which are referred to as genotype skeleton tables. Since skeleton models are proposed to be used for modeling morphology, the template captures the information and relations of these.

Figure 5-23 shows the ERD of genotype template. As shown, each genotype could have many templates (one-to-many relation between 'GT_SMF' and 'GT_template'). A many-to-many relation is specified between template table and each of genotype skeleton tables. Subsequently, one template could be linked to as many as needed skeletal relations, and each skeletal relation could belong to several templates. All these relations are captured in one table, named as 'GT_template_structure'. The repository of the modeling tool and the relational tables in the database can be considered as static part of SMF-TB. While, for designing the dynamic part of it, we need to develop the working algorithms. These algorithms provide basis for Pseudo code development, and consequently, design of procedures for the application software.

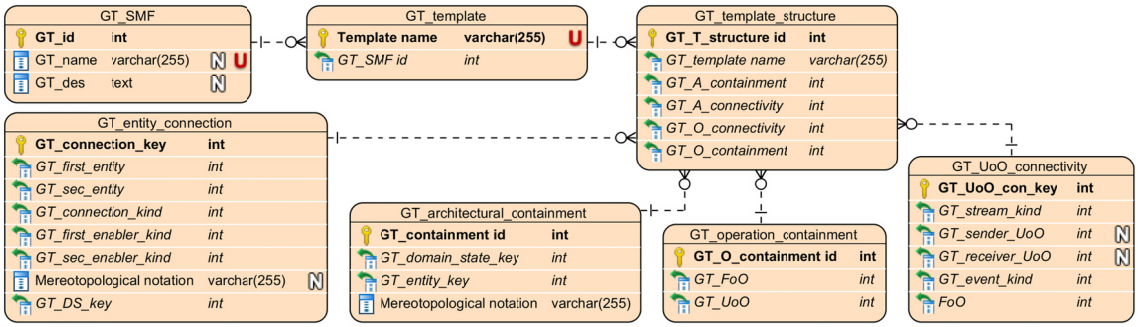


Figure 5-23 ERD of genotype template

5.3.5 Procedure of creating genotypes

As mentioned before, creation of SMFs is a co-creation process, which on the one hand involves several activities performed by knowledge engineer(s), and on the other hand information conversion activities performed by various modules of the SMF-TB. Processing of the user-entered information sets and their relationships, and converting and storing them in the databases requires dedicated procedures [17]. As part of the SMFs co-creation process, GT creation is a sequential process.

Figure 5-24 Procedure of creating a genotype and the actors involved in this procedure

Figure 5-24 presents the proposed computational procedures. The actors involved in creating GTs are: (i) knowledge engineer as the user of SMF-TB in this step, (ii) I/O interface as the module that provides GUI, process the data, and produce output, (iii) GT editor which calls queries to invoke the data from database, and inserts the data, relationships, and inheritance into the GT warehouse, (iv) GT warehouse as repository for storing chunks of information of genotypes and their relations, and (v) ontology tables, as the taxonomical arrangements of concepts which are needed for GT creation.

When a fully-fledged toolbox is available, knowledge engineers select the relevant ontological concepts and specify the relations among them with the support of the toolbox. They enter the information requested about the kind of domain, operation, method, and streams in the GTs creation *input forms* by filling in necessary data into the predefined fields. More than twenty relational data tables are used in the GT warehouse to capture the specific chunks of information required for creating a genotype. Definition of some data depends on the definition of others (e.g. inputting data of entity definition should precede the input of data of entity connection). Therefore, a dependency analysis of the data has been performed to identify the priorities of defining the prerequisite data.

Based on the dependencies, the data aggregation activities were sorted into the following five activity cycles: (i) GT basic definition, (ii) GT operation definition, (iii) GT UoO connection definition, (iv) GT domain-state definition, and (v) GT entity connectivity definition. There is another cycle for overviewing and confirmation of the entered information. The activity cycles of creating GTs are structurally rather similar (with minor differences in threats, queries, and elements). They start with loading GUI components, and then the I/O interface asks for specific pieces of information. The GT editor uses a query to retrieve the data from GT or ontology database. The required data is passed to the I/O interface. A GUI form is rendered and the knowledge engineer fills in the form and selects from the defaulted options. The entered data is checked and passed to the GT editor that fills the chunks of information in the GT database. The following paragraphs explain how the input information is handled by the associated software modules. There are five tabs assigned to the five activity cycles in the user interface (UI) of a GT creation forms. In addition, there is another tab, which provides an overview of the information obtained from the knowledge engineer to confirm the correctness.

Activity cycle one: Definition of GT basics

This cycle specifies basic information about architecture and operations of a genotype. It involves defining the name, description, architectural domain, and operations of GT. In addition, there are two select-box fields for selecting the kind of domain and the kind of operation. The options available in these select-boxes are invoked from the 'domain concept' and the 'operation concept' tables, respectively. Since more than one operation may be added to a GT (due to the defined one-to-many relationship in the database) all fields of the third partition (operation of GT) are recreated as empty fields when the 'add new operation' button is pushed.

Form name: 1. GT basics

1.0. First partition: 1.0.1.GT name <text box>, 1.0.2.GT template <select box>, 1.0.3.Template name <text box>, 1.0.4.GT descriptions <text area>

1.1. Second partition (Architectural domain of GT): 1.1.1.Domain name <text box>, 1.1.2.Domain descriptions <text area>, 1.1.3.Domain kind <select box>

1.2. Third partition (Operation of GT): 1.2.1.Operation name <text box>, 1.2.2.Operation descriptions <text area>, 1.2.3.Operation kind <select box>, Add new operation (1.2) <button>

Activity cycle two: Definition of operation

This cycle specifies the information sets related to operations of a genotype. There are four partitions in this tab. The first part is for selecting operation and selecting kind of possible super-operation. The second part is allocated to define state transition. The kind of 'start state' and 'end state' and the respective 'events' could be chosen from the select-boxes that their options are invoked from ontology tables. The third part is named as I/O transformation. Two select box fields are used to specify kinds of input and output streams which the options are invoked from ontology tables. The fourth part is for defining UoOs. There is a button to add as much as required UoOs. Another button is placed for saving current operation and start with the next one.

Form name: 2. GT operations

2.0. First partition: 2.0.1.Select operation <select box>, 2.0.2.Kind of possible super operation <select box>, Add new super operation (2.0.2) <button>

2.1. Second partition (State transition): 2.1.1.Start state key <text box>, 2.1.2.Start state kind <select box>, 2.1.3.Start event kind <select box>, 2.1.4.End state key <text box>, 2.1.5.End state kind <select box>, 2.1.6.End event kind <select box>, 2.1.7.Transition descriptions <text area>, Add new state transition (2.1) <button>

2.2. Third partition (Input/output transformation): 2.2.1.Kind of input stream <select box>, Add new input stream (2.2.1) <button>, 2.2.2.Kind of output stream <select box>, Add new output stream (2.2.2) <button>

2.3. Fourth partition (Units of operation): 2.3.1.Already defined UoO <select box>, 2.3.2.Name of UoO <text box>, 2.3.3.Kind of UoO <select box>, 2.3.4.UoO descriptions <text area>, Add new UoO (2.3) <button>

Save & next operation (2) <button>

Activity cycle three: Definition of UoO connectivity

This cycle specifies the connections among UoOs. As it is explained in [6], UoOs can only be connected through streams. In fact, this cycle is designed to specify both internal and external streams of a FoO. The first field is a select box for choosing the FoO. Select boxes 'kind of stream' and 'kind of event' (which triggers the stream) could be selected among the options invoked from the 'stream concept' and the 'event concept' tables. Two last select-box fields offer UoOs (already defined in the second cycle) as options for two ends of a stream. These options are invoked from 'GT_UoO' table of genotype database.

Form name: 3. Operation connectivity

3.0. First partition: 3.0.1.Select operation <select box>

3.1. Second partition (UoO connection): 3.1.1.Stream kind <select box>, 3.1.2.Event kind <select box>, 3.1.3.Select sender UoO <select box>, 3.1.4.Select receiver UoO <select box>, Add new connection (3.1) <button>

Activity cycle four: Definition of domain state

This cycle specifies the chunks of information of architecture attributes regarding domain-states. First of all the state should be selected. The options are invoked from 'GT_state' table of genotype database. In the next part, Kind of 'morphology', 'possible super-domain', and 'contract' can be chosen through options invoked from ontology tables. Contract kinds can be selected for both input assumptions and output guarantees. There is an add button to multiply contracts in order to support one-to-many relation. Several domain entities can be defined for a domain-state. For each entity, the name of entity, its description, and the respective UoO should be entered. The options for respective UoO are invoked from the list of UoOs entered in the second cycle. Moreover, the kind of entity and morphology can be chosen from select-boxes, in which the options are invoked from ontology tables.

Form name: 4. Domain-state

4.0. First partition: 4.0.1.Select state key <select box>

4.1. Second partition (Domain-state definition): 4.1.1.Domain morphology kind <select box>, 4.1.2.Possible super domain kind <select box>, 4.1.3.Contract <radio button> options: Input assumption or Output guarantee, 4.1.4.Domain contract kind <select box>, Add new contract (4.1.3 & 4.1.4) <button>

4.2. Third partition (Domain entity): 4.2.1.Entity name <text box>, 4.2.2.Entity kind <select box>, 4.2.3.Entity morphology kind <select box>, 4.2.4.Select respective UoO <select box>, Add respective UoO (4.2.4) <button>, 4.2.5.Entity description <text area>, Add new entity (4.2) <button>

Add a new domain-state (4) <button>

Activity cycle five: Definition of entity connection

This activity cycle processes the information sets concerning the connectivity among the entities. In the first block, 'domain-state' should be selected. The domain-state options are provided from the keys generated in the previous cycle. In the next part, two entities that are connected to each other are selected from the options invoked from 'GT_entity' table of the GT database. In front of 'entity' fields there are two select boxes for specifying kinds of the connection enablers. The options are invoked from ontology tables.

Form name: 5. Entity connection

5.0. First partition: 5.0.1.Select domain-state <select box>

5.1. Second partition: 5.1.1.Select first entity <select box>, 5.1.2.First connection enabler kind <select box>, 5.1.3.Select second entity <select box>, 5.1.4.Second connection enabler kind <select box>, 5.1.5.Connection kind, Add new entity connection (5) <button>

Activity cycle six: Final confirmation

The information is represented in unique versions of AKF and OKF tables. All related chunks of information entered for creation of the genotype of SMF are invoked from the GT database and presented in the GT creation form. The preview is managed and rendered by the I/O interface. When the knowledge engineer clicks on the elements, she will be redirected to the tab where she can modify the parameter(s). Since the parameters are

related together, the other related elements in the table will be highlighted to show the consequences of the modifications.

5.4 INFORMATION CONSTRUCTS FOR CREATION OF PHENOTYPE

5.4.1 Chunks of information needed for creating phenotypes of SMFs

The structural relations of concepts are specified in genotypes, and the attributes and parameters of phenotypes are specified accordingly. Morphologies, materializations, cardinality, positioning, format, conditions, characteristics, and every other kind of qualities related to architectural and operational aspects of a SMF have been made 'manipulatable' through parametrization (i.e. have been considered as 'attributes' in its generic meaning of the word). Information schema constructs (ISCs) have been introduced to help this procedure. The ISCs of a phenotype use 21 ontological concept tables, namely (i) domain concept, (ii) connection concept, (iii) connection enabler concept, (iv) morphology concept, (v) morphology element concept, (vi) morphology parameter concept, (vii) mate concept, (viii) mate parameter concept, (ix) domain attribute concept, (x) attribute parameter concept, (xi) contract concept, (xii) contract parameter concept, (xiii) protocol concept, (xiv) unit concept, (xv) operation concept, (xvi) stream concept, (xvii) event concept, (xviii) operation attribute concept, (xix) operation parameter concept, (xx) state concept, and (xxi) operation layer concept. These concepts create the semantical basis of defining the chunks of information and their relationships in the relational tables of the database of the phenotype warehouse. These read-only relational tables assign PKs to each "kind" defined in the respective ontologies. These PKs are imported to the relational tables of phenotype database in order to specify kinds. The specific chunks of information extracted from AKF are specified below. (The chunks of information inherited from genotype is indicated by the * sign.)

Metadata:	*< kind_of_possible_super_domain >*; < name_of_possible_super_domain >; < description_of_super_domain >; < kind_of_domain >*; < description_of_domain >*; < associated_operation >*; < morphological_kind>*; < representative_state >*; < architectural_attribute_of_domain >; < architectural_attributive_parameter_of_domain >; < value_of_attributive_parameters_of_domain >; < morphological_element_of_domain>; < parameter_of_morphological_element >; < value_of_parameter_of_morphological_element >.*
Domain entities:	*< name_of_entity >*; < entity_description>*; < kind_of_entity >*; < architectural_attribute_of_entity >; < paramter_of_architectural_attribute_of_entity >; < value_of_parameter_of_attribute_of_domain >.*
Entity morphologies:	*< kind_of_entity_morphology>*; < morphological_element_of_entity >; < paramter_of_morphological_element_of_entity >; < value_of_morphological_element >.*
Containment relations:	*< list_of_containment_relations >*; < containment_in_state >.*
Connectivity relations:	*< kind_of_connection >*; < connected_entities >*; < kind_of_connection_enabler>*; < mating_element >; < mating_parameter >; < mating_value >; < connection_in_state >.*
Contracts:	*< kind_of_contract >*; < parameter_of_contract >.*
Constraints:	*< constraint_of_attributive_parameter >; < constraint_of_morphological_parameter < constraint_of_contract_parameter >.*

The specific chunks of information extracted from OKF are as follows:

Metadata:	*< kind_of_possible_super_operation >*; < kind_of_operation >*; < description_of_PT_operation >*; < PT_domain >*; < operation_attribute_of_FoO >; < parameter_of_operation_attribute_of_FoO >.*
States of domain:	*< kind_of_start_state >*; < name_of_ start_state >; < description_of_start_state >; < kind_of_end_state >*; < name_of_end_state >; < description_of_end_state >; < start_event >*; < end_event >*.*
Streams:	*< kind_of_external_stream >*; < kind_of_internal_stream >*; < name_of_stream >; < description_of_stream >; < start_time_stamp_of_stream >; < end_time_stamp_of_stream >; < halt_time_stamp_of_stream >; < resume_time_stamp_of_stream >; < parameter_of_stream >.*
Events:	*< kind_of_event >*; < name_of_event >; < description_of_event >; < stream_to_event >; < state_of_event >; < time_stamp_of_event >.*
Transformative procedure:	*< streams_to_UoOs >*; < direction_of_stream >*.*
Methods of FoO:	*< method >; < element_of_method >; < parameter_of_method >.*
Time stamps:	*< specification_of_time_stamp >; < start_time_stamp >; < end_time_stamp >; < halt_time_stamp >; < resume_time_stamp >; < duration >.*
Scheduling conditions:	*< conditions >; < event_to_condition >; < sequence >; < event_to_sequence >.*
Operational constraints:	*< maximum_value_of_parameter >; < minimum_value_of_parameter >.*
UoOs:	*< name_of_PT_UoO >*; < description_of_PT_UoO >*; < kind_of_UoO >*; < operation_attribute_of_UoO >; < parameter_of_operation_of_UoO >.*
Domain entities of UoOs:	*< list_of_entities_to_UoO >*.*
Methods of UoOs:	*< method >; < elements_of_method >; < parameter_of_method >.*
Operational layers:	*< list_of_layer_ID >, < method_to_layer >.*

5.4.2 Information schema constructs of the phenotypes database

The database of the phenotypes warehouse is organized as entity-relation tables incorporating several various chunks of information. Recall that the tables are related to each other through standard connectors. A relation is valid if the PK of one table is represented as an FK in another table. A large number of relations are needed because of the large number of chunks of information relevant for phenotypes. Actually, the proposed database schemata of phenotypes contain 53 tables and 123 exchanged FKs. This complexity explains the many information schema constructs defined. Since phenotypes inherit various chunks of information from their parent genotype, some ISCs are reusable.

ISC for specification of operation as state transition

Like a genotype, a phenotype also captures information about a domain and the operations associated with it. Therefore, a construct has been designed for this purpose. As shown in Figure 5-25, it includes the "PT_SMF" table that has a one-to-one relation with the "PT_domain" table, and a one-to-many relation with the "PT_operation" table. Directly or indirectly, all operational and architectural warehouse database tables are connected to the "PT_operation" and "PT_domain" tables, respectively. The effects of operations on the architectural domains are modeled as state transitions. Therefore, the operation database tables are connected to the "PT_state_transition table," which includes chucks of information about "start_state," "end_state," "start_event," "end event," and other descriptors of a transition (Figure 13). States are defined by parameters such as "PT_state_name,"

"PT_state_kind," and "PT_des(cription)". Since one operation may cause multiple transitions in architectural states, there is a one-to-many relation between "PT_operation" and "PT_state_transition" table. The "PT_state" table is related to "PT_domain_state" table through a one-to-many connector. The "PT_domain_state" table creates relations between an architectural domain (PT_domain) and its states (PT_states). The "PT_state_transition" table hosts two event keys and two state keys.

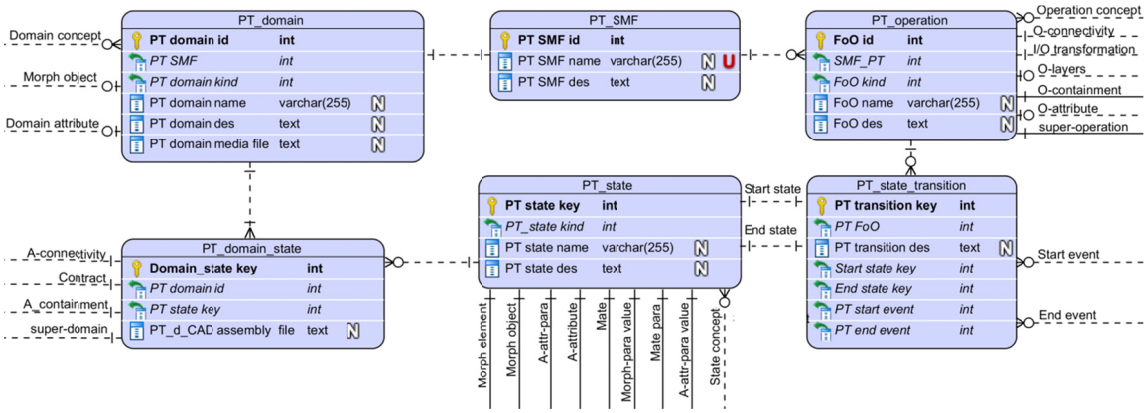


Figure 5-25 Construct for arranging relational tables for capturing state transition by PTs

ISC for specification of operation as I/O transformation

Operations can be seen as transformations of input streams into output ones. The necessary entities and relations are included in the I/O transformation construct (Figure 5-26). A one-to-one relation is defined between the "PT_I/O_transformation" and "PT_operation" tables. Using additional parameters and attributes, streams are defined in more detail in phenotypes than in their parent genotype. There are two instances of many-to-many relations between the "PT_stream" and the "PT_I/O_transformation" tables for both input and output streams, which results in two association tables, namely the "PT_input_stream" and "PT_output_stream."

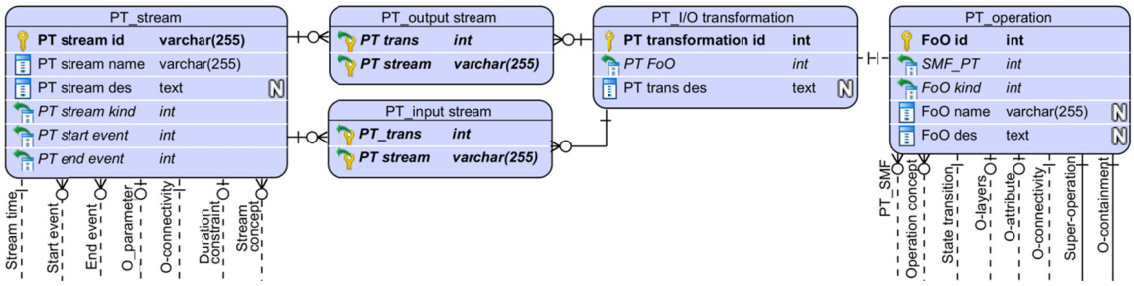


Figure 5-26 Construct for arranging relational tables for capturing I/O transformation by PTs

ISC for specification of operation containment and connectivity

Phenotypes concretize the flows of operations, possible super-operations, and sub-operations (UoO) (Figure 5-27). In this construct, the name of super-operation is the only extra information added to the "PT_possible_super_operation" table. The "PT_UoO" table captures pieces of information related to sub-operations (UoOs). Since one FoO may contain many UoOs, and one UoO may be involved in several FoOs, there is a many-to-many relation between the PT_operation and PT_UoO tables. This relation is captured in

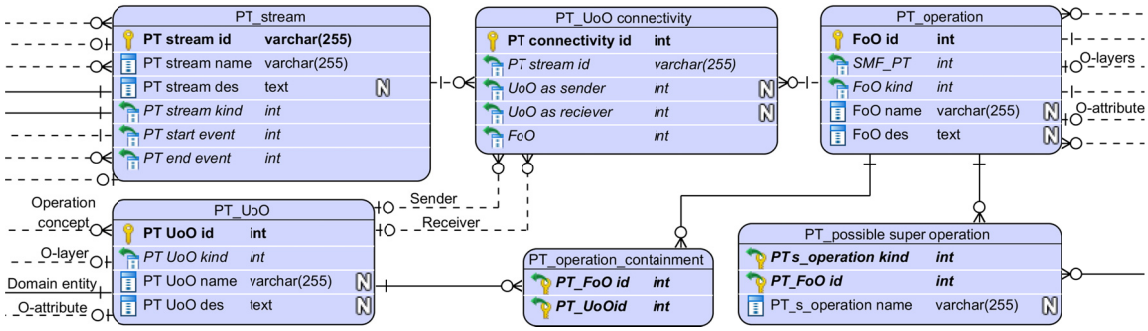


Figure 5-27 Construct for arranging relational tables for capturing operation containment and connectivity by PTs

the "PT_operation_containment" table. The "PT_UoO_connectivity" table captures information about "UoO_as_sender," "UoO_as_receiver," and the streams between them. The connectivity and containment tables of genotypes and phenotypes are more or less similar. The only difference is that the event column is placed into the stream table of phenotype, instead of the connectivity table.

ISC for specification of method of operation

SMFs define physical operations (transformations) in an analytic (numerical) manner and computational operations (transformations) in an algorithmic (logical) manner. Computing of these operations needs different methods. A set of architectural parameter (e.g. morphology, mate, architecture attribute, and contract) tables and operational parameter (e.g. operation attribute, stream, duration, and time) tables are needed to describe the computing methods of a SMF. The chunks of information captured by the tables may be related to each other in various ways (Figure 5-28). The construct, designed for mapping them into the warehouse database, includes: (i) one table to store the methods, (ii) multiple tables for storing different kinds of parameters, and (iii) one table that creates relations between the associated parameters and the elements of the methods.

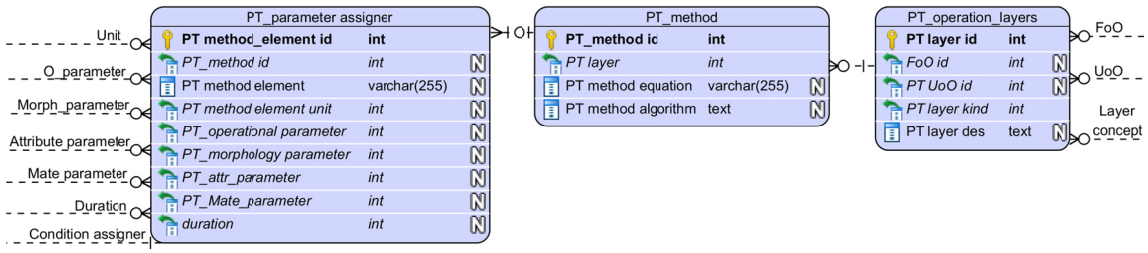


Figure 5-28 Construct for arranging relational tables for capturing methods of operations by PTs

The construct establishes relations between the "PT_operation_layers," the "PT_method," and "PT_parameter_assigner" tables. The "PT_method" table stores both numerical and logical methods. The "PT_method_element" table assigns parameters to the elements of methods. The "PT_parameter_assigner" table establishes a many-to-many relation between the "PT_method" table and the parameter table. One parameter may be involved in multiple methods, and one method may establish connection among many parameters (both architectural and operational).The contents of the parameter table will be explained later.

Since certain operations may occur simultaneously, the opportunity of parallel computing is to be considered. A typical case, where it may be needed, is Multiphysics analysis and

simulation of components of CPSs. To support this from a data management point of view, the concept of layering is applied. Layers lend themselves to simultaneous computing of operations. In the construct designed for this purpose, three kinds of layers are defined as subordinates of the main operation layer. These are specified in the "PT_operation_layers" table. This assigns computing methods from the "PT_method" table of the warehouse database to the layer of operation on which FoOs and UoOs are specified.

ISC for specification of operation parameters and constraints

Streams connect UoOs, and attributive operation parameters describe the actual physical or computational transformations. In SMFs-based modeling, the attributive parameters of UoOs are not since they are used as black boxes. However, if an operation parameter is to be defined for a method, it can be done through a stream, or an attribute, that may share parameters. There is no direct connection between operation attributes and streams. The related construct has been designed based on these assumptions. As shown in Figure 5-29, FKs from the "PT_stream" and the "PT_operational_attribute" tables are accommodated in the "PT_operational_parameter" table. The connection with the "PT_constraint" table makes it possible to define constraints for each parameter. The "PT_operational_attribute" table receives FKs from "PT_operation" and "PT_UoO" tables. Based on this, operational attributes can be defined for both FoO and UoOs.

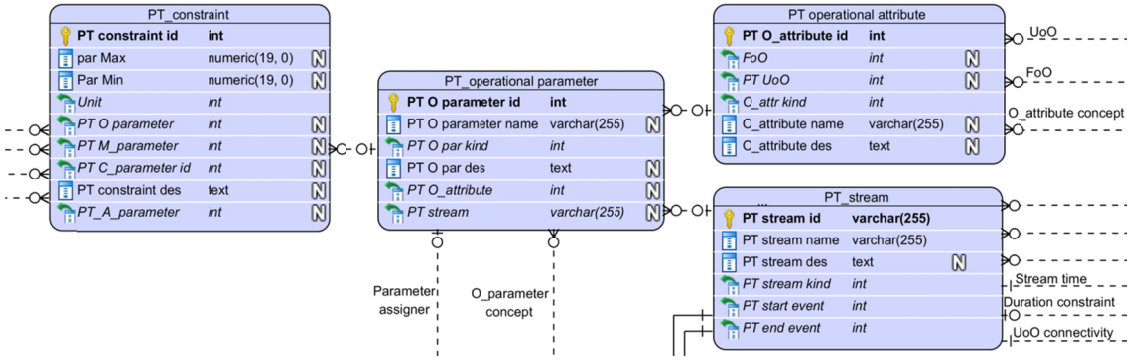


Figure 5-29 Construct for arranging relational tables for capturing operation parameters by PTs

ISC for specification of events

Specification of the streams and state transitions necessitates a forerunning definition of events. In our modeling system, events are triggers of streams and states transitions. Each stream starts with an event, and there is an event when the processing of a stream is completed. Moreover, pauses and resumes of the streams are also regarded as events. Assigning timestamps to all events allows our modeling system to consider operational and state conditions and sequential constraints. In addition, changes of streams and state transitions can be forecasted by tracking events. These operations have been supported by specific ISCs. As shown in Figure 5-30, the "PT_event" table is connected to the "PT_state_transition" and "PT_stream" tables through two connectors: one for the start event and the other for the end event. These connectors establish a one-to-many relation, since each event may be related to multiple streams (e.g., the start event of one stream can be the end event of another stream).

Events typically happen in a subsequent way in various operations, but overlaps and concurrences should also be taken into account. Sequencing can be made by relations such as "before," "after," "coincident with," and "asynchronous with." Pairs of related events have been used to define sequencing constraints for all events. The "PT_sequence_constraint" table makes it possible to define sequential relations by two FKs of related events. The designed construct creates an opportunity to handle conditions of operation through events. Since events are also connected to streams, all operations can be defined conditionally. Conditions are logical statements that are used to make events dependent on the consequences of other events and on parameter values. These logical or mathematical statements are described in the "PT_condition" table of Figure 5-30. The conditions are assigned to events and/or parameters in the "PT_condition_assigner" table. The "PT_condition" table has FKs in the "PT_event" and "PT_parameter_assigner" tables.

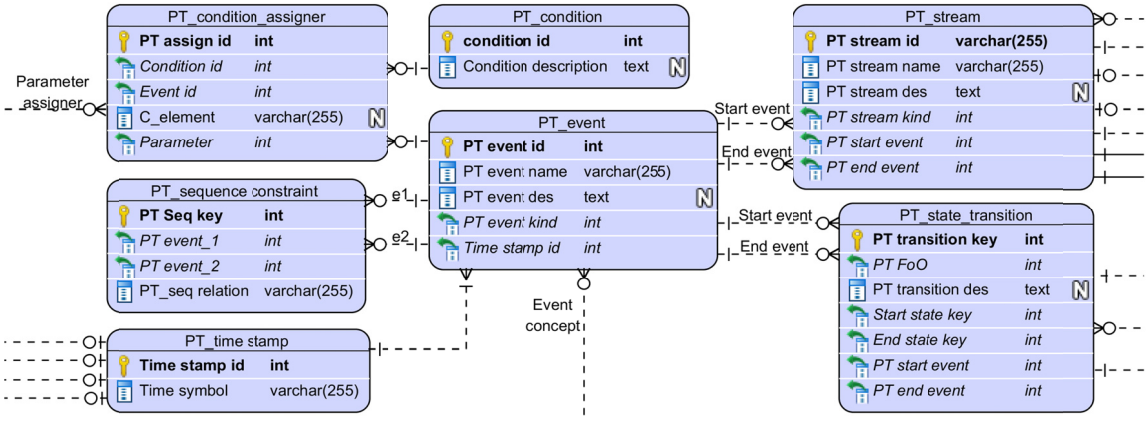


Figure 5-30 Construct for arranging relational tables for capturing events, conditions, and sequence constraints by PTs

ISC for temporal specifications

SMFs are supposed to be able to explicitly handle time aspects of operation, including existence. However, phenotypes are the entity manifestations that used in modeling in specific application context, rather than SMF instances. Therefore, temporal specifications have been included in the phenotypes in order to provide part of the chunks of information that are needed to: (i) compute time-dependent operations of executable instances, (ii) assign points in time and durations to operations, and (iii) time-dependent attributes to streams, events, and entities of SMFs. These are called temporal specification (TS). It is assumed that an event occurs in zero time, and that it is sufficient to assign only one time stamp. However, any change of a stream may need a particular duration, and for this reason a specification of at least two time stamps (begin and end) is necessary. Since streams may be stopped and recommenced, more time stamps may be needed as halts and resumes. The time stamps of the disruption points of streams are taken up by events.

In the construct shown in Figure 5-31, three tables are used for temporal specification. The "PT_stream_time" table captures the start, halt, resume, and end time stamps of streams, which are assigned to the events by the "PT_time_stamp" table. This makes it possible to assign one particular time stamp to multiple events. Together with the condition and sequence constraints assigned to events, this enables time- and condition-based simulation of operations (transformations of streams). The corresponding measurement units and the duration of streams are specified in the "PT_duration_constraint" table. If there is

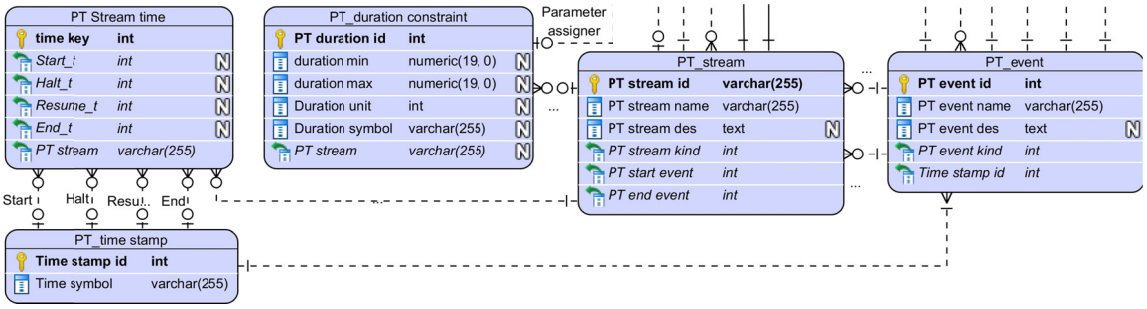


Figure 5-31 Construct for arranging relational tables for capturing temporal specifications by PTs

no constraint on the duration of a stream, then its key will not be included in the "PT_duration_constraint" table.

Figure 5-32 shows an example of temporal specification that has been included in the phenotype database tables. As it can be seen, the timestamp keys 352 and 355 are assigned to the start and the end events, respectively. The same keys are repeated in the table 'PT_stream_time', for the respective streams. Duration constraint specified for this stream is 0 > D1 > 60 seconds. If no duration constraint is needed for a given stream, then the respective '*stream key*' will not be inserted in the 'PT_duration_constraint' table.

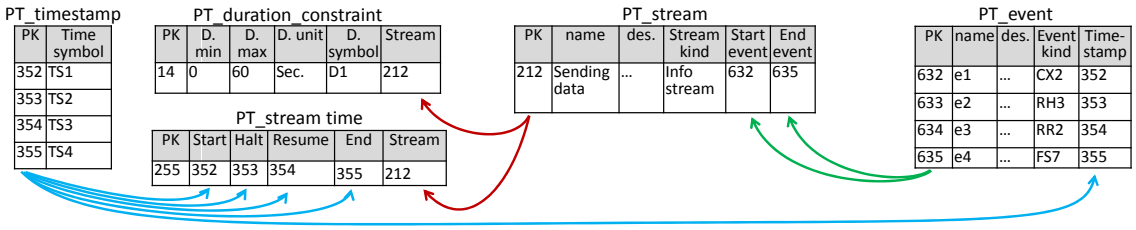


Figure 5-32 Tables regarding temporal specifications of streams and events

ISC for specification of architectural containment and connectivity

According to their supposed operations, the architectural specification of SMFs needs the consideration of three granulation levels, namely: (i) domain, (ii) possible super-domain, and (iii) sub-domain (domain entity) levels. Each domain is an aggregation of multiple domain entities, and can be referred to in several physical contexts (as super-domain). During the operation of an SMF, entities may be added to, changed, or removed from the aggregate represented by a domain. Additionally, the physical context of a domain may also change (due to a change in super-domain). These are considered as state changes of the domain. In a real-life situation, a domain can be embedded in an aggregative hierarchy of multiple super-domains.

The whole of the construct designed for this purpose is shown in Figure 5-33. It also captures the architectural connections with regard to the domain states, because they may change as a result of state changes. The abovementioned requirements necessitate an explicit handling of the containment relations with regard to the "PT_domain_state" table. As a mediator, the "PT_domain_state" table is connected to the "PT_domain" table. Changes of the state can be represented by including respective new keys in the "PT_domain_state" table, which is connected to all architectural tables. The "PT_domain_state" table has a one-to-many relation with the "PT_DS_A_containment"

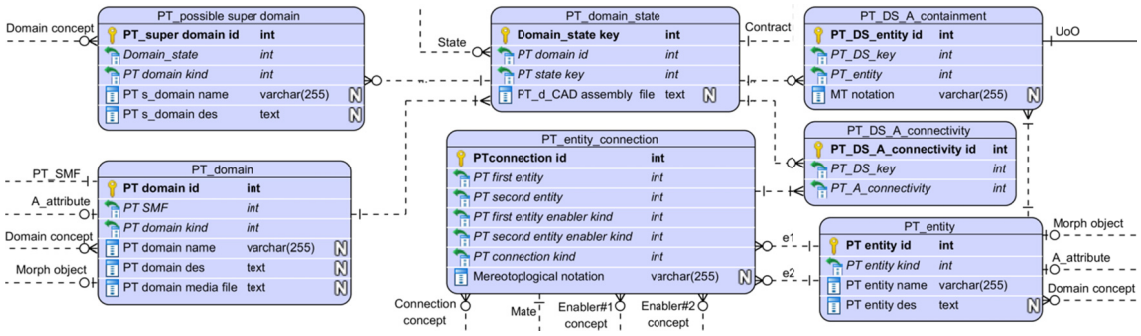


Figure 5-33 Construct for arranging relational tables for capturing architectural containment and connectivity by PTs

table. It means that all domain entities created by domain aggregation and deaggregation can be captured in each state of the domain.

We may think of a video-conferencing application software as an example of a domain. Adding to or removing contacts from the contact list can be regarded as changes in domain entities. Changes in domain entities are interpreted as (architectural and/or operational) states. In the context of this example, when the app (i) resides on the cloud app store, (ii) is installed in the smartphone's memory, (iii) is in-loaded in the virtual machine, (iv) resides in the device RAM, can be considered as cases of having multiple super-domains, which can also be specified related to different domain-states. Therefore, all containment relations should be defined related to domain-state table (Figure 5-34). The possible connections between domain entities are captured in the "PT_entity_connection" table, which includes not only the references to domain entities, but also information about the kind of connection and the kind of connection enabler. For the reason that during different states of the domain, the connections between domain entities may be changed, architectural connection should also be specified with respect to domain-states. If we take the gear shifting mechanism of a bicycle as another example of a domain, the connection between the chain wheels and chain results in different domain-states. To be able to handle these situations, a many-to-many relation is defined between the "PT_domain_state" and the "PT_entity_connection" tables in the construct, through the "PT_DS_A_connectivity" as association table.

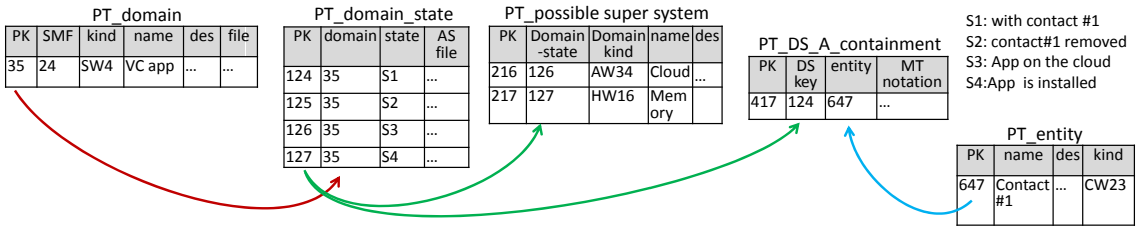


Figure 5-34 Tables regarding architectural containment

ISC for specification of morphology

Morphology of domains and domain entities is explicitly captured in SMF-based modeling, no matter if they are hardware, software, or cyberware components. Morphology describes the space that domains occupy in the 3D space. The form of the occupied space is the palpable shape of the domain. A morphological representation and its formats are managed as high-level variables in data management. With a view to the need of morphologi-

cal representation in the pre-embodiment design phase of CPS development, a combined skeleton-based and closure-box-based modeling is applied for solid-type physically interpretable domains. The skeleton-based modeling exemplifies ports, based on which not only rigid but also liquid and gas domain entities can be modeled. This morphological representation can also capture the morphological characteristics of software and cyberware domains. The construct designed for handling morphologies is able to embed imported CAD files. All these concepts are stated in the "morphology concept" table of the concepts ontology.

The relational tables arranged by this construct are shown in Figure 5-35. The kernel is the "PT_morphological_object" table, which is connected to the "PT_morph (ological)_element" table. It includes the morphological kinds (e.g., solid, liquid, gas, script, code), in addition to "PT_domain" and "PT_entity." The management of morphological models is based on the "PT_morph(ological)_element" (e.g., point, curve, surface) and "PT_morphological_object" tables, which are in a one-to-many relation. The options of element morphologies are stated by the "morphology element concept" table of the concept ontology. For decomposability, there is a reflective one-to-many relation defined for the "PT_morph_element" table. This relation is realized through the nullable "parent element" FK.

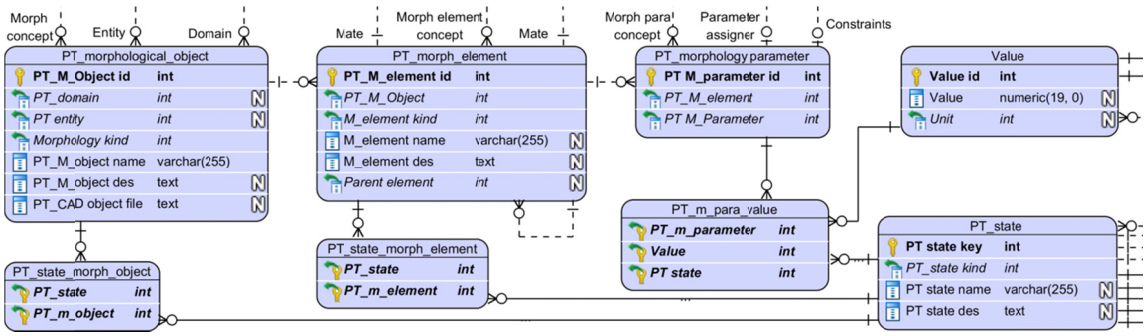


Figure 5-35 Construct for arranging relational tables for capturing morphology by PTs

Description of the morphology of the domain elements needs several parameters (e.g., height, volume) and values (e.g., linear coordinate, angle). The options are stated in the "morphology parameter concept" table of the concept ontology. For many parameters default values are specified, e.g., based on entity relations. A morphological description is valid in the state in which it is described. The morphological elements of the "PT_morph_element" are mapped to the parameters included in the "PT_morphological_parameter" table, and subsequently to the values in the "PT_para(meter)_value" table, to which concrete values can be assigned. The morphology of a domain or an element thereof can change as a result of operation. Therefore, the morphological tables have connection with the operation tables through the domain state table.

ISC for specification of mate

Morphological connection between physical (material) domains and/or entities is called mate. The possible morphological connections are stated in the "architectural connectivity" table of the concept ontology. Skeleton modeling offers ports for these morphological connections. In the construct shown in Figure 5-36, the "PT_morph(ological)_element" table is the basis of specifying mate connections using the content of the "PT_mate" and

"PT_mate_parameter" tables. The "PT_entity_connection" table is referred to by the "PT_entity_connection" key from the "PT_mate" table. Since mate is dependent on the domain states, there is a many-to-many connection between the "PT_mate_state" table and the "PT_state" table of the constructs. Specifying mate connection assumes the existence of an architectural connection between two domains or entities, as well as the selection of two morphological elements.

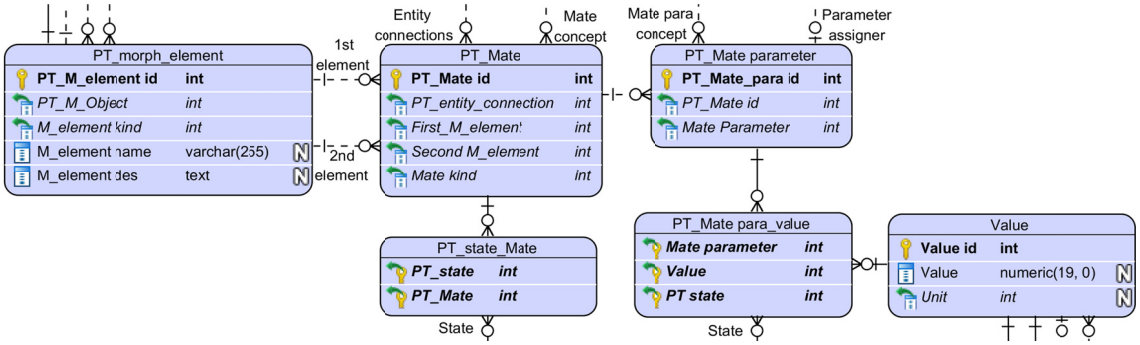


Figure 5-36 Construct for arranging relational tables for capturing mates by PTs

The "mate_kind" key of the "PT_mate" table specifies the mate situation (e.g. tangent mate, perpendicular mate, coincident mate, or concentric mate) between them. Some mate situations, such as distance mate or angle mate, are specified by respective parameters, which are given in the "PT_mate_parameter" table. The kinds of mate parameters are stated in the "mate parameter concept" table of the concept ontology. The values of mate parameters are defined in the "PT_mate para(meter)_value" table, which is connected to the "Value" and "PT_mate" tables, and allows capturing mate parameter values as function of states. There might be a situation in which the mate parameter values should be computed by specified methods. For example, in a capacitive accelerometer, the distance between fixed a movable plates defines the electric current and, consequently, this distance are subject to indirectly measure the applied acceleration. Therefore, this mate parameter is an element in a mathematical equation that is used to calculate the applied acceleration. To support this issue, the "PT_mate_parameter" table is connected to "parameter assigner" table in order to provide access to the "method" table.

ISC for specification of architecture attributes

The construct shown in Figure 5-37 is dedicated to capturing architectural attributes (AAs). These attributes of a domain are varied (e.g., weight, material, cardinality, and flexibility.). All possible attributes are handled by the "PT_domain_attribute" table, which uses keys generated based on the "domain attribute concept" table of the concept ontology. The kind of AA parameter is complemented with parameters, units, and values. Since attribute parameters are used for computing values of operational parameters, the "PT_domain_attribute" table is connected to the "PT_parameter_assigner" table. This is used when parameters are assigned to methods of operation.

The kind of attribute and the attributive parameters may change in different states (over time). For example, viscosity of a lubricant may change due to generated heat. Having this in mind, we provided opportunity for defining the parameter (viscosity) values in different associated states. This explains the many-to-many relations between the abovementioned tables and the "PT_state_attribute" association table, which enables the connection

between attributive parameters and states. In addition, the "PT_domain_attribute parameter" table is connected to the "PT_constraint" table in order to specify the minimum and maximum values of the attributive parameters.

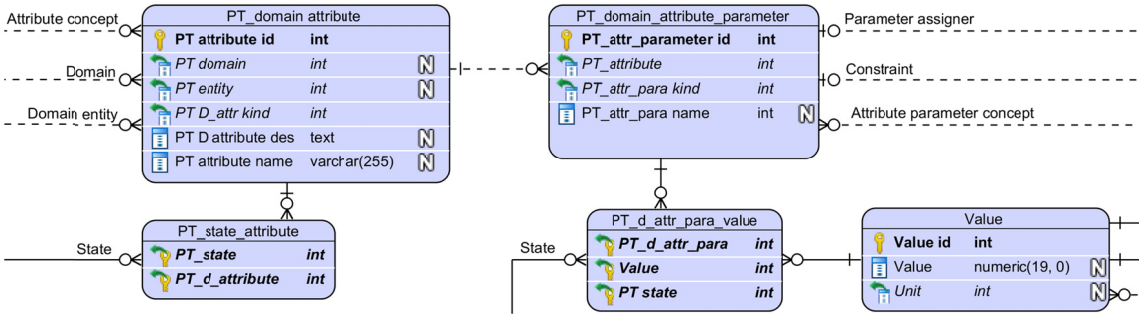


Figure 5-37 Construct for arranging relational tables for capturing architectural attributes by PTs

ISC for specification of contracts

The notion of contracts was adopted to inform about the possibility of establishing connectivity between architectural domains. Contracts are uniquely associated with domains and represent their architectural interfaces. The construct that has been designed to manage contracts is shown in Figure 5-38. There can be parametric contracts and protocol contracts defined. Their concept is stated in the concept ontology. Contract specification first of all embraces the kind of contract. It can be either an input assumption or an output guaranty. These are included in the "PT_contract" table. No matter which hardware, software, or cyberware contracts are concerned, their parameters are captured by the "PT_contract" table. The possible range of parameters is specified in the "PT_constraint" table. Protocol contracts are defined in the "contract protocol" table. They can be operationalized based on the entries of this table. The kinds of applicable protocols are included in the "protocol concept" table. The contracts are also associated with the domain states through the "PT_DS_contract" association table.

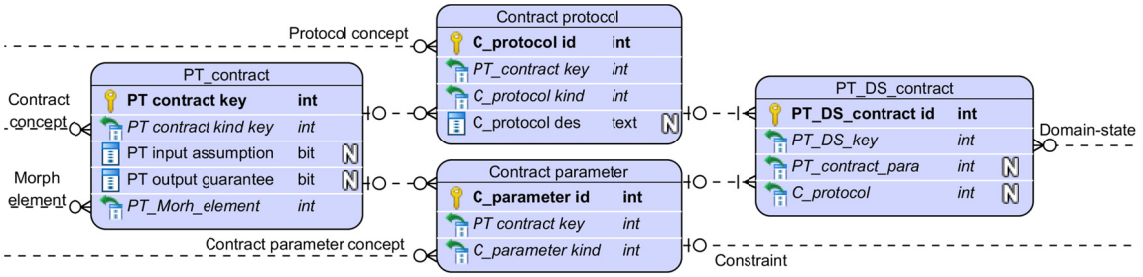


Figure 5-38 Construct for arranging relational tables for capturing contracts by PTs

ISC for specification of constraints

Typical for phenotypes is that they specify a possible range of parameter values, rather than nominal values. It is an issue of instantiation in modeling how to derive the best matching value for an SMF instance from the range of parameter values. This issue is out there even though there may be some constant parameter values (CPVs) included in the phenotype, as input values. From a data management point of view, this is taken care of by the construct shown in Figure 5-39. As a rule, CPVs are defined in the respective association tables (such as "PT_morph_para_value,", "PT_mate_para_value," and "PT_attr_para_

value") of many constructs. The range of values of variable parameters is included in the "PT_constraint" table of the defined construct.

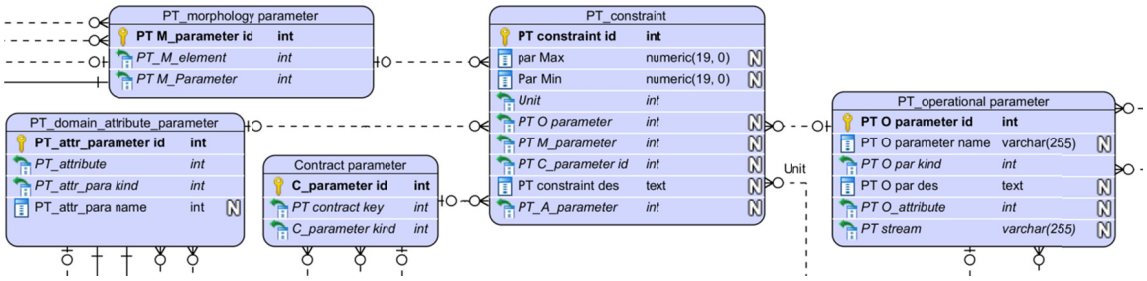


Figure 5-39 Construct for arranging relational tables for capturing parameter constraints by PTs

It may occur in different CPS modeling situations that the range of variation of a parameter value should be specified depending on the range of variation of another parameter value. This needs a specific type of constraint, which sets the range of one parameter according to the range of another parameter that it depends on. These have been called "relative constraints." and are managed in the "methods of operation" construct shown in Figure 5-28. For all variable and constant parameters assigned by the "PT_parameter_ assigner" table, the relative constraints upon ranges or values, are determined by the mathematical or logical equations included in the columns of the "PT_method" table.

Joint points among operation and architecture

The proposed information schema constructs couple certain chunks of operational and architectural information in relational data tables of the warehouse databases. At the highest aggregation level, operations are connected to their architectural domain through the "PT_SMF" table. A second connection is that the operations generated state transitions refer to the states of the concerned architectural domain. This relation is established by means of the "PT_state" and "PT_domain_state" tables. A third connection is that units of operation are connected to domain entities through the "PT_UoO_PT_entity" table. This connection implies relations among subdomains and sub-operations. A fourth connection is captured by the "PT_parameter_assigner" and "PT_method" tables, which assign architecture and operation parameters to numeric and algorithmic methods.

5.4.3 Procedure of deriving phenotypes

The procedure of deriving phenotypes is a complex procedure from a knowledge engineering point of view. All system-level information should be captured and available in this step. Towards this end, knowledge engineers typically use catalogues of components or company product specifications as (the primary) reference in this step. However, there may be need for other pieces of information that should be measured or acquired in other ways in order to complete a phenotype. We mention these issues for the sake of completeness, but do not make efforts to resolve them. We considered the issues related to providing the necessary and sufficient information for phenotype specification being out of scope of this promotion research. We came to this conclusion after analyzing the situation as discussed below.

We conceived three streams of information for conceptualization of the computational procedures (input and checking) of deriving PTs (Figure 5-40). The stream S1 represents the chunks of information inherited form the chosen GT. The stream S2 refers to chunks of information specified (inserted or approved) by knowledge engineers. Options of select boxes are provided by handling the information incorporated in the stream S3 (retrieved either from ontology tables or the PT database (in the latter case, from the information specified and stored earlier by the knowledge engineer).

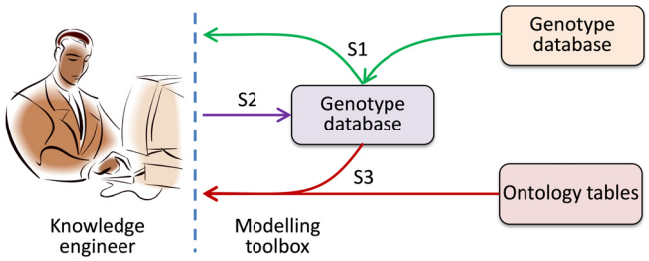


Figure 5-40 Streams of information needed for deriving PTs

The mentioned streams are concurrently involved in the procedure of deriving phenotypes. To complete the workflow of deriving phenotypes, we needed to examine the mentioned streams in each stage of SMFs-based modeling. Based on the results of the analysis of the required chunks of information, eleven sequential cycles of information specification have been proposed. The aim of these cycles is to systematically collect the required information in manageable sets. These cycles (displayed as tabs in the UI) are: (i) phenotype basics, (ii) operations, (iii) units of operation, (iv) operation connectivity, (v) events, (vi) operation parameters, (vii) domain-states, (viii) morphology and attributes, (ix) contracts, (x) architectural connectivity, and (xi) methods.

In the modeling process, deriving a phenotype starts with selecting of a genotype and processing its information contents. The information sets of the selected genotype are automatically copied to the similar relational tables of the PT database. The associated fields of the user interface are filled with the imported information as default data. Five computational modules are involved in deriving phenotypes: (i) the I/O interface that renders the input forms, checks the inserted data types, passes the inserted data, and receives and fills in the data in the forms, (ii) the GT editor that is in charge of invoking genotype information and passing the sorted data to other modules, (iii) the GT warehouse that provides the lists of available genotypes and sends the information set of the selected genotype to the GT editor, (iv) the PT editor that checks the information types, receives information sets, and passes them to and among the I/O interface, PT warehouse, and GT editor, and (v) the PT warehouse that manages the various units of information, and provides lists and datasets of phenotypes.

We have already hinted at the fact that there are two approaches for deriving a new phenotype. In the first approach, a phenotype is directly derived from a genotype. In the second approach the new phenotype is created as an adapted replica of other available phenotype (supposed it has significant similarity, like a functionally similar smartphone with higher internal storage capacity). In this approach the defined phenotype is actually cloned and purposefully modified. These two approaches are generally explained in terms of the procedural and technical details in Figure 5-41. In the rest of this sub-chapter, only the first approach is discussed since the second approach has common elements with it and is less complicated due to involvement of fewer software elements. There is another possibility for editing an available phenotype that more or less similar to the second approach, with this difference that the duplication is not occurred.

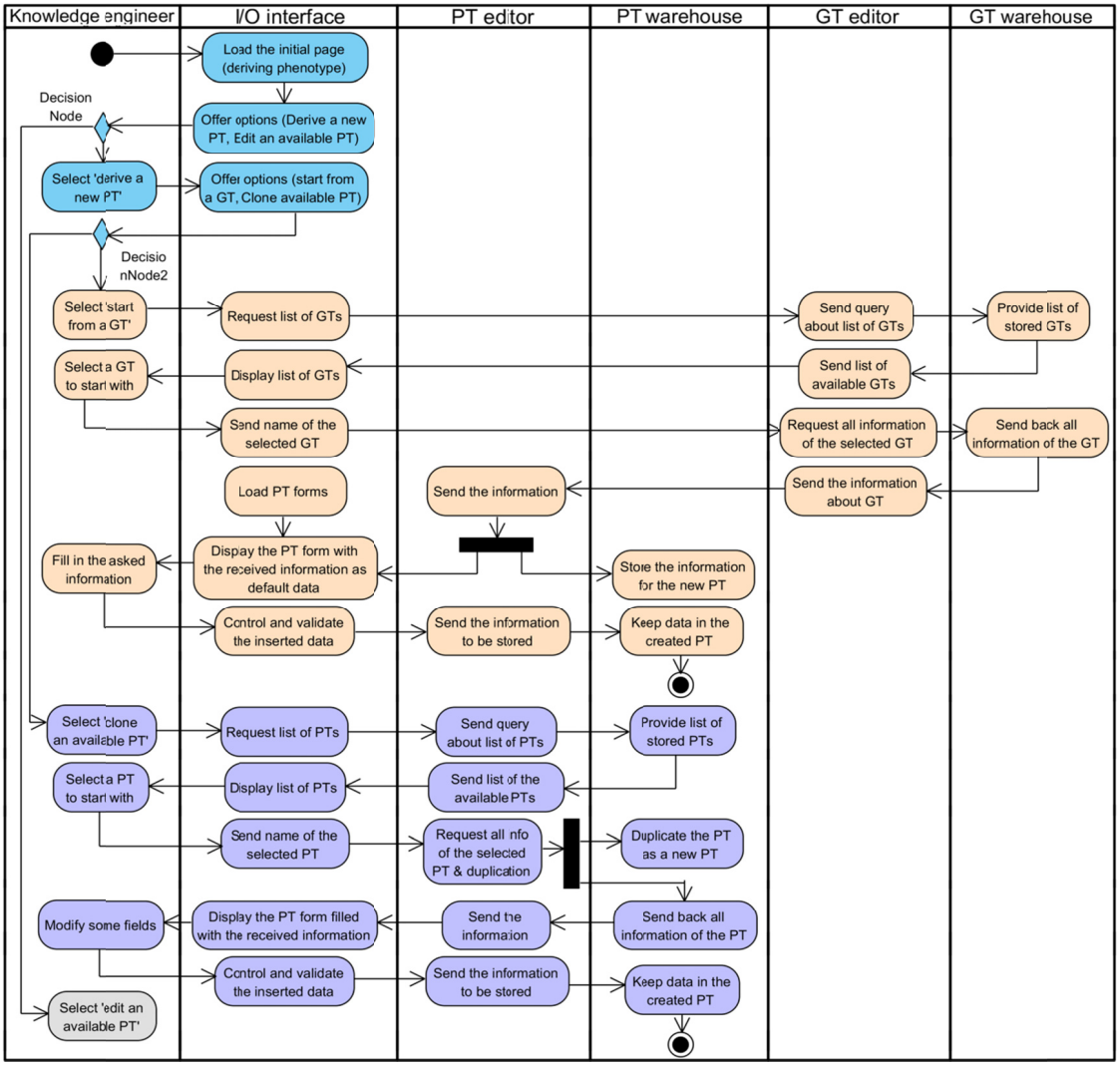


Figure 5-41 Overall procedure of deriving phenotypes

Based of the above considerations, there have been eleven tabs incorporated in the UI. There is another tab that shows overview of the inserted information for confirmation at the end of the activity cycle. Color codes are used to specify where the data comes from. The green color is used for the information inherited from genotype. The red color code is used for the information provided from ontology tables. These chunks of information are used to provide options of the select boxes. Therefore, the red color is only used for the select boxes. The blue color is used for the information invoked from phenotype database. These chunks of information are usually defined in the relative former cycles and used to provide options for making new relations among database entities. Accordingly, they mostly could be seen in the select boxes. It is noteworthy that the red and blue colors both refer to S3. We separate the color codes due to the differences in the source. Purple color code is used for the information should be defined newly for phenotype creation. The following paragraphs explain how the input information is handled by the associated software modules.

Activity cycle one: Definition of PT basics

These activities of defining PTs are largely similar to those of the 'genotype basics' cycle. Most of the fields are filled with default values inherited from the chosen genotype, which could be extended or modified for finer specification of phenotypes.

Form name: 1. PT basics

1.0. First partition: 1.0.1. PT name <text box>, 1.0.2. PT descriptions <text area>

1.1. Second partition (Architectural domain definition): 1.1.1. Domain name <text box>, 1.1.2. Domain descriptions <text area>, 1.1.3. Domain kind <select box>, 1.1.4. Media file <brows file button>

1.2. Third partition (Operation definition): 1.2.1. Operation name <text box>, 1.2.2. Operation descriptions <text area>, 1.2.3. Operation kind <select box>, 1.2.4. Operation attribute name <text box>, 1.2.5. Operation attribute descriptions <text area>, 1.2.6. Operation attribute kind <select box>, Add new operation attribute (1.2.4-1.2.6) <button>, Add new operation (1.2) <button>

Activity cycle two: Definition of PT operations

These activities specify: (i) methods of operation, (ii) state transitions, and (iii) I/O transformations. As demonstrated in Digital Appendix 2, fields are filled with the chunks of information inherited from the chosen genotype. In the beginning, a select box is provided that offers options to choose one of the operations defined in the previous cycle. In fact, the current form is allocated to further elaboration of the defined operations. The first partition of this tab is for specifying possible super-operations. The second partition asks for specifying method of operation as well as the layer the operation belongs to. Methods could be defined in two separate areas, one for mathematic equations, and the other for logical statements. Multiple methods could be defined for each operation. The third and fourth partitions (state transition and I/O transformation, respectively) have several fields filled with GT information. Detailed information about associating events, states and streams should be provided in these partitions.

Form name: 2. PT operations

2.0. First partition: 2.0.1 Select operation <select box>, 2.0.2 Kind of possible super operation <select box>, 2.0.3. Name of super operation <text box>, Add new super operation (2.0.2 & 2.0.3) <button>

2.1. Second partition (Methods of operation): 2.1.1. Method equation <equation area>, 2.1.2. Method algorithm <text area>, 2.1.3.Layer of methods <select box>, 2.1.4. layer description <text area>, Add new layer method (2.1) <button>

2.2. Third partition (State transition): 2.2.1. Start state name <text box>, 2.2.2. Start state kind <select box>, 2.2.3. Start state descriptions <text area>, 2.2.4. Start event name <text box>, 2.2.5. Start event kind <select box>, 2.2.6. Start event descriptions <text area>, 2.2.7. End state name <text box>, 2.2.8. End state kind <select box>, 2.2.9. End state descriptions <text area>, 2.2.10. End event name <text box>, 2.2.11. End event kind <select box>, 2.2.12. End state descriptions <text area>, 2.2.13. Transition descriptions <text area>, Add new state transition (2.2) <button>

2.3. Forth partition (Input/output transformation): 2.3.1. Input stream name <text box>, 2.3.2. Kind of input stream <select box>, Add new input stream (2.3.1 & 2.3.2) <button>, 2.3.3. Output stream name <text box>, 2.3.4. Kind of output stream <select box>, Add new output stream (2.3.3 & 2.3.4) <button> 2.3.5. Transformation descriptions <text area>

Activity cycle three: Definition of UoOs

The aim of this cycle is to gather more information about operational attributes and methods of UoOs. Names, kinds, and descriptions about UoOs are inherited from the chosen GT. The fields in this form are mostly similar to the attribute and method fields of operation (in the first and the second cycles). It offers a field for selecting the operation that the UoOs are part of. There is a button named 'Add new UoO' that makes it possible to define several UoOs for a single FoO. This function is supported by the many-to-many relations among FoO and UoO database tables.

Form name: 3. UoO

3.0. First partition: 3.0.1. Select operation <select box>

3.1. Second partition (Units of operation): 3.1.1. Name of UoO <text box>, 3.1.3. Kind of UoO <select box>, 3.1.4. UoO descriptions <text area>, 3.1.5. Operation attribute name <text box>, 3.1.6. Operation attribute descriptions <text area>, 3.1.7. Operation attribute kind <select box>, Add new operation attribute (3.1.5-1.2.7) <button>, 3.1.8. Method equation <equation area>, 3.1.9. Method algorithm <text area>, 3.1.10.Layer of methods <select box>, 3.1.11. layer description <text area>, Add new layer method (3.1.8 – 3.1.11) <button>, Add new UoO (3.1) <button>

Go to the next operation (3) <button>

Activity cycle four: Definition of operation connectivity

These activities specify the connections among UoOs. The first partition is for selecting the FoO that the connections among the UoOs are realized in. The options for both FoOs and UoOs are provided in select boxes from the previously defined operations and UoOs which stored in the PT database. The third and the fourth partitions are allocated to definition of timestamps of streams and events, respectively.

Form name: 4. Operation connectivity

4.0. First partition: 4.0.1. Select operation <select box>

4.1. Second partition (Stream): 4.1.1. Stream name <text box>, 4.1.2. Stream kind <select box>, 4.1.3. Select sender UoO <select box>, 4.1.4. Select receiver UoO <select box>, 4.1.5. Stream descriptions <text area>

4.2. Third partition (Stream timestamps): 4.2.1. Start timestamp <text box>, 4.2.2. Halt timestamp <text box, add button>, 4.2.3. Resume timestamp <text box, add button>, 4.2.4. End timestamp <text box>, 4.2.5. Duration symbol <text box>, 4.2.6. Duration minimum <text box>, 4.2.7. Duration maximum <text box>, 4.2.8. Duration unit <select box>

4.3. Forth partition (Stream events): 4.3.1. Start event name <text box>, 4.3.2. start event kind <select box>, 4.3.3. Start event timestamp <select box>, 4.3.4.Start event descriptions <text area>, Add new start event (4.3.1-4.3.4) <button>, and 4.3.5. End event name <text box>, 4.3.6. End event kind <select box>, 4.3.7. End event time stamp <select box>, 4.3.8.End event descriptions <text area>, and Add new end event (4.3.5-4.3.8) <button>

Add new connectivity (4) <button>

Activity cycle five: Definition of events

These activities are to elaborate on the previously defined events. In the first partition of this tab, sequential constraints of events are specified. There are two fields for selecting events and for defining their sequential relations (e.g. before, after, coincident, asynchronous). The conditions can be defined in the second partition. The third partition is meant for assigning timestamps to the events.

Form name: 5. Events

5.1. First partition (Sequence constraints): 5.1.1. First event <select box>, 5.1.2. Sequential relation <select box>, 5.1.3. Second event <select box>, Add new sequence constraint (5.1) <button>

5.2. Second partition (Conditions): 5.2.1. Condition description <smart text area>, Add new condition (5.2) <button>

5.3. Third partition (Event timestamp): 5.3.1. Select event <select box>, 5.3.2. Select timestamp <select box>, Add new event-time relation (5.3) <button>

Activity cycle six: Definition of operation parameters

These activities specify the parameters for both streams and operation attributes. The first partition of this form can be used for assigning parameters to streams and/or attributes by selecting them from a select box. The options are invoked from the streams and operation attributes previously defined and stored in the PT database tables. Constraints can be specified on the defined parameters in the second partition of this form.

Form name: 6. Operation parameters

6.1. First partition (O_Parameter): 6.1.1. Select attribute or stream for parametrization <select box>, 6.2.1. Parameter name <text box>, 6.1.2. Parameter kind <select box>, 6.1.3. Parameter descriptions <text area>

6.2. Second partition (Constraint): 6.2.1. Parameter Max value <text box>, 6.2.2. Parameter Min value <text box>, 6.2.3. Parameter unit <select box>, 6.2.4. constraint descriptions <text area>, Add new operation constraint (6.2) <button>

Add new operation parameter (6) <button>

Activity cycle seven: Definition of operation parameters

These activities deal with architectural issues. The first partition is allocated to specify the states of the domain of the given phenotype. The domain-state keys generated here will be used in the following architecture specification forms. Super-domains and domain entities can be defined in the second and third partitions, respectively. The chunks of information included in this form are mostly inherited (i.e. imported) from the chosen genotype.

Form name: 7. Domain-state

7.0. First partition: 7.0.1. Select state key <select box>, 7.0.2. CAD assembly file <brows file button>

7.1. Second partition (Possible super domain): 7.1.1. Super domain name <text box>, 7.1.2. Super domain kind <select box>, 7.1.3. Super domain descriptions <text area>, Add new s-domain (7.1) <button>

7.2Third partition (Domain entity): 7.2.1. Entity name <text box>, 7.2.2. Entity kind <select box>, 7.1.3. Entity descriptions <text area>, 7.1.4.Select respective UoO <multiple select box>, Add new domain entity (7.2) <button>

Add new domain state (7) <button>

Activity cycle eight: Definition of morphology and attributes

These activities define the morphological and values of the attributive parameters. The cycle of activities begins with selecting domain-state or entity-state for which the options are invoked from the PT database. There is a possibility of browsing 3D-model file instead of defining manually. A translator engine is adopted to draw morphological elements from the browsed file and import them to the database.

Form name: 8. Morphology and attributes

8.0. First partition: 8.0.1. Select domain-state <select box>, 8.0.2. Select entity <optional select box>

8.1. Second partition (Morphology): 8.1.1. Morphology kind <select box>, 8.1.2. Morphological object name <text box>, 8.1.3. Morphological object descriptions <text area>, 8.1.4. object CAD file <brows file button>, 8.1.5. Morphological element <select box>, 8.1.6. morphological element name <text box>, 8.1.7. morphological element descriptions <text area>, 8.1.8. Morphological parameter <select box>, 8.1.9. Morphological parameter value <text box>, 8.1.10. M-parameter unit <select box>, 8.1.11. M-parameter value min value <text box>, 8.1.12. M-parameter value max <text box>, Add new M-parameter (8.1.8-8.1.12) <button>, Add new M-element (8.1.5-8.1.12) <button>, Add new M-object (8.2) <button>

8.2. Second partition (Domain attributes):8.2.1. Attribute name <text box>, 8.2.2. Attribute kind <select box>, 8.2.3. Attribute descriptions <text area>, 8.2.4. Attr-parameter kind <select box>, 8.2.2. Attr-parameter name <text box>, 8.2.6. Attr-parameter unit <select box>, 8.2.7. Attr-parameter value <text box>, 8.2.8. Attr-parameter min value <text box>, 8.2.9. A-parameter max value <text box>, Add new Attr-parameter (8.2.4-8.2.9) <button>, Add new attribute (8.2) <button>

Activity cycle nine: Definition of contracts

These activities specify the domain contracts. The cycle starts with selecting domain-state in the first partition. The kind of contracts can be specified in the second partition. The first two partitions are fields with default values inherited from the chosen GT. The third and the fourth partitions are allocated to the specification of more details about parametric contracts and protocol contracts, respectively.

Form name: 9. Contracts

9.0. First partition: 9.0.1. Select domain-state <select box>

9.1. Second partition (Domain contracts): 9.1.1.Contract <radio button> options: Input assumption or Output guarantee, 9.1.2. Domain contract kind <select box>, 9.1.3. Respective morph-element <select box>

9.2. third partition (Parametric contract): 9.2.1. Contract parameter kind <select box>, 9.2.2. Contract parameter min value <text box>, 9.2.3. Contract parameter max value <text box>, 9.2.4. Parameter unit <select box>, Add new contract parameter (9.2) <button>

9.3. Forth partition (protocol contract):9.3.1. Protocol kind <select box>, 9.3.3 Protocol descriptions <text area>, Add new protocol (9.3) <button>

Add a domain-state contract (9) <button>

Activity cycle ten: Definition of architectural connectivity

These activities specify the connections among domain entities. The architectural connections specified in the chosen genotype are imported as default values for the first and the second partition of the form. Mates among the connected domain entities could be defined in the third partition. The query made for invoking the options for morphological elements only consider those domain entities that were selected in the first partition. These morphological elements are defined in the eighth cycle.

Form name: 10. Architecture connectivity

10.0. First partition: 10.0.1. Select domain-state <select box>

10.1. Second partition (Connection): 10.1.1. Connection kind <select box>, 10.1.2. Select first entity <select box>, 10.1.3. First connection enabler kind <select box>, 10.1.4. Select second entity <select box>, 10.1.5. Second connection enabler kind <select box>

10.2. Third partition (Morphological parameter mate): 10.2.1. First Morph-element <select box>, 10.2.2. Second Morph-element <select box>, 10.2.3. Mate kind <select box>, 10.2.4. Mate_parameter <select box>, 10.2.5. Mate_parameter value <text box>, 10.2.6. Mate_para_value unit <select box>, Add mate parameter (10.2.4-10.2.6) <button>, Add new mate (10.2) <button>

Add new entity connection (10) <button>

Activity cycle eleven: Definition of methods

These activities establish relationships between the specified parameters and the beforehand defined methods. In the first partition, a method should be selected from the options invoked from the 'PT_method' table of PT database. The elements of the browsed method are automatically recognized and displayed as options in the select box. The second partition is for assigning parameters to the elements of the chosen method. The parameter options can be invoked from all of the architecture and operation parameter tables. For easier sorting, it is possible to apply grouping or automatic filtering of the irrelevant parameters.

Form name: 11. Methods

11.0. First partition: 11.0.1. Select method <select box>, 11.0.2. Method equation <equation area>, 11.0.3. Method algorithm <smart text area>

11.1. Second partition (Assigning method elements): 11.1.1. Method element <select box>, 11.1.2. Preferred Unit <select box>, 11.1.3. Parameter <select box>, Assign and next (11.1) <button>

Save and next method (11) <button>

5.5 DERIVING INSTANCES OF AN SMF

The interfaces of an SMF enable it to connect with other SMFs. Specification of interfaces has a large impact on the composability of components (resembled by SMFs). The interface parameters of two components to be coupled are evaluated, and the parameter values are shared the interfaces of the coupled SMFs. Under this constraint, the phenotype's interfaces only offer opportunity for coupling, but its feasibility should be checked by evaluating the phenotype and the instance parameters.

Before being able to derive instances from a phenotype, the system designer should place them in an operation context. Coupling of interfaces takes place between multiple phenotypes only when their instances are derived. Interfaces can be considered from two aspects, namely operation interfaces and architecture interfaces. Operation interfaces are external streams and their respective events and states. Architecture interfaces are contracts and respective morphological ports. Moreover, the pertinent parameter constraints should be considered in both interfaces (depending on the type of parameters). Accordingly, when a phenotype is used for instancing, it is composed with other phenotypes, and subsequently, its variable parameters receive values. Compatibility issues should be carefully evaluated prior to composing PTs and deriving their instances, and the resultant instances are stored in the model warehouse.

5.5.1 Procedure of SMF-based system modeling

SMFs-based pre-embodiment design can be best characterized by the term *"cognigeering"*, which expresses that its intent is to simultaneously *"cognize"* (obtain information/knowledge about the constituents and the whole of the planned system) and *"engineer"* (plan and execute a systematic realization of the model of the whole system). SMFs determine the way of availing constituents-related information for designers, and the way of generating a system model. In the cognigeering process, system designers synthesize the architecture and functionality of a system in terms of SMFs, and create a model of the system as a composition of multi-disciplinary SMFs.

SMF-based modeling is a computational implementation of our pre-embodiment design methodology developed with a view on the specificities and demands of CPSs. It includes six major activities of model composition: (i) deciding on the components of the model, (ii) defining the spatial positions and orientations of the components, (iii) determining the functional roles of the components, (iv) architectural connection of components, (v) operational connection of components, and (vi) validation of the composition by checking the constraints. This may seem to be a sequential procedure, but the activities (ii) and (iii), and the activities (iv) and (v), respectively, are done concurrently and in an interrelated manner (Figure 5-42). The methodology also supports the mapping of the pre-embodiment design activities onto formal computational procedures. In the mapping process: (i) the involved human

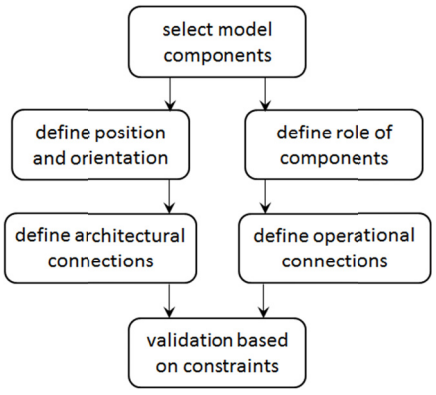


Figure 5-42 Generic workflow of pre-embodiment design

and software actors, (ii) the entry format of the necessary chunks of information, and (iii) the organization scheme of the chunks of information in the warehouse are considered. Based on these factors, an all embracing workflow of SMF-based model composition (an arranged set of computational procedures) has been proposed (Figure 5-43).

Since composition and instantiation of the elements of the system model happens at the same time, there are some confined interactions between the Composer and Instantiator modules in this process. As shown in Figure 43, the Composer module works as a front-end during the model composition process, while the Instantiator module works in the background and support the composer modules with data processing actions. The three horizontal blocks shown in Figure 5-43 include the three activity cycles of model composition. The first activity cycle is concerned with browsing phenotypes from the PT database, and storing them into the model warehouse. The second activity cycle is for establishing operational couplings among the chosen Phenotypes. The third activity cycle is for establishing architectural couplings among the chosen Phenotypes.

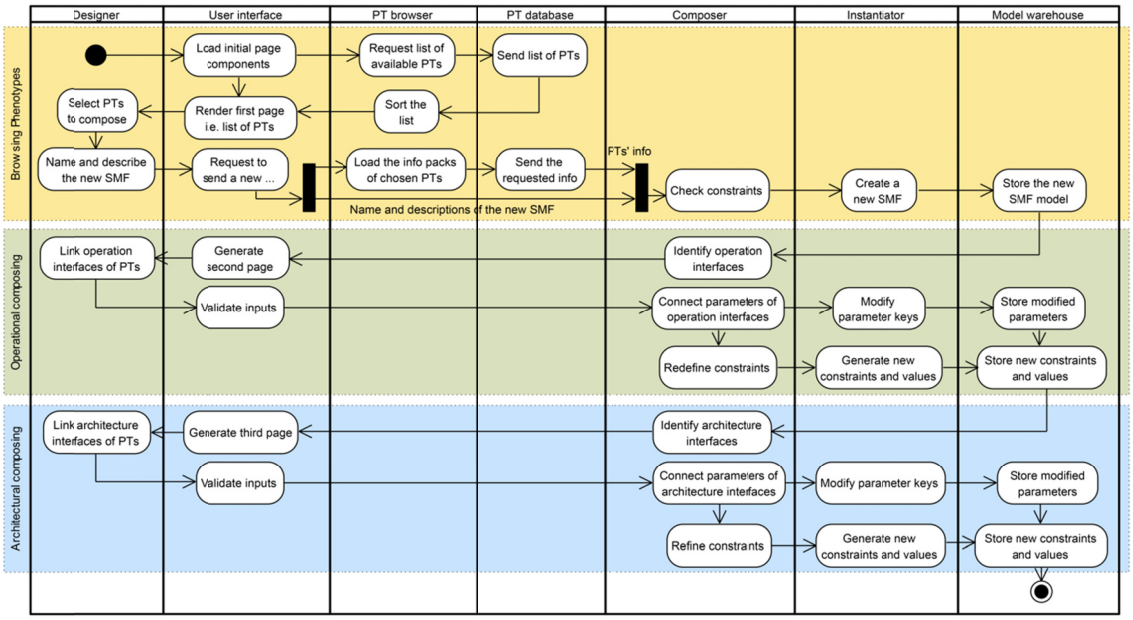


Figure 5-43 Overall workflow of SMF-based model composition

Designers of CPSs can create system models based on the information provided by knowledge engineers, who create and maintain the warehouses of the modeling entities (SMFs) of the SMF-TB [18]. As explained above, composition of system models by SMFs starts with selection of phenotypes by the model creation tools of the Platform part of the SMF-TB. PTs are retrieved from the PT warehouse by the PT browser tool. Instantiation of SMFs is supported by the tools of the Workbench part of the SMF-TB, namely, by the Instantiator, Composer, and Model warehouse manager modules.

The procedure of composition and instantiation of Phenotypes establishes spatial, morphological and operational relationships between selected phenotypes in a pairwise manner. The composition is made possible by the interface specification of the concerned phenotypes. The interface and internal parameters are evaluated so as to make physical composition possible. The modeling software provides the mechanism by means of which

phenotypes can be combined and matching values can be generated for the parameter variables, with the consideration of the specified constraints. The values of the instantiated phenotypes are stored in the model database.

The interfaces of the phenotypes have a decisive role in terms of the possibility of architectural and operational coupling. Architecture interfaces are exemplified by contracts and morphological ports [19]. Operation interfaces provide specifications over external streams, and the related states and events. In addition, parameter constraints are also specified by both interfaces, depending on the types of the parameters. Instances of SMFs take over some chunks of information and their relationships from the relational data tables of phenotypes. Instances share parts of the model database.

5.5.2 Information scheme constructs of the model warehouse

In the previous sub-chapters, the information structure constructs needed to map genotype and phenotype information into relational database schema has been explained. In this sub-chapter we present those additional constructs that are needed for mapping the results of composition and instantiation of phenotypes in the system model. This sub-chapter focuses on these constructs, while in the next sub-chapter those complementing constructs that are needed for a meta-level handling of the system model are presented.

As a first step of discussing the ISCs used for structuring the database of the model warehouse, we present an overview of its external schema in Figure 5-44. The principle of aggregation is the fundamental mechanism that governs data management in the model warehouse. The phenotypes of SMFs imported to the model space are coupled and instantiated in the model space. This changes their architectural and operational relationships. Actually, these relationships should be captured in the database of the model warehouse during coupling and instantiation of phenotypes. From a different viewpoint, aggregation leads to formation of compound SMFs. Together with the simple instance SMFs, the instances of compound SMFs are stored as such in the database of the model warehouse. The aggregation process, i.e. coupling and instantiation of phenotypes of SMFs can be continued without any theoretical limitation. It has to be noted that the current database schemata and storage space requirement are not yet optimized for extremely large system models, and for benefiting from cloud services. On the other hand, 'multi-granularity' of the built SMFs-based models lends itself to an effective composition and decomposition processes.

Phenotypes are placed into architectural and operational relationships when their instances are derived. The ISCs discussed below should capture these established relationships, which do not belong individually to any of the concerned instances, but do belong to both. Coupling of phenotypes means that the parameters related to their interfaces will be coupled and the assigned values will be shared by both. Consequently, all related parameter variables of the concerned phenotypes are evaluated, i.e. values are assigned to the parameters of the created model. In the composition process, the interfaces of phenotypes play two important roles, namely, they are both operation interfaces and architecture interfaces. Operation interfaces handle external streams and the associated events and states. Architecture interfaces handle contracts and respective morphological elements (ports). Constraints on parameter values can be considered in both interfaces (depending on the type of parameters).

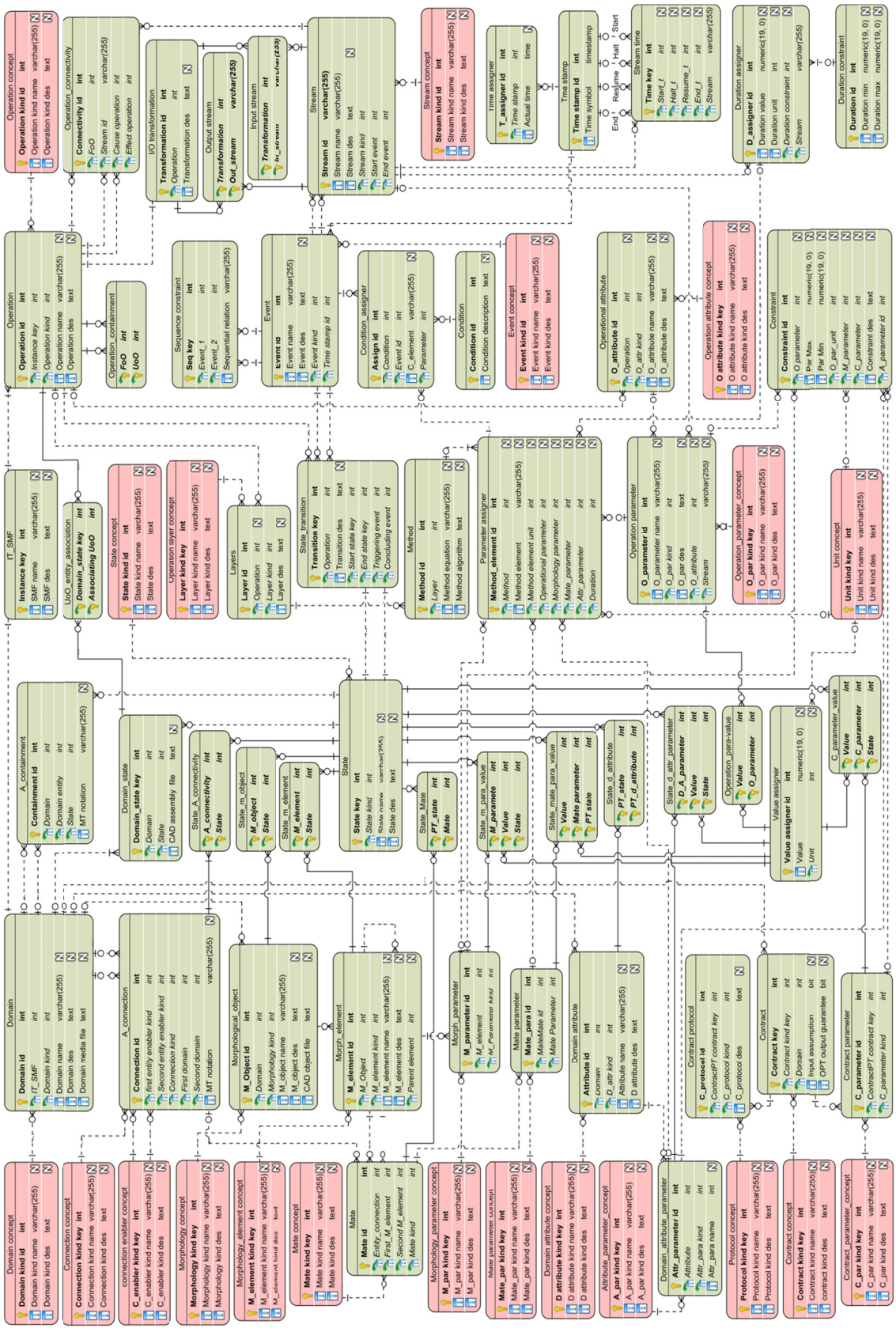
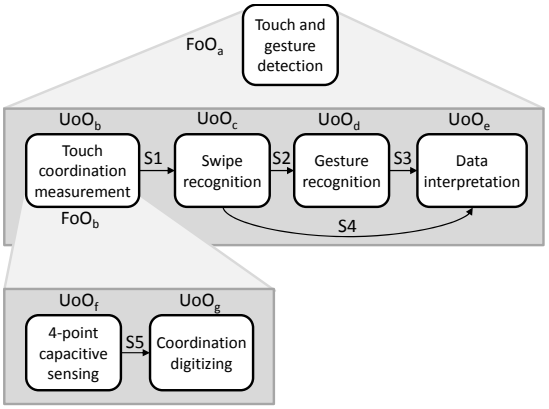


Figure 5-44 Overview of the external schema of the model warehouse

ISC for operation containment and connectivity

The multi-granularity of SMFs makes it necessary to capture aggregation of operations of aggregated phenotypes. Relational tables are included in the database of the model warehouse to capture operation containment. Based on the data stored in the database, multiple UoOs can be represented as a FoO (that can be used as a UoO of the FoO of a higher level operation aggregate.

For example, considering the digitizer module of a smartphone (Figure 5-45), the highest level FoO is *touch and gesture detection*. This FoO can be de-aggregated into four UoOs, namely: (i) touch coordination measurement, (ii) swipe recognition, (iii) gesture recognition, and (iv) data interpretation. We can continue de-aggregation with the new UoOs. Considering the first UoO (touch coordination measurement) as a new FoO, it can be de-aggregated into UoOs namely: (i) four-point capacitive sensing, and (ii) coordination digitizing. The new UoOs can be considered as FoO for the lower level and be de-aggregated more.



Operation connectivity

PK	Stream	Cause	Effect
1	S1	b	c
2	S2	c	d
3	S3	d	e
4	S4	c	e
5	S5	f	g

Operation containment

FoO	UoO
a	b
a	c
a	d
a	e
e	f
e	g

Figure 5-45 Operation containment and connectivity

Though several levels of aggregation should be taken into consideration, we decided to include only one table, which is however able to store all operations without considering their aggregation levels. The level of aggregation (which an operation belongs to) can be reasoned out based on the relation included in the 'Operation_containment' table (Figure 5-46). This table consists of two columns representing FoO and UoO. Each of the rows of this table captures one containment relation. There are two one-to-many relations established between the 'Operation' and the 'Operation_containment' tables. One of these

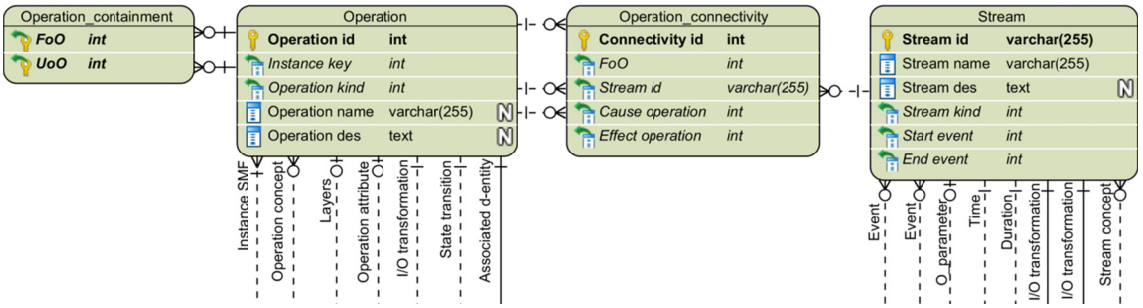


Figure 5-46 Construct for capturing operation containment and connectivity relationships of SMF instances

relations concerns a FoO, and the other indicates one of its UoOs. Operations are physically interrelated by the streams that they manipulate. That is the reason why the table 'Stream' is included in this construct.

Connectivity of operations is captured by the 'Operation_connectivity' table. This table includes five columns: (i) 'Connectivity_id', which is the primary key, (ii) FoO that is formed by the connections of UoOs (iii) 'Stream_id', which is a foreign key imported from the 'Stream' table, and (iv) 'Cause operation', which is a foreign key imported from the 'operation' table, and (v) 'Effect operation', which is another foreign key imported from the 'Operation' table. The 'Cause operation' refers to that UoO, which is a sender of a given stream. The 'Effect operation' refers to that UoO, which is a receiver of a given stream. There are three links established among 'Operation' and 'Operation connectivity' tables. One link allows referring to the 'FoO' key, and two links refer to the keys of UoOs.

ISC for architecture containment and connectivity

In a Euclidean space, the spatial metric of an instance of a SMF is represented by the concerned domain. In the case of a compound instance, the domain is an aggregate of sub-domains (specific domain elements defined in the phenotype of an SMF). When a system model of a CPS is developed, domains can be subjects of multiple aggregation or de-aggregation. The results of this compositional process can be captured in the database as a sequence of containment relations. Consequently, for the purpose of capturing a list of domains the 'Domain' table has been introduced. For capturing the list of containment relations among domains the 'A_containment' table has been specified. They are necessary, but together are also sufficient, to describe the architectural aggregation and de-aggregation hierarchy of a system (Figure 5-47). The 'A_containment' table includes five columns: (i) 'Containment_id', which is the primary key, (ii) 'Domain', which is the foreign key imported from the 'Domain' table, (iii) 'Domain_entity', which is another foreign key from the 'Domain' table (referring to one of the sub-domains of the chosen domain), (iv) 'State', which defines the state of operation, in which a containment relation exists, and (v) 'MT_notation', which represents the mereotopological definition of the captured relation.

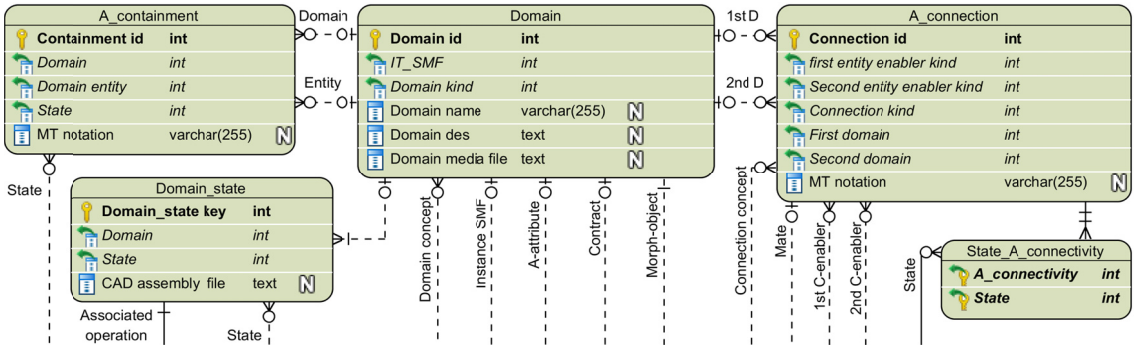


Figure 5-47 Construct for capturing architecture containment and connectivity of SMF instances

Due to the exclusion of self-containment, which is necessitated by the applied strictly physical view, a domain cannot be included in one row of this table as domain and a domain entity concurrently. Moreover, an instance of a component cannot be part of two separate components at the same time. However, it is possible and allowed to be part of several components in various states (time sections). The 'State' column has been intro-

duced because of the need for capturing this situation. It specifies if a 'Domain entity' is part of a 'Domain' in a particular state, and if it is a part of another 'Domain' in another state. If two domains are architecturally connected, they should be registered as 'First domain' and 'Second domain' in one row of the 'A_connection' table. This table is similar to the table included in the architecture containment and connectivity construct of SMF phenotypes.

The usefulness of this construct can be exemplified by the case of transferring a media file from a memory stick (MS) to a hard drive (HD) (Figure 5-48). The operation can be specified as 'moving file', which transfers the start state (file on MS) to the end state (file on HD). In this example, there are two domains (MS and HD) and one domain entity (the media file). The media file is part of both MS and HD, but not in the same state. Figure 5-48 shows how these relations can be captured in the proposed relational tables. The $file_m$ stands for the media file which is moved. The $file_w$ stands for a *Word file* that exists on HD in both states. And the $file_p$ is a Presentation file that is deleted; therefore there is no 'part-of' relation for it in the second state.

Architecture containment

PK	State	Domain	Entity
1	S1	MS	$file_M$
2	S2	HD	$file_M$
3	S1	HD	$file_T$
4	S2	HD	$file_T$
5	S1	HD	$file_P$

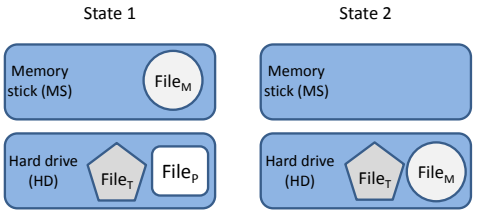


Figure 5-48 An example for architecture containment

ISC for handling assigned metric and attributive values

The major difference between an instance of a SMF and its phenotype is in the explicit and dynamic value specification of instances. In the case of a phenotype, it is sufficient to specify the value range of variables (i.e. maximum and minimum values). Technically, there are two ways of specifying values of architecture and operation parameters: by their (highest and lowest values) constraints, and by their precise values. Typically, constraints are therefore used in specification of phenotypes, and mostly precise values are used for specification of instances. The values of parameter variables may be of two kinds: constant values and variable values. The parameters with constant values may be set to default by the phenotype manager, but cannot be changed by input and output operations. Values of some parameters may change continuously according to physical variations of the related parameters.

The designed construct is shown in Figure 5-49. The main considerations were as follow: Due to interaction with other associated instances, variable values may temporarily change in SMF instances. For example, in the case of an electromotor built into a model, the output values of speed and torque will change according to the input value for the electric current. These dependencies are captured as numerical relations between inputs and outputs, and described by 'methods' stored as mathematical equations. The temporal changes of variable values can be specified either by methods or states of operation. These two ways of capturing the changes support both 'continues-timing' and 'discrete event-based' simulations.

In the process of instantiation, there are two sorts of parameters whose values should be specified: operation-related parameters and architecture-related parameters. Operation-related parameters are: (i) time stamps, (ii) duration, (iii) stream parameters, and (iv) opera-

tion attribute parameters. The values of operation-related parameters can be: (i) defined in interaction with other SMF instances, (ii) calculated based on the mathematical equations stored in the 'Method' table, (iii) specified by the simulation engine, and (iv) inserted manually.

Architecture-related parameters are: (i) morphological parameters, (ii) mate parameters, (iii) architecture attribute parameters, and (iv) contract parameters. All of these parameters (likewise in the case of a phenotype) are linked to the 'State' table. The reason is to provide the possibility of changing the architectural parameters in different states of operation. As an example, let us assume that a given amount of ice is melting due to some heating effect in a CPS. In the start state, we have a solid domain, which can be defined by parameters such as length, wide, and height. In the end state, we have a liquid domain, which can be measured by volume. These types of state changes may frequently occur in cyber-physical systems, which deeply penetrate into real life physical, chemical, biological and technical processes. Therefore, from computational and data management perspectives, architecture parameters and their values should be defined based on the associated states (of the components or whole of the system at hand). In the organization of the model database, we have to take into consideration that architecture parameters may be different in start states and end states. This also applies to the descriptive architectural parameters of domains. Revisiting the abovementioned example, the values assigned to the temperature of the domain attribute should be associated with the start and the end states.

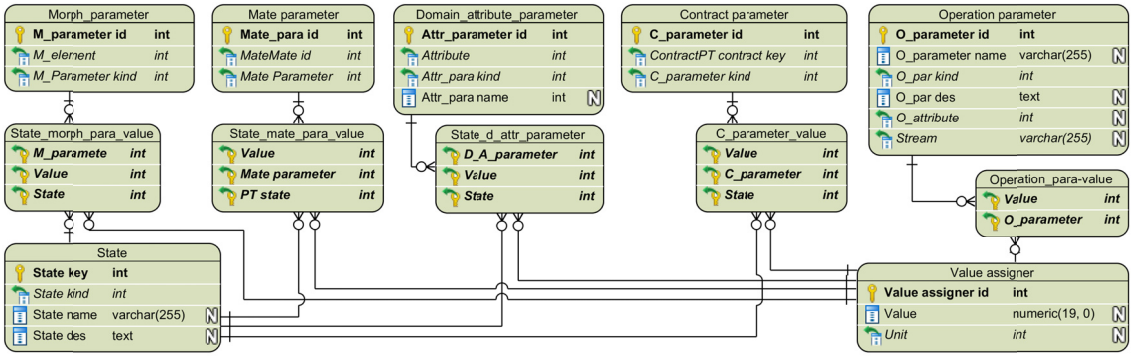


Figure 5-49 Construct for capturing assigned metric and attributive values of SMF instances

In the database of the model warehouse, the 'Value assigner' table is in charge of assigning values to all architecture and operation parameters. This table can be considered as an input gate for values received from the Instantiator module, the Simulation engine, and the User interface. It has a many-to-many relation to: (i) the morphology parameter table, (ii) the mate parameter table, (iii) the architecture attribute parameter table, (iv) the contract parameter table, and (v) the operation parameter table. Five associated tables are allocated to capture the many-to-many relations. The association tables of architectural parameters accommodate another column to assign the states. The foreign key of 'State' is imported from the 'State' table.

ISC for handling values assigned to duration and point of time parameters

The designed construct is shown in Figure 5-50. As one approach, temporal and event values are partly handled by the 'Sequence_constraint' table. This table captures transposition and chronological relations among events. The states are linked to events, and events

are defined based on temporal specifications. Some temporal constraints can be defined by the link between the 'Event' and the 'Time stamp' tables. Another way of setting the temporal values is to specify actual time of timestamps. The actual time specifications are handled by the Simulation engine.

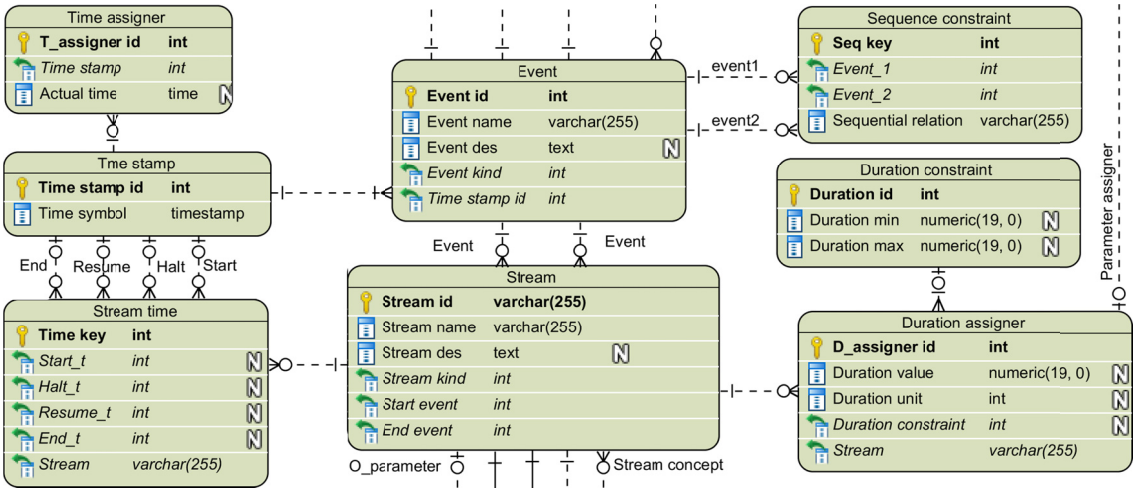


Figure 5-50 Construct for capturing values assigned to duration and point of time parameters

The 'Time_assigner' table is created and linked to 'Time_stamp' in order to support assignment of values to temporal parameters. The 'Time_assigner' table is considered as a gatekeeper for specifying the time stamp values by the Simulation engine. In order to specify the accurate duration of an operation, the 'Duration assigner' table has been created and linked to the 'Parameter assigner' table. The values of this table are calculated based on the actual time of the start, end, and halt and resume events. The 'Duration constraint' table supports the process of checking the composability of temporal constraints.

5.5.3 Information schema constructs of the meta-level knowledgebase

The model warehouse module also includes the computational mechanisms for meta-level management of SMF instances-based system models in its database. The warehouse keeps track of and records (i) from which genotype the instantiated phenotypes have been derived, (ii) which phenotypes have been composed and instantiated as parts of the model, (iii) what interfaces of the instantiated phenotypes are coupled, and (iv) the history of assigning values to their parameters. As a first step of discussing the ISCs used for structuring the meta-level model database of the model warehouse, we present an overview of its external schema in Figure 5-51. It shows the logic of the connections among the relational tables. The primary and foreign keys of these tables mostly refer to the relational tables of the model database. Maintaining these chunks of information facilitates addition and/or subtraction of SMFs, and any in-process partial modification of the model.

Creation of a (SMF instances-based) system model involves coupling, instantiation, and composition of chosen phenotypes. In order to couple phenotypes operationally, their interfaces (i.e. external streams and their associated events and states) should be connected. This process is called unification. Depending on the subject, various unification mechanisms have been defined (e.g. stream unification, event unification, and state unification).

They imply specific database operations. For instance, In order to describe complex operation processes, the end event of the first operation should be unified with the start event of the next operation. In association with this, information about mating of the phenotypes in the architectural aggregation process should be recorded.

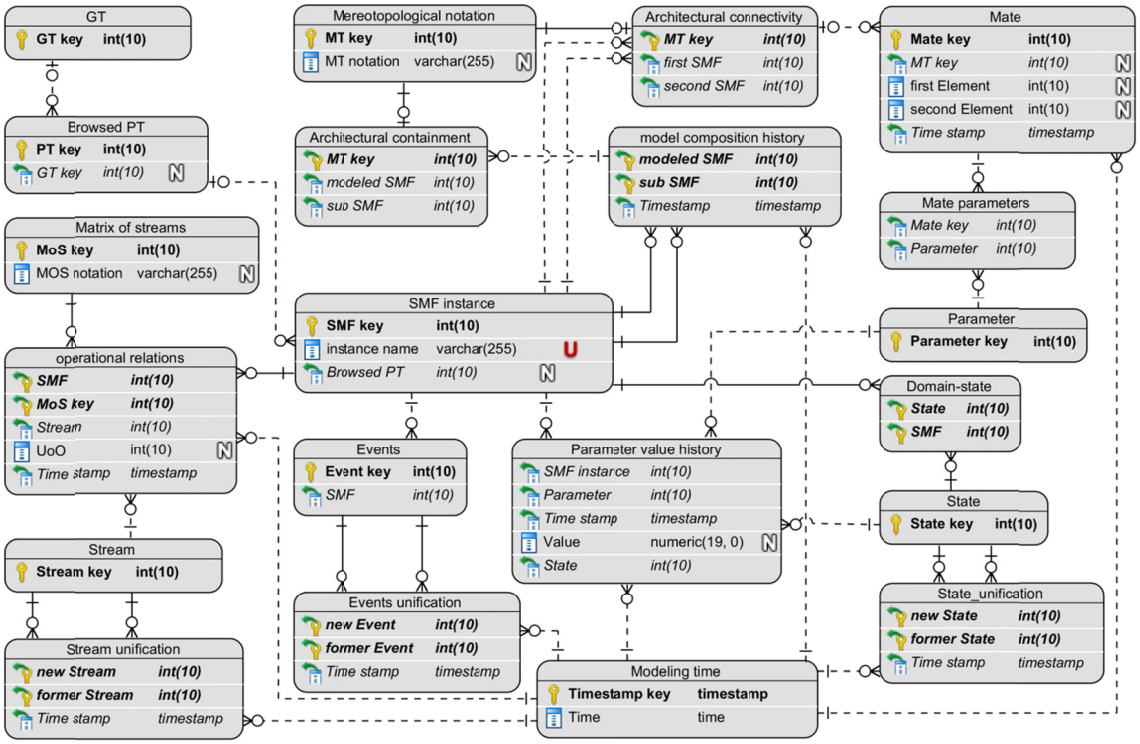


Figure 5-51 Overview of the meta-level external schema of the model warehouse

ISC for recording the origin of the phenotypes and handling the registry of SMFs

Figure 5-52 shows the construct designed for recording the phenotypes included in the system model. Typically, phenotypes are derived from genotypes. As an alternative of this, phenotypes that are stored in the phenotype warehouse can also be used for composing parts of or complete system models. The parts of system models reusable as 'macros' are considered 'compound' SMF instances. In the case of compound SMF instances, there is no need to record the identifier of the parent genotypes, since this information can be retrieved from the database of the concerned phenotypes. Therefore, in this case, the 'GT key' in the 'browsed PT' table is 'unnullable'. Another consideration is that, by alternative structural parameterization, multiple SMF phenotypes can be derived from a particular genotype and multiple SMF instances can be derived from one phenotype. That explains why there are one-to-many relations among the relational tables shown in Figure 5-52.

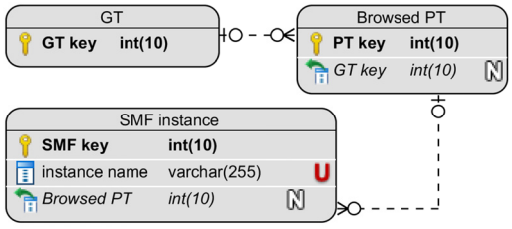


Figure 5-52 Construct for recording the origin of the phenotypes included in the system model

ISC for recording the composition and parameterization history

Figure 5-53 shows the construct designed for recording the history of phenotype composition and parameterization. Actually, this is the main construct of supporting meta-level model database management. It consists of two history recording tables. The 'Model composition history' table stores the information about the composition of SMFs during the modeling process, and the 'Parameter value history' table captures the changes of the values of the various parameters in the course of model composition.

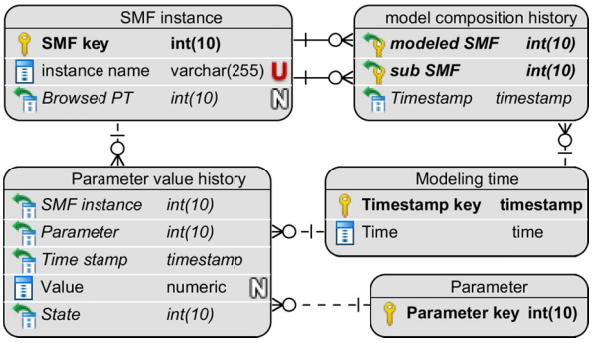


Figure 5-53 Construct designed for recording the history of phenotype composition and parameterization

There are two groups of SMF instances stored in the 'SMF instance' table. The first group includes those instances, which are derived by instantiation of browsed phenotype. Therefore, the 'browsed PT' keys are specified for this group. The second group consists of the compound SMFs that are created by composing several phenotypes in the model database. The 'Model composition history' table captures the relations among these two implements of SMFs. The 'modeled SMF' key refers to the newly generated SMFs and the 'sub SMF' key refers to the instantiated Phenotypes. This table also captures the timestamp of composition and consequently the sequence of composing SMF instances into a model. For the reason that a compound SMF can be treated computationally as a novel SMF, they have been identified in the construct as 'new SMFs'. The value of 'browsed PT' is null for the new SMFs.

The composition of SMFs changes their parameter values and attributes. Some of the parameters should be redefined, while values of some others should be updated according to the results of model composition. For example, the center of gravity of a component may change due to building in additional components into the model, or the fuel consumption changes the weight in airplanes when its behavior is simulated. In the first example, the parameter changes should be captured according to the modeling timestamp, while in the second example, the changes of parameters should be captured according to the simulated operational states. The 'Parameter value history' table has been included in the construct captures all pieces of information required for tracking these changes in the model. The parameters shared among the instantiated Phenotypes and the new SMFs are captured by a many-to-many relation between the 'SMF instance' and the 'Parameter' tables. This relation is realized by the 'Parameter value history' association table, which accommodates timestamps in order to track history of changes.

ISC for recording state, event and stream unification history

A phenotype can specify two types of internal and external states. The internal states are those states that are produced by UoOs, whereas the external states are those states that are produced by FoOs. External states are 'observable' from outside and are regarded as part of interfaces of phenotypes. When multiple phenotypes are coupled in a model composition process, the associated FoOs are considered as UoOs associated with the

compound SMF. Therefore, some of the external states may be turned into internal states. Just as an example, if FoO_1 of PT_1 is also associated with PT_2 as FoO_2, then the end-state (S^e_1) of FoO_1 is considered as the start-state (S^s_2) of FoO_2. In the model composition process these two states are unified into a mid-state (S^m_3) of the created compound SMF (new SMF).

The construct designed to capture the records of state unification and the related timestamps is shown in Figure 5-54. This construct facilitates the ordered storage of all states, including the internal and external states of instances of phenotypes, as well as the states newly created as the results of state unifications are captured in the model database. However, primary keys of the former external states of the instantiated phenotypes and the newly created states are mirrored in the 'State' table of the meta-level knowledgebase. The records of state unifications and their respective timestamps are accommodated in the 'State_unification' table. The states are linked to their SMFs through an association table named as 'Domain-state'.

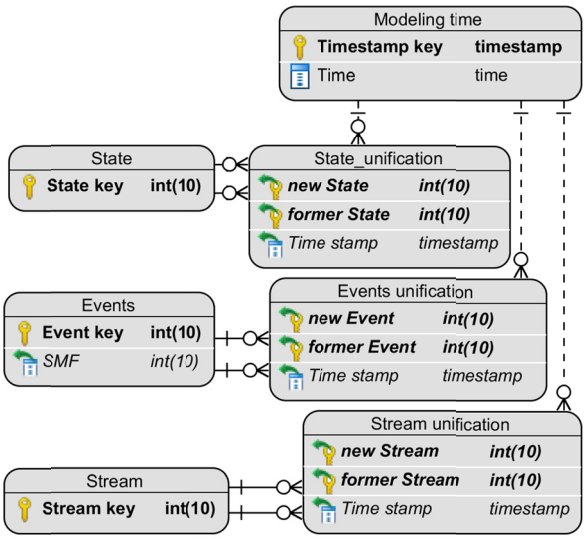


Figure 5-54 Construct for arranging relational tables for recording state, event and stream unification history

When a start-event and an end-event are unified, a new middle-event are created and related to both coupled SMFs. Accordingly, the former start and end events are no longer related to the coupled SMFs. However, the information about the relations of the former events is needed for modification of the model. Consequently, the history of event unification should be captured. The construct that captures event unification history is similar to the construct of state unification. If the output stream of one PT is similar to the input stream of the other one, they can be operationally coupled. For composition, the two external (input and output) streams should be unified. The stream unification replaces the external streams with an internal stream in the database. Capturing the history of stream unification facilitates separation of SMFs, which is needed for model modification.

The construct named as 'Stream unification history' resembles the other constructs related to interfaces of SMFs. The main difference is that the streams are indirectly connected to the SMFs. In fact, the streams are connected to SMFs through an association table named as 'Operational relations', which is a part of the constructs presented in the next paragraph. It is worth mentioning that the values and parameters related to the constraints of streams may need to be changed due to the outcome of SMF instance composition. These changes are captured by the 'Composition and parameterization history' construct, which has been introduced and explained above.

ISC for recording operation containment and connectivity on meta-level

As it explained in Chapter 4, the concept of matrix of streams (MoS) is used as a means for capturing operation relations. A MoS specifies operational relations within a FoO by

capturing its UoOs and the streams connecting them. Considering FoOs as operations of the modeled CPS, UoOs are operations of instances of SMFs composed into the model. The streams (which are connecting these instances) are the newly created streams (as a result of stream unification).

This construct is shown in Figure 5-55. MoSs are stored in the 'Matrix of streams' table. The streams and UoOs are defined by the association table named as 'operational relations'. This table accommodates imported SMF and MoS keys as compound primary keys. The stream key is imported from the 'stream' table. Moreover, the 'time stamp' key that is imported from the 'modeling time' table specifies the sequence of model composition. In the case of model modification, the time stamp also helps to identify the records in multiple tables that are related to one model composition action.

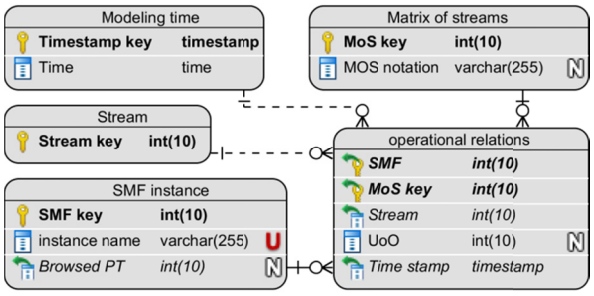


Figure 5-55 Construct for capturing operational relations in the composed system model

ISC for recording architectural containment and connectivity on meta-level

This construct is shown in Figure 5-56. The architectural relations are captured by mereotopological notations. These notations are stored in the 'Mereotopological notation' table. The elements of the notations are the SMFs that are stored in the 'SMF instance table'. The containment notations are linked to SMFs through the 'Architectural containment' table. The connectivity notations are linked to SMFs through the 'Architectural connectivity' table.

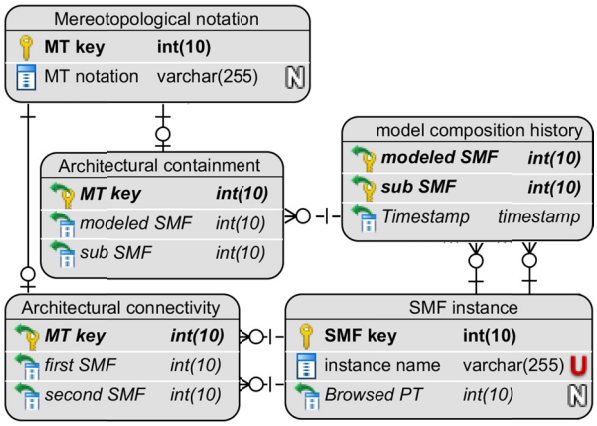


Figure 5-56 Construct for capturing architectural relations in the composed system model

ISC for capturing mating elements and history of mates

This construct is shown in Figure 5-57. Recording the mate history is needed for tracking the physically-driven aggregation of SMFs, as well as the modification of the model whenever instances of SMFs are added to or subtracted from the model. Another benefit of having the 'mate history' construct is that it provides opportunity for calculating the mating constraints. For example, in a practical pre-embodiment design case, the tables of this construct may keep a record of the occupation of the hard disk space by the installed software applications, or the space remained on a surface after connecting some components.

Mates are associated with the architectural connectivity relations. Specifically, each of the architectural connectivity relations may be described by more than one mate. That is the reason why one-to-many relation is established between the 'Architectural connectivity' and 'Mate' tables. The 'Mate' table accommodates keys of: (i) two morphological elements imported from the model database, (ii) the timestamp when mate was created in modeling, and (iii) the mereotopological relations of them. The mate parameters are captured by an association table that supports many-to-many relations among the 'Mate' and the 'Parameter' tables.

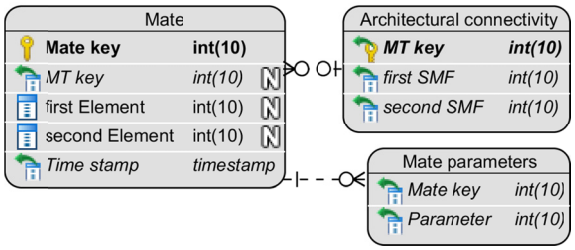


Figure 5-57 Construct for capturing mating elements and history of mates

Although the precise value of parameters should be defined for instances of SMFs, the constraints imported from Phenotypes should be kept and updated. These constraints are needed for control of value assignment (e.g. during simulation) and analysis of constrains satisfaction [20]. The Composer module of the SMF-TB is responsible for checking and validating all constraints before and during composing instances into a system model, as well as for updating them after composing. However, in order to point out weak points or bottlenecks, the total performance analysis of the CPS might need redundant constraint analysis. Additionally, the Simulation engine is in charge of calculation of the values that should be specified. Sometimes, the purpose of simulation is to understand the situations that some constraints are violated. The resultant operational and architectural data are assigned to the model by the Instantiator module. When, the model is updated, multiple relational tables are updated concurrently.

5.6 ISSUES OF COMPOSING MODELS FROM SMF INSTANCES

While operations of the new SMF are defined, there might be similar operations of the phenotypes that could be merged together. For example, several phenotypes that all have 'conducting electricity' as one of their operations, might be used to create 'conducting electricity' as one of the operations of the new SMF. The main difference between these two operations of 'conducting electricity' is in their domains of operation. This issue can be easily captured by our models, just by specifying domains associating in a particular operation. Therefore, there is no need for extra actions.

Another issue refers back to the multi-layered operations that are enabled in our modeling tool. There are some operations that affect surrounding domains as well (e.g. heating, vibration). These effects could be formulated as relations between domain parameters and stream parameters. All relations could be defined and stored as methods. Dealing with these post-modeling specifications is postponed to simulation. Both browsing and searching of SMFs require classification and sorting them according to many aspects. Information about kernel and interfaces of SMFs might be used as criteria for searching or browsing SMFs in order to select the best options for modifying or composing a CPS. The most important issues that searching could run based on are listed below:

- **Input/output of SMFs:** Based on I/O streams, events, and states, CPS designers could find out what they need for a particular operation composition. If a component is replaced, the new component should be connected operationally to the streams the former detached from.
- **Morphology and size:** Physical positioning and dimensional limitations of a component could be a serious issue sometime. For example, if an App should be installed for navigation, the size of the App should be considered regarding the available storage space on the disc.
- **Efficiency (performance):** The methods of operations stored in SMFs could be used as searching criteria in order to find SMFs with a particular performance.
- **Constraints:** Many architectural and operational parameters might be defined by their constraints (e.g. flexion, temporal, volume), and the constraints could be used for searching.
- **Similarity (kind):** Instances of SMFs could be traced back to their phenotypes, genotypes, and genotype templates. Accordingly, SMFs could be search based on their architectural and operational similarities. For example, 'photovoltaic' energy producing could be used as kind search.

Following, we elaborate on the major issues of composing models from SMF instances.

5.6.1 Collecting chunks of information by the entry forms

The entry form makes the modeling process interactive and conversational. There are three entry forms designed for collecting the required chunks of information from the system designers, each of which realized in a separate tab of entry form. These entry forms successively collect chunks of information concerning: (i) list of phenotypes that are composed, (ii) operational composition of the selected phenotypes, and (iii) architectural composition of the selected phenotypes.

In the first entry form, CPS designers need to select the phenotypes that will be used in the system model. This model is considered as a very higher level SMF, which can be used to compose a system of systems (SoS) model requiring multi-level composition. Name and description of any new or SoS instance of SMFs should be included into the first partition of the form. In the second partition, the designers will be asked to provide a name and description for the architectural domain of the concerned SMF. There is an option for uploading a media file. In the third partition, name and descriptions of operations of the SMF should be determined. Since there might be several operations for an SMF, for each operation, associating UoOs of phenotypes should be selected.

For example, for composing a new SMF of a smartphone, phenotypes of battery, screen, camera, processor, etc., should be instantiated. Multiple operations could be defined for the new SMF e.g. navigation, gaming, video calling, photo shooting, music playing. For all operations of the new SMF, all operations of the aggregates are not needed. For example, navigating (as an operation of smartphone SMF instance) will not need visual sensing (as and operation of camera phenotype). Accordingly, in the third partition of the first page, operations of the new SMF (FoOs) could be defined by selecting multiple operations

(UoOs) imported from the chosen phenotypes. The following declarations represent the fields of the first instance form.

Form name: 1. Composition basics

1.0. First partition: 1.0.1. Composed SMF name <text box>, 1.0.2. Composed SMF descriptions <text area>, 1.0.3. Select phenotypes to compose <select box, add button>

1.1. Second partition (Architecture of composed SMF): 1.1.1. Composed domain name <text box>, 1.1.2. Composed domain descriptions <text area>, 1.1.4. Composed domain Media file <brows file button>

1.2. Third partition (Operation of composed SMF): 1.2.1. Composed operation name <text box>, 1.2.2. Composed operation descriptions <text area>, 1.2.3. Select operations of phenotypes to compose <select box, add button>,

Add new operation (1.2) <button>

The second entry form is created for configuring operations of the model. It means, each operation should be defined as state transitions (i.e. as an input/output transformation). In the second tab of the form, the firs partition is allocated to select an operation (among the operations defined in the first tab). The state transition should be defined in the second partition. There is no need for creating states; they could be selected from the states formerly defined in phenotypes. The I/O transformation should be defined in the third partition. The streams represented as options in this partition are invoked from external streams of the chosen phenotypes. In this way, the interfaces of the phenotypes that remain as interfaces of the new SMF could be specified. The next partition specifies the external streams of the phenotypes that should be turned into internal streams in the new SMF. In order to couple the operation interfaces of the chosen phenotypes, the external streams of them should be linked together. We named this action as stream unification. It means that FoOs of the phenotypes (which are considered here as UoOs) should be linked together as procedural elements of FoOs of the new SMF.

Sequential constraints of the events are already defined in the phenotypes. However, after having composed them, we need to define sequential constraints among the events in various phenotypes in order to instantiate them. For instance if end state of one phenotype is the start state of the other one, their respective events should be defined as coincident. In the fifth partition, the sequential relations of the events will be specified. The same is applies to defining conditions: there may be a need to define some additional conditions that regulate the operational relationship among phenotypes. This kind of conditions can be defined in the sixth partition. The events needed to be selected in partitions five and six, should be invoked from the database. It means, there is no need to generate a new event. The reason is that all external events are formerly created in phenotypes.

In the second tab of the form, the procedure of the operations of the newly composed SMF will be defined by composer module. Methods of operation will be derived automatically in the background by the *Instantiator* module. The following declarations represent the fields of the second instance form.

Form name: 2. Operational composition

2.0. First partition: 2.0.1 Select operation <select box>

2.1. Second box (State transition):2.1.1. Start state <select box>, 2.1.2. End state <select box>, 2.1.2. Transition descriptions <text area>, Add new state transition (2.1) <button>

2.2. Third partition (Input/output transformation): 2.2.1. Input streams <select box, add button>, 2.2.2. Output streams <select box, add button>, 2.2.3. Transformation descriptions <text area>

2.3. Fourth partition (Stream unification): 2.3.1. Stream <select box, add button>, is the same as <text>, 2.3.2. Stream <select box, add button>

2.4. Fifth partition (Event sequence constraints): 2.4.1. Events <select box, add button>, 2.4.2. Sequential relation <select box>, 2.4.3. Events <select box, add button, Add new sequence constraint (2.4) <button>

2.5. Sixth partition (Conditions): 2.5.1. Condition description <smart text area>, Add new condition (2.5) <button>

New composed operation (2) <button>

Following the specification of the state transitions in the second tab, the first partition in the third tab (architectural composition) is to select the state that the architectural relations should be specified in. The other partitions are allocated to specify architecture containment (partition 2), architecture connectivity (partition 3), and morphological mates (partition 4).

Since the included domain entities and the connections among them are specified in association with states, the changes in architectural domain of the new SMF should be defined in this tab. The domain entities of the new SMF are the domains of the chosen phenotypes that are being instantiated here. The contract defined in the phenotype step could be used for defining connections in this step. The architectural composition of the new SMF is the result of this configuration and instantiation action completed by the CPS designer. After specifying the architectural connections, morphological mates should be specified. This partition is more or less similar to the partition allocated to morphological mates in the phenotype derivation step. The following declarations represent the fields of the third instance form.

Form name: 3. Architectural composition

3.0. First partition: 3.0.1. Select state key <select box>, 3.0.2. CAD assembly file <brows file button>

3.1. Second partition (Architecture containment): 3.1.1. Contained domain <select box, add button>

3.2. Third partition (Architecture connection): 3.2.1. Connection kind <select box>, 3.2.2. Select first domain <select box>, 3.2.3. First connection enabler kind <select box>, 3.2.4. Select second domain <select box>, 3.2.5. Second connection enabler kind <select box>

3.3. Fourth partition (Morphological mate): 3.3.1. First Morph-element <select box>, 3.3.2. Second Morph-element <select box>, 3.3.3. Mate kind <select box>, 3.3.4. Mate-parameter <select box>, 3.3.5. Mate_parameter value <text box>, 3.3.6. Mate_para_value unit <select box>, Add mate parameter (3.3.4-3.3.6) <button>, Add new mate (3.3) <button>

Add new connection (3.2) <button>

Add new domain state (3) <button>

5.6.2 Information processing by the Composer and Instantiator modules

As it is discussed before and indicated in Figure 5-58, structure of the two databases are almost similar. Accommodating all level of operations in the same table (table 'operation'), as well as accommodating all level of domains in the same table (table 'domain') resulted in removing four tables. On the other hand, three tables are added to the model database that does not exist in the phenotype database. The tables 'duration assigner' and 'time assigner' are in charge of assigning temporal values to the model variables. The table 'operation para value' connects 'value assigner' table to the 'operation parameter' table. Value assigner is the gate for values should be assigned to all kind of parameters.

Operation tables		Architecture tables	
Phenotype database	**Model database**	**Phenotype database**	**Model database**
PT_SMF	IT SMF	PT domain	Domain
PT operation	Operation	PT possible super domain	Domain
PT possible super operation	Operation	PT entity	Domain
PT UoO	Operation	PT entity connection	A connection
PT operation containment	Operation containment	PT DS A connectivity	State A connectivity
PT operation connectivity	Operation connectivity	PT DS A containment	A containment
		PT UoO PT entity	UoO-entity association
PT I/O transformation	I/O transformation	PT domain-state	Domain-state
PT output stream	Output stream		
PT input stream	Input stream	PT morphological object	Morphological object
PT stream	Stream	PT state morph object	State m object
		PT morph element	Morph element
PT event	Event	PT state morph element	State m element
PT sequence constraint	Sequence constraint	PT morphology parameter	Morph parameter
PT condition	Condition	PT m para value	State m para value
PT condition assigner	Condition assigner		
PT state transition	State transition	PT mate	Mate
PT state	State	PT state mate	State mate
		PT mate parameter	Mate parameter
PT time stamp	Time stamp	PT mate para value	State mate para value
PT stream time	Stream time		
PT duration constraint	Duration constraint	PT domain attribute	Domain attribute
	Duration assigner	PT state attribute	State d attribute
	Time assigner	PT domain attr. parameter	Domain attribute parameter
		PT d attr. para value	State d attr parameter
PT operation layer	Operation layer	Value	Value assigner
PT method	Method		
PT parameter assigner	Parameter assigner	PT contract	Contract
PT operation parameter	Operation parameter	Contract protocol	Contract protocol
PT constraint	Constraint	Contract parameter	Contract parameter
PT operational attribute	Operational attribute	PT DS contract	C parameter value
	Operation para value		

Figure 5-58 projecting tables of phenotype database to model database

Modeling process of a CPS is considered as an ordered set of systematic procedures that several actors are involved in. Accordingly, this procedure can be investigated in many aspects. The user workflow, interaction among modules of modeling tool, and information

processing are examples of these aspects, which are in focus of our research. Obviously, there are multiple aspects that are excluded from our research, in order to be able to reduce it to a manageable amount of work.

In order to capture the procedure of model composition and instantiation, concept of procedural computational schema (PCS) has been created. PCSs represent algorithmic activity flows that could be implemented through programming languages. In the following, the PCSs of generating SMF-based models are demonstrated. The Composer and Instantiator modules are considered, which are main modules directly involved in system modeling. Consequently, we mainly focus on the activities of these two modules. The three steps represented in Figure 5-43 will be revisited and referred to in below elaboration of the process of modeling. The process starts with copying content of the relational tables of the chosen phenotypes from the phenotype database to the model database. This information transfer is performed according to the mapping table presented in Figure 5-58.

First step of instantiation

In the first step of instantiation, the information sets of the chosen phenotypes should be copied to the model warehouse. A new SMF also should be created and stored in the same relational tables. The containment and connectivity relations (that should be specified in the next steps) will connect the new SMF to the copied phenotypes. The only connection between the new SMF and the copied phenotypes are stored in the 'operation containment' table (as it is specified in the first step). The following declarations represent the procedure construct proposed to be employed by the Composer and the Instantiator in the first step.

PCS 1: Check constraints (actor: Composer)

- Start with a newly specified operation (FoO)
- Extract the operations (UoOs) of the chosen phenotypes that composed for delivering the new operation (FoO)
- Compare the 'operation kind' (specified for the new operation) with list of the 'operation kind' defined for the possible super operations of the chosen phenotypes
- Notify the designer about the mismatches
- Receive approves or changes
- If changes applied, repeat the comparison
- Create operation containment relations
- Record the relations in the meta-level knowledgebase of the model warehouse

PCS 2: Create a new SMF (actor: Instantiator)

- Receive the name and descriptions of the new SMF (and its domain), from the entry form and store it in the model warehouse
- Receive the data sets of the chosen phenotypes and store them in the respective table of the model database
- Receive the operation containment relations among the new SMF and the chosen phenotypes, made by Composer, and store them in the 'operation containment' table
- Create a report and store it in the meta-level knowledgebase of the model warehouse

Second step of instantiation

In the second step, the Composer identifies 'operation interfaces', and sends them to the user interface. After receiving back the user decision concerning the coupling of the interfaces, the parameters should be merged and constraints should be redefined by the Composer. Subsequently, the Instantiator modifies respective keys of parameters in order to link them to the methods and to the newly defined constraints and values. The generic algorithms used by the Composer and the Instantiator are represented below.

PCS 3: Identify operation interfaces (actor: Composer)

- Group operations of the chosen phenotypes regarding to their participation in the operations defined for the new SMF
- Perform the rest of this process for each group separately
- Identify external streams of the selected operations of the phenotypes
    - Identify the streams associating with 'I/O transformation' table
    - Identify the streams linked to 'UoO connectivity' table that are missing one side of either cause operation or effect operation
    - Group them as external 'input streams' and 'output streams'
- Identify the interface events of the selected operation of the phenotypes
    - Trace the events associating with external streams represented in the 'stream' table as 'start event' and 'end event'
    - Identify the events represented in 'state transition' table as 'triggering event' and 'concluding event'
    - Group them as external 'start events' and 'end events'
- Identify the interface states of the selected operation of the phenotypes
    - Extract list of states associating with the chosen events in the table of 'state transition'
    - Group them as 'start states' and 'end states'
- Create a report and store it in the meta-level knowledgebase of the model warehouse

All of the selected and sorted streams, events, and states should be sent to the interface for being displayed to the user. The user decision will be captured by the entry form. The entered information will be sent back to the Composer to be processed and composed. Streams, events, and states that are identified as 'external' may remain as external stream, events, and states of the new SMF, if they will not be connected to each other. They should be turned into the internal ones if they are connected, unified, or merged together.

PCS 4: Connect operation parameters & redefine constraints (actor: Composer)

- Identify the parameters that should be merged
    - Identify the streams of the phenotypes that are unified (turned to the internal streams)
    - Trace the 'operation parameters' that host the foreign keys of the unified streams
- Consider the identified parameters as merge-able parameters
- Compare the 'parameter kinds' of the merge-bale parameters
- If there is any difference, notify the designer

- Trace the 'constraints' associating with the parameters
- If there is any mismatch in constraints, notify the designer
- Redefine new constraints by combining the constraints of the parameters
- Send the result of parameters composition and new constraints to Instantiator
- Create a report and store it in the meta-level knowledgebase of the model warehouse

PCS 5: Modify operation parameter keys and generate new constraints and values (actor: Instantiator)

- Link the chosen states of the phenotypes to the new 'state transitions' associating with the operations of the new SMF
    - Create new 'state transitions' for the operations of the new SMF (specified by the user)
    - Link the states chosen by the user to the created state transitions
- Link the streams chosen as input and output of the new I/O transformations to the operations of the new SMF
    - Create an I/O transformation for each of the operations defined for the new SMF
    - Link the external streams of the phenotypes (selected by the user to be considered as input and output of the new operations) to the created I/O transformations
- Update names and relations of the unified streams
- Modify relations between timestamps and events
    - Trace the events associated with the unified streams
    - Unify the events through changing their names and relations
    - Assign the respective time stamps to the newly unified events
- Calculate new durations and link them to the streams
- Refine the parameters merged by Composer
- Refine the constraints associating with the merged parameters
- Update the sequential constraints of the events
- Update conditions and generate new ones according to the user input
- Create a report and store it in the meta-level knowledgebase of the model warehouse

Third step of instantiation

The third step is typically about the architectural aspect of instantiation and model composition. For providing contents of the user interface in this step, the Composer should identify the architecture interfaces of the composed phenotypes. The states specified in the previous step will be used for defining the architecture relations. Containment and connectivity relationships are the most important issues regarding to the domain states. On the other hand, contracts should be used as architecture interfaces that provide options for domain connections. The morphological elements that are associated with the contracts will be seen as ports that could provide connection with other domains. The pieces of information filled in by the user will be sent to the Composer for connecting the parameters and refining the respective constraints. Subsequently, Instantiator will create

new database entity and modify the available primary keys. The generic algorithms used by the Composer and the Instantiator are represented below.

PCS 6: Identify architecture interfaces (Composer)

- Identify the list of states regarding to the operations defined for the new SMF
- Identify list of domains respective to the phenotype operations participating in the new operations
- Group and sort the domains according to the associating operations
- Trace the contracts regarding to each of the listed domains
- Identify the complementing contracts in different domains for creating smart suggestions
- Trace the domain elements associating with the contacts
- Create a report and store it in the meta-level knowledgebase of the model warehouse

According to the chunks of information collected by the third entry forms, three group of information will be defined by the user at this step related to (i) architecture containment, (ii) architecture connectivity, and (iii) morphological mate. The 'connection kind' and 'connection enabler kind' that are suggested to the user are derived according to the contracts of the selected domains. The user inputs should be processed by the Composer and the Instantiator according to the below algorithms.

PCS 7: Connect architecture parameters and redefine constraints (actor: Composer)

- Identify the contracts used for creating the connections
- Identify the morphological elements that associate with the connections
- Compare morphological elements associating with the contracts to the ones associating in the mates
- Notify the user if there is any difference
- Trace the new mate parameters and the former contract parameters that are the same
- Check the constraint of the former contract parameters with the value of the new mate parameters
- Notify the user if there is any mismatch
- Create a list of modification and send it to Instantiator
- Derive the list of contract that are not used for connections according to the domain state of the new SMF
- Make decision if the contracts that converted to connections are still available as contracts for the new SMF
- Send the list of available contracts to Instantiator
- Create a report and store it in the meta-level knowledgebase of the model warehouse

PCS 8: Modify architecture parameter keys and generate new constraints and values (actor: Instantiator)

- Assign new architecture containment relations regarding to the states of the new SMF
- Assign new architecture connectivity relations regarding to the states of the new SMF

- Assign new morphological mates to the specified connections
- Replace the former contract parameters with the new mate parameters in 'parameter assigner' table
- Assign newly specified values to mate parameters
- Summarize and send the report to the model warehouse

The process of composing the model of an SMF is done here. However, functions of Composer and Instantiator are not finished. Instantiator should assign values to the variable parameters during simulation and Composer should check all constraints at the same time.

5.6.3 Modifying SMF-based system models

'Modification' is a generic term for all kind of changes needed to be applied manually in the process of or after composing a model. It could be divided into two groups of modifications: (i) parameter and context modification, and (ii) partial modifications of the SMFs instances-based model.

Parameter value and context modification

This involves changing the values of the parameter variables. These parameters belong to a particular instance of a phenotype, composed into the model. For example, if a multifunctional phenotype is instantiated in a model for using only one of its functions, changing in its operating mode could be considered as parameter modification. Or, if a phenotype with some knobs for adjusting the operation or architecture parameters is instantiated in a model, setting up the knobs could also be considered as parameter modifications. Accordingly, parameter modification could be handled both by simulation engine and manually. Simulation of embedded customization could be enabled through parameter modification. The options are embedded in phenotypes as interface events which are linked to various streams and states.

Context modification implies changing of the operating contexts of the modeled CPS [21]. Simulation engine defines contexts of operation through specifying variable parameters that are defined for this purpose (e.g. parameters of the external streams that define input and output of the CPS). The context modification might be done in commination of the simulation engine with the CPS designer. The Instantiator module is the main actor in the course of the above described parameter and context modifications. It captures all applied changes and stores them in the meta-level knowledgebase of the model warehouse. The knowledgebase is accessible by the simulation engine if it is required.

Modification of SMF instances-based models

SMF modification implies Subtracting, adding, or replacing SMF instances composed in a model. As it is explained, for modeling a CPS, SMF phenotypes are instantiated and composed to create higher level SMFs, these SMFs could again composed repeatedly till the desired model of a CPS is made. For re-designing of a CPS, several times might be needed to replace or add SMF instances in the CPS model, and simulate the system to check the performance enhancement. Therefore, SMF modification can be considered as an intrinsic part of CPS design and modeling. Adding a new SMF to the model is considered as an action similar to the model composition, consisting of two phenotypes (the model and the

new phenotype). Replacing a SMF can be seen as firstly subtracting a SMF and then adding and instantiating a new phenotype. Consequently, it can be computationally realized by combining a subtraction mechanism with the procedure of model composition explained before. In the following paragraphs procedure construct of subtraction has been explained.

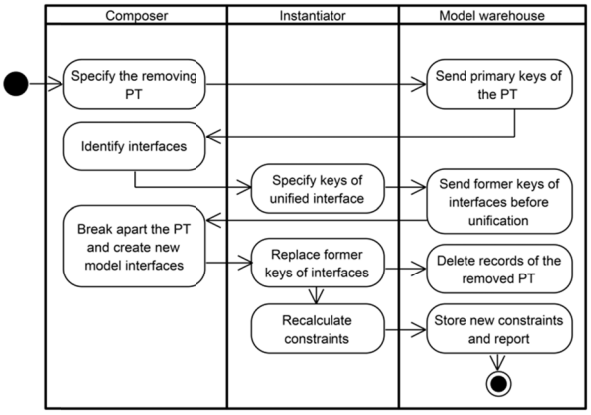


Figure 5-59 Procedural computational schema of phenotype subtraction

This procedural computational schema has three actors namely the *Composer*, the *Instantiator*, and the *Model warehouse* (Figure 5-59). The main challenge in SMF subtraction is separating unified interfaces (i.e. events, streams, states, and mates) of the composed phenotypes that currently turned into internal events and streams, etc. After identifying the unified interfaces, they should be separated and the former interfaces should be rebuilt. Former interfaces imply the interfaces of Phenotypes before composition. The procedure starts with receiving a command from the designer indicating the SMF that should be subtracted from the model. The composer specifies the phenotype composed in the model and asks for it to be removed. List of the related primary keys, which are stored in the meta-level knowledge base of the warehouse are invoked from the 'Events_unification', 'Stream_unification', 'Mate_parameters', and 'State_unification' tables, and sent back them to the composer. The composer is now in charge of identifying the interfaces of the SMF which is being removed. In tight interaction with the composer, the Instantiator specifies the keys of interfaces. These changes are applied in the 'Stream_unification', 'Events_unification', and 'State_unification' tables. Although in the model, these streams and events are internal streams and events currently, they will be turned into external streams and events after subtraction.

The 'Model_composition_history' table and the 'Parameter_value_history' tables contain all required chunks of information about the composed SMFs and their parameters. The architectural and operational subtraction of SMFs can be reasoned out from the information stored in these tables, particularly for subtraction of the SMFs that are earlier composed into the model. The former keys of interfaces are extracted from this knowledgebase and sent the composer. Composer uses this information to break apart the SMFs and create the new model interfaces in cooperation with the Instantiator. The keys are being replaced and all records of the subtracted SMF are removed from the model database.

The last activity of the Instantiator module is to recalculate constraints of the parameters in the model after modification. The constraints are updated in the 'Constraint' table of the model database. A report about the subtraction is stored in the log file of the warehouse for later usage. All modifications stored in the relational tables of the meta-level database are assigned with the timestamp key of the 'Modeling_time' table. This is the modeling timestamp and should not be misunderstood with the timestamp of the modeled CPS (which is stored in the database).

5.6.4 Simulation of the behavior of the modeled CPSs

Simulation involves assigning values to parameters, as well as calculating the outcomes and examining the consequences based on the formulated relations among parameters. The model, loaded into the model space can operate according the values defined by the user or a scenario generator, according to the time factors imposed by the simulation engine. The simulation engine is in a direct interaction with designers to capture the required information. On the other hand, it includes a scenario generator module to automatically generate values for variable parameters in the defined range and according to a particular strategy. Parameters that are manipulated by the simulation engine include all variables related to architecture, operation, timing and conditions. The simulation engine needs to consider three aspects of parameters namely, their (i) values, (ii) constraints, and (iii) relations. These three aspects are captured in the model database. Values can be set through the value assigner, time assigner, and duration assigner units. Constraints of the parameters are captured in 'constraint' table. Relations among parameters are captured in 'method' table regarding to the operation layers.

Simulation of one instance could not be separated from simulation of the whole model, and vice versa, since all parameters of the instances get connected together after composition (by the Composer module). The relations between them are handled by methods. For example, the change in a morphological parameter of an instance will most probably affect a parameter of an output stream of that instance, which is most likely unified with an input stream of another instance, and so forth. Parameters are connected either through methods (as various kinds of internally related parameters), or stream, event, and state unification (as externally related same kind of parameters). This results in a huge network of relationship among parameters in various kinds. Changing one of the parameters will have consequences on other parameters that should be calculated by simulation engine.

CPS designers should have access to the parameters for determining their values. Based on the input and the other default values, simulation demonstrates (i) the consequences of operation, (ii) the constraints that are violated, (iii) the diagnoses that are needed, (iv) the optimizations that are possible, and so forth. Simulation techniques were not placed in the center of this research currently, and the only purpose of this section was to give an overview of the challenges and implementation of simulation. Further investigations and elaborations are needed in this domain of the follow up research.

5.7 CONCLUDING REMARKS

This chapter reported on the development of the computational constructs of the SMF-based modeling toolbox. Targeting the feasibility test of SMF-based modeling (as the main aim of this research cycle), AKF and OKF were taken as information structures for computational implementation. The main objective of the research completed in this research cycle decomposed to three sub-objectives, namely (i) systematically recognizing logical and relations among pieces of information introduced by AKF and OKF, (ii) designing database schema for realizing information structure proposed by AKF and OKF, and (iii) devising workflow algorithms required for developing information processing application.

The objective was accomplished by proposing step-wise process of SMF creation. The main novelty achieved in this research cycle is the proposed three-stage procedure. Based on this: (i) a major part of repetitive works can be prevented by selecting options from a default value list, (ii) the workload of manual entering of the data can be reduced by inheritance of values, (iii) inheritance also prevents human error in entering data (even considering double checking conformation), (iv) the database size can be reduced due to preventing repetition of the assigned pieces of information, and (v) categorization of SMFs is possible for future uses. The genotypes and phenotypes of SMFs include specific sets of information and associative relationships. The information structural relationships are inherited from genotypes through phenotypes to specific instances. This inheritance mechanism contributes to an effective architectural and operational processing of the SMFs information.

The proposed computational constructs were intended to realize SMF knowledge frames and have been confirmed through critical system thinking and projecting them to numerous practical design cases and the real-world processes. However, it must be emphasized, a final utility validation will only be possible through an all-inclusive implementation of the SMF-based modeling and simulation toolbox. In a broader research perspective, the current chapter can be considered as the computational feasibility confirmation of SMF theory.

From the perspective of **usability**, the proposed constructs and computation enablers operationalize the theoretical specifications that are inevitable for a system-level configuration and conceptualization of CPSs. Usability should be judged with the eyes of CPS designers and having their requirements in mind, since they are the decisive stakeholders. These issues are considered below:

- CPS designers do not have to bother with definition of SMFs. They are available for them in a relatively high level of preprocessing. The task of specifying the required chunks of information about components, and accommodating them into SMFs, is delegated to knowledge engineers. Furthermore, computational concerns and the design concerns have been considered concurrently, rather than in separation. They can pay their attention to the creative part of system-level configuration and conceptualization of CPSs in the pre-embodiment design phase. At the same time the design thinking and composition process is supported by the systematic computational workflow.
- The proposed SMF-TB supports various user activities such as SMF definition, content conversion, CPS model composition, CPS modification, model simulation, and so forth, in an integral manner. This has been made possible by the rigorously defined theoretical/methodological frameworks, and the uniform implementation of the data processing and process execution resources.
- The information structures and defined contents of SMFs are reusable. On the other hand, capturing and processing system-level information of components are demanding with regards to the needed computational resources, and also effort-intensive from a knowledge engineering perspective. However, in the end, large scale reuse of information structures and contents of contributes to efficiency and transparency both in the design and the knowledge engineering activities.

- SMFs can be retrieved, adapted and manipulated throughout the whole CPS model composition and modification process. This is particularly important in the pre-embodiment design phase. The SMFs-based implementation even allows suggesting components according to the list of requirements of designers. The taxonomically aided warehouse management helps designers find the most appropriate components for model composition.

From the perspective of **utility**, the whole methodology and proposed constructs support CPS designers to tackle system composability challenges. It can also be claimed that SMFs also facilitate reducing compositionality issues by raising the modeling element level from low level constituents to high level system features. The major indicators of utility are as follows:

- Our methodology assumes that CPSs are composed of SMFs. It imposes a strictly physical view, which is obeyed by the ISCs and PCSs. This physical view has been transferred to the organization of the warehouse databases, which are managed on the level of chunks of information, rather than on the level of entities of information.
- The proposed constructs play the role of cognitive enablers of programming. The proposed construct oriented thinking introduces a meso-level in between the micro-level entity management and the macro-level model management. Advantages of this can be easily identified from both implementation and application points of view.
- The created CPS models are extensively and rapidly modifiable. The proposed constructs capture the architectural and operational data of models through the meso-level chunks of information and relations. On the other hand, they can manage structural re-parameterization and attribute changes in a holistic way [22].
- The proposed constructs lend themselves to an effective constraints management, which capability originates in their abovementioned meso-level manifestation. They support not only modeling, but also simulation of operations. The resources needed for simulation (e.g. methods, relations, parameters, temporal relations) are all accommodated in ISCs.
- The proposed construct-driven schemas and database management allows the interoperation of the SMF-TB with other modeling and simulation software tools. The chunks of information and the relations can be converted to the entities and relations of micro-level database schemas. However, in this case, many of the advantages of SMF instances-based modeling are lost.

5.7.1 Ignoramus et ignorabimus?

This Latin maxim (meaning 'not known and not knowable') was used in the nineteenth century to express the position on the limits of knowing. In the context of this chapter, it fairly expresses our position about validation of the information schema constructs proposed for instantiation and composition of system manifestation features in pre-embodiment modeling of cyber-physical systems. Although we have argued with defendable reasons that the proposed computational constructs oriented thinking supports

feasibility, utility and usability of the SMF-TB, we must recognize the fact that only a fully fledge implementation can provide practical evidences for these.

Using information schema constructs for database development and management for instantiation and composition of system manifestation features is a novel and affordance-rich approach. ISCs help tackle the heterogeneity problem and support multi-granularity. In the current stage of our research, we do not have the means that would be needed for a comprehensive and systematic validation of the proposition. The testbed implementation provided some initial insight in the above mentioned three aspects, but it cannot replace a long term testing in benchmark applications. This remains for our future work.

Our reported work concentrated on the methodology of computation, rather than on the methodology of using the proposed system manifestation features-based modeling in various application contexts. For this reason it is understandable for us that the presented stage of development gives floor to some skepticism. But, the major question is whether the impact of the concept of ISCs on the targeted SMFs-based toolbox is just 'ignoramus' or 'ignorabimus'. Without being able to building it completely, applying it widely, and testing it exhaustively, we tried to consider everything based on which we could forecast its merits from feasibility, usability and utility perspectives. Based on our recent findings, it seems that our arguments will be defendable. But the Catch-22 situation and the dilemma related to that cannot be resolved.

The SMFs-based modeling process has been decomposed to intermittent and terminal process elements. First of all, the process divides into two sub-processes, which are called 'defining SMF entities' and 'composing CPS models'. The former sub-process includes the procedure of creating genotype and the procedure of deriving phenotypes. The procedure of creating genotype includes several cycles of activities, such as (i) basic definition of genotypes, (ii) operational definition of GTs, and (iii) connectivity definitions of the units of operation of GTs. The latter sub-process includes the procedure of composition and instantiation of phenotypes and the procedure of model management. As the name implies, every cycle of activities consists of specific modeling activities, which further decomposes to specific actions.

Multiple modules of the SMF-TB have been defined to support the execution of instantiation, for instance: (i) the phenotype browser module that is in charge of: (a) finding the SMF phenotypes that can be used for modeling of the CPS at hand, (b) presenting the information sets to the user, and (c) sending the information sets to the Composer module, (ii) the Composer module, that: (a) receives the data packages of the chosen phenotypes, (b) updates them with new information, and (c) sends the up-dated data packages to the Instantiator module, (iii) the Instantiator module, that which determines and assigns values and constrains to parameter variables, and (iv) Model warehouse module, that stores the consolidated and tested system model data.

The task of initial creation SMFs is delegated to knowledge engineers. This is in line with the trends of setting up and updating complex information technological systems, such as smart cyber-physical systems. The promotion research addressed the challenge of supporting both the methodological and computational processes of creating very-high level semantic entities for next-generation system architecting and functional design. The computational constructs proposed for the development of the SMF-TB paid attention to

the different system user-roles, as well as to the issue of information reusability. The knowledge sources and infrastructures needed for the implementation of SMFs are available, and will be even more sophisticated in the future. The concept of ISCs allows a language independent specification of information structures for both information engineering and knowledge. Nevertheless, the main objective remains to be a comprehensive support of the creative work of system designers. A fully fledge implementation and validation of SMF-TB will show to what extent this goal has been accomplished.

While this promotion research could not reach so far as proving utility and efficacy in various application contexts and considering different designers and tasks, the feasibility and usability issues have been comprehensively addressed in the completed research. This chapter showed that the proposed AKF and OKF provide sufficient information and are convertible to computational constructs, which in turn can frame both the computational process and the design process. Although the process of creation of SMFs has been described as a manual step-wise process, future research however may address the opportunities of a semi-automated or full automated process. In the case of the manual approach, several solutions exist to discount the overall effort of SMF creation:

- Open source libraries:
  By providing open source online libraries, the SMFs defined by one can be used by others. Sharing the outcomes of knowledge engineering works contributes to a rapid growth of database contents.
- Company catalogues:
  All new products come out with their digital catalogues. Companies developing components tend to produce digital replicas and catalogues of their products, which can be used as the basis of SMFs. If SMF creation forms become standardized, this can give impetus to SMF-s based modeling, since nobody knows more about their own products than the producer companies.
- Marketing opportunities:
  Since designers search for components that are appropriate from the aspect of their requirements in designing and consequently in modeling, CPSs, availability of SMFs as open access can improve visibility of products. Accordingly, defining SMFs of products and sharing them can be exploited by companies as a marketing strategy.
- Creating SMFs market:
  Creating SMFs may stimulate a new focus for undertaking, and compiling catalogues of standard definition (interchangeable) and selling them can be a novel form of motivation for entrepreneurship for knowledge engineers. Nowadays, purchasing a virtual car in a race game is already a common thing - thinking about monetizing SMFs would not be a dream.
- Automatic SMF creation:
  Studies concerning the opportunities of a semi-automated or a full automated generation of SMFs and compiling digital SMF catalogues on the web are still not even in a premature stage, but this idea may get more emphasis when research in system-level features-based modeling of complex heterogeneous (transdisciplinary) systems is further progressing. Some sort of smart collaborating agents systems and the smart cyber-physical systems themselves can be considered for

the task of facilitating automatic conversion of digital catalogues into open access SMFs warehouses.

5.8 REFERENCES

[1] Scacchi, W., (2001), "Process models in software engineering", Encyclopedia of software engineering.

[2] Arbaoui, S., Derniame, J.-C., Oquendo, F., and Verjus, H., (2002), "A comparative review of process-centered software engineering environments", Annals of Software Engineering, Vol. 14 (1-4), pp. 311-340.

[3] Garg, P.K., Mi, P., Pham, T., Scacchi, W., and Thunquest, G., (1994), "The SMART approach for software process engineering", Proceedings of the Proceedings of the 16th international conference on Software engineering, IEEE Computer Society Press, pp. 341-350.

[4] Rolland, C., (1998), "A comprehensive view of process engineering", Proceedings of the Advanced Information Systems Engineering, Springer, pp. 1-24.

[5] Madhavji, N.H., (1991), "The process cycle [software engineering]", Software Engineering Journal, Vol. 6 (5), pp. 234-242.

[6] Pourtalebi, S., and Horváth, I., (2016), "Towards a methodology of system manifestation features-based pre-embodiment design", Journal of Engineering Design, Vol. 27 (4-6), pp. 232-268.

[7] Chaudron, M., Eskenazi, E., Fioukov, A., and Hammer, D., (2001), "A framework for formal component-based software architecting", Proceedings of the OOPSLA Specification and Verification of Component-Based Systems Workshop, Citeseer, pp. 73-80.

[8] Lorsz, M., (2002), "The software process: Modeling, evaluation and improvement", Handbook of Software Engineering & Knowledge Engineering: Fundamentals, Vol. 1, p. 193.

[9] Pandit, S., and Honavar, V., (2010), "Ontology-guided Extraction of Complex Nested Relationships", Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence, Institute of Electrical and Electronics Engineers (IEEE).

[10] Fioukov, A.V., Eskenazi, E.M., Hammer, D.K., and Chaudron, M.R., (2002), "Evaluation of static properties for component-based architectures", Proceedings of the Euromicro Conference, 2002. Proceedings. 28th, IEEE, pp. 33-39.

[11] http://catalog.miniscience.com/catalog/motors/RF370CA15.html, (2015), "RF370CA15 DC Motor", Vol. 2015, MiniScience Inc., Clifton, USA, 2015.

[12] Edwards, J.R., and Bagozzi, R.P., (2000), "On the nature and direction of relationships between constructs and measures", Psychological methods, Vol. 5 (2), p. 155.

[13] Lee, G., Sacks, R., and Eastman, C., (2007), "Product data modeling using GTPPM — A case study", Automation in Construction, Vol. 16 (3), pp. 392-407.

[14] Richter, G., (1981), "Utilization of data access and manipulation in conceptual schema definitions", Information Systems, Vol. 6 (1), pp. 53-71.

[15] Petter, S., Straub, D., and Rai, A., (2007), "Specifying formative constructs in information systems research", Mis Quarterly, pp. 623-656.

[16] Gavrilescu, M., Magureanu, G., Pescaru, D., and Doboli, A., (2010), "Accurate modeling of physical time in asynchronous embedded sensing networks", Proceedings of the IEEE 8th International Symposium on Intelligent Systems and Informatics, Institute of Electrical and Electronics Engineers (IEEE).

[17] Schumann, M., and Michael, J.B., (2009), "Statechart based formal modeling of workflow processes", Proceedings of the System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on, IEEE, pp. 1-5.

[18] Petnga, L., and Austin, M., (2016), "An ontological framework for knowledge modeling and decision support in cyber-physical systems", Advanced Engineering Informatics, Vol. 30 (1), pp. 77-94.

[19] Bhave, A., Krogh, B., Garlan, D., and Schmerl, B., (2010), "Multi-domain modeling of cyber-physical systems using architectural views".

[20] Munir, S., Stankovic, J.A., Liang, C.-J.M., and Lin, S., (2013), "Cyber physical system challenges for human-in-the-loop control", Proceedings of the Presented as part of the 8th International Workshop on Feedback Computing.

[21] Seiger, R., Keller, C., Niebling, F., and Schlegel, T., (2015), "Modelling complex and flexible processes for smart cyber-physical environments", Journal of Computational Science, Vol. 10, pp. 137-148.

[22] Hadorn, B., Courant, M., and Hirsbrunner, B., (2015), "Holistic System Modelling for Cyber Physical Systems", Proceedings of the The 6th International Multi-Conference on Complexity, Informatics and Cybernetics (IMCIC 2015), accepted paper.

Chapter 6

BENCHMARKING

the theoretical framework and methodological fundamentals against existing systems

The previous chapter discussed the computational and procedural implementation of SMFs-based modelling and the SMF-TB. The objective of the research was to investigate computational feasibility of the proposed SMFs-based modeling approach and toolbox, and to provide technical information about a possible form of implementation. This chapter focuses on validation of the competitiveness and practical cogency of the proposed theoretical framework and computation methodological fundamentals. The preferred method of validation was a systematic and critical comparison and evaluation (benchmarking) of the theoretical framework and methodological fundamentals with that of some other, commercial or proprietary, tools or systems supporting system-level design. The completed comparative benchmarking was used to evaluate if the theoretical and methodological novelties are seen competitive or not by domain experts and if the foreseen benefits indeed have cogency. Thus, the fifth research cycle was devoted to benchmarking and providing evidential information about the strong points and weak points of the SMFs-based modeling approach in view of the functional and utility capabilities of the selected benchmark tools and systems.

The first sub-chapter explains the main research questions that express our interest in validation of the competitiveness, and the organization of benchmarking. In sub-chapter 6.2, the cogency of the proposed theoretical framework and computation methodological fundamentals is reviewed. Sub-chapter 6.3 discusses the planned conduct of benchmarking and choosing comparable system design, modeling and representation support tools, systems or languages. In Sub-chapter 6.4, we present the aspects of assessing the conceptual framework and methodological approach in benchmarking. The results of survey are demonstrated in sub-chapter 6.5. The major findings of benchmarking are presented and discussed in Sub-chapter 6.6, and discussed in detailed in sub-chapter 6.7. Our reflections and conclusions concerning the value profile and possible enhancement opportunities are presented in Sub-chapter 6.8.

6 BENCHMARKING THE THEORETICAL FRAMEWORK AND METHODO-LOGICAL FUNDA-MENTALS AGAINST EXISTING SYSTEMS

6.1 INTRODUCTION: OUR INTEREST IN VALIDATION

This chapter reports on the validation of the developed SMFs-based modeling approach. The main intention of the related research was to see how much the proposed theoretical framework and computation methodological fundamentals are competitive if they are compared with those of other commercial or laboratory tools, systems, and languages. Or, in other words: 'Do they have sufficient cogency from a practical perspective?' This validation complements those actions that have been completed in the fourth research cycle to prove the computational and methodological feasibility of the proposed approach. We developed a specific research design for the validation in the form of comparative benchmarking and critical analysis. This has been done with awareness of the fact that a comprehensive validation would necessitate at least a fully functional and full-scale prototype implementation of the system modules and the modeling toolbox. Obviously, this wishful fully-fledged implementation would require a large team of experts involved in ontology development, software architecting, software programming, and system prototyping, as well as sufficient financial and laboratory resources.

In order to be able to conduct a comparative benchmarking, we had to investigate (i) the modeling framework of currently available modeling tools and languages which are comparable with the proposed theoretical framework and computational methods, and (ii) the aspects of comparison that are important to show competitiveness and cogency at modeling and design of CPSs. For the above mentioned reasons, we had to limit the comparative benchmarking to framework level instead of the level of detailed functions. Consequently, first we had to understand the theoretical assumptions behind the available tools selected for benchmarking, instead of the functional features provided by the tools. In terms of their implementation, many tools have common roots, or use the same theoretical and computational resources. For instance, several tools have been developed based on the Modelica language, which are only slightly different in terms of the functional features provided by them. However, the framework which supports Modelica as a language is the same for all these derived tools.

The frameworks determine the underlying logical or semantic structure, or a hypothetical description of a complexity, which lends itself to deriving models, representations, procedures, and methods of complex things or phenomena. The implications of the framework are presented in the 'working model' of the specific tools. This allowed us to make a kind of 'reverse engineering' towards identifying the enablers of the frameworks of the benchmarked tools. This was actually inevitable since the frameworks were not explicitly documented, or were just partially specified in descriptive documents. There were some similarities observed in terms of conceiving, processing and handling information among the compared modeling frameworks, the combination of these and the entailed working model makes the frameworks distinctive from the others. These specifications were the main subject of benchmarking in the fifth research cycle. To be able to do so, we needed to (i) summarize the novel specifications of SMFs-based modeling framework, (ii) find unique

specifications of the other frameworks based on the available knowledge sources, (iii) find and list the aspects of comparison based on the requirements of pre-embodiment design of CPSs, and finally (iv) compare the specifications of the available frameworks with novel specifications of SMF-based modeling framework based on the derived comparison aspects. The mentioned steps are presented in the following sub-chapters respectively.

6.2 NOVELTIES OF THE PROPOSED FRAMEWORK

The conceptual, computational and methodological elements of the proposed SMFs-based modeling framework were comprehensively discussed before. In this sub-chapter, we expose those novelties that we believe are the most important from the point of view of evaluating the possible scientific and professional impacts of the proposed modeling framework from computational, implementation and application perspectives. This exposition helps comparing the SMFs-based modeling framework with those that underpin the other benchmarked modeling tools.

N1 Imposing strictly physical view:

In contrast with the frameworks that prefer logically-based modeling and neglect capturing the observable physicality of the modeled systems (i.e. physical interaction of components and morphological considerations), our framework captures everything in a cyber-replicate of the physical world. Imposing pure physicality supports aggregation of components without the need for applying abstraction. Subsequently, all parameters of lower level components are directly determined and affect features of the higher level components. SW and CW constituents are also considered as physical entities (instead of logical or virtual ones) and can be aggregated with all other constituents.

N2 Enforced concurrent consideration of architecture and operation:

SMFs capture both architectural and operational aspects of components concurrently, and without any preference. Consequently, creation of a SMFs-based CPS model simultaneously means creating the architectural and the operational models of a system in interaction, as part of a single process. This is in contrast with most of the logically- and mathematically-based modeling tools, which put emphasis on the functional aspects of components.

N3 Amalgamating of architectural and operational aspects:

All information sets that define relations among architectural and operational aspects are captured within SMFs (e.g. through constraints, parameters, state-transitions and associating events, timing, conditions, and logical relations). This results in a robust approach of modeling in which the changes due to operation of components are directly reflected on their architectural attributes, and the effect of architectural alterations is directly reflected on the performed operations of components.

N4 Using uniform information structures for heterogeneous components:

The knowledge frames and the corresponding database schema for accommodating pieces of information are the same for all types of components. This means that hardware, software, and cyberware constituents are practically stored in the same structures and relational tables. As a result, the relations among them are being processed uniformly. The

differences between the various constituents are made by their semantics, not by their descriptive information structure. It allows avoiding unnecessary simplification and abstraction (that are typically used by the currently available tools to tackle components' heterogeneity in modeling).

N5 Multi-level multi-granularity:

Multi-granularity means the possibility of moving from coarse-grained model to fine-grained model in the process of modeling, and vice versa. The proposed SMFs-based modeling framework considers multiple levels of aggregation and de-aggregation in the modeling space. SMFs have the same information structure in all aggregation levels. Accordingly, containment and connectivity relations in all levels are captured without applying abstraction and simplification. This results in linking the parameters of SMFs in multiple various levels, and therefore, considering effects of changing values of parameter in both top-down and bottom-up manners.

N6 Multi-aspects coupling among SMFs:

The specification of both containment and connectivity relations of SMFs are supported through interfaces of SMFs, neglecting the types of components (e.g. if SW, HW, CW, or AW). Consequently, the integration is supported by several aspects of multiple coupling. SMFs can be connected together through states of operation, events (i.e. timestamps), streams, parameters, physical mates, contracts, and so forth. Higher level SMFs can also be created if multiple couplings among the lower level SMFs are created. The dynamic integrity is also captured through utilizing concept of spatiotemporal mereotopology, and consequently, state-based definition of architectural relations and parameterizations.

N7 Multi-stage model composition:

In SMFs-based modeling framework, models are not created in one go. Generating genotypes, deriving phenotypes, instantiation of phenotypes and model composition are the stages that an SMF should go through to eventually end up in a model. The relations among these stages of SMFs life cycle provides many advantages that have been discussed before. The most important benefit from model composition perspective is that manual information input is reduced in the conceptual design phase, since the kernel (default) information is already included in SMFs by the knowledge engineers.

N8 Multi-purpose system-level features:

Reasoning with higher level semantics, trans-disciplinary fusion of knowledge, and harmonization of pre-embodiment design and computational methodology are the multiple purposes that are fulfilled with SMFs as system-level features. Higher-level semantics equips with ability to capture those information sets that are needed for system level conceptualization. Trans-disciplinary fusion of knowledge enables comprehensiveness of the model and communication of experts. The modeling approach is harmonized with the computational methodology and the conceptual design methodology.

N9 Multiple application contexts:

SMFs as novel system-level features support both system-level conceptualization and system-level engineering (analyzing and optimization). They also could be used as means of communication among experts when the discipline related tasks are delegated to them.

SMFs are reusable modeling entities that could be added, modified, subtracted, and regenerated from a model. It makes them suitable for mass customization, problem diagnoses, adaptability tests, and so forth.

N10 Multi-component warehouses:

The SMFs are organized into warehouses, which facilitate handling and storing of the interconnected instances of SMFs and support the feature creation and model composition process. The proposed database component provides relational tables to store data records of the included SMFs. The database management component takes care of storing, querying, invoking, and removing records as well as generating keys to track relations among information stored in the relational tables. The meta-level knowledgebase component is in charge of capturing track of model composition and modification.

N11 Benefiting from active ontologies:

Semantic-richness of the SMF-TB is supported by active ontologies. Continuous updating of the ontological records is necessary due to emerging of new technologies, materials, attributes, protocols, and so forth. Although, effort-intensiveness of this task can be considered as a drawback, the usefulness of the results cannot be disregarded.

6.3 AVAILABLE RELATED MODELING TOOLS

6.3.1 On the selection criteria

As discussed in Chapter 2, numerous modeling tools are being used for designing, evaluation and optimization of complex technical systems. These tools have been developed for general applications, while others are offered for specific applications and specific system context. Therefore, it was not easy to choose the tools, which are the most appropriate for designing CPSs. To systematize the selection of the relevant tools, we assumed some selection criteria, as follows:

- Appropriateness for system-level design purposes.

We intended to find general system composition tools for comparison. However, not all of the found modeling tools were developed for the purpose of system design. Actually some of them are not really suitable for effective system design. For example, some tools specifically support designers in optimization of systems, while others were developed for circuit design, or software architecting, just to mention a few examples.

- Acceptance by designers of complex technical systems.

Some tools are not known enough by designers. The reason can be that they are in the beginning of their academic and industrial use, or are intended to other experts than designers. We wanted to find tools that specifically support pre-embodiment design and have been tested by designers in this context. Otherwise, finding enough evidences for benchmarking is not possible. Fortunately, there are many publications about the known modeling tools, which evaluate them and we also found numerous experts to be asked about their experiences of using the tools.

- Representing different categories of modeling tools.

There are several methods of modeling that have been used to underpin the development of tools and languages. Therefore the tools represent rather diverse categories. Each modeling method has its own advantages and disadvantages, which remains likely the same in all tools developed based on it. We tried to select the most dominant tools in each category for comparison. A categorization of the CPS modeling tools is discussed in [2]. In this publication, six categories of tools were recognized:

- Model checkers, which are suitable for automatic verification of specification and exhaustive model verification in the phase of an early model exploration
- Block diagram languages, which define a model as a composition of graphs and nodes, and can be used for modeling both cyber and physical parts of a CPS
- Equation-based object-oriented (EOO) languages, which are acausal (meaning that modeling is based on equations that do not specify which variables are inputs and which ones are outputs) [3]
- Reactive languages, which are suitable for the cyber parts and their continuous reaction to the physical environment
- Hardware description languages (HDLs), which are suitable for modeling and specifying digital circuits
- Multi-formalism languages and tools, which employ models of computation (MoC) and combine them for actor-oriented modeling of CPSs.

Figure 1 graphically shows the idea of classification presented in [2]. It can be recognized that this categorization reflects a combined control point of view and software design view. Consequently, this classification was considered just as a guide, and was not adopted as a reasoning framework from the perspective of system-level design and conceptualization. In order to be able to select comparable modeling frameworks for benchmarking, we concentrated on the formalisms, which were in combination associated with various languages and tools. In the end, one modeling tool (or language) was selected from each of the shown categories.

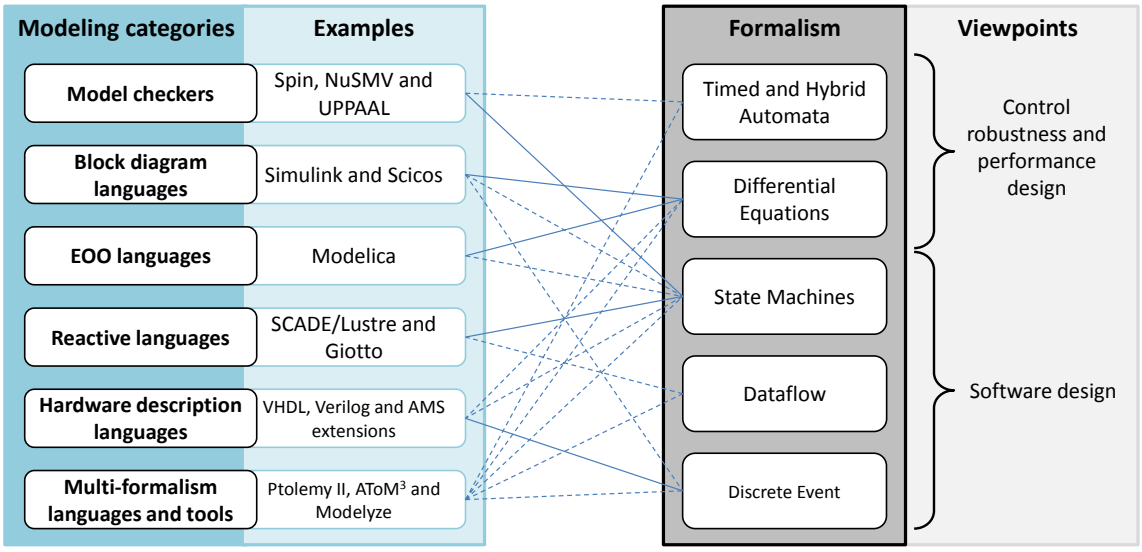


Figure 6-1 The viewpoints and formalisms applied in the case of system modeling tools and languages (based on [2])

Based on the mentioned criteria, five commercial and experimental modeling tools and languages were selected, namely Simulink, Modelica, Ptolemy II, SysML, and LabVIEW. These are mostly being used for system-level design purposes, widely accepted by designers of complex technical systems, and each of them represents a different category of modeling tools. Bellow explains the reasons for this selection:

- **Simulink** belongs to category of block-diagram languages, which is also considered as an actor-oriented tool [4]. It is one of the most known modeling and simulation tools, which is widely used in various contexts. The strength of Simulink is in the function-oriented modeling of systems through differential equations.

- **Modelica** is a popular modeling language which uses mathematical equations for modeling relations of physical attributes and parameters. In our benchmarking, it represents acausal EOO languages. Unlike Simulink, Modelica is a language that was used as a basis for development of several modeling and simulation tools. It also partly supports capturing information about architecture of systems.

- **Ptolemy II** is a non-commercial tool which uses models of computation for applying multi-formalism approach. Although it was originally developed for modeling embedded control systems and systems on a chip, many efforts have been done recently to adopt it for CPS design. In our benchmarking it represents a unique category which employs MoC for modeling systems. Moreover, it is the only non-commercial tool in our benchmarking.

- **SysML** is a logical modeling language, which is developed based on UML for generic modeling of systems. It also supports more specific modeling concerns with the aid of its add-on profiles. Since SysML is an open-source modeling language, many commercial tools have been developed based on it. It has been selected for its high-level widely logic-based system modeling which provides more freedom in conceptual design phase.

- **LabVIEW** is originally developed for data acquisition. Although it is not considered as a modeling tool used for system conceptualization, there are evidences of using LabVIEW in system development [5] [6]. LabVIEW is selected for comparison due to its wide use in system design and development, as well as its great contrast to the other selected tools.

In the following the specifications of the chosen tools and languages are elaborated.

6.3.2 Simulink

Simulink is a widely used tool for modeling and simulation of dynamic systems. It provides a block diagram environment with a graphical editor and uses customizable block libraries for model-based design [7]. The customized functions can be written as MATLAB, Fortran, Ada, or C code into a model, or browsed from libraries including, structural blocks (e.g. Mux, Switch, and Bus), algorithmic blocks (e.g. Sum, Product, Lookup Table), and continuous and discrete dynamic blocks (e.g. Integration, Unit Delay) [8]. Simulink mainly supports modeling functions of systems. However, there are possibilities to connect the functions defined in Simulink to the architectural models of other tools. The functions in Simulink

can be defined hierarchically. It means that a set of related block diagrams can be encapsulated in a single block as its subsystems. A Simulink model consists of both signals and parameters. Signals are represented by lines that connect blocks together. Parameters define system dynamic and behavior by assigning coefficients. Simulink uses solvers to perform simulation. During simulation, the system dynamics should be computed over time according to numerical integration algorithms which are called as Solvers. Multiple solvers are provided by Simulink to support the simulation of a broad range of systems, including continuous-time (analog), discrete-time (digital), and hybrid (mixed-signal).

Through the libraries and add-on products, Simulink supports modeling physical systems with mechanical, electrical and hydraulic components. Accordingly, it could be used for multiple purposes, i.e., control, aerospace, signal and image processing, and communications. Simulink also supports hardware-in-loop approach which is known as built-in support. It means that algorithms designed in Simulink can be connected to low-cost hardware (e.g. Arduino, LEGO Mindstorms NXT, and Raspberry Pi) for running models. This approach can be used for control systems, robotics, and other applications in real time manner. Both continuous-time and discrete-time methods of simulation are supported by Simulink. It uses differential equations for continuous-time, and discrete time difference equations for discrete-time simulation [2]. These equations are wrapped in diagrams presented with blocks and signals. The equations are solved with numerical algorithms built into the vendor's tool environment, which process that the Simulink model is analytic [9]. Simulink is not supporting multi-formalism itself. However, in combination with other tools such as Stateflow and SimEvents can combine several formalisms [2].

These auxiliary tools also provide particular kinds of blocks which can be used in Simulink block diagramming. Stateflow and Simscape are two famous tools. Stateflow allows the users to model decision logic based on the state machine and flow chart formalisms. Simscape provides fundamental building blocks from various domains (such as electrical, mechanical, and hydraulic) that can be combined to model a physical plant [10]. In order to support both hardware and software design, numerous efforts have been done. An example, in which UML is used for software parts and connected to Simulink for hardware parts, is reported in [11]. In this case, UML was used for discrete computation system modeling and Simulink was used for continuous physical system modeling.

The Embedded Modeling Language (ESMoL) environment is a suite of domain-specific modeling languages and a set of tools, developed for generating virtual prototypes and facilitating design of embedded real-time control systems [12]. By importing a model created in Simulink into ESMoL, it becomes the functional specification for instances of software components [13]. Moreover, there are some add-on products for Simulink that could generate C and C++, HDL, or PLC code directly from Simulink model. Simulink is also very useful to support a decision of design concept with minimum of effort on some software phase such as developing and testing rapid virtual prototype [14]. Mathworks official website has briefed the features of Simulink [8] as tools for importing C and C++ codes into models, function blocks for importing MATLAB algorithms into models, multiple tools for model analysis and model refining, project and data management tools, simulation engine with fixed- and variable- steps solvers, several libraries and graphical editor.

6.3.3 Modelica

Modelica is considered as a resemblance of object-oriented programming languages. However, there are two main differences. Firstly, Modelica can be considered as modeling language rather than a programming language. Secondly, Modelica uses different type of compilation, which allows it to translate classes into objects which can be implemented by a simulation engine [15]. Modelica is characterized as acausal, object-oriented, and domain neutral. These characteristics make it suitable for system-level simulation [16]. Modelica has major differences in comparison with Simulink in many aspects. For instance, modeling entities in Modelica are the architectural components of a system, in contrast with Simulink models in which blocks represent functions of a system. That is reason that models in Modelica are easy to understand. In addition, due to acausal nature, causality derivation process is not required in Modelica, which makes it faster [17].

In contrast with actor-oriented tools such as Simulink and Ptolemy II, that produces models in which components have input and output ports, Modelica is object-oriented acausal language which considers equations in modeling [3]. Mathematical functions in Modelica allow acausal modeling, high level specification, and increased correctness. Likewise other object oriented languages, it benefits from its general concept. It uses a syntax that resembles that of Java and Matlab. However, its efficiency is comparable to C [18]. The visual component programming in Modelica supports its hierarchical system architecture characteristics. "Multi-domain modeling" is mentioned as one of the characteristics of Modelica. It combines electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc. in one model [17]. However, the currently available libraries are mostly hardware components of a system.

Since software design and higher-level system conceptualization is not supported sufficiently in Modelica, several researches are performed to connect Modelica to UML [19] [20]. Modelica Modeling Language (ModelicaML) is a profile of UML that supports description of time-continuous and time-discrete/event-based system dynamics [21]. It generates executable Modelica code based on the UML diagrams [22]. On the other hand, it provides modeling entities for presenting the composition, connection, inheritance or behavior of classes in UML for graphical model definition. As a result, ModelicaML supports formalizing and evaluating system requirements using simulations [23].

Owing to the fact that Modelica is an open source language, it is exploited by many different tools. Dymola, MapleSim, SystemModeler, and OpenModelica are the most known tools created based on Modelica. Each of these tools offers different advantages by providing specialty in terms of the tasks addressed. For example, Dymola is a tool primarily for Modelica coding. It also integrated in CATIA system V6 for supporting 3D interaction [24]. MapleSim is more focused on modeling without relying on hand coding, while still having the flexibility of accessing code when it is needed [25]. Multi-domain modeling, hierarchical modeling, and hybrid systems modeling are mentioned as features of SystemModeler. The numerical solvers of this environment detect and handle discontinuities in hybrid systems, so models with sudden events such as switches, collisions, or state transitions are correctly simulated [26]. In the tools such as SystemModeler the Modelica language is not only used as a design facilitator, but it also supports system analysis and engineering.

6.3.4 Ptolemy II

Ptolemy II is an open-source modeling tool which has been under development since 1996 [27]. The first generation version of this software is called Ptolemy Classic. In the beginning, it was an extension of the interface of Gabriel [28] which was written in Lisp and aimed at signal processing [29]. Ptolemy classic is implemented in C++ language [30]. In the Ptolemy classic, the key idea was to mix models of computation rather than trying to develop one, all-encompassing model [31]. This idea was later on incorporated by Ptolemy II in order to focus on component-based design to facilitate reusing of components in modeling [32]. Ptolemy II creates models as clustered graphs of modeling building blocks and relations among them [33].

In Ptolemy II, The basic building blocks of a system are actors [34], which represent actions of the components of a system. Consequently, a model is a hierarchical aggregation of actors. That is the reason that Ptolemy II is considered as actor-oriented modeling framework. Actors contain a thread of control and provide interfaces to communicate with other actors. An actor can create other actors, send messages, and modify its own local state [35]. Actors are implemented as java classes and stored in a library. The actors composed in a model can execute concurrently and communicate through messages sent via interconnected ports [27]. The actor-oriented view of a system allows separation of data transmission from control transferring which makes it different from object-oriented modeling. The emphasize of actor-oriented design is on concurrency and communication between components [32].

Semantics is handled by models of computation In Ptolemy. MoCs are implemented by a software component named as director. A domain is composed of one or more actors which are governed by a director that executes a MoC. Consequently, hierarchical composition of several domains with various MoCs in a model supports modeling of heterogeneous systems. Polymorphism and modal models underpin this ability. Polymorphism means that components are designed to be able to operate in multiple domains, and modal models means that finite state machines can be combined hierarchically with other MoCs [32], e.g. Synchronous Data Flow, Kahn Process Networks, Discrete Event, and Continuous Time [27]. Actors can be either atomic or composite, which are both executable. Composite actors are executed as a composition of executions of their contained actors which facilitates managing heterogeneity of systems' models [34].

Models in Ptolemy are constructive which are often used to describe behavior of a system in response to stimulus from outside the system [32]. It means that the models as formal representation of systems are defined as computational procedures that mimics a set of properties of the system [32]. This type of models is called executable. Executable models are made based on MoCs which represent sets of physical principles that regulate interactions among actors. The physical principles are provided by one or more semantics included in MoCs. The included semantics might be distinct sets of rules that impose identical constraints on behavior [32]. Combining a large variety of MoCs and hierarchical nesting of the models are supported by Ptolemy II [36].

In Ptolemy II, models can be created either by writing Java codes that instantiates components, parameterizes them, and interconnects them or by using Vergil as a graphical editor [27]. An XML schema named MoML is used by Vergil in order to store models in ASCII file

format [37]. Several specialized tools are available for Ptolemy II, namely HyVisual (for hybrid systems modeling) [38], Kepler (for scientific workflows) [39], VisualSense (for modeling and simulation of wireless networks) [40], Viptos (for sensor network design) [41]. The visual depictions of systems provided by these tools helps to tackle the complexity introduced by heterogeneous modeling [42].

Since operations are defined as event-based communication, causal equation-based modeling of physical components is not supported sufficiently [72]. Ptolemy also does not sufficiently support architectural aspects of the modeled systems [71]. Ptolemy does not separately capture attributes of the physical architecture and the focus is on the operation of the components. Metronomy is a good example of integrating other frameworks to expand functionality. Metronomy uses Ptolemy for capturing functional model, while the architectural aspects are supported by MetroII [71].

6.3.5 SysML

SysML was brought into existence as a profile of UML in 2003. Then, the Object Management Group (OMG) has adopted it as OMG SysML and developed it as a standalone product. SysML is one of the most popular frameworks for Model-Based Systems Engineering (MBSE) Purposes due to its simplicity, visual modeling capability, and general purpose application [43]. SysML shares many similarities with UML which is a standard general-purpose modeling language [44]. UML is an object-oriented visual modeling language, initially developed for software engineering. It provides sets of graphical notations for visual modeling of software systems.

SysML provides methods to specify internal structure of a system and its behavior [45]. Two sets of structure and behavior diagrams are provided by SysML. The structure diagrams represent architecture of a system and its components through e.g. block definition diagram, package diagram, internal block diagram and parametric diagram. The behavior diagrams represent functions of a system and its components through e.g. activity diagram, sequence diagram, state machine diagram, and use case diagram [46]. SysML diagrams are not sufficiently supported by an explicit handling of semantics. However, there are many add-in products that enhance the capabilities of SysML in specification, analysis, design, verification, and validation of system models [47]. Figure 6-2 shows the types of diagrams offered by SysML for system modeling.

Parametric diagram is a new diagram type that shows mathematical relationships among components of a system and helps in analyzing performance, verification, and validation [47]. However, the issues related to parametric design in SysML are still not sufficiently developed. High level of abstractions, insufficiency of semantics, and inability of processing numerical constraints are the major issues that SysML is suffering from. Different levels of abstraction can be applied in SysML which means different level of details can be chosen [47]. However, in the system level design, designers need to apply more abstraction to reduce the detail level to a manageable amount. Numerical constraints can be presented in SysML; however, these constraints are not evaluated and therefore can only be used as a kind of information for system designers [9]. In addition, SysML does not provide any means to formally analyze the constraints [48].

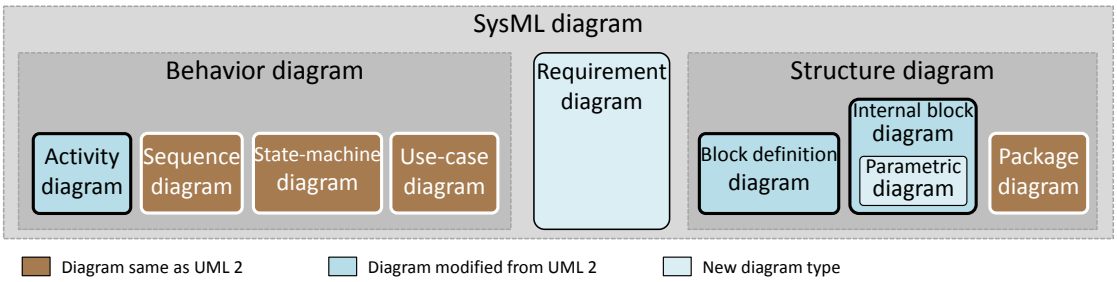


Figure 6-2: Diagrams offered by SysML [46]

All these shortcomings are resulted in the endeavor to develop complementary tools such as ModelicaML [49], SysML-Sec [50], TEPE [51]. ModelicaML adds Modelica's formal executable modeling for analyses and simulation to SysML diagrams [22]. SysML-Sec employs a model-driven approach to focus on security issues during system design [52]. TEPE is a graphical temporal property expression language based on SysML parametric diagrams which incorporate notion of physical time and unordered signal reception [51].

In SysML, functional modeling is supported by two types of ports: (i) standard port is used for event-driven communication and (ii) flow port is used for continuous flow of data or material [53]. The attributes of a flow are described by using flow properties. However, to support engineering of cyber physical systems, a clear distinction between different classes of types is needed [54] which is not supported by semantics. However, it was an effort towards employing some kind of semantic entities in SysML modeling [54].

6.3.6 LabVIEW

LabVIEW stands for Laboratory Virtual Instrument Engineering Workbench. It is a visual programming language from National Instruments that provides an environment for system-design and development. The main application of LabVIEW is for data acquisition, instrument control, and industrial automation [55]. LabVIEW can be also used for developing human machine interface (HMI), sequential control, and safety interlock [56]. Besides its main functionality which is data acquisition, LabVIEW is a useful tool for simulation purposes [57]. In order to support CPS simulation, it also can be used together with other tools [58].

National Instruments introduces LabVIEW as “the ultimate system design software used by engineers and scientists to efficiently design, prototype, and deploy embedded control and monitoring applications” [59]. Libraries of LabVIEW included of the codes needed for off-the-shelf hardware (usually from National Instruments). It also provides a variety of programming approaches including graphical development, and connectivity to existing ANSI C and HDL code [59]. The graphical programming environment offered by LabVIEW is mentioned as a user-friendly interface that strongly supports technical design and visual simulation [60].

A dataflow programming language (named as G) is used in programming LabVIEW [61]. Several package options are provided in LabVIEW as libraries to support e.g. data acquisition, signal generation, mathematics, statistics, signal conditioning, analysis, along with numerous graphical interface elements [62]. In LabVIEW, continuous systems are formulat-

ed and represented as ordinary differential equations or differential algebraic equations; and discrete systems are expressed as difference equations [63].

LabVIEW has numerous built-in functions that facilitates creating user interfaces, data processing, and communication with various instruments [64]. There are also enormous numbers of advanced mathematic blocks for functions such as integration, filters, and other specialized capabilities usually associated with data capture from hardware sensors [65]. Furthermore, a text-based programming component is provided by LabVIEW (name as MathScript) which provides extra functionality for signal processing, analysis and mathematics. The syntax used by MathScript is compatible with MATLAB [65]. As the literature shows, LabVIEW is preferred by many users with regards to MATLAB, when a functional and intuitive graphical user interface or interaction with hardware (signal acquisition and generation) is needed [66]. Although LabVIEW is referred to as a candidate tool for modeling and simulation of the behaviors of hybrid systems [67], since it cannot consider morphological, architectural and physical attributes of components, it is not an appropriate tool for CPS design. It is usually used with other tools to support modeling aspects.

6.3.7 On the frameworks of the chosen modeling tools

In order to be able to compare our modeling framework with the chosen modeling tools, we have to compare them either in the framework level or in the tool level. Since the tool level is not possible due to unavailability of SMF-TB, the only way of getting to the conceptual frameworks of the chosen modeling tools is based on a kind of 'conceptual reverse engineering' of the available tools and using the result of this action in the benchmarking. The reasoning logic of the mentioned conceptual reverse engineering is shown in Figure 6-3. The chosen modeling tools and languages are either distributed commercially, or implemented experimentally.

As discussed in 6.2.1, the five tools selected for benchmarking represent different conceptual frameworks. However, the original frameworks based on which they were developed are usually not documented with sufficient technical details. It is also an issue that the frameworks have been adapted to specific objectives in many follow up tool developments and they are actually not separable from the core of these tools. Furthermore, various add-in products intended to make these modeling tools more sophisticated and expanded their applicability, but also deformed the originally proposed conceptual frameworks. In order to overcome this difficulty, a detailed list of benchmarking (comparison) aspects has been set up. The list includes five main aspects which have been further articulated by a number of subordinate aspects and the related indicators. This composition of these subordinate aspects allowed us to get information about the theoretical and methodological concepts that were used in the development of the benchmarked tools and languages. These aspects are discussed below.

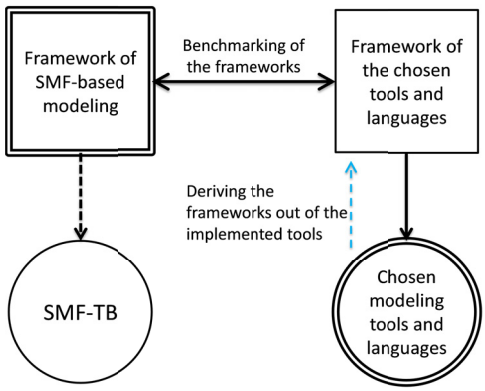


Figure 6-3 Deriving frameworks of the chosen tools

6.4 ASPECTS AND INDICATORS OF COMPARISON

In order to comparing the modeling frameworks, we needed to specify the aspects and indicators of the relevance and sufficiency (or suitability) of the framework-implied characteristics of the frameworks for designing CPSs in the pre-embodiment design phase. Otherwise, it is obvious that the chosen tools and languages are the best in their categories and provide proper assistances for their users as specific groups of system designers with particular design concerns. Our comparison targeted the appropriateness of these tools according to the specific of characteristics CPSs in the pre-embodiment design phase and based on all of the assumptions discussed before. The aspects and the related indicators were also supposed to reflect the design concerns originating in the system composability challenges and in multi-functionality. Consequently, five groups of aspects of comparison are specified which address: (i) comprehensiveness, (ii) heterogeneity, (iii) integrity, (iv) complexity, and (v) implementation. In the following paragraphs, the reasons of choosing these aspects and their criteria and indicators are further discussed.

A1. Addressing heterogeneity of CPSs

Heterogeneity is one of the major characteristics of CPSs that makes them difficult to model and simulate. Heterogeneity of CPSs implies that they are composed of tightly coupled HW, SW, CW constituents and AW components. It entails that in addition to modeling their architecture and operations, a proper modeling tool should also support architectural interfacing of heterogeneous components, as well as their interoperations. This characteristic was taken into consideration seriously throughout our research. The indicators of suitability related to this aspect have been specified as follow:

a. Method of tackling heterogeneity:
   In the context of modeling, the term "tackling heterogeneity" is associated with the capability (and the method employed by) the concerned tool to capture, formulate, and represent interfacing and interoperation among heterogeneous components in a system. The main method used by the benchmarked modeling tools was simplification of the various components to their least common denominators, or imposing certain abstractions on relations among components.

b. Level of applied abstraction:
   The level of the applied abstraction was chosen by us as an indicator. Abstractions are made from various perspectives. Most of the modeling tools use a specific (predetermined) level of abstraction for modeling. The advantages of abstraction are uniform representation of modeling building blocks, and capturing heterogeneous components despite of their inherent differences. The downside of abstraction is that plenty of proper and useful chunks of information that can be used for simulation, analysis, and optimization of the system are neglected.

c. Captured relations:
   As mentioned above, models are only simplified representations of the modeled CPSs, and simplification can be used up to different extents in various modeling tools. Capturing the relations among systems components is largely influenced by the extent of simplification. As a consequence, the heterogeneous physical relations are reduced to a

kind of uniform relations (such as logical relations, mathematical relations, or call-response relations) in different systems.

d. Captured types of constituents:
The last considered indicator was the ability of the modeling tools to capture the different types of constituents. Since CPSs are composed of all types of HW, CW, SW and AW components, the desired modeling tool should be able to capture all of these. The system which is able to capture large level of heterogeneity in modeling is seen as better appropriate for modeling CPSs.

A2. Addressing integrity of CPSs

Integrity of components in CPSs is also an important characteristic of these systems. Nevertheless, it is rather difficult to consider this issue in modeling. Integrity is interpreted as completeness and consistency of capturing the important chunks of information required for representing architectural and operational containment and connectivity relations among components of a system. Achieving a higher level of integrity means the consideration of more types of information (which is in contrast with simplification). The indicators of this aspect are as follows:

e. Architectural containment:
This indicator informs about if a modeling tool is able to capture architectural part-of relations in a system, and to what extent it is able to do so. Due to the latter, it also describes how tightly the components are structurally aggregated to form a system.

f. Operational containment:
This indicator represents the ability of a modeling tool to capture operational containment relations. It also represents the mechanism that the operational containment relations are captured by the modeling tool.

g. Architectural connectivity:
This indicator is related to the described amount of architectural relations among components and the types of architectural relations that can be specified by the modeling tool. The parametric relations among the components are also considered by this indicator.

h. Operational connectivity:
This indicator is related to the described amount of operational relations among components and the types of operational relations that can be specified by the modeling tool. It also indicates with what fidelity operational connectivity is described by parameters and attributes. Finally, it boils down to the ability of a modeling tool to capture all required attributes and parameters.

i. Architecture-operation relation:
Operations may change the architecture of a component its, and changing architectural attributes will affect the operations provided by a component. The range of parametric relations and the domain-state relations among operations and architecture of a component are captured by this indicator.

A3. Addressing complexity of CPSs

Complexity is an intrinsic characteristic of CPSs and it affects modeling and simulation opportunities. It implies large number of functional and structural dependencies among the components within the system, and among the system component and the embedding environment. Complexity typically treated by a reductionist approach, but in the case of intrinsically connected CPSs this usually cannot be applied. Complexity is against simplification of the information structures that the system designers should specify and/or process in their design tasks. As the common solution for facilitating model specification tasks, available modeling tools provide model-building libraries. The indicators of this aspect are given below:

j. Component libraries:
   Mentioned above, a common solution used by modeling tools to address compositional complexity is providing libraries of a wide range of potential components as modeling elements. The elements and the libraries are usually manually created with various level of support from the modeling tools. This indicator is supposed to provide information about how wide and sophisticated the components libraries offered by the systems are, and if they provide any semantical grouping and interface specifications.

k. Component creation:
   The main task of CPS designers is system-level conceptualization through tackling composability challenges. Involving them in creation of component models would distract them from their main task and would draw them into an unnecessary loss of awareness of the system as a whole. Therefore, this indicator considers how much and in what forms the designer is supposed to be engaged in component creation procedures in the case of various tools.

l. System-level view:
   Designing of cyber-physical systems requires the parallel application of a system level-view and a component-level view. Nonetheless, many systems provide more support for working in component-level views, than for working in system-level views. This indicator is introduced to provide information about how much the methods implemented in a modeling tool support system-level operations and representations. It is also supposed to provide information about how large amount of information is needed to work on system-level in addition to that is needed for working on component level.

m. Handling Multiphysics:
   Modeling the operation of cyber-physical systems makes consideration of multi-physical processes inevitable. In order to provide proper support for operation modeling, modeling tools are supposed to be able to handle multiple physical phenomena and processes in synergy. This indicator is introduced to describe the extent with which a modeling tool can capture multi-physics and to provide the information about the amount of data that is needed for the computations.

A4. Addressing information comprehensiveness

CPS modeling needs huge amount and various pieces and structures of information. This is strongly related to the needed fidelity of architectural and operational modeling. In simple

words, the indicators introduced in association with this aspect specify how completely (sufficiently) a CPS can be modeled by the set of information processed by a tool. Like in the case of many of the above discussed indicators, this can obviously be only a qualitative indication, rather than a quantitative account. It has to be also added that comprehensiveness of information processing is obviously a relative concept, thus we needed to specify the information supposed to be required for a comprehensive modeling a system. Our assumptions concerning this issue have been discussed in the previous chapters. In our investigation we considered it as reference. Based on this, the following indicators have been introduced:

n. Operation/architecture-based modeling:
   There is a need for a proper composition of the model descriptive information to facilitate both aspects of CPS modeling. Since some tools target behavioral and operational modeling of systems, they mainly focus on capturing information related to this objective. Other tools aim at architectural modeling and collect and process information for this purpose only. This indicator was introduced to express the balance between the two information sets, as well as the quality of interoperation of these two sets without any concrete numerical measures.

o. Temporal considerations:
   Both the operation in the physical space and the operation in the cyber space of CPSs need to consider temporal aspects (e.g. timestamps, durations, touchpoints, etc.). Accordingly, two main approaches and methods of handling temporal issues in a modeling tool have been identified: (i) continuous-time-based and (ii) discrete event-based specification of operations. (Continuous time specification is normally useful for capturing functions of HW constituents and discrete-time specification is needed for SW and CW constituents.) This indicator expresses how much these timing approaches are treated in synergy, and how much they complement each other in the functionality of modeling tools.

p. System-level conceptualization:
   This indicator evaluates the ability of a tool to support system-level conceptualization, architecting, and behavioral design of a system. It reflects the functional elements of the support tools that are dedicated to particular conceptualization tasks and address composability. (It has to be mentioned that our investigation did not extend to functional elements that would support compositionality in conceptualization of CPSs).

q. System-level engineering:
   System-level engineering involves not only the development of a system, but also life cycle management of CPSs. In the context of this promotion research only the firstly mentioned activities and the technological knowledge domain in which systems engineers operate were considered. This indicator evaluates ability of a tool to provide analytical, and optimization facilities for a deep investigation of the functionality (operation and behavior) of the modeled cyber-physical system. It also evaluates how much the detailed information about the components supports it.

r. Inclusiveness of information:
   Physicality-oriented modeling of cyber-physical systems necessitates the availability of not only architecture and operation information, but also morphological, geometric,

attributive, appearance, performance, etc. information. These various sets of information should be 'processable' by the tool without causing a cognitive overload for designers. This indicator is intended to express how comprehensively the information of the above clusters are captured by the systems, transferred to the model, and presented in views of the modeling tools.

A5. Addressing tool implementation

The last aspect of comparison is concerned with the issues related to implementation of modeling tools. Six indicators have been considered from this aspect. Some of these reflect identity characteristic of the modeling tools, while other capture usability and utility characteristics. The following indicators are associated with implementation.

s. Category of modeling tool:
Recall that modeling tools can be categorized into different groups based on their modeling mechanisms. Actor-oriented tools, block diagram tools, acausal equation-based tools, and feature-based tools are examples of categorization according to implementation. Each category has strengths and weaknesses with regards to application in pre-embodiment modeling of CPSs that are intrinsic due to the chosen method of modeling. This indicator has been considered to correlate the implementation category of tools with their ability to support pre-embodiment modeling of CPSs.

t. Manual programming facility:
Manual programming helps designers extend the abilities of a modeling tool for solving emergent modeling problems. The commercial and proprietary modeling tools usually have their own programming languages and syntax. These syntaxes are typically similar to standard programming languages, e.g. C, Java. This indicator is intended to express how much the (manual) programming facilities incorporated in a tool allows tailoring it to pre-embodiment modeling of CPSs.

u. Modeling view:
It is common that modeling tools support one or multiple views of modeling. The supported views are associated with particular sets of information and also influenced by the specific system design tasks. These views often appear as visual means of interaction between system designers and modeling tools. This indicator intends to capture the range of views that are supported by the modeling tools, as well as the measure of supporting system design tasks in the offered views.

v. Hardware-in-the-loop support:
The real-time data acquisition and control is a typical feature of all CPSs. Hardware-in-the-loop can be used for providing inputs for the modeled system. This assumes a relatively short bridge between data acquisition devices and system interfaces. This indicator is considered to express the opportunity and the extent of including real-life data sources in inputting data for system-level conceptualization and architecting.

w. Programming language:
The language used for programming a modeling tool very much affects the performances of the modeling tool. For instance, using Java reduces the speed of information processing in comparison with C codes. The performance becomes more crucial in simulation of large-scale complex models, when a huge amount of information is to be

processed instantly, or at designing timed data transfer on networks. In these cases process parallelization also plays a role. This indicator is to express the relationship of performance of the modeling tools and the language(s) used in its implementation.

x. Semantic richness:
Semantics and interpretation of semantics play a crucial role in future CPS modeling systems. Semantics, on the one hand, helps interpretation by the modeling tool, on the other hand, it can control decision making in the entire process of system modeling. Semantics processing-enabled modeling supports interpretation of modeling elements, symbols, and data. Semantics goes beyond operations on syntax level (combinatorics of modeling units and elements) and paves the way to pragmatic approaches (and modeling tools), which also consider the optimal way of achieving the objectives of system modeling. This indicator was defined to express how much attention processing of semantics received in the development of a particular modeling tool, what functionalities have been included for this purpose, and what level of support designers can profit from.

6.5 RESULTS OF THE SURVEY

6.5.1 Execution of the comparative study and processing the responses

The conducted comparative study benefited from two sources. There was a literature study conducted with a focus on the nature of the benchmarked modeling tools and languages, their functionality, strengths and weaknesses. Altogether more than 300 papers were studied and some 65 were found absolutely relevant. This not only provided sufficient insights about the frameworks, but also helped us to find information gaps in the literature. Finding proper explanations in the case of these information gaps and confirming the findings of literature study encouraged us to include a second source of knowledge. Accordingly, as a second approach a survey was planned with experts, who were requested to fill in an online questionnaire. The questionnaire included six groups of questions - starting with general questions and asking step by step more concrete technical questions. In order to validate the correctness of the responses, multiple questions were formulated (from the same aspect, but with different inquiry perspectives). The questionnaire included questions with multiple-choice, check-boxes, linear scale, and grid choice answers. Filling in the questionnaire took 10-15 minutes in average for true experts.

The link to the questionnaire was shared with expert groups and individual experts/professional on LinkedIn, ResearchGate, and other technical forums/websites. The databases of Google Scholar, Scopus, Web of science, and Springer were used to identify relevant scholars, who recently authored publications about the modeling frameworks chosen for benchmarking. The information about the link was sent to the assembled contact list one by one, to invite the scholars to participate. Altogether 145 responses were recorded at the time of downloading the results in the form of spreadsheet. The spreadsheet is primarily organized in MS Excel and imported to IBM SPSS for thoroughly processing. Several filters were applied to increase reliability of answers, excluding the errors and outliers. The questionnaire used in the study is available in Appendix 1. The responses were analyzed, compacted, and summarized as below:

6.5.2 Processing the responses to questions about the basics

The first part of the questionnaire included a question (Q1.1) for selecting a modeling framework. The respondents were asked to choose one or more modeling tool or language they preferred to provide information about. Then, a question (Q1.2) was asked about the background knowledge and expertise of the respondents in order to see their competences and relevance to the chosen modeling framework. Based on the responses, the expertise related to modeling tools diagram has been generated, which is shown in Figure 6-4. It can be seen that a high percentage of people who provided information about Simulink and Modelica were involved in mechatronics and CPSs design. However, the respondents who had experience with SysML were mostly software architects. In general, the respondents who were familiar with using LabVIEW had electronic engineering background. In the case of Ptolemy II, the background knowledge was related to embedded systems and software design. However, the respondents who have chosen Ptolemy II mostly have expertise related to CPS design.

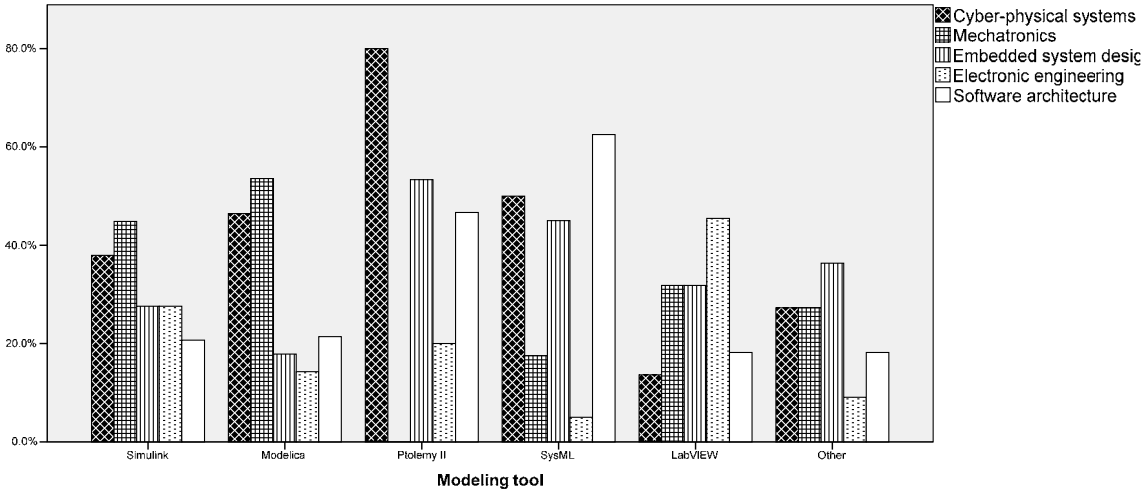


Figure 6-4 Expertise/modeling tool relations

The third question (Q1.3) concerned to tools developed based on Modelica and SysML. As it mentioned before, Modelica and SysML are popular programming and modeling languages which have been considered as the basis of many other tools. Therefore, the rationale behind this question was that supposingly many variants of the tools in question may be used by the respondents. The Modelica-based tools most frequently used by the respondents were Dymola and OpenModelica, respectively. The variety of the SysML-based tools was much wider. As examples, MagicDraw, Eclipse Papyrus, and IBM Rational Rhapsody were identified as the most frequently used ones in this family. The complete overview of the responses is shown in Figure 6-5.

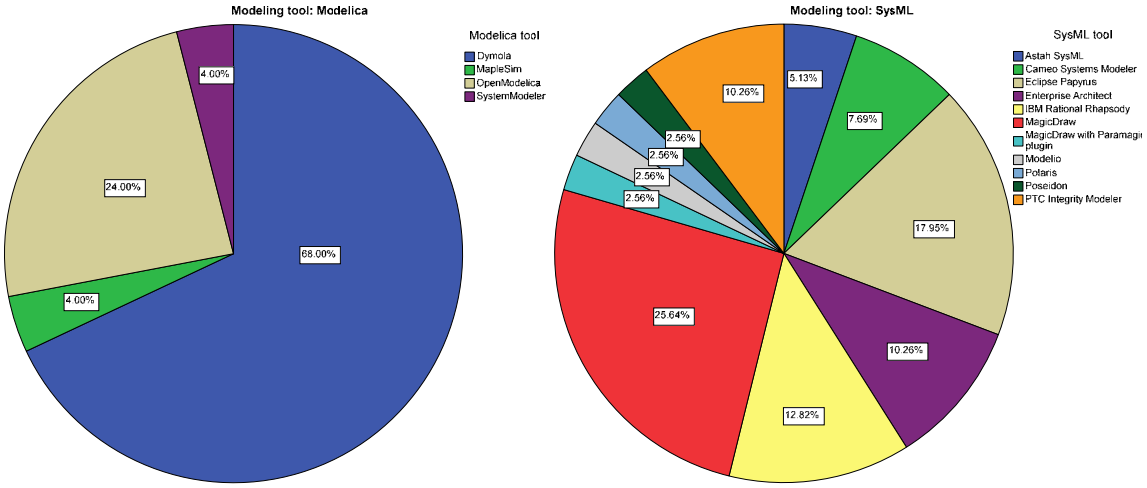


Figure 6-5 Tools used more frequently based on Modelica and SysML

6.5.3 Processing the responses to questions about the concerns of design

The next questions (Q2.1-Q2.2) focused on the concerns of design addressed by the tools. Although, the respondents were system experts and researchers, their answers were somewhat biased. For this reason we used a third question (Q2.3) to confirm their responses. In addition, their responses were compared with the more detailed questions at the end of questionnaire and filtered accordingly. The obtained data showed that system-level conceptualization and system engineering were seen as two sides of spectrum of modeling concerns. Figure 6-6 shows the opinion of the respondents about suitability of the tool used by they for (i) system-level conceptualization concern, for (ii) system engineering concern, and (iii) the balance between these two research variables.

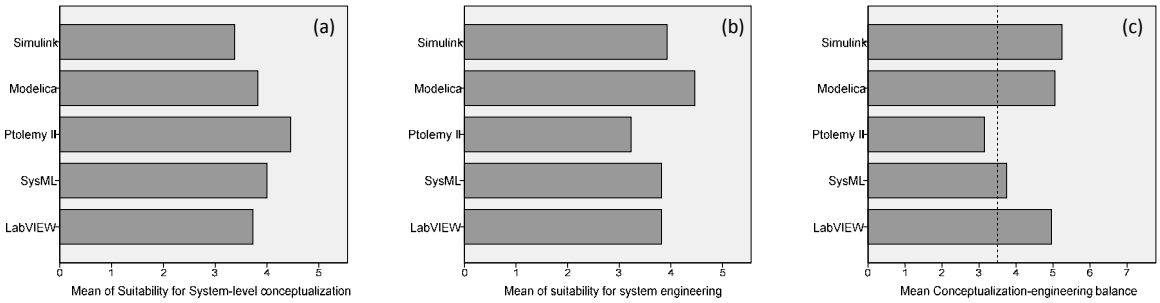


Figure 6-6 Comparison of modeling tools according to the design concerns

It can be seen that Modelica was selected as the most suitable tool for system engineering. However, Ptolemy II was claimed to be the most suitable one for system-level conceptualization. With regards to the balance of appropriateness for both conceptualization and engineering, the respondents were asked to make an evaluation according to a scale of 0 – 7. Here 0 means that no support is provided for system engineering activities, and 7 means that no support is provided for system conceptualization and design activities (Figure 6-6.c). The assessment of the respondents was that Simulink, Modelica and LabVIEW were better in system engineering, and Ptolemy II and SysML kept a balance between the two concerns. It should be noted that Ptolemy II was the only tool that was found to be slightly better in system-level conceptualization.

6.5.4 Processing the responses to questions about the modeling entities

The next questions (Q3.1-Q3.3) interrogated about the modeling entities used by the assessed modeling frameworks. The first question (Q3.1) revealed what constituents could be modeled by them. The modeling tools and languages were not able to equally capture all types of constituents. This is due to the fact that originally these frameworks were developed for specific types of constituents. Figure 6-7 shows the responses of the experts in this regard. For instance, the frameworks of Simulink and Modelica were originally developed for capturing HW constituents.

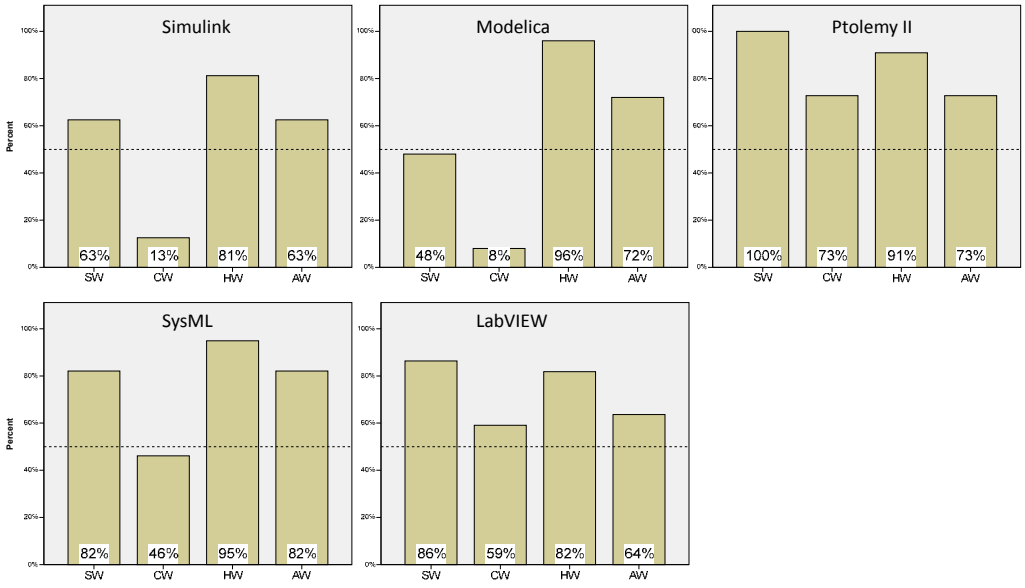


Figure 6-7 Constituents that can be captured by modeling tools

The framework of Ptolemy II and LabVIEW covered almost all types of constituents in modeling, though SW and HW were more supported. SysML could also capture all types of constituents, except CW constituents, which do not receive enough support. As shown in Figure 6-8, the assessed modeling tools were almost similar in terms of their ability to capture HW and AW constituents, but Modelica and Simulink was ranked at the bottom of the list in terms of capturing SW and CW constituents.

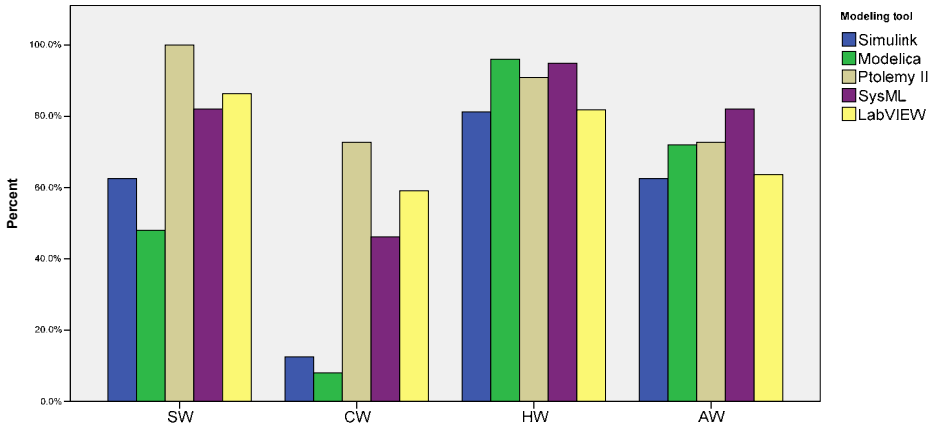


Figure 6-8 Comparison of tools in supporting modeling of various constituents

The question (Q3.2) was concerning modeling the interoperation of constituents captured by the modeling tools resulted in an interesting insight. Figure 6-9 shows the answers to this question. Diagram (a) shows the percentages of the claims that interoperations of heterogeneous constituents is supported, and diagram (b) shows the percentage of the claims that interoperations of heterogeneous constituents is not supported by the respective tools, or if answer was not provided by the respondents. Based on the answers we concluded that the frameworks of Modelica and Simulink were not designed to capture heterogeneous interoperation of SW constituents. However, both frameworks gave consideration to HW and AW interoperations. SysML and Ptolemy II showed higher degree in capturing interoperations, which is not surprising due to the fact that they apply higher level abstraction, but capturing semantics-richness is rather limited in the case of these frameworks. Since they do not consider intrinsic differences of HW, SW, and AW constituents in modeling, capturing interoperations could be realized easier.

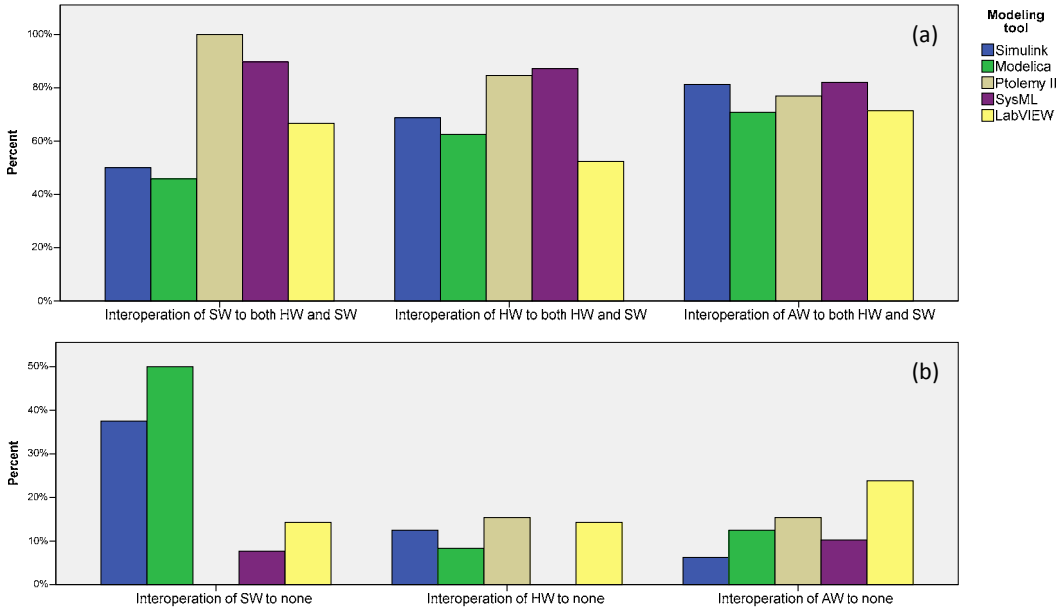


Figure 6-9 Comparison of the modeling tools in supporting interoperation of constituents

In the next question (Q3.3) ,the respondents were asked to identify the abilities of the modeling tools to capture three main types of architectural attributes: (i) topological relations among components of a system, (ii) morphology (e.g. 3D shapes, volume) of components, and (iii) physical attributes (e.g. material, flexibility, resilient) of components. According to the responses, Modelica was the best framework for capturing architectural attributes, and the only framework that was developed for capturing morphological attributes of components (Figure 6-10). The other frameworks, however, partially support it with the aid of some add-on products, or in connection with other complementary tools. On the other side of spectrum, respondents argued, Simulink did not have the ability of capturing architectural attributes, as its main focus was on functional attributes. The physical attributes of components were partially included in the modeling functions of Simulink. There was a remark that SysML could not capture morphological attributes of components. Ptolemy showed a reasonable capability to capture topological relations, but morphological and physical attributes of components are mostly neglected by its framework. Our conclusion is that architectural attributes could not be handled by most of the modeling tools due to the abstractions they applied.

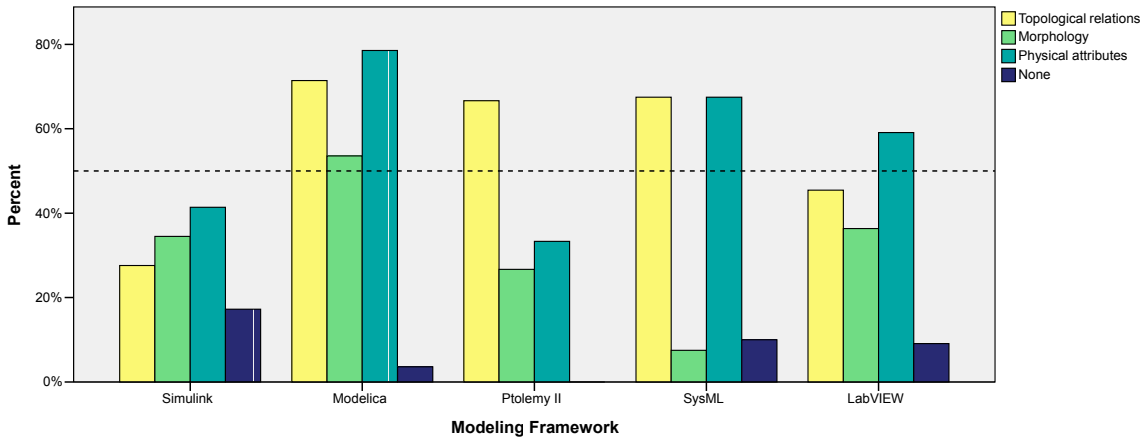


Figure 6-10 Comparison of abilities of modeling tools in capturing architectural attributes

6.5.5 Processing the responses to questions about the modeling aspects

Related to the modeling aspects, there was a question (Q4.1) posed about the basis modeling. The respondents were asked to specify how the modeled system is captured in the chosen tool. The provided options were: (i) as composition of architectural components, (ii) as composition of functions of its components, and (iii) as integration of functional and architectural aspects. As it is shown in Figure 6-11, Simulink is the only tool, which received more votes for the second option, than for the third option. Furthermore, it has the lowest percentage with respect to supporting architecture-based design. This result implies that architecting of components was not considered initially in the framework of Simulink. A high percentage of the respondents answered that Modelica and Ptolemy model a system based on integration of architecture and function. However, the pieces of information captured about architectural attributes are of significantly different meaning in the case of these two frameworks. An interesting outcome was that SysML had the highest percentage of votes with regards to architecture-based modeling, and obtained no vote for function-based modeling. The answers of some respondents indicated that they did not believe that the facilities provided by SysML for modeling functions of components are fully integrated with the architectural aspects of components. The responses discussed below provided further clarification on this issue.

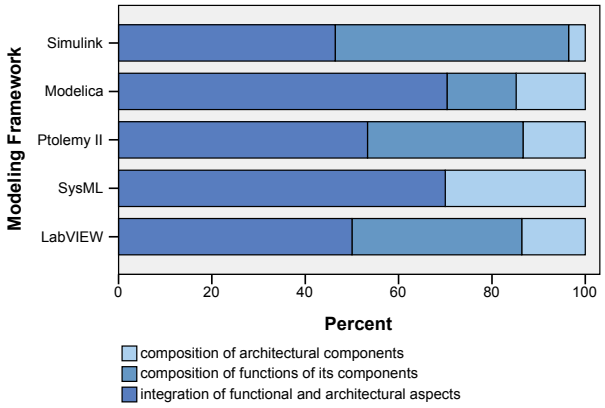


Figure 6-11 Comparison of the modeling basis of the frameworks

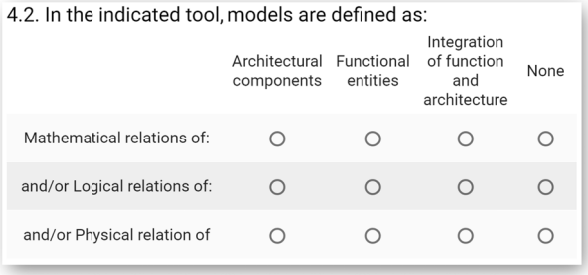


Figure 6-12 The grid-type question about types of relations

The multi-choice question (Q4.2) concerning the types of relations handled in modeling was visually presented as a grid type of question shown in Figure 6-12. By clicking on the respective buttons, the respondents could specify what type of relations could be specified among components, and in what form were they captured by the modeling tools. The options in the rows ask about the relations. The answers could be selected from the following options: (i) mathematical relations, (ii) logical relations, (iii) physical relations, or (iv) a combination of them. The results were filtered according to the answers to the previous question, since the options in the columns resemble the options that were addressed by the previous question.

Figure 6-13 shows four diagrams that visualize the results of responses to question (Q4.2). As diagram (a) shows, the architectural relations of components were mostly prescribed as physical relations by modeling tools. However, the highest percentage of votes, about 28 percent, was in the case of SysML, and this most probably happened owing to some add-ons functionality. It was claimed by the respondents that LabVIEW and Ptolemy capture the relations among architectural components as logical relations. It can be seen in diagram (b) that Simulink was claimed to capture functions of components as logical and mathematical relations. The same claim was made in the case of LabVIEW. This graph also shows that in the case of Ptolemy functional relations were said to be mathematically captured. The lowest percentage of votes on the approach to capturing functional relations was for SysML. Diagram (c) shows that Modelica received the highest number of

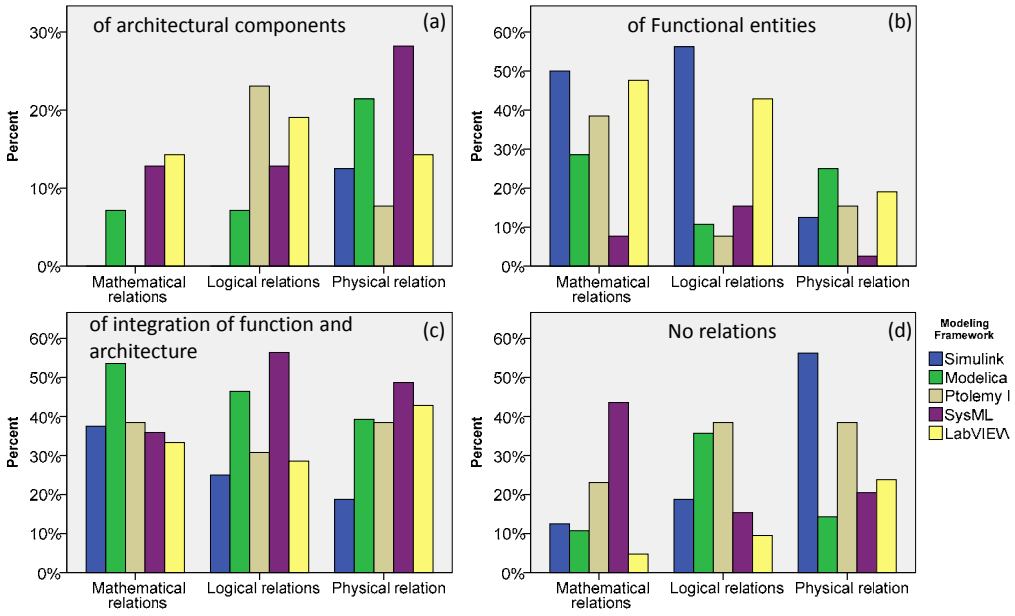


Figure 6-13 Comparison of the four columns included in the grid-type question

votes concerning the fact that it mathematically specifies architecture-function integration. At the same time, SysML was mostly claimed to capture architecture-function relations logically. Simulink was found as the weakest in physically capturing architecture-function relations. For the sake of completeness, we included diagram (d) in Figure 6-13. This indicates the percentage of the answers which did not provided information about the capability of the system managing relations. According to the answers, with regards to mathematical relations, SysML received the largest number of votes, while with regards to capturing physical relations Simulink and Ptolemy received the largest number of votes.

Unfortunately, based on the selections it was not possible for us to conclude if this meant that the tool in question was not able to handle relations, or the respondents did not have proper information about this.

Concerning the managed function-architecture relations, the question (Q4.3) interrogated about the way of establishing relations by the modeling system. The selectable options were: (i) there is no link between these two aspects, (ii) linked by timing contract, (iii) linked through functional and architectural variable parameters, (iv) state-transitions are linked to states of architectural domains, (v) each function is defined linked to an architectural domain, (vi) other form of function-architecture relation specification is applied. The percentages of the votes on how the function aspect was associated with the architecture aspect were very different (Figure 6-14).

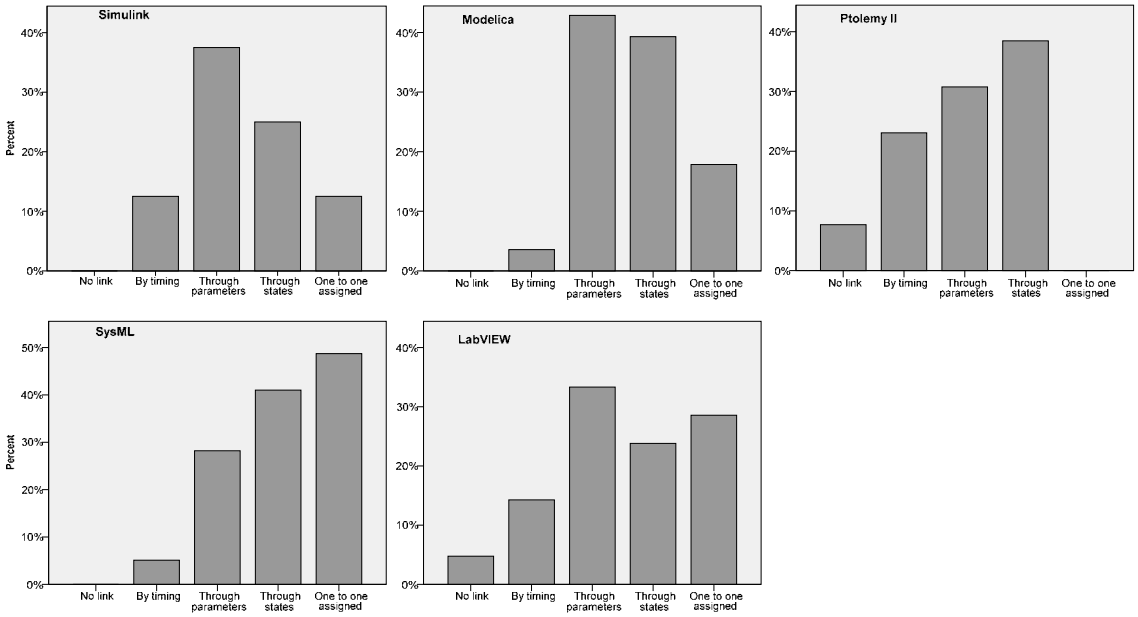


Figure 6-14 Comparison of four columns of the questions about function-architecture relation

It could be concluded the Simulink and Modelica managed these relations through parameters, while Ptolemy captured the relations mostly through states, and partially through parameters and timing. Ptolemy obtained the highest percentage of votes concerning the fact that it created timed relation between functionality and architecture. The respondents indicated that SysML created one-to-one relations by assigning the functional entities and architectural components. It seems that LabVIEW is more balanced with regards to linking functionality and architecture through parameters, states, or one-to-one assignment.

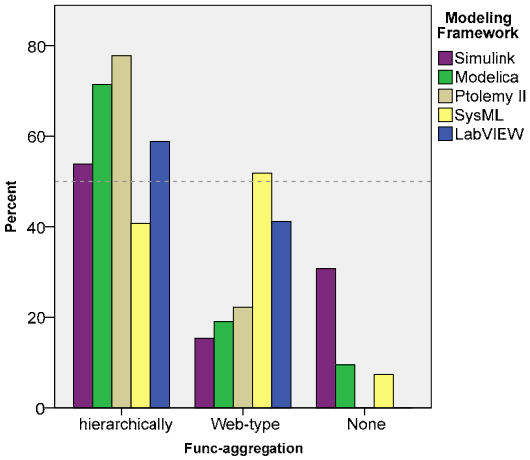


Figure 6-15 Comparison of methods of function aggregation

The next question (Q4.4) concerned the way of function aggregation by the modeling frameworks. In general all of them provided support

for function aggregation as shown in Figure 6-15. The question offered three options: (i) hierarchical, (ii) web-type, or (iii) none. The responses indicated the most of the systems supported hierarchical aggregation of functions, but were also to some extent believed to able to manage web-type aggregation of functions. SysML proved to be the most supportive tool with regards to this type of function aggregation.

6.5.6 Processing the responses to questions about handling information

The next cluster of questions was related to information handling capabilities of the benchmarked tools. The first question (Q5.1) inquired about the kind of information that could be captured by the studied modeling frameworks. The respondents were asked to share their opinion about what kinds of information were them. The responses are shown in Figure 6-16. It seems that the conceptual frameworks of all modeling tools made them capable to process information about (i) architectural connectivity, (ii) functional connectivity, (iii) function-related parameters, (iv) event-based timing, and (v) parameter constraints. However, none of the tools were claimed to sufficiently manage information about the morphology of components. Simulink was mentioned as having limitations in handling architecture containment, architecture related parameters, and morphology of components. The restriction of SysML in capturing morphology of components, handling continues timing, and multi-physics were also commented upon.

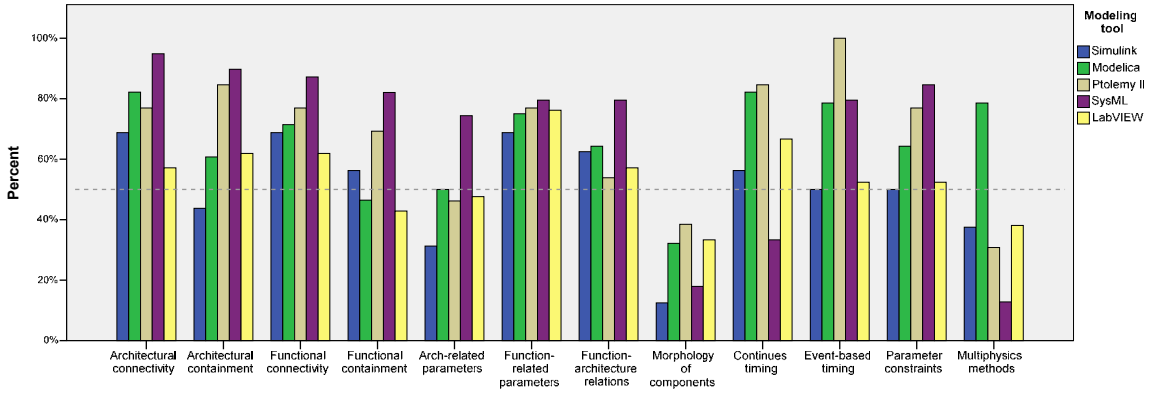


Figure 6-16 Distribution of the opinions on the capability of handling different types of information

The next question (Q5.2) concerned the type of views offered for modeling by the various tools. Figure 6-17 presents the results in this regards. Assembly view is supported by SysML only. Physical 2D view is provided by LabVIEW and Modelica. With the exception of Model-

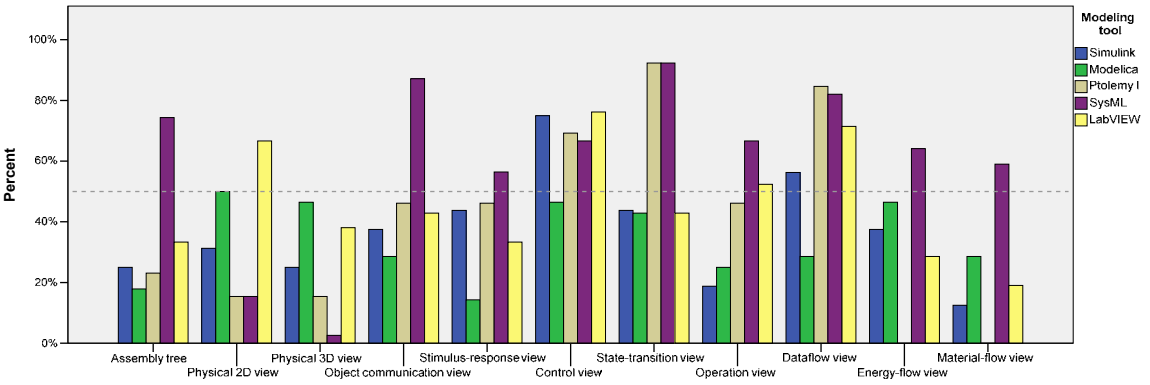


Figure 6-17 Comparison of the views provided by the modeling frameworks

ica, none of the tools provided 3D view. Object communication, energy flow, and material flow views were not considered in any of the tools, apart from SysML. In the case of Ptolemy II, only data-flow view is supported from the abovementioned three views. The answers of the respondents were not always converging, most likely because various add-on and plug-in tools have been incorporated in the benchmarked tools.

6.5.7 Processing the responses to questions about model composition

Related to the cluster of questions concerning model composition, one of the questions (Q6.1) inquired about the possible information exchange links among modeling components. Eight selection options were provided to the respondents to identify the possible information links. Presented in Figure 6-18, SysML supports most of the link options, except assumption-guarantees. The reason is that using semantics was not the objective of the developers of this tool. Information exchange linking through control streams, states, and events are supported by all tools. Linking through physical interaction, energy flow, and material flow information can only be realized by Modelica and SysML. Apparently, linking through assumptions-guarantees was ignored by the modeling frameworks of the studied tools.

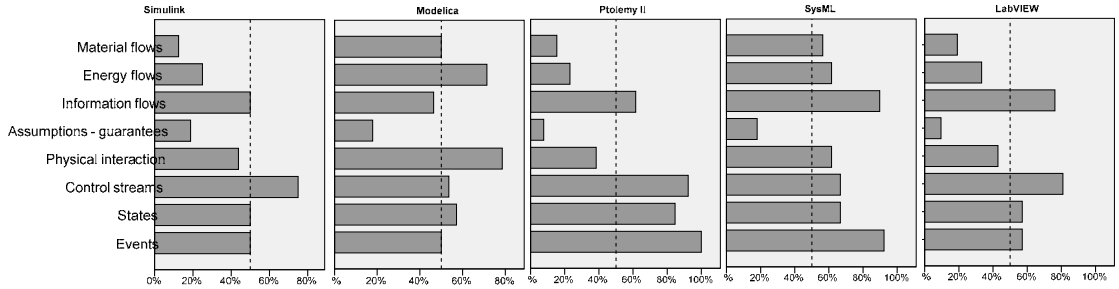


Figure 6-18 Possible information exchange links among modeling components

The next question (Q6.2) was about the effects of using abstraction. Actually, this question was a complementary question that was supposed to prove many assumptions. It specifically asked the respondents to identify information sets that they believed were ignored in favor of applying abstraction. As presented in Figure 6-19, the information about the shape of components is the major issue, since it is ignored by most of the studied frameworks. The information about physical attributes was also neglected, especially by LabVIEW.

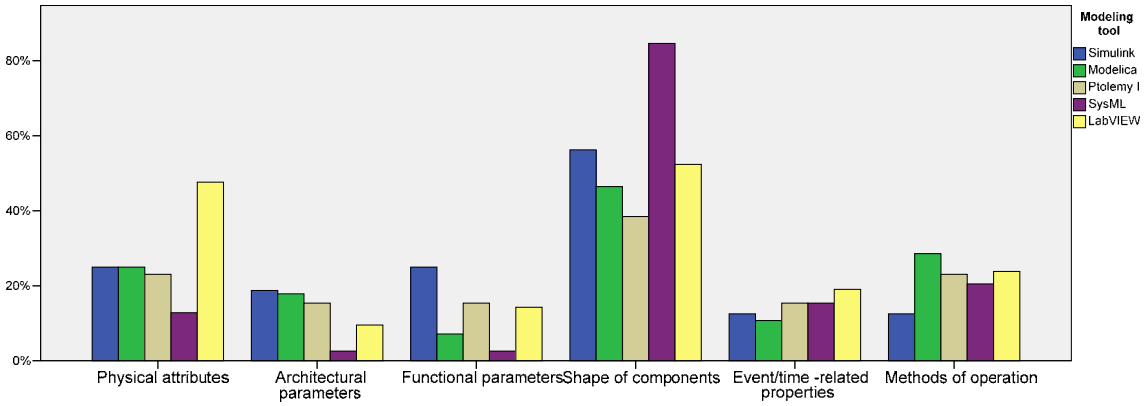


Figure 6-19 Information sets ignored because of abstraction

6.6 MAJOR FINDINGS OF BENCHMARKING

6.6.1 Mapping the results into indicators

The results have been imported into SPSS and research variables have been created based on the questions. Table 6-1 shows the variables and maps them onto indicators the answers imply. There are many interdependencies among the indicators - therefore drawing a sharp demarcation line was not possible. Moreover, projecting the answers to the indicators circumscribed in Sub-section 6.4 could not be done otherwise, but by subjective semantic interpretation and critical reasoning. We tried to use multiple evidences in our reasoning in order to end up with a reasonably sound mapping. The main issue was the multiplicity of the possible mapping actions. With regards to Table 1 it entails that each variable may be associated with several indicators.

Table 6-1 Mapping of the SPSS variables into comparison indicators

Question #	SPSS variables	indicators	Aspects
2.1 System-level conceptualization concern	System-level conceptualization	l, p	A3, A4
2.2 System engineering concern	System engineering	l, q	A3, A4
2.3 Conceptualization-engineering balance	Conceptualization-engineering balance	l, p, q	A3, A4
3.1 Constituents that can be modeled	Supports SW constituents	a, d	A1
	Supports CW constituents	a, d	A1
	Supports HW constituents	a, d	A1
	Supports AW constituents	a, d	A1
3.2 Interoperation of constituents	Interoperation of SW to	a, c	A1
	Interoperation of HW to	a, c	A1
	Interoperation of AW to	a, c	A1
3.3 Ability of capturing architectural attributes	Topological relations	a, g	A1, A2
	Morphology	a, g	A1, A2
	Physical attributes	a, g	A1, A2
	None	a, g	A1, A2
4.1 Modeling basis	Composition of architectural components	i, n	A2, A4
	Composition of functions of components	i, n	A2, A4
	Integration of architecture and functions	i, n	A2, A4
4.2 Types of modeling relations	Mathematical relations of	c, i	A1, A2
	Logical relations of	c, i	A1, A2
	Physical relation of	c, i	A1, A2
4.3 Function-architecture relations	No link	i	A2
	By timing	i	A2
	Through parameters	i	A2
	Through states	i	A2
	One to one assigned	i	A2
4.4 Function aggregation	Function-aggregation	f, p	A2, A4
5.1 The information that can be captured	Architectural connectivity	g, p, r	A2, A4
	Architectural containment	e, p, r	A2, A4
	Functional connectivity	h, p, r	A2, A4
	Functional containment	f, p, r	A2, A4

	Arch-related parameters	e, g, q, r	A2, A4
	Function-related parameters	f, h, q, r	A2, A4
	Function-architecture relations	i, p, r	A2, A4
	Morphology of components	g, p, q, r	A2, A4
	Continues timing	h, o, p, q, r	A2, A4
	Event-based timing	h, o, p, q, r	A2, A4
	Parameter constraints	q, r	A4
	Multiphysics methods	m, q, r	A3, A4
5.2 Provided views	Assembly tree	n, r, u	A4, A5
	Physical 2D view	n, r, u	A4, A5
	Physical 3D view	n, r, u	A4, A5
	Object communication view	n, r, u	A4, A5
	Stimulus-response view	n, r, u	A4, A5
	Control view	n, r, u	A4, A5
	State-transition view	n, r, u	A4, A5
	Operation view	n, r, u	A4, A5
	Dataflow view	n, r, u	A4, A5
	Energy-flow view	n, r, u	A4, A5
	Material-flow view	n, r, u	A4, A5
6.1 Links among modeling components	Events	h, r	A4
	States	g, h, r	A2, A4
	Control streams	h, r	A2, A4
	Physical interaction	g, r	A2, A4
	Assumptions - guarantees	g, r	A2, A4
	Information flows	h, r	A2, A4
	Energy flows	h, r	A2, A4
	Material flows	g, h, r	A2, A4
6.2 Negative effects of abstraction	Physical attributes	b	A1
	Architectural parameters	b	A1
	Functional parameters	b	A1
	Shape of components	b	A1
	Event/time -related properties	b	A1
	Methods of operation	b	A1

6.6.2 Comparing the frameworks according to the benchmarking aspects

The results were mapped and summarized in the following four tables. The first aspect that was mapped is 'addressing heterogeneity' (Table 6-2). Three different approaches of tackling heterogeneity were identified in our studies: (a_1) ignoring the constituents that cause heterogeneity, (a_2) ignoring attributive differences, e.g. shape and physical attributes, and (a_3) ignoring interoperation. The first approach (a1), ignoring constituents such as CW and partly SW constituents, was used by Modelica (as shown in Figure 6-7). This is the approach which was also used by Simulink and SysML. The second approach was used by Ptolemy, LabVIEW, Simulink, and SysML as shown in Figure 6-10. The third approach is used by Simulink and Modelica to address the heterogeneity problem in modeling (Figure 6-9).

Applying abstraction and the level of abstraction have a direct implication on the sets of information that are not considered as a result of. Figure 6-19 shows the related comparison. Six issues were considered for comparing the applied abstraction from different perspective. The percentages of the importance are mentioned in the table. Figure 6-19 shows that SysML had the highest percentage in the benchmarking with regards to ignoring morphological issues and LabVIEW had the highest percentage in ignoring physical attributes due to the applied abstraction.

Table 6-2 Comparison of the modeling frameworks from aspects of addressing heterogeneity

Addressing heterogeneity	Simulink	Modelica	Ptolemy II	SysML	LabVIEW	SMF-TB
a. Method of tackling heterogeneity	a1, a2, a3	a1, a3	a2	a1,a2	a2	Uniform information structure
b. Level of applied abstraction	25.0%	22.6%	21.8%	23.0%	27.8%	None
c. Captured relations	Functional/logical relations	Mathematical/physical relations	Multi-formalism/ logical relations	Logical relations	Control relations	Physical/logical/ mathematical relations
d. Captured types of constituents	HW, SW, AW	HW, AW	SW, HW, AW, CW	HW, AW, SW	SW, HW, AW, CW	Equally all

The second aspect to consider was the aspect of 'addressing integrity'. This is closely related to the specification of architectural and operational containment. Architectural containment was considered by all modeling frameworks except by that of Simulink. It seems that operational containment is not sufficiently considered in the conceptual framework of LabVIEW and Modelica. However, they support some level of hierarchical operation containment relations. Simulink and Ptolemy II supported hierarchical operation containment. The only framework that supported web-type operation containment was SysML.

According to the first bar-set of Figure 6-16 (architectural connectivity), all frameworks supported architectural connectivity. The fifth bar-set (arch-related parameters) and the eighth bar-set (morphology of components) display that the respondents were aware of the ability of the modeling tool to handle various component relationships. However, by this indicator we intend to express if all related information about the attributes of connections are captured. Our conclusion has been that the framework of Simulink was not developed to capture architectural attributes. Morphological connection attributes are not supported by any of the modeling frameworks. As shown in Figure 6-10, topological information about architectural connectivity can be captured by Modelica, Ptolemy, and SysML. However, Figure 6-18 shows that assumption-guarantee was not considered in the frameworks of the studied modeling tools (e.g. for defining architectural connectivity constraints).

All studied modeling frameworks support functional connectivity. In Figure 6-16, the third and sixth bar-sets are the evidence for this. However, as it can be seen in the same figure, the ninth bar-set shows that the framework of SysML is not able to formulate continuous timing (CT) in operation connectivity. Figure 6-18 provides more information about handling functional connectivity relations. Although all frameworks consider events (E), states (S), and control streams (CS) in functional connectivity, Modelica framework does not

sufficiently support information flows (IF), and the frameworks of Simulink, Ptolemy, and LabVIEW have not been developed to consider energy flows (EF) and material flows (MF) among operations. Nevertheless, they are able to partially capture these pieces of information with the aid of add-on products, manual formulation, or connection to other software tools.

It has been mentioned before that all studied frameworks, except that of Simulink, support integration of architecture and operation (Figure 6-11). Though, the integrity of components varies according to the information captured to represent architecture-operation relationships. Figure 6-14 provided further information to evaluate the integrity of components, namely: through parameter (P), through states transition (ST), though timing contracts (TC), and one-to-one (O) relations.

Table 6-3 Comparison of the modeling frameworks from aspects of addressing integrity

Addressing integrity	Simulink	Modelica	Ptolemy II	SysML	LabVIEW	SMF-TB
e. Architectural containment	No	Yes	Yes	Yes	Yes	Yes
f. Operational containment	Hierar-chical	Hierarchical (partly)	Hierarchical	Web-type	Hierarchical (partly)	Web-type
g. Architectural connectivity	No	Topology	Topology	Topology	Information and control	Morphologi-cal/physical interaction, assumption-guarantee
h. Operational connectivity	CT, E, S, CS, CT, IF	CT, E, S, CS, EF, MF	CT, E, S, CS, IF	E, S, CS, IF, EF, MF	CT, E, S, CS, IF	CT, E, S, S, CS, IF, EF, MF
i. Architecture-operation relation	P	P	P, ST, TC	O	P, ST, O	P, ST, O

In the third stage of our benchmarking, the aspects of 'addressing complexity' were considered (Table 6-4). Modelica had the largest library of components considering all other modeling tools. A wide variety of components are covered by the commercial or non-commercial libraries of Modelica. The libraries of LabVIEW contains components that are developed and produced by NI, i.e. codes of NI components, data-acquisition, signal generation, mathematics, statistics, signal conditioning, and analysis libraries. The other frameworks also provide libraries that are named differently, e.g. block library in the case of Simulink (i.e. functions block, structural block, algorithmic block, and dynamic block), actor library for Ptolemy, and unit library for SysML. There is no indicator for comparing libraries defined and organized according to different modeling frameworks. With a view to addressing composability and complexity, the libraries of Modelica were recognized as the most comprehensive and intraoperative. However its libraries are mostly developed for HW and AW constituents. LabVIEW has a more limited library. Simulink library is a library of functions, rather than a library of components. Ptolemy II and SysML have very limited libraries – this aspect was not sufficiently considered initially in their frameworks.

Regarding the 'component creation' indicator, Modelica was found to be better than the other tools, and LabVIEW is the second to that. In the other modeling frameworks, the components should be created manually. However, there is a possibility of reusing components in Ptolemy II. The next indicator (system-level view) evaluates the ability of the

modeling tools to reduce the complexity implied by the details of components in system-level design. According to Figure 6-6.a Ptolemy II, SysML, Modelica, LabVIEW, and Simulink have been sequentially ranked on the 1st to 5th places. Modelica had the only framework that was developed to support Multiphysics. Although there are add-on tools in the case of the other tools that facilitate processing Multiphysics, their frameworks were initially not designed to handle it.

Table 6-4 Comparison of the modeling frameworks from aspect of addressing complexity

Addressing complexity	Simulink	Modelica	Ptolemy II	SysML	LabVIEW	SMF-TB
j. Component libraries	3rd (block library)	1st (model library)	4th (actor library)	4th (unit library)	2nd (NI products)	PT library
k. Component creation	4th	1st	3rd	4th	2nd	Delegated to knowledge engineers
l. System-level view	5th	3rd	1st	2nd	4th	2nd
m. Multiphysics	No	Yes	No	No	No	Yes

The fourth aspect implies a group of indicators that address 'information comprehensiveness' (Table 6-6). As a starting point, we considered the bases of modeling. Simulink is primarily a function-based modeling tool. Modelica has an architecture-oriented (component-based) framework that also captures operations of components. Ptolemy II is built on an actor-based conceptual framework, in which actors represent operations of components in a system. Although actors represent domains, they do not capture sufficient information for representation of architectural attributes. SysML works according to both an architecture-based and an operation-based modeling framework. However, integration of the two sides is not sufficiently supported. LabVIEW is a function (control)-based modeling framework that logically links operations to architectural domains (components). About the temporal consideration, all studied frameworks, except SysML, support continuous timing (CT). Event-based (EB) timing is also supported by all studied modeling frameworks.

Regarding 'system-level conceptualization' indicator, we considered the votes of respondents with the general applicability (possibility to apply in general system conceptualization) of the tools. The reason is that, apparently, the meaning of system was different for the respondents of the questionnaire. For instance, Ptolemy was originally dedicated and had many applications in the design of embedded system, and in system-on-a-chip design. As a consequence, 'system conceptualization' means different for its responding users, than for genuine CPSs designers. According to our reasoning, SysML had the best framework for system-level conceptualization purposes, followed at a considerable distance by Modelica, Ptolemy II, LabVIEW and Simulink, respectively. For system-level engineering, Modelica and Simulink were found to be the best, followed by LabVIEW, SysML and Ptolemy II, in the given order.

For evaluating comprehensiveness of captured information, responses of four questions (Q5.1, Q5.2, Q6.1, and Q6.2) were analyzed and refined to provide scores. Table 6-5 shows the mean each piece of information receives according to the answers of the respondents. SysML has the highest score (18.54), and followed by Ptolemy II (15.47), Modelica (14.49), LabVIEW (13.6), and Simulink (11.91).

Table 6-5 Percentage of the information comprehensiveness according to the respondents

Ques-tions	indicators	**Simulink**		**Modelica**		**Ptolemy II**		**SysML**		**LabVIEW**	
5.1. The information that can be captured	Architectural connectivity	0.69		0.82		0.77		0.95		0.57	
	Architectural containment	0.44		0.61		0.85		0.9		0.62	
	Functional connectivity	0.69		0.71		0.77		0.87		0.62	
	Functional containment	0.56		0.46		0.69		0.82		0.43	
	Arch-related parameters	0.31		0.5		0.46		0.74		0.48	
	Function-related parameters	0.69		0.75		0.77		0.79		0.76	
	Function-architecture relations	0.63		0.64		0.54		0.79		0.57	
	Morphology of components	0.13		0.32		0.38		0.18		0.33	
	Continues timing	0.56		0.82		0.85		0.33		0.67	
	Event-based timing	0.5		0.79		1		0.79		0.52	
	Parameter constraints	0.5		0.64		0.77		0.85		0.52	
	Multiphysics methods	0.38		0.79		0.31		0.13		0.38	
5.2. Provided views	Assembly tree	0.25		0.18		0.23		0.74		0.33	
	Physical 2D view	0.31		0.5		0.15		0.15		0.67	
	Physical 3D view	0.25		0.46		0.15		0.03		0.38	
	Object communication view	0.38		0.29		0.46		0.87		0.43	
	Stimulus-response view	0.44		0.14		0.46		0.56		0.33	
	Control view	0.75		0.46		0.69		0.67		0.76	
	State-transition view	0.44		0.43		0.92		0.92		0.43	
	Operation view	0.19		0.25		0.46		0.67		0.52	
	Dataflow view	0.56		0.29		0.85		0.82		0.71	
	Energy-flow view	0.38		0.46		0		0.64		0.29	
	Material-flow view	0.13		0.29		0		0.59		0.19	
6.1. links among compo-nents	Events	0.5		0.5		1		0.92		0.57	
	States	0.5		0.57		0.85		0.67		0.57	
	Control streams	0.75		0.54		0.92		0.67		0.81	
	Physical interaction	0.44		0.79		0.38		0.62		0.43	
	Assumptions - guarantees	0.19		0.18		0.08		0.18		0.1	
	Information flows	0.5		0.46		0.62		0.9		0.76	
	Energy flows	0.25		0.71		0.23		0.62		0.33	
	Material flows	0.13		0.5		0.15		0.56		0.19	
6.2. negative effect of abstraction	Physical attributes		0.25		0.25		0.23		0.13		0.48
	Architectural parameters		0.19		0.18		0.15		0.03		0.1
	Functional parameters		0.25		0.07		0.15		0.03		0.14
	Shape of components		0.56		0.46		0.38		0.85		0.52
	Event/time -related properties		0.13		0.11		0.15		0.15		0.19
	Methods of operation		0.13		0.29		0.23		0.21		0.24
	Sum	13.42	1.51	15.85	1.36	16.76	1.29	19.94	1.4	15.27	1.67
	Total	**11.91**		**14.49**		**15.47**		**18.54**		**13.6**	

It is worth mentioning that the total scores do not show the real ability of the frameworks, since it is affected by the opinion of the users and might be biased. Moreover, the respondents evaluate the modeling tools which equipped with multiple add-in products. Consequently, reasoning back to the ability of the frameworks in this case is not possible and the scores do not reflect valid evaluation of the frameworks.

Table 6-6 Comparison of the modeling frameworks from aspect of addressing information comprehensiveness

Reflecting info comprehensiveness	Simulink	Modelica	Ptolemy II	SysML	LabVIEW	SMF-TB
n. Opera-tion/Architecture based modeling	Function-based	Component-based (mathematically linked to operations)	Actor-based (representing operations of components)	Logically linked architecture-based and function-based	Function-based (logically linked to components)	SMF-based (tight integration of operations and architecture)
o. Temporal considerations	CT, EB	CT, EB	CT, EB	EB	CT, EB	CT, EB
p. System-level conceptualization	3th	3rd	3rd	1st	4th	2nd
q. System level engineering	2nd	1st	5th	4th	3rd	Comparable with Modelica
r. Comprehensive information	11.91	14.49	15.47	18.54	13.6	NA

The last aspect of benchmarking concerned the 'implementation of the tools'. It was mentioned that Simulink was implemented as a block diagram language, Modelica as an acausal mathematic equation-based language, Ptolemy as an actor oriented modeling tool, SysML as a logical modeling language, and LabVIEW originally as a data-acquisition design tool. If we compare the proposed SMF-TB with these, the principal difference is that it is being implemented as a system-level feature-based modeling tool. It is worth mentioning that in some of the frameworks, the possibility of manual programing is also considered, as shown in Table 6-7.

Various views were considered in the frameworks of the benchmarked tools. In the questionnaire, the hypothesized views were: assembly tree view (AT), physical 2D view (P2), physical 3D view (P3), object communication view (OC), stimulus-response view (SR), control view (CV), state-transition view (ST), operation view (OV), dataflow view (DF), energy-flow view (EF), and material-flow view (MF). The results generated by the synthesis of the answers of respondents are presented in. It should be noted that many views are provided by the tools and add-in products that are not considered here.

Table 6-7 Comparison of the modeling frameworks from aspects of tool implementation

Referring to tool implementation	Simulink	Modelica	Ptolemy II	SysML	LabVIEW	SMF-TB
s. Modeling tool category	Block diagram language	Acausal mathematic based	Actor-oriented multi-formalism	logical modeling	data-acquisition	System-level feature-based
t. Manual programming facility	Matlab, Fortran, ADA, C	Java-, Matlab-like syntax	Java	NA	MathScript component	NA
u. Modeling view	CV, DF	P2, P3, CV, EF	ST, DF, CV	AT, OC, CV, ST, OV, DF, EF, MF	P2, VC, OV, DF	all views are supported
v. Hardware-in-loop support	Yes	Yes	No	No	Yes	No
w. Codes used for tool develop-ment	Matlab	NA	Java	NA	G	NA
x. Semantic richness	moderate	moderate	moderate	low	moderate	high

Some of the frameworks such as LabVIEW are initially developed to support hardware-in-the-loop for real-time simulation and data acquisition purposes. There is no information about Ptolemy and SysML whether there was an intention to consider HW-in-the-loop in their framework. The other issue is the programming languages used for tool development. Simulink is created based on the Matlab codes, Ptolemy II is written by Java, and LabVIEW is mostly developed based on G language. Modelica and SysML are considered as visual languages that are implemented through different tools. Semantic richness is an important but complicated issue that is difficult to evaluate. We know that SysML is suffering from the lack of proper semantics. The other frameworks are working based on a certain level of semantics, which is in comparison lower than that targeted by SMF-TB, since the latter strongly counts on semantics and ontology.

6.7 ANALYSIS AND DISCUSSION ON THE FINDINGS

The challenges of CPS design are mentioned and discussed before. The requirements to tackle these challenges have also been extracted. The aspects of comparison (Sub-chapter 6.4) are the results of grouping the requirements that manifested as indicators. The indicators were used to specify to what extend the available modeling frameworks address the CPS design challenges and reflected in the questionnaire. We believe that the novelties proposed and implemented in our work provide a proper solution to address the mentioned challenges. The novelties of SMF-based modeling are briefed in Sub-chapter 6.2. Figure 6-20 represents the mapping of these novelties to the comparison aspects and the questions that regarded to them. The following paragraphs represent how our modeling framework fulfils the aspects of comparison in contrast with the studied modeling frameworks. The results of this projection are summarized in Table 6-8. We note that the pieces of information obtained from the two sources (responses to questionnaire, briefed in Sub-chapter 6.6, and the literature study, summarized in Sub-chapter 6.3) were blended.

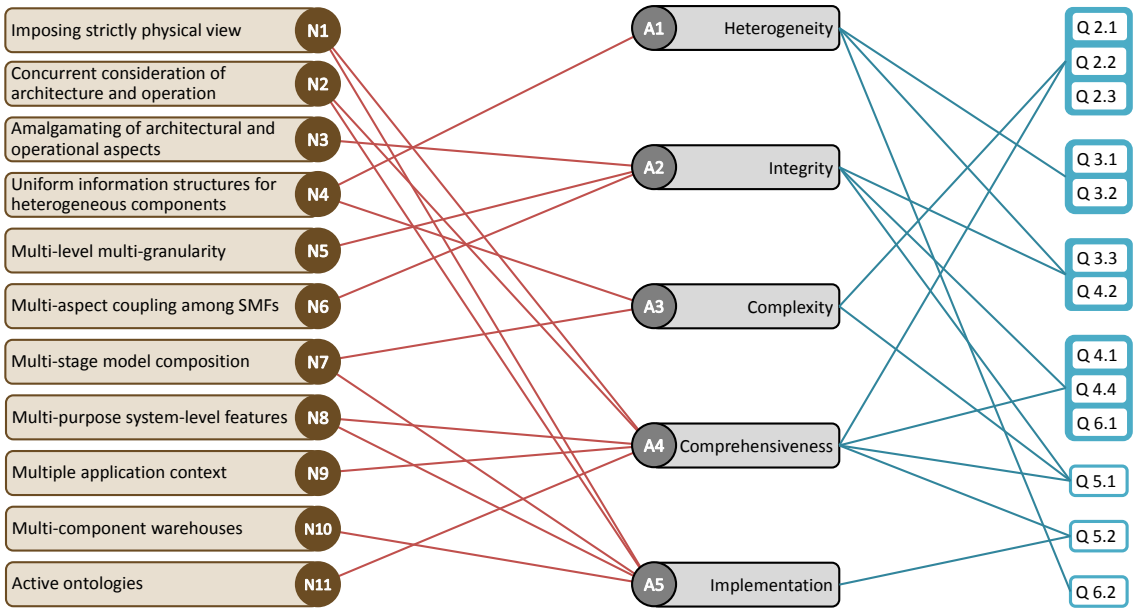


Figure 6-20 Mapping of the novelties of the SMF-TB framework to the aspects of modeling CPSs; (N = novelty; A = aspect of comparison; Q = question of the survey)

6.7.1 Findings concerning imposing a strictly physical view:

Physical view implies that nothing can be abstracted and simplified in system level. This view has the same significance for all constituents (i.e. SW and CW). This novelty addresses two aspects namely *information comprehensiveness* and *implementation*. Among others, physical interaction of components and morphological considerations are the main sets of information that neglected by available modeling tools. Simulink reduces a system into its functions and relations among its parameters. Modelica simplifies a system into mathematical relations among the parameters of its components. Ptolemy II does the same, in the form of actor oriented modeling by ignoring physicality of components. SysML framework is not developed to capture morphology and physicality of components and concentrates on the logical relations among components. The main issue in LabVIEW is control and signal streams within a system by ignoring physical view on system models.

6.7.2 Findings concerning enforced concurrent consideration of architecture and operation:

SMFs capture both architectural and functional aspects of components concurrently and without any preference. That is the reason that SMF-TB is neither components-based nor function-based modeling tool, but hybrid SMF-based modeling tool that architecture and operation aspects are inseparable. In view of that, aspect of *information comprehensiveness* is well supported by this novelty. Furthermore, information structures of SMFs as capsulations of architecture- and operation- related information facilitate *implementation* of both. In comparison, Simulink is a function-based modeling framework. Modelica is a component-based framework that mathematically links architecture aspect to operations. Ptolemy is actor-based modeling framework that put emphasis on operation of components. SysML supports architecture and function aspects that are logically linked. LabVIEW is a function-based modeling framework that supports logical links to architectural aspect.

6.7.3 Findings concerning amalgamating of architectural and functional aspects:

All chunks of information that defines relations among architectural and operational aspects are captured within SMFs. Although it does not refer to integration of components within a system, this novelty supports the *integrity* issues in modeling by proving internal amalgamating of information. SMFs capture relations among architecture and operations through constraints, parameters, state-transitions and associating events, timing, conditions, and logically by methods of operation. In comparison, Simulink and Modelica do it through limited parameters. State-transition and timing contract are used by Ptolemy. SysML creates one to one relations among diagrams related to architecture and operation aspects. LabVIEW benefits from linking through parameters, state transition, and logical coupling. Comparing the studied modeling frameworks to SMF-based modeling framework shows a considerable contrast, and that is, in SMF-based modeling architecture aspects and operation aspects cannot be separated.

6.7.4 Findings concerning using uniform information structures for heterogeneous components:

The intrinsic differences of SW, HW, CW constituents enforced using abstraction and simplification in modeling to facilitate capturing interrelation of heterogeneous compo-

nents. The studied modeling frameworks use multiple solutions to encounter this issue. Simulink neglects capturing CW constituents, and neglects capturing architectural aspects of components towards this aim. Framework of Modelica is designed for considering HW and AW components. Ptolemy II and LabVIEW ignore physical differences of components. SysML makes a higher abstraction and ignores differences of components. SMF-based modeling framework tackle this problem in a unique way by providing uniform information structures for capturing interoperation of various constituents. This novelty results in avoiding unnecessary simplification and abstraction.

6.7.5 Findings concerning multi-level multi-granularity:

Moving from higher level of aggregation to lower levels and vice versa is supported by almost all modeling tools. All studied frameworks support this issue through some kinds of logical relation and by applying abstraction. Consequently, they do not sufficiently support relations of the parameters in different levels of aggregation (parameters composability). SMF-based modeling framework supports multi-level modeling through multi-granularity of SMFs. This novelty implies that the information structures of SMFs are designed to fully support physical aggregation of them through containment operational and architectural coupling.

6.7.6 Findings concerning multi-aspects coupling among SMFs:

SMFs can be coupled together in order to create a higher level SMF. And, this can be continued till a model of a CPS is created. The knowledge structures of SMFs provide interfaces that support multi-aspect coupling. By 'multi-aspect' we mean through states of operation, events (i.e. timestamps), streams, parameters, physical mates, contracts, and so forth. These interfaces address fully integration of modeled components. In contrast, the other studied modeling frameworks create limited number of coupling links among their modeling building block which have been mentioned before. Simulink provides functional/logical relations among modeling entities. Modelica creates mathematical/physical relations among the components. Ptolemy II uses multi-formalism to create logical and functional relations among actors. SysML connects blocks through logical relations and LabVIEW connects components through control and signal relations.

6.7.7 Findings concerning multi-stage model composition:

In our modeling framework, models are not created in one go. Firstly, modeling building blocks have to be created, and after that they can be composed together in the modeling space by system designer. It reduces the tasks of the system designer by delegating the time-consuming tasks to knowledge engineers. Consequently, the system designer can focus more on the conceptualization tasks and tackling composability challenges. This novelty addresses the complexity aspect. The other frameworks deal with this aspect by providing components and functions libraries that is discussed before. In comparison, we recognized that in the studied modeling frameworks many tasks are assigned to the system designers that distract them from their main responsibility.

6.7.8 Findings concerning using multi-purpose system-level features:

SMFs are smart system-level features that support many purposes. Reasoning with higher-level semantics relieves them from dealing with unnecessary details which are not important in system-level conceptualization. The studied modeling frameworks are not supported by this level of semantic-richness. Trans-disciplinary fusion of knowledge is another purpose that is fulfilled by SMFs which is rare among the studied modeling frameworks. Harmonization of design and computational methodologies (elaborated in the previous chapter) shows remarkable differences with the other modeling frameworks that usually can be used in a particular design phase. SysML is the only framework that partially supports the last two issues. However, it suffers seriously from the lack of semantic-richness.

6.7.9 Findings concerning explicit support of multiple application contexts:

The two main application contexts that could be supported by the modeling frameworks are system-level conceptualization and system-level engineering. According to our reasoning, SMF-TB supports both application contexts better than others. SMF-based modeling novelty relies on the information structures that comprehensively capture all pieces of information required for working in these two application contexts. However, the information captured by the other frameworks is specified for a particular design purposes. According to Table 6-6, Modelica is the most comprehensive among the other studied frameworks. It is followed by SysML, Simulink, LabVIEW, and Ptolemy II, respectively.

6.7.10 Findings concerning managing multi-component warehouses:

The libraries in modeling tools are used for providing ready-to-use modeling elements. In SMF-TB the libraries are supported by multi-component warehouses. These warehouses differ from others in some aspects. Firstly, they physically composed of three components of (i) the relational database, (ii) the management component, and (iii) the meta-level knowledgebase. Secondly, the changes of the model and SMFs are tracked and recorded in the meta-level knowledgebase. This gives opportunity of including historical information in SMFs, and transferring and reusing them into other models (same as using second-hand components in a system). Thirdly, instances of SMFs are not separated from their GTs and PTs. It means not only pieces of information are inherited, but also the information about inheritance and the source is included inside SMFs. Since it is handled by meta-level knowledgebase, the performances of simulation are not affected by the information overload. Finally, meta-level knowledgebase facilitate accessibility to some kinds of information that is more frequently used by simulation engine. This ability improves simulation performance and speed. None of the abovementioned aspects are observed in other modeling frameworks.

6.7.11 Findings concerning uses active ontologies:

SMFs are created by using 21 kinds of ontological entities. SMF creation highly dependents to ontologies, and has the advantage of semantic-richness. In particular, every attribute should be defined and parameterized regarding a concept imported from ontological tables. That means pieces of information are not open to interpretation and can be processed smartly by computational programs. Consequently, we can expect highly intelli-

gent tools derived from SMF-based modeling frameworks. Although the development of ontologies is excluded from this work, it can be foreseen that active ontologies give an outstanding advantage to SMF-based modeling over other modeling frameworks. It means that by emerging of new kinds of technologies, material, attributes, etc., the ontological tables can be updated and accordingly the new SMFs will be already understandable for the modeling tool.

Table 6-8 Summary of the results of comparing the frameworks

Novelties in SMF-TB	**Simulink**	**Modelica**	**Ptolemy II**	**SysML**	**LabVIEW**
Imposing strictly physical view	Simplified to functions modeling	Mathematical-based	Actor-oriented function-based	Logical-based	Control-based
Enforced concurrent consideration of architecture and operation	Function-based	Component-based (mathematically linked to operations)	Actor-based (representing operations of components)	logically linked architecture-based and function-based	Function-based (logically linked to architecture)
Amalgamating of architectural and functional aspects	Through limited parameters	Through parame-ters	Through parameters, state transition, and timing contracts	One-to-one logical coupling	Through parame-ters, state transition, and logical coupling
Uniform information structures for heterogeneous components	Ignoring some constituents, and differences of constituents	Ignoring some constituents, and their interopera-tion	Ignoring physical differences of constituents	Ignoring some constituents, and differences of constituents	Ignoring physical differences of constituents
Multi-level multi-granularity	Abstraction	Abstraction	Abstraction	Abstraction	Abstraction
Multi-aspects coupling among SMFs	Functional/logical relations	Mathemati-cal/physical relations	Multi-formalism/ logical relations	Logical relations	Control relations
Multi-stage model composition	functions, structural block, algorithmic block, and dynamic block libraries	Huge commercial and open source component libraries	Actor libraries	Unit libraries	Data-acquisition, signal generation, mathematics, statistics, signal conditioning, analysis libraries
Multi-purpose system-level features	Not supported	Not supported	Not supported	partially support-ed	Not supported
Multiple application contexts (1st)	3rd	2nd	6th	4th	5th
Multi-component warehouses	Simple libraries	Simple libraries	Simple libraries	Simple libraries	Simple libraries
Active ontologies	Limited ontologies	Finite ontologies	Limited ontologies	No ontology	Limited ontologies

6.8 CONCLUDING REMARKS

The performed comparison allowed us to expose the characteristic features and the used modeling and computational concepts of the underpinning conceptual frameworks of benchmarked modeling tools. These modeling frameworks were chosen as the best and most known ones in their categories. Although the studied modeling frameworks are supporting sufficiently the purposes that they are designed and developed for, the com-

parative study expressively clarified that they are not appropriate for the purpose of pre-embodiment design of cyber-physical systems as they cannot sufficiently address the main requirements in this particular application context. These frameworks are mostly suffers from the high level of abstraction applied in modeling, lack of support for heterogeneous components, ignoring morphological and physical attributes of components, not fully supporting integrity of components, inability to manage high complexity of components in modeling, and insufficient ontological supports.

In addition, this comparative study approved that the novelties introduced by SMF-based modeling framework sufficiently address the challenges of pre-embodiment design of CPSs. The 11 novelties explained in this chapter have been projected to the aspects of comparison and accordingly compared with the frameworks of the available modeling tools. Based on this comparison, those have been approved as novel solution towards tackling the regarded modeling and design challenges. It has to be noted that the current states of the studied modeling tools are the results of development for a long time. Many built-in tools are added to them and many add-in products are developed for enhancing their functionalities. By considering SMF-based modeling as novel and proper solution for the determined application, it could be expected that the further developments and implementations could sufficiently support system-level modeling of CPSs.

6.9 REFERENCES

[1] Sangiovanni-Vincentelli, A., (2007), "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design", Proceedings of the IEEE, Vol. 95 (3), pp. 467-506.

[2] Broman, D., Lee, E.A., Tripakis, S., and Törngren, M., (2012), "Viewpoints, formalisms, languages, and tools for cyber-physical systems", Proceedings of the Proceedings of the 6th International Workshop on Multi-Paradigm Modeling, ACM, pp. 49-54.

[3] Fritzson, P., (2014), Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach, John Wiley & Sons.

[4] Lee, E.A., Neuendorffer, S., and Wirthlin, M.J., (2003), "Actor-oriented design of embedded hardware and software systems", Journal of circuits, systems, and computers, Vol. 12 (03), pp. 231-260.

[5] Kehtarnavaz, N., and Kim, N., (2011), Digital signal processing system-level design using LabVIEW, Newnes.

[6] Shetty, D., and Kolk, R., (2010), Mechatronics System Design, SI Version, Cengage Learning.

[7] website, M.o., (2016), "Simulation and Model-Based Design, http://www.mathworks.com/products/simulink/index.html, Access date March 03, 2016".

[8] website, M.o., (2016), "Features of Simulink, http://www.mathworks.com/products/simulink/features.html, Access date: March 03, 2016", 2016.

[9] Nakajima, S., Furukawa, S., and Ueda, Y., (2012), "Co-analysis of SysML and Simulink Models for Cyber-Physical Systems Design", Proceedings of the Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on, pp. 473-478.

[10] Tariq, M.U., Florence, J., and Wolf, M., (2014), "Design Specification of Cyber-Physical Systems: Towards a Domain-Specific Modeling Language based on Simulink, Eclipse Modeling Framework, and Giotto", Proceedings of the ACESMB@ MoDELS, pp. 6-15.

[11] Wang, Y., Zhou, X., and Liang, D., (2012), "Study on Integrated Modeling Methods toward Co-simulation of Cyber-Physical System", Proceedings of the High Performance Computing

and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, IEEE, pp. 1736-1740.

[12] Eyisi, E., Zhang, Z., Koutsoukos, X., Porter, J., Karsai, G., and Sztipanovits, J., (2013), "Model-based control design and integration of cyberphysical systems: an adaptive cruise control case study", Journal of Control Science and Engineering, Vol. 2013, p. 1.

[13] Zhang, Z., Porter, J., Eyisi, E., Karsai, G., Koutsoukos, X., and Sztipanovits, J., (2013), "Co-simulation framework for design of time-triggered cyber physical systems", Proceedings of the Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems, ACM, pp. 119-128.

[14] Son, H.S., Kim, W.Y., Kim, R.Y., and Min, H.-G., (2012), "Metamodel Design for Model Transformation from Simulink to ECML in Cyber Physical Systems", in: Computer Applications for Graphics, Grid Computing, and Industrial Environment, Springer, pp. 56-60.

[15] Wikipedia, (2015), "Modelica, https://en.wikipedia.org/wiki/Modelica, Access date: Dec. 10, 2015", 2015.

[16] Casella, F., (2015), "Simulation of large-scale models in modelica: State of the art and future perspectives", Proceedings of the 11-th International Modelica Conference, Paris, France, pp. 459-468.

[17] Fritzson, P., and Bonus, P., (2006), "Introduction to object-oriented modeling and simulation with OpenModelica", IDA-The Department of Computer and Information Science.

[18] Fritzson, P., (2011), Introduction to modeling and simulation of technical and physical systems with Modelica, John Wiley & Sons.

[19] Nytsch-Geusen, C., (2007), "The use of the UML within the modelling process of Modelica-models", Proceedings of the EOOLT, pp. 1-11.

[20] Paredis, C.J., Bernard, Y., Burkhart, R.M., Koning, H.P., Friedenthal, S., Fritzson, P., Rouquette, N.F., and Schamai, W., (2010), "5.5. 1 An Overview of the SysML Modelica Transformation Specification", Proceedings of the INCOSE International Symposium, Vol. 20, Wiley Online Library, pp. 709-722.

[21] Schamai, W., Fritzson, P., Paredis, C., and Pop, A., (2009), "Towards unified system modeling and simulation with ModelicaML: modeling of executable behavior using graphical notations", Proceedings of the Proceedings 7th Modelica Conference, Como, Italy, pp. 612-621.

[22] Schamai, W., (2009), "Modelica modeling language (ModelicaML): A UML profile for Modelica".

[23] Pop, A., Akhvlediani, D., and Fritzson, P., (2007), "Towards Unified System Modeling with the ModelicaML UML Profile", Proceedings of the EOOLT, Citeseer, pp. 13-24.

[24] Cellier, F., (2005), "Dymola: Environment for Object-oriented Modeling of Physical Systems", Retrieved July, Vol. 5, p. 2011.

[25] Maplesoft, (2015), "MapleSim, http://www.maplesoft.com/products/maplesim/compare/index.aspx, Access date: Nov. 28, 2015", 2015.

[26] Wolfram, (2015), "Modeling & Simulation with SystemModeler, http://www.wolfram.com/system-modeler/features/modeling-simulation.html, Access date: Nov 26, 2015".

[27] Brooks, C., Lee, E.A., and Tripakis, S., (2010), "Exploring models of computation with Ptolemy II", Proceedings of the Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on, IEEE, pp. 331-332.

[28] Bier, J.C., Goei, E.E., Ho, W.H., Lapsley, P.D., O'Reilly, M.P., Sih, G.C., and Lee, E.A., (1990), "Gabriel: A design environment for DSP", Micro, IEEE, Vol. 10 (5), pp. 28-45.

[29] Lee, E.A., Ho, W.-H., Goei, E.E., Bier, J.C., and Bhattacharyya, S., (1989), "Gabriel: A design environment for DSP", Acoustics, Speech and Signal Processing, IEEE Transactions on, Vol. 37 (11), pp. 1751-1762.

[30] Ortega, R., (1992), "Operational Event Timing Constraints in Ptolemy".

[31] Evans, B., Kamas, A., and Lee, E.A., (1994), "Design and Simulation of Heterogeneous Systems using Ptolemy", Proceedings of the Proceedings of ARPA RASSP Conference, Citeseer.

[32] Brooks, C., Lee, E.A., Liu, X., Neuendorffer, S., Zhao, Y., Zheng, H., Bhattacharyya, S.S., Cheong, E., Davis, I., and Goel, M., (2008), "Heterogeneous concurrent modeling and design in java (volume 1: Introduction to ptolemy ii)", DTIC Document, 2008.

[33] Liu, X., Xiong, Y., and Lee, E.A., (2001), "The Ptolemy II framework for visual languages", Proceedings of the null, IEEE, p. 50.

[34] Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y., (2003), "Taming heterogeneity-the Ptolemy approach", Proceedings of the IEEE, Vol. 91 (1), pp. 127-144.

[35] Brooks, C., Lee, E.A., Liu, X., Neuendorffer, S., Zhao, Y., Zheng, H., Bhattacharyya, S.S., Cheong, E., Davis, I., and Goel, M., (2008), "Heterogeneous concurrent modeling and design in java (volume 2: Ptolemy ii software architecture)", DTIC Document, 2008.

[36] John Davis, I., Ii, J.D., Goel, M., Hylands, C., Kienhuis, B., Lee, E.A., Liu, J., Liu, X., Muliadi, L., and Neuendorffer, S., (1999), "Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java".

[37] Lee, E.A., and Neuendorffer, S., (2000), MoML: A Modeling Markup Language in SML: Version 0.4, Citeseer.

[38] Brooks, C., Cataldo, A., Lee, E.A., Liu, J., Liu, X., Neuendorffer, S., and Zheng, H., (2005), Hyvisual: A hybrid system visual modeler, Electronics Research Laboratory, College of Engineering, University of California.

[39] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S., (2004), "Kepler: an extensible system for design and execution of scientific workflows", Proceedings of the Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on, IEEE, pp. 423-424.

[40] Baldwin, P., Kohli, S., Lee, E.A., Liu, X., Zhao, Y., Ee, C., Brooks, C., Krishnan, N., Neuendorffer, S., and Zhong, C., (2005), "Visualsense: Visual modeling for wireless and sensor network systems", Citeseer, 2005.

[41] Cheong, E., Lee, E.A., and Zhao, Y., (2005), "Viptos: a graphical development and simulation environment for tinyos-based wireless sensor networks", Proceedings of the SenSys, Vol. 5, pp. 302-302.

[42] Davis II, J., Goel, M., Hylands, C., Kienhuis, B., Lee, E.A., Liu, J., Liu, X., Muliadi, L., Neuendorffer, S., and Reekie, J., (1999), "Overview of the Ptolemy project", ERL Technical Report UCB/ERL, 1999.

[43] SysML, (2016), "SysML official website, http://sysml.org/, access date: Feb. 13, 2016", 2016.

[44] Object-Management-Group, (2016), "OMG official website, http://www.omg.org/, access date: Feb. 13, 2016", 2016.

[45] Zhang, L., (2014), "Modeling large scale complex cyber physical control systems based on system of systems engineering approach", Proceedings of the ICAC 2014 - Proceedings of the 20th International Conference on Automation and Computing: Future Automation, Computing and Manufacturing, pp. 55-60.

[46] OMG-SysML, (2016), "OMG SysML official website, http://www.omgsysml.org/, access date: Feb. 13, 2016", 2016.

[47] Bombieri, N., Ebeid, E., Fummi, F., and Lora, M., (2013), "On the reuse of heterogeneous IPs into SysML models for integration validation", Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 29 (5), pp. 647-667.

[48] Nakajima, S., Furukawa, S., and Ueda, Y., (2012), "Co-analysis of SysML and simulink models for cyber-physical systems design", Proceedings of the Proceedings - 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2012 - 2nd Workshop on Cyber-Physical Systems, Networks, and Applications, CPSNA, pp. 473-478.

[49] Schamai, W., Fritzson, P., Paredis, C.J., and Helle, P., (2012), "ModelicaML value bindings for automated model composition", Proceedings of the Proceedings of the 2012 Symposium

on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium, Society for Computer Simulation International, p. 31.

[50] Apvrille, L., and Roudier, Y., (2013), "SysML-Sec: A model-driven environment for developing secure embedded systems", Proc. of SARSSI 2013, Mont-de-Marsan, France.

[51] Knorreck, D., Apvrille, L., and de Saqui-Sannes, P., (2011), "TEPE: a SysML language for time-constrained property modeling and formal verification", ACM SIGSOFT Software Engineering Notes, Vol. 36 (1), pp. 1-8.

[52] Apvrille, L., and Roudier, Y., (2015), "Designing safe and secure embedded and cyber-physical systems with SysML-Sec", Communications in Computer and Information Science, Vol. 580, 2015, pp. 293-308.

[53] Kawahara, R., Nakamura, H., Dotan, D., Kirshin, A., Sakairi, T., Hirose, S., Ono, K., and Ishikawa, H., (2009), "Verification of embedded system's specification using collaborative simulation of SysML and simulink models", Proceedings of the International Conference on Model-Based Systems Engineering, MBSE 2009.

[54] Slomka, F., Kollmann, S., Moser, S., and Kempf, K., (2011), "A multidisciplinary design methodology for cyber-physical systems", ACESMB 2011, p. 23.

[55] ZHANG, B.-c., LIU, L., GAO, G.-f., and ZHAO, P., (2007), "Data Acquisition and Signal Analysis Based on LabVIEW [J]", Instrument Technique and Sensor, Vol. 12, p. 74.

[56] Xiong, Y., Qin, B., Wu, M., Yang, J., and Fan, M., (2007), "LabVIEW AND MATLAB-based virtual control system for virtual prototyping of cyclotron", Proceedings of the Particle Accelerator Conference, 2007. PAC. IEEE, IEEE, pp. 281-283.

[57] Chen, B., Pattanaik, N., Goulart, A., Butler-Purry, K.L., and Kundur, D., (2015), "Implementing attacks for modbus/TCP protocol in a real-time cyber physical system test bed", Proceedings of the Proceedings - CQR 2015: 2015 IEEE International Workshop Technical Committee on Communications Quality and Reliability.

[58] Prist, M., Freddi, A., Longhi, S., and Monteriu, A., (2015), "An integrated simulation module for wireless cyber-physical system", Proceedings of the 2015 IEEE 15th International Conference on Environment and Electrical Engineering, EEEIC 2015 - Conference Proceedings, pp. 1397-1402.

[59] National-Instruments, (2016), "NI LabVIEW official web-page, http://www.ni.com/labview/applications/embedded/, access date: Feb. 19, 2016", 2016.

[60] Jamal, R., and Wenzel, L., (1995), "The applicability of the visual programming language LabVIEW to large real-world applications", Proceedings of the Visual Languages, Proceedings., 11th IEEE International Symposium on, IEEE, pp. 99-106.

[61] Sousa, T.B., (2012), "Dataflow programming concept, languages and applications", Proceedings of the Doctoral Symposium on Informatics Engineering, Vol. 7, p. 13.

[62] Swain, N.K., Anderson, J.A., Singh, A., Swain, M., Fulton, M., Garrett, J., and Tucker, O., (2003), "Remote data acquisition, control and analysis using LabVIEW front panel and real time engine", Proceedings of the SoutheastCon, 2003. Proceedings. IEEE, IEEE, pp. 1-6.

[63] Jensen, J.C., Chang, D.H., and Lee, E.A., (2011), "A model-based design methodology for cyber-physical systems", Proceedings of the IWCMC 2011 - 7th International Wireless Communications and Mobile Computing Conference, pp. 1666-1671.

[64] Lipovszki, G., and Aradi, P., (2006), "Simulating complex systems and processes in LabVIEW", Journal of mathematical Sciences, Vol. 132 (5), pp. 629-636.

[65] LabVIEW-Wikipedia, (2016), "Wikiperdia article about LabVIEW, https://en.wikipedia.org/wiki/LabVIEW, access date: Jan. 23, 2016", 2016.

[66] Tasner, T., Lovrec, D., Tasner, F., and Edler, J., (2012), "Comparison of LabVIEW and MATLAB for Scientific research", Annals of the Faculty of Engineering Hunedoara, Vol. 10 (3), p. 389.

[67] Bauer, K., and Schneider, K., (2013), "Teaching cyber-physical systems: Programming approach", Proceedings of the Proceedings of the 2012 Workshop on Embedded and Cyber-Physical Systems Education, WESE 2012.

[68] Derler, P., Lee, E.A., and Vincentelli, A.S., (2012), "Modeling Cyber-Physical Systems", Proceedings of the IEEE, Vol. 100 (1), pp. 13-28.

[69] Akkaya, I., Liu, Y., and Lee, E., (2015), "Modeling and Simulation of Network Aspects for Distributed Cyber-Physical Energy Systems", in: Cyber Physical Systems Approach to Smart Electric Power Grid, Khaitan, S.K., McCalley, J.D., Liu, C.C. (Eds.), Springer Berlin Heidelberg, pp. 1-23.

[70] Davare, A., Densmore, D., Guo, L., Passerone, R., Sangiovanni-Vincentelli, A.L., Simalatsar, A., and Zhu, Q., (2013), "metroII: A design environment for cyber-physical systems", ACM Trans. Embed. Comput. Syst., Vol. 12 (1s), pp. 1-31.

[71] Liangpeng, G., Qi, Z., Nuzzo, P., Passerone, R., Sangiovanni-Vincentelli, A., and Lee, E.A., (2014), "Metronomy: A function-architecture co-simulation framework for timing verification of cyber-physical systems", Proceedings of the Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on, pp. 1-10.

[72] Bhave, A., Krogh, B., Garlan, D., and Schmerl, B., (2010), "Multi-domain modeling of cyber-physical systems using architectural views".

Chapter 7

CONCLUSIONS, REFLECTIONS, & FUTURE RESEARCH

7 CONCLUSIONS, REFLECTIONS, and FUTURE RESEARCH

7.1 CONCLUSIONS

The original aim of this promotion research was to deal with customization issues of cyber-physical consumer durables. The conducted investigations cast light on the fact that the currently available design support tools could provide only a limited support when truly heterogeneous, complex and multi-faceted systems, such as cyber-physical systems, are at stake. CPSs represent the next generation of advanced engineered systems such as mechatronic systems, robotic systems, embedded systems, AI systems, IoT systems, and distributed networked sensor, actuator and control systems. In the case of CPSs the concerns of system design have shifted from discipline-related challenges to inherent interdisciplinary and cross-disciplinary challenges. CPS designers either should be equipped with the knowledge of all related system domains, or should be a (transdisciplinary) coordinator of system experts, in particular at designing CPSs that are systems of systems. We assumed in our research that the main task of CPS designers is to address composability challenges. The root of this assumption is that conceptual design of CPSs includes: (i) architectural and functional composition of at least conceptually defined system definitive components, (ii) specification of the required characteristics, interaction and interoperation of the coupled components, and (iii) retrieving them from repositories or delegating their embodiment, detail design, and development to concerned experts.

The objective of the presented research was to develop a supporting tool for designers to be able to model heterogeneous complex-systems such as CPSs in the pre-embodiment design phase. Such kind of tools is needed not only for original architecting and operation design of CPSs, but also for preparing them for embedded customization. Towards this end, five research cycles have been planned, which respectively concentrated on: (i) investigation of the state of the art in customization and computer tool support, (ii) ideation of a conceptual framework, (iii) development of information structures, (iv) elaboration of computational constructs, and (v) benchmarking of the proposed functional and methodological framework. Two complementary theories (the MOT and the SMF theories) have been developed to underpin the proposed modeling framework. Preliminary feasibility, utility, and usability tests have been completed on the modeling framework and a benchmarking study was performed against five widespread modeling tools and languages. Following text provides detailed explanation about the performed research.

7.1.1 Research cycle 1: Investigation of the state of the art in designing CPSs

The main objective of the first research cycle was to acquire fundamental insights in the studied general phenomenon and to provide comprehensive and rigorous descriptive knowledge. The key research questions in this RC were:

- What is the state of the art and what are observable knowledge gaps in designing CPSs in the pre-embodiment design phase?
- Why do the conventional design methods not sufficiently support CPS designers to tackle composability issues in system synthesis?

In the conducted study, five domains of interest were addressed: (i) CPSs as specific engineered systems, (ii) design approaches, (iii) design principles, (iv) design supporting tools, and (v) features technology. The study gained sufficient insight about the achieved overall progress and the knowledge gaps from these different perspectives. It has been proved that the conventional design principles are not adequate for designing CPSs. The reasons of insufficiently of the current modeling tools for supporting pre-embodiment design of CPSs were also explored. The following paragraphs summarize the main findings of the first research cycle:

1. *Some important sets of modeling information have been neglected due to simplifications and abstractions applied by the current CPS modeling tools.*

CPSs consist of intrinsically different components such as hardware, software, and cyberware. These constituents may also be combined as aggregateware. The interactions and interoperations among these intrinsically different components cannot be captured, unless the components are modeled and processed in a uniform manner. Consequently, simplifications and abstractions are applied by the conventional modeling tools.

2. *CPSs are tightly integrated internally as well as with their surrounding physical environment. Accordingly, their internal and external relationships should be defined as precisely as possible. This enables a high fidelity modeling CPSs, which is beneficial in the stage of pre-embodiment design. The physical nature and realization of CPSs can be captured by imposing (a strictly) physical view and considering morphological attributes. This has not been sufficiently implemented or has even been neglected in most of the current CPS modeling tools.*

Some of the current modeling tools (e.g. Ptolemy II) were originally developed for modeling and simulation of engineered systems other than CPSs (e.g. system on a chip and embedded systems). These exemplified systems do not physically (and mechanically) interact with their embedding environment, as they typically deal with data flows and transformations. Consequently, the original frameworks of the current modeling tools did not address the physicality issues (attributes, relationships, and impacts). However, considering internal physical relationships and external physical interactions with the surrounding environment is a must for conceptualization of CPSs.

3. *The main concerns of the current modeling tools are system engineering, analysis, and optimization. However, the main concerns in design of CPSs are such as system conceptualization, system architecting, and ensuring system level performances (e.g. dependability, safety, security, maintainability, adaptability, economy, and energy-efficiency).*

System engineering aims at solving relatively discipline-related problems. However, system conceptualization deals with inter-disciplinary and/or transdisciplinary challenges. Since current CPS modeling tools are originally developed for specific system engineering

purposes, their functional scope have remained the same, even after extending them with enablers for the CPS design arena. The abstractions and simplifications remained the basis of the working methodology of these extended modeling tools. However, a fully-fledged CPS modeling tool should support designers to tackle all logical and physical composability challenges (e.g. functional conceptualization and architecting, and feasibility and interoperation investigations).

4. *System-level feature-based modeling is capable of capturing interdisciplinary and multicultural aspects of designing CPSs in the pre-embodiment design phase. The opportunity of managing physicality of components is the main advantage of this kind of modeling.*

In contrast with traditional part level orientation, system-level modeling requires dealing with composability of components on the system level. Feature technology and feature-based modeling has many advantages for system-level conceptualization and development of CPSs. System-level features allow an explicit specification and representation of physical attributes and morphological properties of components in conceptual modeling and early simulation. SMFs integrate artifact- and process-related knowledge of the concerned disciplines and aspects, and help interfacing heterogeneous components.

7.1.2 Research cycle 2: Development of a theoretical framework

The second research cycle aimed at the investigation of novel concepts and proposed a theoretical framework for pre-embodiment design of CPSs. The key research questions in this research cycle were:

- How to consider all required interdisciplinary and multicultural aspects of designing CPSs in pre-embodiment design phase?
- What kind of information should to be considered towards a uniform modeling heterogeneous CPSs?

This research cycle started with empirical studies. After identifying requirements for modeling heterogeneous system components and systems as a whole, theories that could support constructing a theoretical framework have been explored. As a result, the mereo-operandi theory (MOT) has been introduced as one important component of the sought for theoretical framework. This theory underpinned the rest of our research by shifting the paradigm of modeling complex systems to system-level feature-based modeling. MOT introduced all information engineering fundamentals that were needed for a uniform 'above-the-disciplines' specification of the components and the whole of CPSs. The following paragraphs present the main findings of the second research cycle:

5. *For system-level modeling of CPSs, architecture and operation aspects should be considered simultaneously, as well as the interrelations of these aspects.*

A proper modeling and simulation tool should capture what the components are (architecture) and what they do (operation). Accordingly architecture and operation are considered as the major complementing aspects of modeling and simulation, each of which goes through the specification of an ordered set of parameters and values. Architectural attributes of components influence their operations, and the operational attributes change their

architectural states. Consequently, the architectural and operational parameters have been considered as intertwined.

6. *In order to capture the operations of components in a comprehensive manner, operation prescriptive concepts have been introduces such as: (i) flow of operation, (ii) units of operation, (iii) morphological attributes, and (iv) physical phenomena.*

The operations of SMFs are specified by: (i) the procedure of operation, (ii) the procedural elements, (iii) the morphological and physical attributes of components, and (iv) the principles of the physical and computational phenomena that the operation is based on. The flow of operation (FoO) describes the procedure of operation, and the units of operation (UoOs) specify the procedural elements. Furthermore, the phenomena, which cause the operation, are captured as numerical or algorithmic methods of operation. Moreover, operations were interpreted both as state-transition and as input/output transformation. The former entails architectural changes according to the performed operation, and the latter implies the transformation of inputs into outputs caused by the operation. The proposed framework takes into consideration both of these views, and handles them in relation with other operational attributes.

7. *Architectural relations of all types of components of a CPS (i.e. hardware, software, cyberware, and aggregateware) are specified by using the principles of spatiotemporal mereotopology in order to provide a uniform formulation of diverse relations.*

By definition, the basis of components of a CPS is an architectural domain that lends itself to specific system operations. In addition the morphological and physical attributes (physical manifestations) of the individual domains, the relations among them also have influence on the operation of components. These relations can be described by the formalism that has been developed based on the notions and notations of spatiotemporal mereotopology. The main advantage of the mereotopological formulation of the architectural relations is the opportunity of a uniform representation of relations among all types of components. The differences of these relations can be specified on by means of semantic annotations. There are two kinds of architectural relations defined among domains of a system, namely (i) part-of relation and (ii) connected-to/with relation. The first kind is referred to as architectural containment and the second kind is called architectural connectivity. It has been found that these two kinds of relations were necessary and sufficient to fulfill our requirements for describing architectural relations.

8. *The procedural relations of operations can be captured by and described as streams of information, energy, and material.*

Considering FoOs as sets of related UoOs, the relations between a FoO and each of the contained UoOs can be described by 'operational containment' relations. In harmony, the relations among the UoOs were described by 'operational connectivity' relations. The operational connectivity relations also express dependency of operations. Streams are representatives of operational connectivity relations. Consequently, procedural relations of a FoO are specified by defining the streams among its UoOs.

7.1.3 Research cycle 3: Development of information structures

The theory of system manifestation features was introduced in the third research cycle. As a complement and extension of MOT, the SMFs theory offers a formal description for all required aspects of the proposed high-level modeling elements of CPSs and their components and constituents in a uniform way. The key research questions in this RC were:

- What information structure describes the modeling units?
- Which pieces of information should be captured by knowledge frames in order to computationally model components and constituents of CPSs?
- How the pieces of information need to be combined in order to create modeling building blocks?

The main result of the conducted research is the clarification of the role of different pieces of information in the context of SMFs-based modeling of CPSs, and the required construction of the information structures. These have been explained by the SMF theory. The information structures have been captures by both architecture and operation knowledge frames. The following paragraphs cast light on the main findings of the third research cycle:

9. *System manifestation features resemble real-world components of CPSs and are defined as modeling building blocks, which encapsulate all pieces of information and their relations required for modeling and simulation of CPSs.*

According to the traditional feature technology, a model of a part can be captured as a combination of part-level features. Likewise, a system can be modeled as an aggregation of system-level features. In CPS modeling, SMFs are considered as models of assumed or real-world components, which can be connected together through their predefined interfaces. Interfaces of SMFs include both architecture and operation related pieces of information. Besides the information about interfaces, SMFs contain the kernel information sets, which explain how the architecture and operation attributive parameters are related with each other.

10. *SMFs can capture the required pieces of information and the relations among them uniformly for all different types of constituents and components.*

SMFs encapsulate the required information in a uniform (standard) format for all types of components. Based on this uniform representation, SMFs can treat hardware, software, cyberware, and aggregateware components in the same way (i.e. as similar modeling building blocks). These building blocks can be connected with the same compositional mechanisms to build higher level building blocks, which can also be considered as new SMFs.

11. *Information structures of SMFs are realized through interrelated architecture knowledge frames (AKFs) and operation knowledge frames (OKFs).*

The information sets of SMFs and their relations are captured in a stereotyped (standard) format called information structure. The standardization of this information structure would support the uniformity of SMFs. The information structure of SMFs was realized as a

combination of architecture and operation knowledge frames. The pieces of information included in these two knowledge frames are linked together. SMFs in various aggregation level and the operational and architectural relations among them were captured uniformly by AKFs and OKFs. This is supported by vertical associations of knowledge frames. Vertical association means that each OKF (which represents a FoO) may be associated with several lower level OKFs (as representatives of its UoOs). Since there may be numerous aggregation levels, the UoOs can be considered as FoOs for their lower-level SMFs. Consequently, each of the OKFs of the UoOs can be associated with both higher and lower level OKFs. A similar rule applies to the architectural side. That is, each AKF represents a domain that associates with a higher level AKF and some lower-level AKFs. Horizontal association, on the other hand, refers to connectivity among domains in a same level of aggregation (represented by AKFs) and connectivity among operations in a same level of aggregation (represented by OKFs). The information about connectivity among the entities of a domain and the connectivity among the UoOs of a FoO were captured by the corresponding AKF and OKF, respectively. Accordingly, the horizontal association of AKFs and OKFs can also be found in their higher level AKF and OKF, respectively.

12. *Layers of operation were defined in SMFs in order to facilitate handling of Multiphysics operations. Multiple numerical (mathematical) and algorithmic methods can be assigned to different layers of operation for computing multiple* interacting physical processes.

We considered methods as means of representing functional relations among (attributive, metric, or temporal) parameters. Multiphysics operations raise the need for a parallel computation of multiple interacting physical processes. These processes are allocated to various computational layers, among which there is a real time data transfer and exchange. An example of this situation can be a smartphone, where the heat generated by the processor has a relation with the processing load. The processing load dependent estimation of the generated heat can be allocated to one computational layer. Another computational layer can be allocated to the computation of the effects of the heat on the processing performance. In this example we have two methods on two layers that share some parameters (e.g. heat). Without this sharing it is not possible to realize Multiphysics computation. Therefore, the assigned methods include parameters that receive their values as results of computation on (several other) layers. Three computational layers of operation have been considered in the proposed framework to capture Multiphysics, namely: (i) concurrent operation, (ii) subordinate operation, and (iii) software manifestation.

7.1.4 Research cycle 4: Elaboration of information and computational constructs

The fourth research cycle focused on how the information structure of SMF could be converted into computational units and processed computationally. The specific aim was to propose information schema constructs (that were used in the development of the relational databases of the conceived SMFs warehouses) and computational constructs (that were used for the implementation of the computational procedures of SMF specification and composition). From a practical point of view, the existence of the needed information constructs supports the feasibility of the MOT and SMF theories. The key research questions in this RC were:

- How the pieces of information and the relations among them can be captured for an effective computation?
- What computational procedures are needed and possible for creating modeling building blocks?
- What procedures need to be implemented for creating a CPS model by composing modeling building blocks?

The process of generating SMFs was decomposed to the phases of genotype creation, deriving phenotypes, and deriving instances of SMFs. Accordingly, three databases of genotypes, phenotypes, and models were designed. Information schema constructs were defined to map necessary information sets (and the relations among them) to relational database contents. This research cycle also determined the procedures of SMF creation and model composition in combination with the entry forms and the warehouses. The following paragraphs explain the outcomes of the fourth research cycle:

13. *The chunks of information enclosed in AKFs and OKFs, and their relations were transferred to computational constructs.*

Computational construct were defined by identifying a specific set of variables and their semantics-implied relations. In general, computational construct were designed to facilitate data organization, and execution of computational procedures. The data organization issue appeared in the context of developing relational data tables for warehouse databases. These constructs have been named as information scheme constructs (ISC). The procedures for function operationalization were named as computational processing constructs (CPC). At the beginning of designing the abovementioned constructs, the interrelated pieces of information in AKF and OKF were grouped into logically related (consistent) chunks of information that described specific aspects of SMFs. These groups were formalized as constructs and converted into relational database tables, which accommodated the designated chunks of information. The ISCs specified the necessary relational data tables and the connections among them. The procedures of receiving chunks of information from the designer and mapping them into relational tables were also developed. These procedures were specified with the help of CPCs.

14. *SMFs were created and composed into a model through the three steps of genotype creation, deriving phenotypes, and descending instances.*

The need for entering a large amount of data during the process of SMF creation is a challenge for knowledge engineers. In addition, consistency of the symbols, notations, and relations should be maintained throughout all phases of creating and specifying SMFs. Reducing the cognitive workload of knowledge engineers and processing the related information in a consistent way were two main objectives to be achieved by a step-wise procedure of SMF creation. According to our assumption, the formal specification of concepts and entities related to the definition of the architecture and operations of SMFs are obtained from dedicated ontologies. It is also assumed that the SMFs ontologies are comprehensive enough to capture all concepts and entities necessary for defining the hardware, software, cyberware, and aggregateware ingredients of SMFs, and thus the need for run-time definition of non-warehoused modeling entities is reduced.

15. SMF genotypes were introduced to specify structural relations among operation- and architecture-related pieces of information.

Genotypes resemble categories of components, e.g. browsers, gearboxes, screws, and modems. A genotype defines and portrays a variable (parameterized) structure. It works with s specific (unique) set of possible structural entities and connections among them, whose concepts are captured in relevant ontologies. In other words, a genotype is defined by arranging structural concepts included in ontological classes (as genes), and internal structural relationships among them. These concepts may refer both to architectural and operational attributes.

16. Attributive parametrization and information sets of SMFs were captured as phenotypes.

Phenotypes resemble particular components with predefined parameterization. Phenotypes of SMFs were derived based on choosing an appropriate structure included in the parent genotype. By doing so, the structural attributes became specified for each phenotype, and this allowed selecting and assigning morphological and attributive parameters to them.

17. Instances of a SMF were conceptualized as descendants of phenotypes, which are evaluated for their attributive values and composed into a CPS model.

The phenotypes manifested in a particular system composition are called instances of an SMF. Instances are always derived in a given architectural and operational context of system modeling. As a preceding action, phenotypes are first coupled and then instantiated in order to create a specific part of a system model. By coupling instances of SMFs, their parameters become shared. By placing the modeled system in a working context, the variable parameters take values according to the interface specifications, morphological parameters, or simulation outcomes. One phenotype can be instantiated in multiple places of a model. For example, five wheels of an office chair are name-distinguished instances of the same phenotype.

18. Composed parts of models or entire models can be stored as a new SMF and made available for use in other models. Accordingly, the concept of system of systems is supported in modeling and system architecting by SMFs.

The part of models or the entire models appearing as the results of composing and instantiation process of phenotypes are supposed to be stored in the model warehouse. After disconnecting them from their embedding environment, they can be considered as phenotypes and can be stored back in the phenotype warehouse. The phenotypes which are not directly inherited from parent genotypes are called compound phenotypes. Using compound phenotypes reduces the workload of system design.

19. The meta-level knowledgebase of the model warehouse is in charge of keeping track of model compositions and modifications.

The model warehouse comprises of three parts: (i) the database, which is in charge of accommodating pieces of relational information that specify the SMFs and the composed models, (ii) the database management system, which is in charge of writing, reading,

removing, and editing the data in the database, and (iii) the meta-level knowledgebase, which is in charge of keeping track of the SMFs imported into the model, history of model modification, and history of variable changes.

The procedures of creating models, modification of models, and other model-level activities are facilitated and supported by the meta-level knowledgebase. It also supports model adaptation, optimization, and customization. Working tightly with the simulation engine, it allows the meta-level knowledgebase to provide information even about accumulative effects such as aging, wearing, fatigue and erosion. In the case of smart and self-adaptive systems, the evolution of system could also be supported by the meta-level knowledgebase.

7.1.5 Research cycle 5: Benchmarking the proposed functional and methodological framework

The aim of the fifth research cycle was to validate the theoretical efforts through benchmarking against five commercial and academic modeling tools and languages. The key research questions in this research cycle were:

- What are the aspects of comparison and indicators of appropriateness of CPS modeling tools and languages in benchmarking?
- What are the advantages and disadvantages of the introduced modeling framework in pre-embodiment design of CPSs, in comparison with the modeling frameworks of the benchmarked systems?

The research work included comparing the specificities of the modeling tools and their conceptual frameworks, as well as reasoning about their appropriateness in the context of benchmarking. The aspects of comparison and the indicators of appropriateness were chosen according to the characteristics of CPSs and the requirements of pre-embodiment design. The aspects of comparison have been detailed in a questionnaire which was presented to expert users of the benchmarked tools and languages. The answers to the questionnaire were processed by critical reasoning as well as by descriptive statistics. The following paragraphs discuss our takeaways from the fifth research cycle:

20. *The comparison of SMF-based modeling with other commercially and experimentally available modeling tools had to be performed on the framework level.*

Comparison of a proposed framework with that of an implemented tool is not theoretically possible, unless there is a comprehensive and detailed description of the underpinning assumptions and concepts. This was not the case in our context. Since the modeling framework proposed in this research is not fully implemented as SMF-TB, many of its functionalities cannot be measured explicitly. There were only limited amount of information about the frameworks based on which the available modeling tools were implemented. Therefore, we applied a conceptual reverse engineering strategy, and concluded about the possible underpinning theories and concept of the benchmarked systems based on their documented and observable structural and functional features. We used a literature study and an expert questionnaire for a multi-source information collection and, more importantly, for a cross validation of the results, in order to reduce the uncertainty caused by the reasoning.

21. *The benchmarking of the current CPS modeling tools has demonstrated differences in terms of their ability to fulfill the assumed requirements of CPS conceptualization and architecting.*

The defined requirements of pre-embodiment design of CPSs were grouped into five categories of comparison aspects and a relatively large number of appropriateness indicators. The categories of aspects of benchmarking and for comparison of the modeling tools were (i) addressing heterogeneity of CPSs, (ii) addressing integrity of CPSs, (iii) addressing complexity of CPSs, (iv) addressing information comprehensiveness, and (v) addressing tool implementation. The comparison was evaluated based on the synthesized results of analyses of the literature and the responses to the questionnaire. The results revealed that the chosen modeling tools (i) work based on different principles, (ii) capture dissimilar sets of information, and (iii) formulate relations among components differently. Accordingly, appropriateness of the studied modeling tools for supporting pre-embodiment design of CPSs was identified.

22. *The analyses of the findings of the conducted comparison study, and the projection of the novelties of the SMF-based modeling framework onto the appropriateness indicators showed the uniqueness of our proposed solution and the huge potentials it carries.*

We specified a large set of appropriateness indicators in five overall aspects of benchmarking. By objectively comparing the strong and weaker point of the benchmarked tools and languages with that of the proposed framework and computational solution, we intended to achieve a sound comparison and an objective image. The novelties introduced by the SMF-based modeling were assessed according to the same set of indicators. We could identify in which sense SMF-based modeling can go beyond the current state of the art, but also those points where it needs further attention and elaboration. In general, the novelties introduced by our framework makes a better fulfilment of the requirements possible. The novelties allow us arriving at a more-fledged modeling toolbox for pre-embodiment design of CPSs.

7.2 PERSONAL REFLECTIONS ON RESEARCH AND RESULTS

This research started four years ago (in 2012) in a somewhat humble way. The initial target was determined as facilitating mass customization (MC) of cyber-physical consumer durables. For about a year, I performed deep exploration of the approaches of mass customization and regarding design principles. Moreover, characteristics of CPSs were extensively studied and causalities of mismatches of MC design principles with characteristics of CPSs were investigated. The results showed a deep gap between the available MC design principles and the required ones. According to the planned research target, we had to derive new design principles that fulfill the requirement of designing customizable CPSs. At that moment, we realized that the real knowledge gap is actually is not concerning the principles, but the computer-based tools that could support it in an application-independent and efficient way. This recognition turned our attention to theoretical issues, computational methodology, modeling entities, information structures and constructs, and practical computer aided modeling processes. As a consequence, an ambitious decision was made, which increased my curiosity and devotion to this new issue of developing

a supporting tool for CPS designers, instead of supporting only designers of customizable CPSs. We believed that developing a successful modeling tool is possible if, and only if, the major characteristics of CPSs are taken into consideration with regards to pre-embodiment design, as well as the design thinking of CPS designers and architects. However, during the time of promotion research, I had to tackle a wide range of challenges and learn the relevant and needed bodies of knowledge of related disciplines such as software technology, information engineering, computational Multiphysics, warehouse technologies, and ontology technologies, just to mention the most important ones. The line of my research has significantly deviated from the initial line already at the beginning. I do hope that the achievements in developing SMF-based modeling framework prove the worthiness of our decision. For us it showed what real research is all about. Doing, learning and thinking, and then thinking, learning and doing again.

7.3 RECOMMENDATIONS FOR FUTURE RESEARCH

Since the 1980s feature-based modeling has been employed mostly in the context of designing mechanical and software parts using part-level features. The concept of feature-technology is revived in our work and conveyed to the system-level feature-based design. The research about system-level feature is still in its infancy and a proliferating future is expected in this scientific context. The advantages of system-level feature in modeling complex systems are confirmed through our research, and we recommend it as a novel topic for future scientific efforts.

Our research has introduced many merits of system-level feature-based modeling which cannot be used, tested, and validated until the full-fledged implementation of the developed modeling framework. We believe that the implementation of SMF-TB helps to reveal real advantages of SMF-based modeling, and shifts the current state of the modeling tools of complex system to a new level. The computational constructs created in our research is ready to be implemented by software tool programmers. The following research could be about practically implementation of the SMF-TB, evaluation of the results, providing concrete feedbacks and improving the concept for further functionalities. In addition, creation of ontological contents is the most critical research work that should be done towards implementing SMF-TB. Development of the information schema construct is performed by assuming the availability of the ontological contents in the relational tables. Researching on ontological contents and its implementation is suggested as a part of the research work towards SMF-TB realization.

After implementation of the SMF-TB many further opportunities could be disclosed. Among them, we can consider different types of SMFs such as human manifestation feature (HMF) and environment manifestation feature (EMF), with similar information structure and complementary interfaces. Providing open source online libraries of SMFs is another opportunity. It means that the SMFs defined by one can be shared with others. It helps rapid growth of databases due to sharing outcome of knowledge engineering works. Another implementation opportunity is to work on coupling of the SMF-TB with other external tools in order to extend its functionality.

SMFs can be used as means of interdisciplinary communication which opens a window for follow up researches. Step-wise definition of SMFs implies multiple level of concreteness in

formal definition of components. This potential characteristic makes SMFs suitable for being used as means of communication in multiple design phases, from conceptual ideation to concrete elaboration and detail design. In a complex system design project, usually, several engineering disciplines are involved in research, design, and development tasks; while there are various sequences and dependencies between their activities. Some of them use conceptual and ambiguous terminology, while others need some kind of fact numbers to work with. Interdisciplinary, multi-step and context-free nature of SMFs makes them suitable as a communication medium in this kind of situations. This is another ambitious research topic that could be elaborated as future research.

LIST OF ABBREVIATIONS

AKF	Architecture knowledge frame
ASIC	Application-specific integrated circuit
AW	Aggregated-ware
CBD	Component-based design
CBED	Case-based evolutionary design
CDFMC	Concurrent design for mass customization
CFBD	Conventional feature-based design
CPC	Cyber-physical computing
CPCD	Cyber-physical consumer durable
CPS	Cyber-physical system
CSC	Computational schema construct
CT	Continuous time
CW	Cyberware
DB	Databases
DBC	Design by customer
DE	Discrete event
DFA	Design for assembly
DFM	Design for manufacturing
DFMC	Design for mass customization
DFMP	Design for mass personalization
DIR	Design inclusive research
DOT	Dynamic operation in time
EMO	Engineering modus operandi
EOO	Equation-based object-oriented
EOP	Event-oriented programming
ERT	Entity-relationship tables
FK	Foreign key
FoO	Flow of operation
FSM	Finite state machine
GT	Genotype
HDL	Hardware description language
HMI	Human machine interface
HW	Hardware
ISC	Information schema construct
MBSE	Model-based system engineering
MC	Mass customization
MD	Modular design
MDA	Model driven architecture
MEMS	Micro-electro-mechanical system
MLaaS	Machine learning-as-a-service
MM	Mathematical morphology
MMR	Minimal morphological representation
MoC	Model of computation
MoS	Matrix of stream

MT	Mereotopology
NSF	National Science Foundation
ODP	Order decoupling point
OKF	Operation knowledge frame
PBD	Platform-based design
PCS	Procedural computational schema
PFA	Product family architecture
PFM	Product family modeling
PIM	Platform independent model
PK	Primary key
PLC	Product life cycle
PN	Process network
PSF	Paradigmatic system feature
PSM	Platform specific model
PT	Phenotype
RCs	Research cycles
RDT	Relational data tables
RIDC	Research in design context
SCPS	Social-cyber-physical system
SFMF	System-level feature-based modeling framework
SLF	System-level feature
SLFBD	System-level feature-based design
SMF	System manifestation feature
SMF-TB	System manifestation features-based modeling toolbox
SMT	Stratified mereotopology
SOCE	System-On-Chip Environment
SoS	System of system
ST-MT	Spatiotemporal mereotopology
SW	Software
TCD	For traditional consumer durable
TDL	Timing definition language
TOS	Task-oriented scheduling
TS	Temporal specification
UI	User interface
UoO	Unit of operation

LIST OF FIGURES

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

LIST OF TABLES

PUBLICATIONS BASED ON THE RESULTS OF THE PROMOTION RESEARCH

1. Pourtalebi, S., Horváth, I., & Opiyo, E. (2013). Multi-aspect study of mass customization in the context of cyber-physical consumer durables. In Proceedings of the International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. ASME, pp. V004T05A006-V004T05A006.

2. Pourtalebi, S., Horváth, I., & Opiyo, E. (2014). New features imply new principles? deriving design principles for mass customization of cyber-physical consumer durables. In Proceedings of the TMCE, Budapest, Hungary, pp. 95-108.

3. Pourtalebi, S., Horváth, I., & Opiyo, E. Z. (2014a). First steps towards a mereo-operandi theory for a system feature-based architecting of cyber-physical systems. In GI-Jahrestagung, pp. 2001-2006.

4. Pourtalebi, S., Horváth, I., & Opiyo, E. Z. (2014b). New features imply new principles? Deriving design principles for mass customization of cyber-physical consumer durables. In Proceedings of the 10th International Symposium on Tools and Methods of Competitive Engineering TMCE 2014, Budapest, Hungary, May 19-23, 2014, Delft University of Technology. pp. 129-142.

5. Horváth, I., & Pourtalebi, S. (2015). Fundamentals of a mereo-operandi theory to support transdisciplinary modeling and co-design of cyber-physical systems. In Proceedings of the International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. AMSE, pp. V01AT02A005-V01AT02A005.

6. Pourtalebi, S., & Horváth, I. (2016a). Towards a methodology of system manifestation features-based pre-embodiment design. Journal of Engineering Design, pp. 1-37.

7. Pourtalebi, S., & Horváth, I. (2016b). Information schema constructs for defining warehouse databases of genotypes and phenotypes of system manifestation features. Frontiers of Information Technology & Electronic Engineering, vol. 17, no. 9, pp. 862-884.

8. Pourtalebi, S., & Horváth, I. (2016c). Information schema constructs for instantiation and composition of system manifestation features. Frontiers of Information Technology & Electronic Engineering, vol. x, no. x, pp. x-x.

9. Pourtalebi, S., & Horváth, I. (2016d). Benchmarking the conceptual framework of a system-level manifestation features-based toolbox, (in preparation for journal publication), pp. x-x.

SUMMARY

BACKGROUND

The research reported in this thesis book has opened a new door towards modeling intricate technical systems. It focused on system-level feature-based architecting and modeling of cyber-physical systems (CPSs) in the pre-embodiment phase of design. The challenges have been addressed ambitiously in the conducted research by providing the required underpinning theories, implementation methodologies, and processes, as well as a conceptual framework for the toolbox and a unique strategy of validation. In the theoretical realm, two remarkable theories have been developed. With regards to implementation, the main contribution is in development of the concept of warehousing, the procedures of knowledge engineering, and the new pre-embodiment design approach of CPSs. The developed framework was benchmarked against those of the available commercial and experimental system development tools.

The main challenge of our research with regards to development of a supporting tool for architecting/designing CPSs was to achieve and maintain a balance between a comprehensive system-level view and the consideration of the operational and architectural attributes and performance of the lower-level components. This issue is crucial in designing complex, integrated, and heterogeneous CPSs. According to our assumption, system-level consideration is unquestionably necessary for pre-embodiment design (conceptualization and configuration) of CPSs, while designing of the components can even be delegated to other experts. This also helps address the compositionality and composability of CPSs simultaneously.

RESEARCH DOMAIN AND ADDRESSED RESEARCH PROBLEM

The emergence of the concept of CPSs shifts the attention from component to system, from fraction to the whole, from efficiency to effectiveness, from means to goals, and from product to service. In this new paradigm, "design" implies system-level conceptualization, architecting, and configuration. In the future, a part of design will even be delegated to self-aware and self-adaptive CPSs for run-time execution. Presently, components are mostly considered as black boxes, and composability issues are the main challenges of the CPS designers. The discipline-oriented view had a dominant influence on developing conventional design and modeling tools. Consequently, the problem of system-level design of CPSs remained ill-solved - nevertheless it has been recognized and emphasized that system-level design needs a holistic and interdisciplinary view.

The available system design and modeling tools suffer from the lack of a parallel consideration of the innate details of components and the system as a whole, and ignore physicality of CPSs in interaction with other systems and the embedding operational environments. Though widely used for designing CPSs, these modeling tools are mostly based on logical, analytical, and mathematical formulation. In order to model CPSs they typically apply some sorts of abstraction and simplification, which ignore a huge amount of attributes, constraints, and contracts. In fact, system designers are being pushed to formulate the relationship between components in an abstract level. Moreover, the available system-level modeling tools consider operations and architecture of a system as two separate aspects. Although this approach simplifies the inherent complexity of CPSs, ignoring mutual relations between architecture and operations is the side effect.

In our research, we assumed that by targeting three main characteristics of CPS, namely complexity, heterogeneity, and integrity, a novel tool would be able to sufficiently support CPS designers. However, the common solutions of applying excessive abstraction and simplification, and neglecting physicality of components should be avoided. Consequently, system-level feature-based

modeling of CPSs was chosen as the focus of our research. The required high-level semantic is supported by strong ontological resources. The theoretical fundamentals, the conceptual framework and the computational methodological approach have all been elaborated. However, the development of the supporting ontology and fully-fledged implementation of the conceptual modeling toolbox were excluded in our research due to the time limitation.

OVERALL RESEARCH APPROACH

The research started with studying the state-of-the-art in the domain of CPS conceptualization and design supporting tools. The explorative studies in five domains resulted in obtaining sufficient insight about the knowledge gap and helped narrowing down the phenomenon to system-level feature-based modeling of CPSs. In the second research cycle, our investigations were centered on developing the required underpinning theories. As a result, based on integration and enhancement of several classic theories, the mereo-operandi theory (MOT) was proposed to underpin our ideas and logical argumentation. It provided the required theoretical framework for development of the system manifestation feature (SMF) theory in the third research cycle. The SMF theory was meant as a complementary to MOT and explained how it could be methodologically applied. The SMF theory specified foundational information structures required for creation of the building blocks (named as SMFs) for system modeling.

In the fourth research cycle, the required information structures, the methodology of implementation, the computational constructs, and the procedures of processing information have been created in order to move towards a computational implementation. Finally, the developed conceptual framework of our modeling toolbox was benchmarked against those of a number of leading commercial and experimental tools. The results revealed that there are gaps between the functionalities provided by the available modeling tools and the requirements of pre-embodiment design of CPSs. Moreover, it has been evidenced that the novelties introduced by the proposed SMFs-based modeling framework in properly and effectively addresses the identified challenges. It has also been revealed that the proposed conceptual framework provides a higher potential in supporting multi-disciplinary pre-embodiment design of CPSs, and can be the basis of a knowledge-intensive smart system design application of the future.

RESULTS AND NOVELTIES

The conceptual, computational, and methodological elements of our SMFs-based modeling framework are comprehensively discussed in the thesis book. The following text briefly mentions the novelties of the proposed modeling framework from both computational application and implementation perspectives:

N1 **Imposing strictly physical view:** The imposed physical view supports aggregation of all components (HW, SW and CW) without the need for applying abstraction. Subsequently, all parameters of lower level components directly determine and affect the features of the higher level components.

N2 **Enforced concurrent consideration of architecture and operation:** It results in a simultaneous creation of both architectural and operational models of a system in interaction. This is in contrast with most of the logically- and mathematically-based modeling tools, which put emphasis on the functional aspects of components.

N3 **Amalgamating of architectural and operational aspects:** All information sets that define relations among architectural and operational aspects are captured within SMFs. This results in a robust approach of modeling, in which the changes due to the operation of the components are directly reflected on their architectural attributes, and vice versa.

N4 **Using uniform information structures for heterogeneous components:** The knowledge frames and accordingly the database schema for accommodating pieces of information are the same for all types of constituents. Consequently, the relations among them are being uniformly processed, and the various constituents are differentiated semantically, not structurally.

N5 **Multi-level multi-granularity:** It offers the possibility of moving from coarse-grained to fine-grained elaboration in a system model, and vice versa. In the SMFs-based modeling framework, SMFs have the same information structure on all aggregation levels. Accordingly, containment and connectivity relations are captured likewise on all levels, without applying abstraction and simplification.

N6 **Multi-aspects coupling among SMFs:** Both containment and connectivity relations of SMFs are supported through interfaces of SMFs, neglecting types of components. Consequently, the integration is supported by several aspects of multiple coupling. The dynamic integrity is also captured through utilizing concept of spatiotemporal mereotopology.

N7 **Multi-stage model composition:** In the proposed framework, building blocks of system models are not created in one go. Generating genotypes, deriving phenotypes, instantiation of phenotypes, and model composition are the stages that an SMF should go through to eventually end up in a model. As a result the manual information input is minimized by selection and mostly excluded in the conceptual design phase.

N8 **Multi-purpose system-level features:** Trans-disciplinary fusion of knowledge, reasoning with higher level semantics, and harmonization of pre-embodiment design and computational methodologies are the multiple purposes that are fulfilled with SMFs.

N9 **Multiple application contexts:** SMFs as semantically rich and self-contained system-level building blocks support both system-level conceptualization and system-level engineering (analyzing and optimization). They can also be used as means of communication among experts. SMFs are reusable and modifiable modeling entities that are suitable for various design and engineering approaches.

N10 **Multi-component warehouses:** Warehouses of the SMF-based toolbox benefit from three interconnected components to support SMF creation and model composition. In this way, the tracks of model composition and modification can be captured.

N11 **Benefiting from active ontologies:** Semantic-richness of the SMF-based toolbox is supported by active ontologies. Continuous updating of the ontological records is seen however necessary due to emergence of new technologies, materials, attributes, protocols, and so forth. Although, effort-intensiveness of this task can be considered as a drawback, the usefulness of the results cannot be disregarded.

VALIDATION AND CONCLUSIONS

The theoretical results of our research have been validated through benchmarking against five commercial and academic modelling tools and languages. The aspects of comparison were chosen according to the characteristics of CPSs and the requirements of pre-embodiment design. Since the modeling framework introduced by this research is not fully implemented as a prototype toolbox, many of its functionalities could not be measured definitely. Consequently, it was compared with the characteristics of the frameworks of the available modeling tools, which were derived by a kind of reversed engineering. To reduce the uncertainty caused by the reasoning, we used literature study and relied on the opinion of CPS modeling experts who participated in our survey. The comparison of strengths and weaknesses of the modeling tools and the analyses of the findings of the conducted study approved the uniqueness of the proposed solution, as well as its sufficiency to support pre-embodiment design of CPSs.

The analysis of the findings of our research confirmed the expectable advantages of using system-level features in modeling complex systems. Putting together everything, our impression has been that though the research concerning system-level features is in general still in its infancy, a rapid and wholescale proliferation may be expected in the near future. Based on the findings, we propagate this scientific phenomenon and knowledge exploitation context as a novel topic for future scientific efforts. Though our research has introduced and elaborated on many merits of system-level feature-based modeling, it cannot be used, tested, and validated in practical application until a full-fledged implementation of the proposed modeling approach is available.

PROPOSITIONS

- In contrast with the abstractions applied by logic-oriented and analytical modeling methodologies, system manifestation features-based modeling is able to capture the physicality of cyber-physical systems (CPSs) and their tight interaction with the surrounding physical world in the pre-embodiment design phase. Δ
- For system-level modelling of CPSs, architecture aspects and operation aspects should be considered concurrently, together with their interrelations. Δ
- While the architecture of CPSs can be decomposed hierarchically, the decomposition of their operations cannot be hierarchical. Δ
- As computational building blocks of high level semantics, system manifestation features can encapsulate all pieces of information required for modelling and simulation of the architecture and operation of CPSs. Δ
- The growing number of Internet repositories, the popularity of social platforms, and the ease of network access to digital contents raise the feeling that the future of learning is searching. Δ
- The current technological and social trends significantly change the relationship of individuals and communities to science. Δ
- Using the available computational design tools for conceptualization of truly complex systems is similar to tightening up a bolt with a hammer. Δ
- Overcoming complexity, integrity and heterogeneity challenges needs complex, integral and heterogeneous thinking. Δ
- There is no common understanding about the essence of cyber-physical systems among CPS experts. Δ
- The best' is a time-dependent notion - there comes 'a better' always. Δ

These propositions are regarded as opposable and defendable, and have been approved as such by the supervisor prof. dr. Imre Horvath.

SAMENVATTING

ACHTERGROND

Het onderzoek in dit proefschrift biedt nieuwe benaderingen om ingewikkelde systemen te modelleren. Het richtte zich op architectuurbepaling en modelleren van cyberfysische systemen (CFS'en) op kenmerkniveau (feature level), in de pre-materialisatiefase van het ontwerpen. Door de benodigde onderbouwende theorieën, implementatiemethodologieën en processen voor de toolbox te verstrekken, vergezeld van een conceptueel kader en een unieke validatiestrategie, zijn de uitdagingen in het uitgevoerde onderzoek op ambitieuze wijze aangepakt. De theoretische bijdrage bestaat uit twee opmerkelijke theorieën. Wat de implementatie betreft bestaat de belangrijkste bijdrage uit de ontwikkeling van het concept warehousing, kennistechnologie-procedures, en een nieuwe pre-materialisatie-ontwerpaanpak voor CFS'en. Het ontwikkelde raamwerk is getoetst door vergelijking met bestaande op de markt verkrijgbare en experimentele systeem¬ontwikkelingstools.

Wat de ontwikkeling van een ondersteuningsgereedschap voor architectuurbepaling en ontwerp van CFS'en betreft, was de voornaamste uitdaging in ons onderzoek het realiseren en handhaven van evenwicht tussen beschouwing op het niveau van enerzijds het gehele systeem en anderzijds dat van operationele en architecturale attributen van de deelcomponenten. Het gehele ontwerpen van complexe, geïntegreerde en heterogene CFS'en draait om deze kwestie. Waar het ontwerp van de componenten ook aan andere deskundigen kan worden overgelaten, zijn wij ervan uitgegaan dat beschouwing op systeemniveau in het pre-materialisatiestadium (conceptualisatie en configuratie) van CFS-ontwerp onontbeerlijk is. Op die manier kan ook gelijktijdig rekening gehouden worden met met de compositionaliteit en samenstelbaarheid van CFS'en.

ONDERZOEKSTERREIN EN AANGEPAKT ONDERZOEKSPROBLEEM

De opkomst van het begrip CFS doet de aandacht verschuiven van component naar systeem, van deel naar geheel, van efficiëntie naar effectiviteit, van middel naar doel, en van product naar dienst. Onder dit nieuwe paradigma impliceert "ontwerpen" conceptualisatie, architectuurbepaling en configuratie op systeemniveau. In de toekomst zal een deel van het ontwerp zelfs in runtime worden uitbesteed aan CFS'en die zijn uitgerust met zelfbesef en –aanpassingsvermogen. Nu nog worden componenten vooral beschouwd als black boxes, en vormen samenstelbaarheidsproblemen de belangrijkste uitdaging voor CFS-ontwerpers. De disciplinegebonden benadering heeft veel invloed gehad op de ontwikkeling van conventionele ontwerp -en modelleertools. Bijgevolg is het probleem van CFS-ontwerp op systeemniveau maar deels opgelost – desalniettemin werd het onderkend en benadrukt dat voor ontwerp op systeemniveau een holistische en interdisciplinaire insteek nodig is.

De beschikbare tools voor systeemontwerp en –modellering getuigen van een gebrek aan simultane beschouwing van enerzijds ingewikkelde componentdetails en anderzijds het gehele systeem, en zij negeren de fysische aard van CFS'en in de interactie met andere systemen en de operationele omgevingen waarin ze zijn ingebed. Hoewel ze alom gebruikt worden zijn deze tools te zeer gebaseerd op logische, analytische en wiskundige formulering. Voor het modelleren van CFS'en zijn dan bepaalde abstracties en simplificaties nodig, waarmee een enorme hoeveelheid attributen, restricties en verbanden verloren gaat. Eigenlijk worden systeemontwerpers zo gedwongen om relaties tussen de componenten op een abstract niveau te formuleren. Bovendien beschouwen modelleertools op systeemniveau de werking en de architectuur van het systeem als onderling gescheiden. Zo wordt de inherente complexiteit van CFS'en gereduceerd ten koste van de wederzijdse relaties tussen de architectuur en de werking.

In dit onderzoek hebben we aangenomen dat, door zich te richten op drie hoofdkenmerken van CFS'en namelijk complexiteit, heterogeniteit en compleetheid, een nieuwe tool in staat zou moeten zijn om CFS-ontwerpers toereikend te ondersteunen. Daarbij moet wel worden vermeden dat, zoals in de gebruikelijke aanpak, buitensporige abstractie en simplificatie worden afgedwongen, en de fysische aard van componenten wordt genegeerd. Daarom is het modelleren van CFS'en op systeemniveau uitgaand van features gekozen als onderzoeksfocus. De benodigde semantiek op hoog abstractieniveau wordt ondersteund door middel van robuuste ontologische kennismiddelen. De theoretische fundamenten, het conceptuele raamwerk en de computationeel-methodologische aanpak zijn alle volledig uitgewerkt, maar vanwege de beperkt beschikbare tijd zijn de ontwikkeling van de achterliggende ontologie en de volwaardige implementatie van de modelleertoolbox buiten beschouwing gelaten.

ALGEHELE ONDERZOEKSAANPAK

Het onderzoek is begonnen met het in kaart brengen van de state-of-the-art in CFS-conceptualisatie en tools voor ontwerpondersteuning. Verkennende studies in vijf domeinen hebben voldoende inzicht opgeleverd in de kenniskloof en hebben zo geholpen om het fenomeen in te perken tot kenmerkgebaseerd modelleren van CFS'en op systeemniveau. In de tweede onderzoekscyclus richtten onze studies zich op het ontwikkelen van de benodigde onderbouwende theorieën. Daaruit kwam, op basis van integratie en verbetering van verscheidene klassieke theorieën, de mereo-operanditheorie (MOT) voort, die onze ideeën en logische argumentatie moest onderbouwen. Deze verleende het benodigde theoretische kader om in de derde onderzoekscyclus de systeemmanifestatiefeatures(SMF)-theorie te ontwikkelen. De SMF-theorie is bedoeld als aanvulling op de MOT om te verklaren hoe deze methodologisch kon worden toegepast. De SMF-theorie specificeert de basisinformatiestructuren die nodig zijn om de bouwstenen voor systeemmodellering, die SMF's genoemd worden, aan te maken.

In de vierde onderzoekscyclus zijn de benodigde informatiestructuren, de implementatiemethodologie, de computationele constructies, en de informatieverwerkingsprocedures vastgelegd, om de stap naar implementatie te kunnen maken. Als laatste is het ontwikkelde conceptuele raamwerk gebenchmarkt tegenover de conceptuele raamwerken van een aantal toonaangevende op de markt verkrijgbare en experimentele tools. De uitkomsten toonden hiaten aan tussen enerzijds de functionaliteiten geboden door de beschikbare modelleertools en anderzijds de eisen vanuit de pre-materialisatiefase van CFS-ontwerp. Bovendien is bewezen dat de noviteiten die het SMF-gebaseerde modelleringskader biedt de geconstateerde uitdagingen deugdelijk en effectief aanpakken. Ook is aangetoond dat het voorgestelde conceptuele kader een groter potentieel biedt qua ondersteuning van multidisciplinair pre-materialisatie-CFS-ontwerp, en de basis kan vormen voor toekomstige kennisintensieve smart-system-ontwerpsoftware.

N1 **Voorschrijven van een strikt fysische beschouwingswijze:** De voorgeschreven fysische beschouwingswijze ondersteunt de samenvoeging van alle componenten (hardware, software en cyberware) zonder abstractie te vereisen. Bijgevolg bepalen alle parameters van componenten op laag niveau direct de kenmerken van de componenten op hogere niveaus.

N2 **Geforceerde gelijktijdige beschouwing van architectuur en werking** resulteert in het simultaan ontstaan van zowel architecturale als operationele modellen van een interacterend systeem. Dit in tegenstelling tot de meeste logica- en wiskundegebaseerde modelleertools die de functionele aspecten van componenten benadrukken.

N3 **Samensmelting van architecturale en operationele aspecten**: Alle informatieverzamelingen die relaties tussen architecturale en operationele aspecten vastleggen zijn vertegenwoordigd in SMF's. Zo ontstaat een robuuste modelleerbenadering waarin verander-

ingen als gevolg van de werking van componenten direct weerslag hebben op hun architecturale eigenschappen en vice versa.

N4 **Gebruikmaking van uniforme informatiestructuren voor heterogene componenten:** de kenniskaders en overeenkomstig het databaseschema om stukjes informatie onder te brengen zijn dezelfde voor alle soorten bestanddelen. Daardoor worden alle onderlinge relaties op uniforme wijze verwerkt, en de verscheidene bestanddelen worden niet structureel maar semantisch onderscheiden.

N5 **Meervoudige granulariteit over meerdere niveaus** biedt de mogelijkheid om grofkorrelige en fijnkorrelige uitwerking af te wisselen. In het SMF-gebaseerde modelleerraamwerk hebben SMF's dezelfde informatiestructuur op alle aggregatieniveaus. Dienovereenkomstig worden omvattings- en verbindingsrelaties ook zo vastgelegd op alle niveaus zonder abstractie en vereenvoudiging toe te passen.

N6 **Multi-aspectkoppeling tussen SMF's:** Zowel omvattings- als verbindingsrelaties van SMF's worden door interfaces van SMF's ondersteund, waarbij componenttypes genegeerd worden. Zo wordt de integratie ondersteund door verscheidene aspecten van meervoudige koppeling. De dynamische integriteit wordt ook vastgelegd door het begrip spatiotemporele mereotopologie toe te passen.

N7 **Meertraps-modelsamenstelling:** in het voorgestelde raamwerk worden bouwstenen niet in een keer aangemaakt. Het aanmaken van genotypen, het afleiden van fenotypen, het instantiëren van fenotypen en modelsamenstelling zijn de stappen die een SMF moet ondergaan om uiteindelijk een model op te leveren. Bijgevolg wordt het handmatig moeten invoeren van informatie geminimaliseerd door selectie, en grotendeels uitgesloten in de conceptuele ontwerpfase.

N8 **Multipurpose features (kenmerken) op systeemniveau:** Transdisciplinaire samensmelting van kennis, redeneren met semantiek op hoog niveau, en harmonisatie van prematerialisatieontwerp met computationele methodologieën vormen de meervoudige doelstellingen die door SMF's gerealiseerd worden.

N9 **Veelvoud aan contexten qua toepassing:** SMF's als semantisch verrijkte onafhankelijke bouwstenen ondersteunen zowel conceptualisatie als engineering op systeemniveau (analyse en optimalisatie). Ze kunnen ook worden gebruikt als communicatiemiddel tussen experts. SMF's vormen herbruikbare en modificeerbare modelleereenheden die geschikt zijn voor verscheidene ontwerp- en engineeringsbenaderingen.

N10 **Meer-componenten-depots:** Depots (warehouses) van de SMF-gebaseerde toolbox profiteren van drie onderling verbonden componenten die het ontstaan van SMF's en het componeren van modellen ondersteunen. Op die manier kunnen de sporen van modelsamenstelling en –modificatie worden vastgelegd

N11 **Profiteren van actieve ontologieën:** de rijke semantiek van de SMF-gebaseerde toolbox wordt ondersteund door actieve ontologieën. De vastgelegde ontologische gegevens moeten echter voortdurend worden bijgewerkt in verband met de opkomst van nieuwe technologieën, materialen, protocollen etc. Hoewel de arbeidsintensiviteit van die taak als een nadeel kan worden opgevat moet de bruikbaarheid van de resultaten niet worden onderschat.

VALIDATIE EN CONCLUSIES

De theoretische resultaten van dit onderzoek zijn gevalideerd d.m.v. benchmarking in vergelijk met vijf op de markt verkrijgbare en in de academische wereld voorgestelde modellertools en –talen. De vergelijkingsaspecten zijn gekozen met het oog op de kenmerken van CFS'en en de eisen

vanuit het pre-materialiserend ontwerpen. Omdat het voorgestelde modelleerraamwerk nog niet als toolbox-prototype geïmplementeerd is, konden veel functionaliteiten niet concreet getoetst worden. Daarom is het vergeleken op basis van karakteristieken, die van de modelleerraamwerken van beschikbare tools zijn afgeleid d.m.v. een soort reverse engineering. Om de daarmee geïntroduceerde onzekerheid te reduceren hebben we een literatuurstudie gedaan en zijn we uitgegaan van de oordelen van CFS-modelleringsdeskundigen aan wie we een enquête hebben voorgelegd. De vergelijking van sterktes en zwaktes van de modellertools en analyse van de uitkomsten van het onderzoek bewezen zowel de uniekheid van de voorgestelde oplossing als haar geschiktheid om pre-materialisatieontwerp van CFS'en te ondersteunen

Analyse van de resultaten van het onderzoek bevestigden de verwachtbare voordelen van het gebruik van kenmerken op systeemniveau bij het modelleren van complexe systemen. Alles bij elkaar beschouwend is onze indruk dat, hoewel ontwerp naar kenmerken op systeemniveau nu nog in zijn kinderschoenen staat, er in de nabije toekomst een snelle uitbreiding naar toepassing op volle schaal kan worden verwacht. Op basis van de uitkomsten willen we dit wetenschappelijke fenomeen en deze context van kennisexploitatie naar voren brengen als een onderwerp van toekomstig vernieuwend wetenschappelijk onderzoek. Hoewel het onderzoek vele voordelen van kenmerkgebaseerd ontwerpen op systeemniveau heeft geïntroduceerd en uitgewerkt kan het nog niet in de praktijk worden gebruikt, getest en gevalideerd zo lang er nog geen volstandige implementatie van de voorgestelde modelleerbenadering beschikbaar is.

STELLINGEN

- In tegenstelling door logica-gerichte en analytische modelleertools die abstracties hanteren, kan systeemmanifestatiefeatures-gebaseerd modelleren de fysische aard van cyber-fysische systemen (CFS'en) en de nauwe interactie met de fysieke omgeving bevatten. Δ
- Voor het modelleren van CFS'en op systeemniveau moeten architectuur- en werkingsaspecten, alsmede de verwevenheid daarvan, gelijktijdig beschouwd worden. Δ
- De architectuur van CFS'en kan hiërarchisch worden ontleed maar hun werking niet. Δ
- Als computationële bouwstenen van semantiek op hoog abstractieniveau kunnen systeemmanifestatie-features alle brokken informatie omvatten die nodig zijn voor het modelleren van simuleren van de architectuur en de werking van CFS'en. Δ
- Het toenemende aantal internetarchieven, de populariteit van sociale netwerken en het gemak van netwerktoegang tot digitale content doen het gevoel opkomen dat zoeken de toekomst van leren is. Δ
- De huidige technologische en sociale trends veranderen de verhouding van individuen en gemeenschappen tot de wetenschap. Δ
- Het gebruiken van de beschikbare computationële ontwerptools voor het conceptualiseren van waarlijk complexe systemen is vergelijkbaar met het gebruik van een hamer om een bout aan te draaien. Δ
- Het overwinnen van complexiteits-, integriteits- en heterogeniteitsuitdagingen behoeft complex, integraal en heterogeen denken. Δ
- Er bestaat onder CFS-deskundigen geen overeenstemming omtrent de essentie van cyber-fysische systemen. Δ
- 'Het beste' is een tijdsafhankelijk begrip. Er komt altijd een 'beter.' Δ

Deze stellingen worden opponeerbaar en verdedigbaar geacht en als zodanig goedgekeurd door de promotor, prof. dr. Imre Horváth.

Shahab Pourtalebi

March 31, 1981, Iran

Education

Sep. 2012 – Apr. 2017
PhD
Design Engineering department, IDE, TU Delft

- Topic: System-level feature-based modeling of Cyber-Physical Systems

Sep 2003 - July 2006
MASTER's DEGREE
Art University of Tehran

- Major: Industrial design engineering
- Thesis topic: Perseverance in sport, designing a multipurpose fitness station for home use
- Top student in Industrial design

Sep. 1999 – Sep. 2003
BACHELOR's DEGREE
Sahand University of Technology

- Major: Industrial design engineering
- Honor project: Design of citrus juice extractor
- Top student in Industrial design

Projects and awards

- Research project: feasibility study: design and implementation of Tabriz MiniCity (2011).
- Research project: artificial intelligent aided product design methods (2009-2011).
- Research project: implementation and feasibility test of distance learning through learning management system in Tabriz IA University (2008-2010).
- Best paper award: conference on new technologies in home appliances, Isfahan, Iran (2012).
- Best master thesis: perseverance in sport and designing of multipurpose fitness station for home use (2006).
- The winner of Khwarizmi Award for the Invention of Rassam; a new digital drawing device (1996).

Experience

Sep. 2012 - present
GUEST RESEARCHER
Delft University of Technology

Jul. 2006 – Sep. 2012
LECTURER
Industrial Design Department, Tabriz IA University

Feb. 2008 – Feb. 2011
HEAD OF DEPARTMENT
Industrial Design Department, Tabriz IA University

Dec. 2006 – Feb. 2008
DEPUTY DIRECTOR
Industrial Design Department, Tabriz IA University

Jan. 2004 – Jul. 2006
DESIGNER / PROJECT MANAGER
NAK interior design studio

Sep. 2002 – Sep. 2004
DESIGNER (part-time)
PARS ZARASA household design and industrial company

Sep. 2000 – Sep. 2002
DESIGNER (part-time)
PARS KHAZAR home appliance design and production company

ACKNOWLEDGEMENT

I would never have been able to finish my thesis without the guidance of my promoter and committee members, help from friends, and support from my family and wife. It is not possible to thank everyone who helped me. However, I would like to specially thank the greatest helps.

Firstly, I would like to express my sincere gratitude to my promoter Prof. Dr. Imre Horvath for the continuous support of my research, for his patience, motivation, and immense knowledge. I learned and enjoyed a lot from our discussions. I'll never forget how the concept of MOT and SMF were created from our long discussions. His commitment to this research was incredible, and his guidance helped me in all the time of research and writing of this thesis.

My sincere thanks also go to Dr. Eliab Z. Opiyo who helped me in the first two years of this research. It was unfortunate that we were not able to continue our collaboration.

A very special gratitude goes out to all colleagues and friends in CPS research group for helping and providing support for the work: Wilhelm Frederik van der Vegte, Zoltán Rusák, Chong Li, Yongzhe Li, Garrett Keenaghan, Santiago Ruiz-Arenas, Elizabeth Rendon-Velez, Azrol Kassim, Ellemieke Van Doorn,Fatima-Zahra Abou Eddahab, Kanter van Deurzen, Fabian Mulder, Els Du Bois, and also to everyone in the Design Engineering department of IDE faculty.

With a special mention to my friends in Delft, Babak Raji, Mohammad Taleghani, Farzaneh Fakhreddin, Saleh Heidary Shalmany, Samira Amani, Milad Mehrpou, Bahar Barati, Vahid Arbabi, Firouzeh Farokhzad, Amin Fatemi, Amin Firouzi, Fatima Anisi, Hadi Jamali-Rad, Dewit Ivo, Argun Cencen, Asli Boru, Mostafa Hajian, Amin Amani, Ehsan Baha, Zjenja Doubrovski, and many others who have not been mentioned.

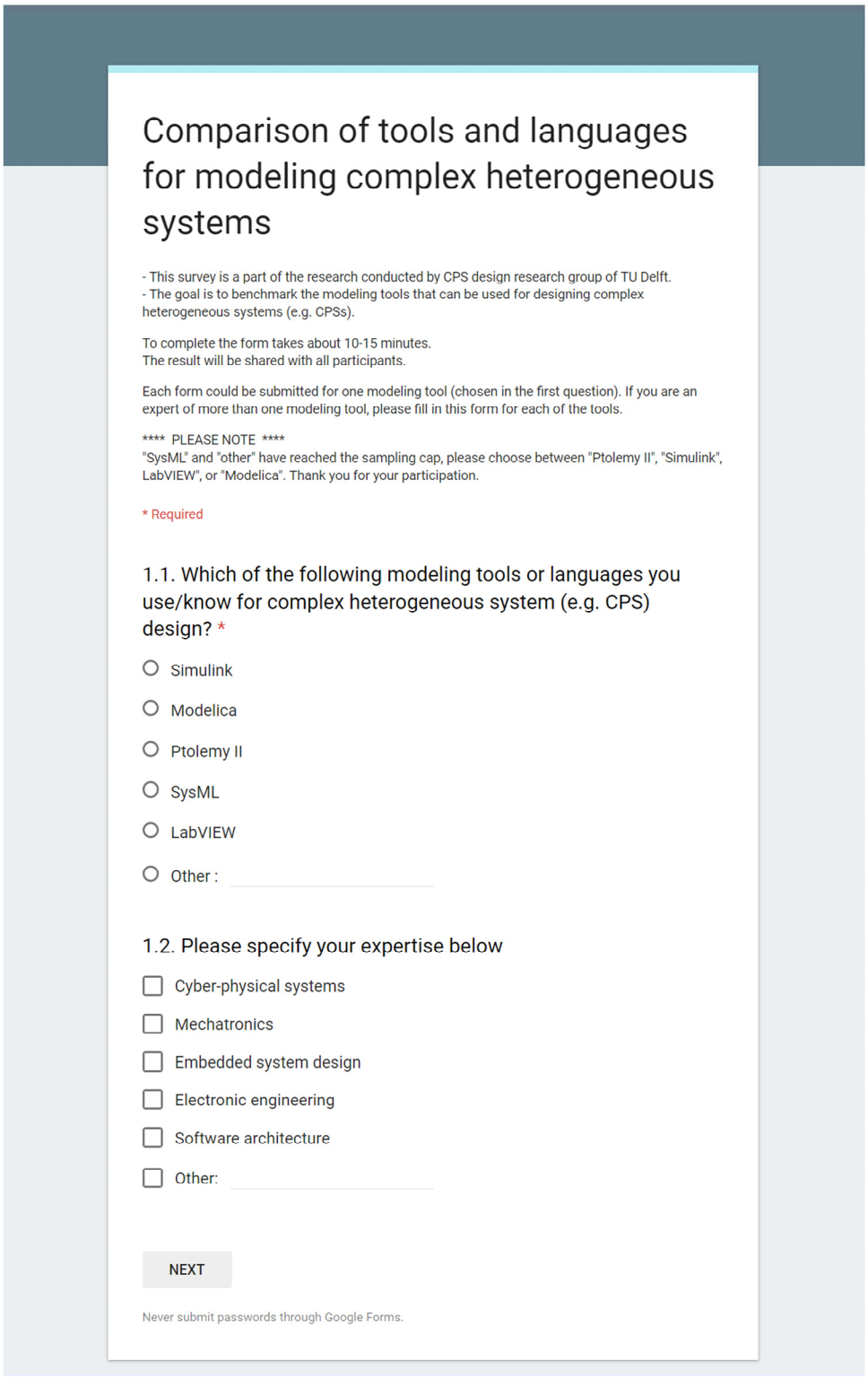
And finally, last but by no means least, I am grateful to my wife, Sanaz, who was always there cheering me up and stood by me through the good times and bad. And thanks to my little Daniel, who encouraged me to finish my PhD. I am also grateful to my family members, my parents and brothers Arash and Arad, as well as my in-laws specially Daryoush who were always supporting me and encouraging me with their best wishes along the way.

Thanks for all your encouragement and support!

APPENDICES

APPENDIX 1: Questionnaire

Page 1: Q1.1 and Q1.2



Comparison of tools and languages for modeling complex heterogeneous systems

- This survey is a part of the research conducted by CPS design research group of TU Delft.
- The goal is to benchmark the modeling tools that can be used for designing complex heterogeneous systems (e.g. CPSs).

To complete the form takes about 10-15 minutes.
The result will be shared with all participants.

Each form could be submitted for one modeling tool (chosen in the first question). If you are an expert of more than one modeling tool, please fill in this form for each of the tools.

**** PLEASE NOTE ****
"SysML" and "other" have reached the sampling cap, please choose between "Ptolemy II", "Simulink", LabVIEW", or "Modelica". Thank you for your participation.

* Required

1.1. Which of the following modeling tools or languages you use/know for complex heterogeneous system (e.g. CPS) design? *

- ○ Simulink
- ○ Modelica
- ○ Ptolemy II
- ○ SysML
- ○ LabVIEW
- ○ Other :

1.2. Please specify your expertise below

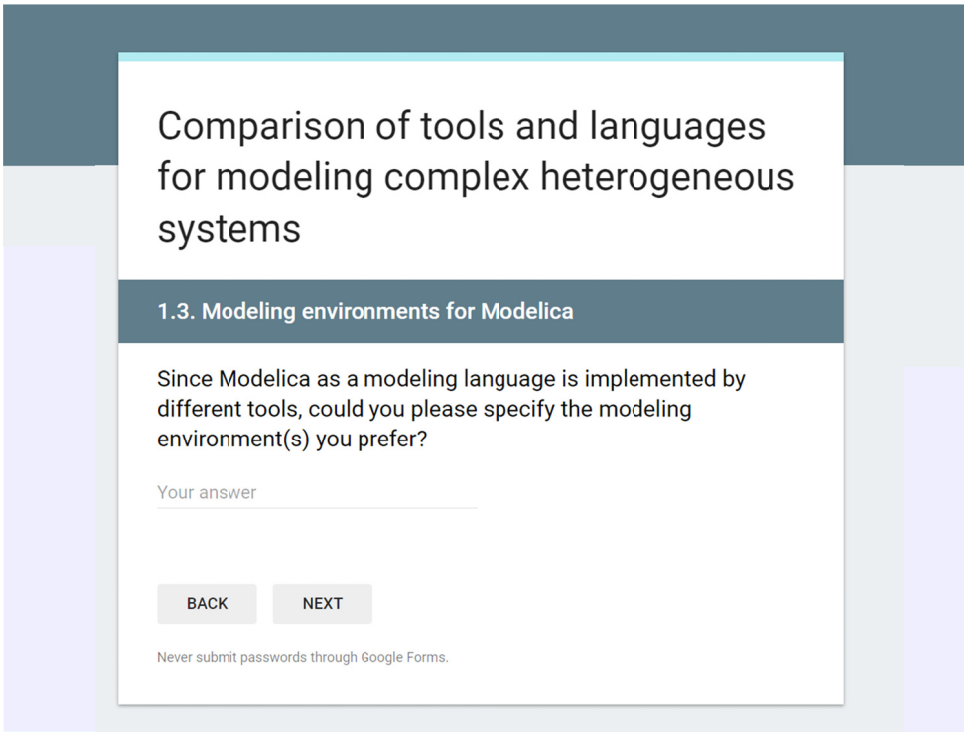
- ☐ Cyber-physical systems
- ☐ Mechatronics
- ☐ Embedded system design
- ☐ Electronic engineering
- ☐ Software architecture
- ☐ Other:

NEXT

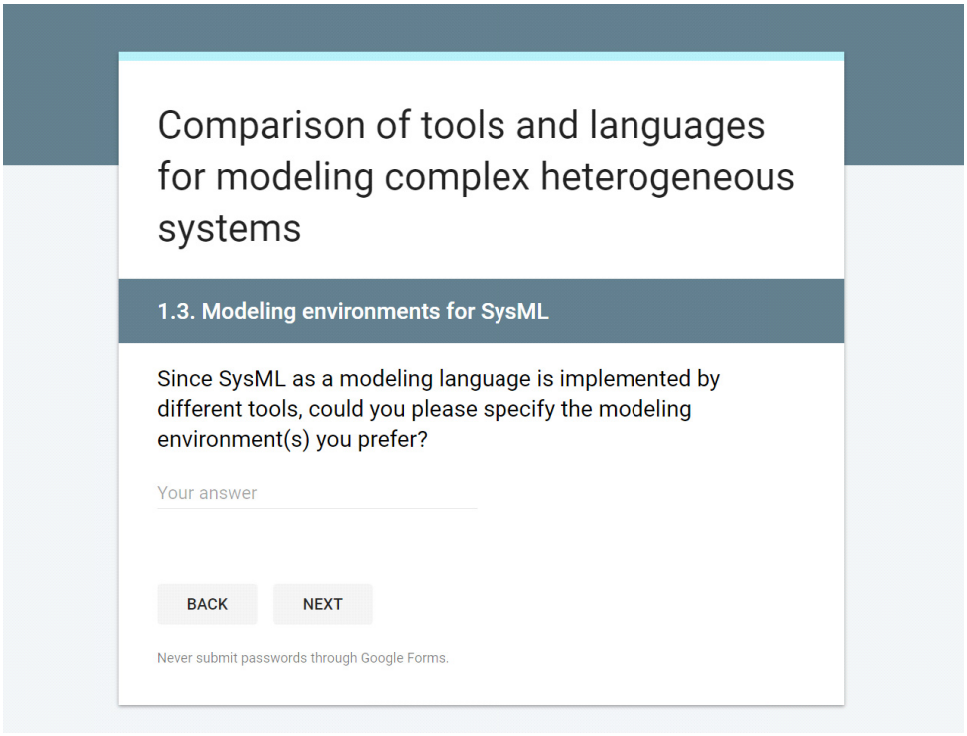
Never submit passwords through Google Forms.

Page 1: Q1.3

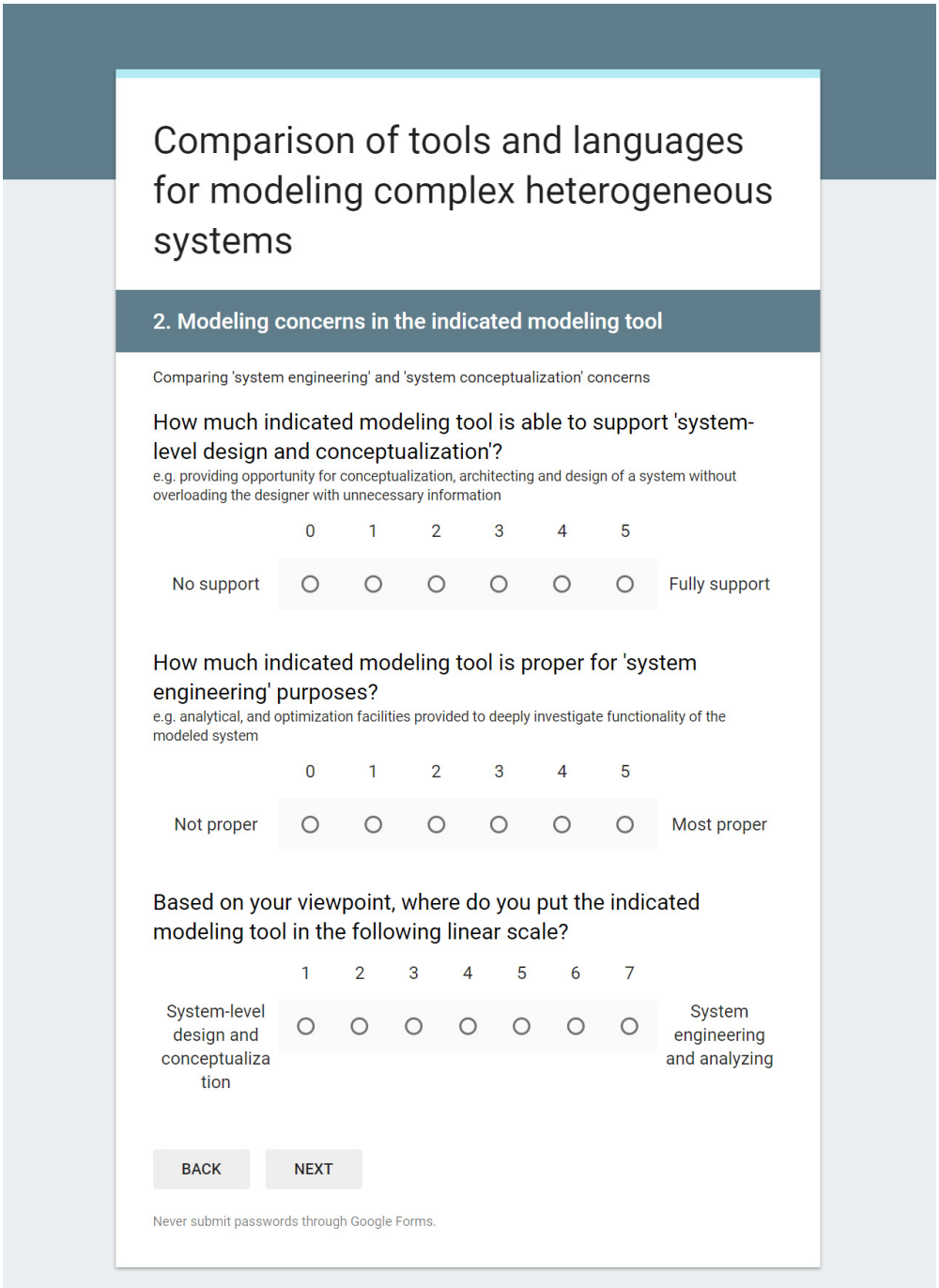
If a respondent selects Modelica in Q1.1, s/he is redirected to this page in order to specify the modeling environment (tool) that is preferred.



The same link created to the other page for SysML, as it is a modeling language which is incorporate in several modeling environments.



Page 3: Q2.1, Q2.2, and Q2.3



Comparison of tools and languages for modeling complex heterogeneous systems

2. Modeling concerns in the indicated modeling tool

Comparing 'system engineering' and 'system conceptualization' concerns

How much indicated modeling tool is able to support 'system-level design and conceptualization'?

e.g. providing opportunity for conceptualization, architecting and design of a system without overloading the designer with unnecessary information

	0	1	2	3	4	5	
No support	○	○	○	○	○	○	Fully support

How much indicated modeling tool is proper for 'system engineering' purposes?

e.g. analytical, and optimization facilities provided to deeply investigate functionality of the modeled system

	0	1	2	3	4	5	
Not proper	○	○	○	○	○	○	Most proper

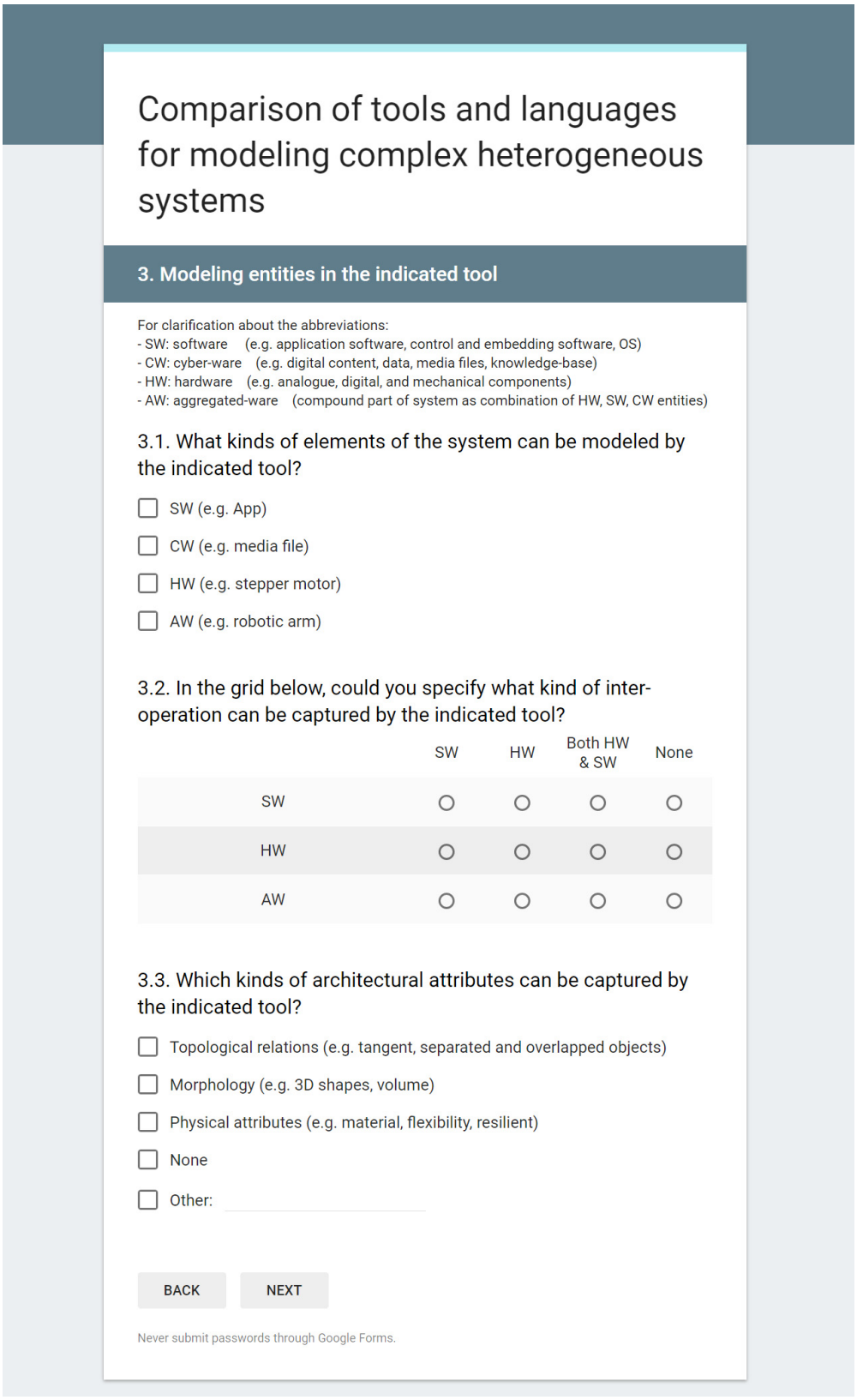
Based on your viewpoint, where do you put the indicated modeling tool in the following linear scale?

	1	2	3	4	5	6	7	
System-level design and conceptualization	○	○	○	○	○	○	○	System engineering and analyzing

BACK NEXT

Never submit passwords through Google Forms.

Page 4: Q3.1, Q3.2, and Q3.3



Comparison of tools and languages for modeling complex heterogeneous systems

3. Modeling entities in the indicated tool

For clarification about the abbreviations:
- SW: software (e.g. application software, control and embedding software, OS)
- CW: cyber-ware (e.g. digital content, data, media files, knowledge-base)
- HW: hardware (e.g. analogue, digital, and mechanical components)
- AW: aggregated-ware (compound part of system as combination of HW, SW, CW entities)

3.1. What kinds of elements of the system can be modeled by the indicated tool?

- [ ] SW (e.g. App)
- [ ] CW (e.g. media file)
- [ ] HW (e.g. stepper motor)
- [ ] AW (e.g. robotic arm)

3.2. In the grid below, could you specify what kind of inter-operation can be captured by the indicated tool?

	SW	HW	Both HW & SW	None
SW	○	○	○	○
HW	○	○	○	○
AW	○	○	○	○

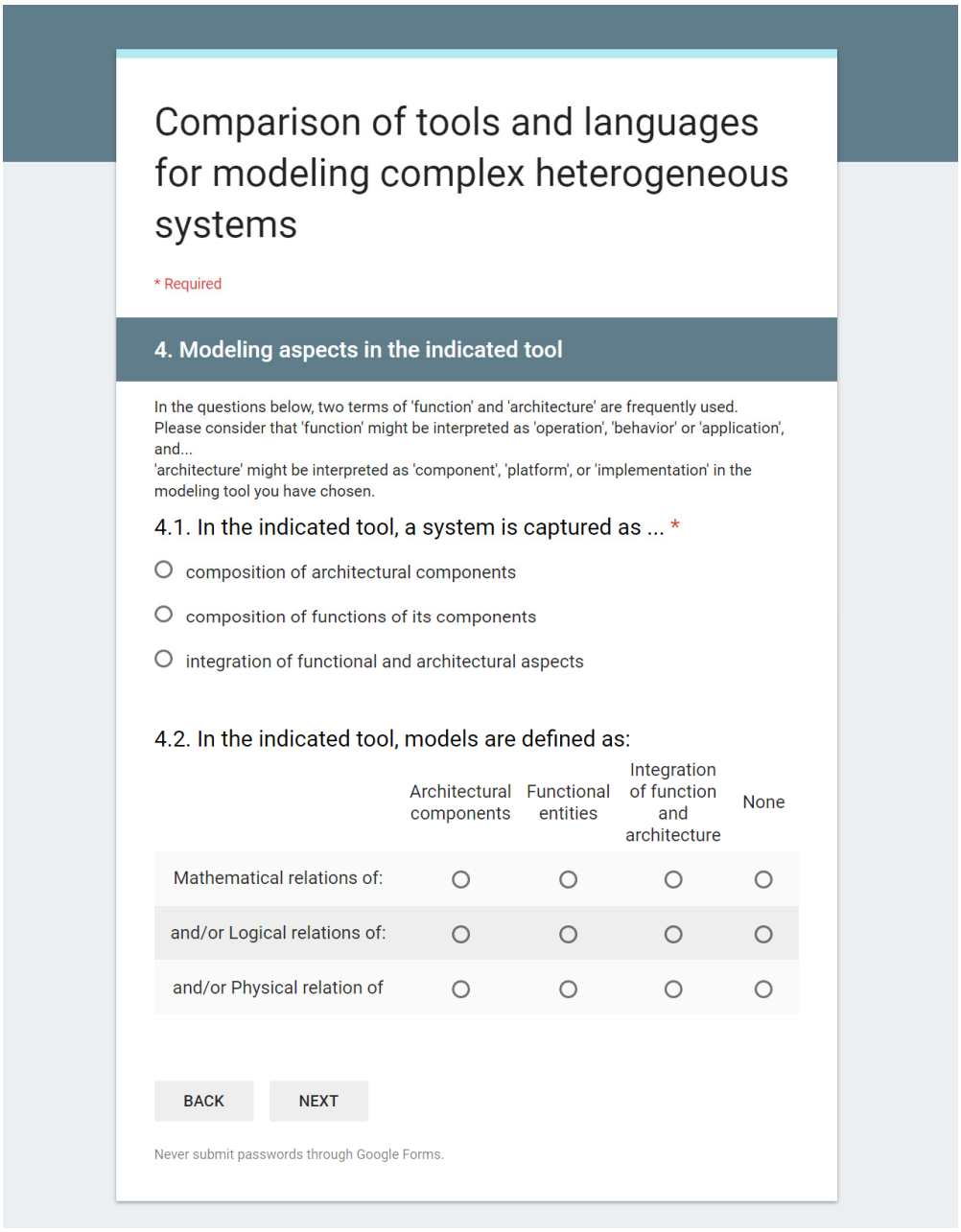
3.3. Which kinds of architectural attributes can be captured by the indicated tool?

- [ ] Topological relations (e.g. tangent, separated and overlapped objects)
- [ ] Morphology (e.g. 3D shapes, volume)
- [ ] Physical attributes (e.g. material, flexibility, resilient)
- [ ] None
- [ ] Other:

BACK NEXT

Never submit passwords through Google Forms.

Page 5: Q4.1 and Q4.2



Comparison of tools and languages for modeling complex heterogeneous systems

* Required

4. Modeling aspects in the indicated tool

In the questions below, two terms of 'function' and 'architecture' are frequently used.
Please consider that 'function' might be interpreted as 'operation', 'behavior' or 'application', and...
'architecture' might be interpreted as 'component', 'platform', or 'implementation' in the modeling tool you have chosen.

4.1. In the indicated tool, a system is captured as ... *

- composition of architectural components
- composition of functions of its components
- integration of functional and architectural aspects

4.2. In the indicated tool, models are defined as:

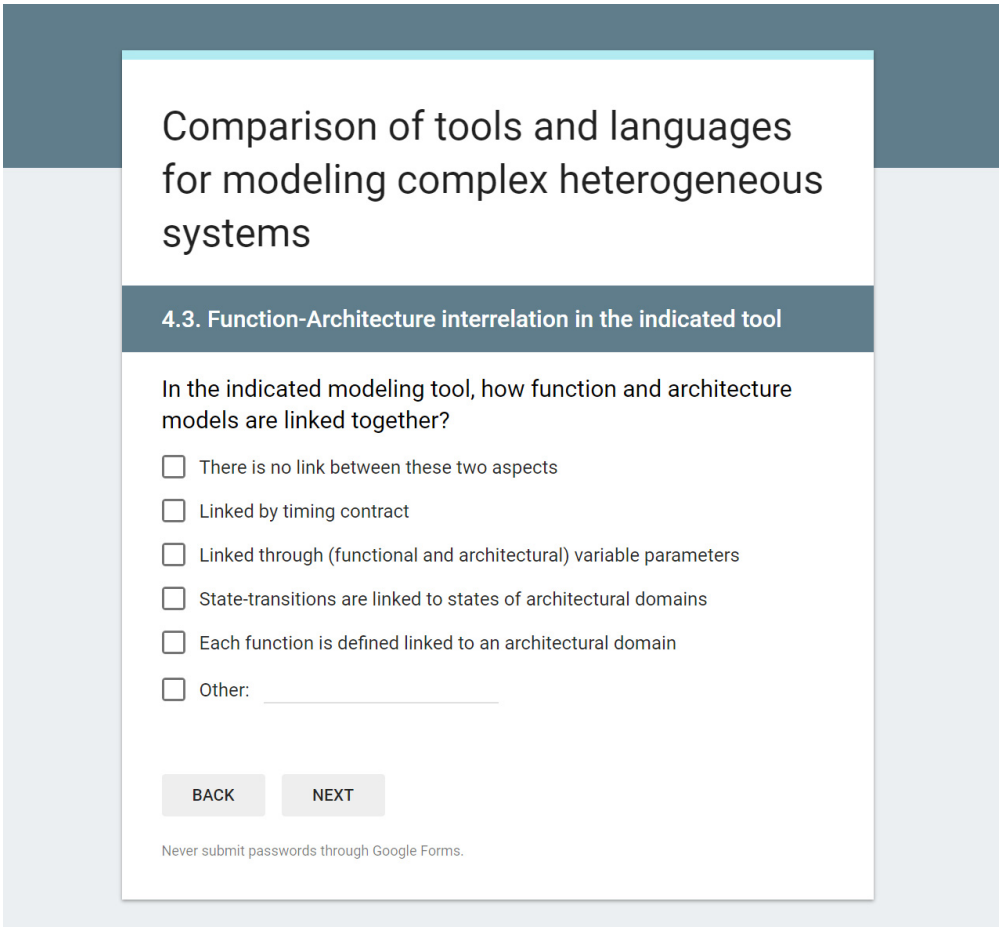
	Architectural components	Functional entities	Integration of function and architecture	None
Mathematical relations of:				
and/or Logical relations of:				
and/or Physical relation of				

BACK NEXT

Never submit passwords through Google Forms.

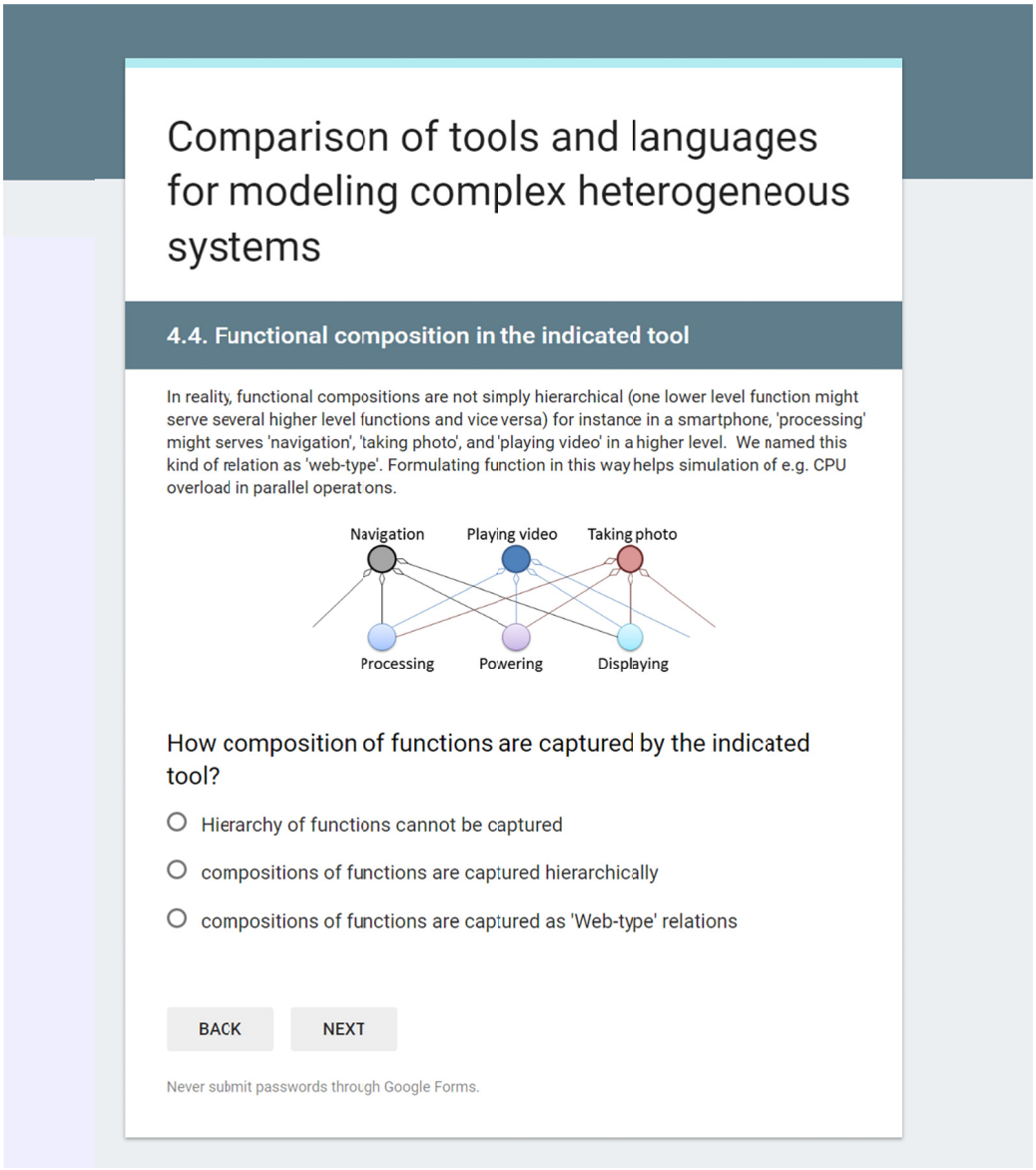
Page 6: Q4.3

If a respondent selects the third option in Q4.1 s/he is redirected to this page

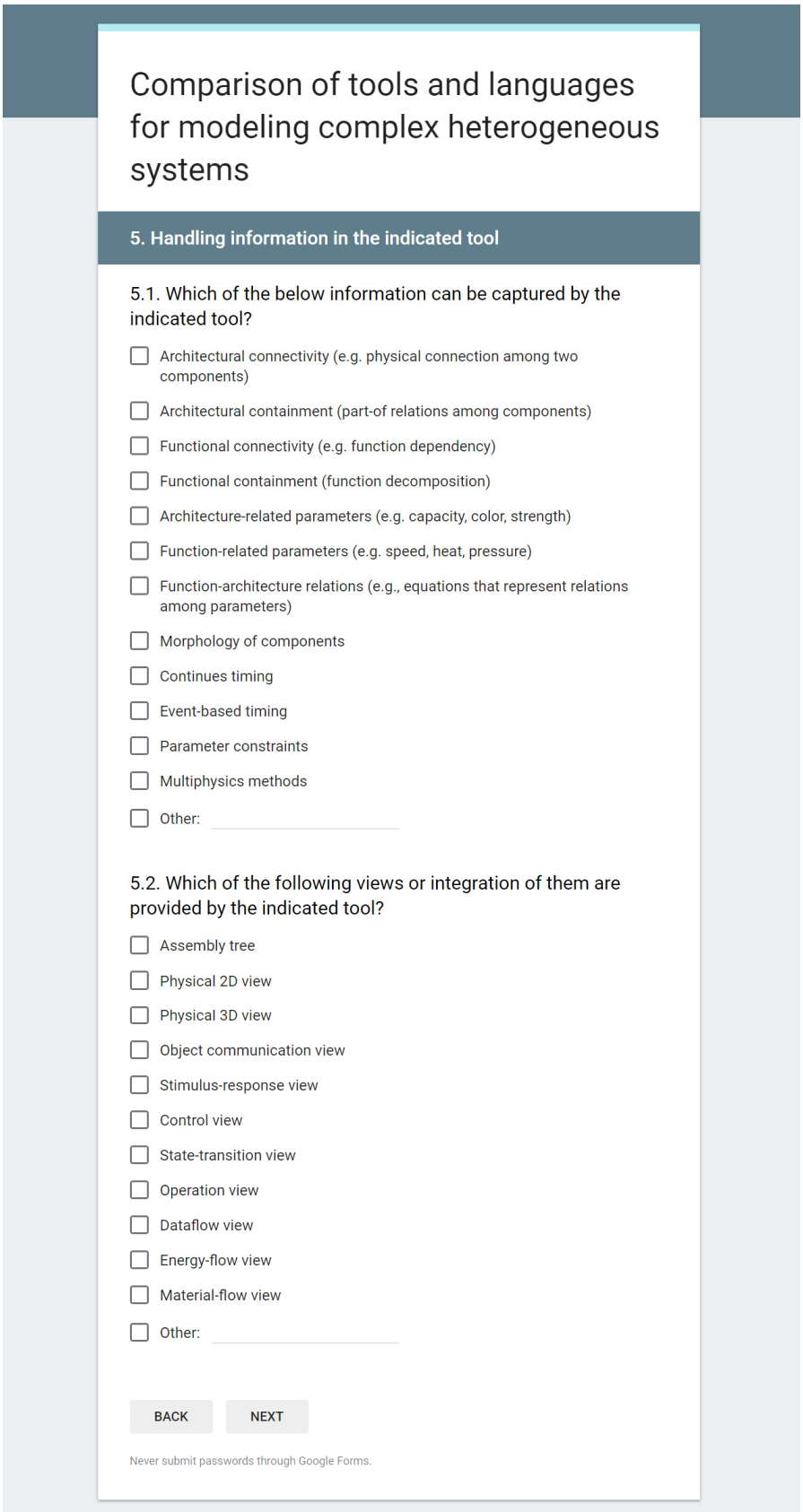


Page 7: Q4.4

Respondents are redirected to this page only if they select options 2 and 3 in Q4.1



Page 8: Q5.1 and Q5.2



Comparison of tools and languages for modeling complex heterogeneous systems

5. Handling information in the indicated tool

5.1. Which of the below information can be captured by the indicated tool?

- [ ] Architectural connectivity (e.g. physical connection among two components)
- [ ] Architectural containment (part-of relations among components)
- [ ] Functional connectivity (e.g. function dependency)
- [ ] Functional containment (function decomposition)
- [ ] Architecture-related parameters (e.g. capacity, color, strength)
- [ ] Function-related parameters (e.g. speed, heat, pressure)
- [ ] Function-architecture relations (e.g., equations that represent relations among parameters)
- [ ] Morphology of components
- [ ] Continues timing
- [ ] Event-based timing
- [ ] Parameter constraints
- [ ] Multiphysics methods
- [ ] Other: _____

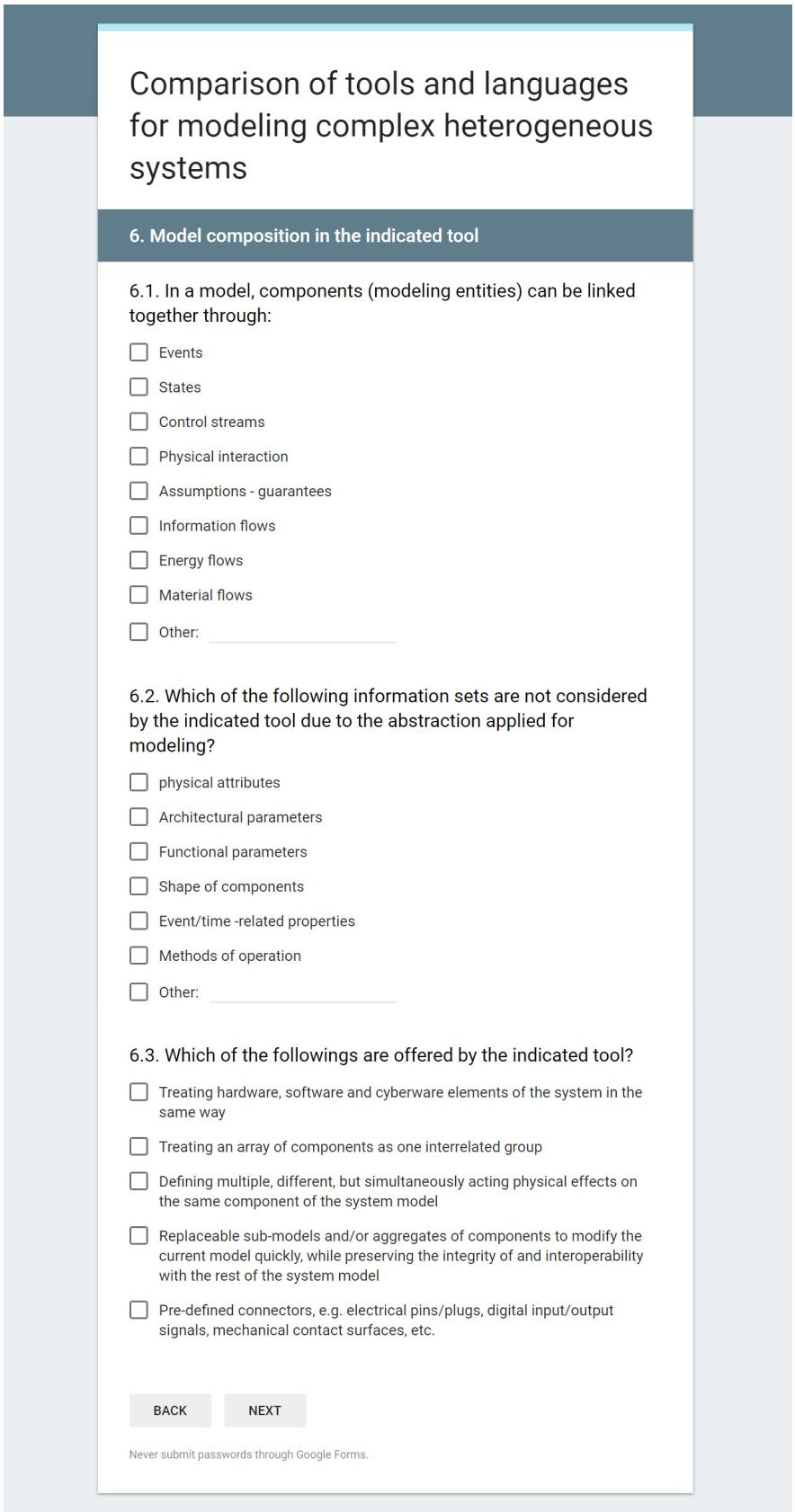
5.2. Which of the following views or integration of them are provided by the indicated tool?

- [ ] Assembly tree
- [ ] Physical 2D view
- [ ] Physical 3D view
- [ ] Object communication view
- [ ] Stimulus-response view
- [ ] Control view
- [ ] State-transition view
- [ ] Operation view
- [ ] Dataflow view
- [ ] Energy-flow view
- [ ] Material-flow view
- [ ] Other: _____

BACK NEXT

Never submit passwords through Google Forms.

Page 9: Q6.1, Q6.2, and Q6.3



Comparison of tools and languages for modeling complex heterogeneous systems

6. Model composition in the indicated tool

6.1. In a model, components (modeling entities) can be linked together through:

- [ ] Events
- [ ] States
- [ ] Control streams
- [ ] Physical interaction
- [ ] Assumptions - guarantees
- [ ] Information flows
- [ ] Energy flows
- [ ] Material flows
- [ ] Other: _____

6.2. Which of the following information sets are not considered by the indicated tool due to the abstraction applied for modeling?

- [ ] physical attributes
- [ ] Architectural parameters
- [ ] Functional parameters
- [ ] Shape of components
- [ ] Event/time -related properties
- [ ] Methods of operation
- [ ] Other: _____

6.3. Which of the followings are offered by the indicated tool?

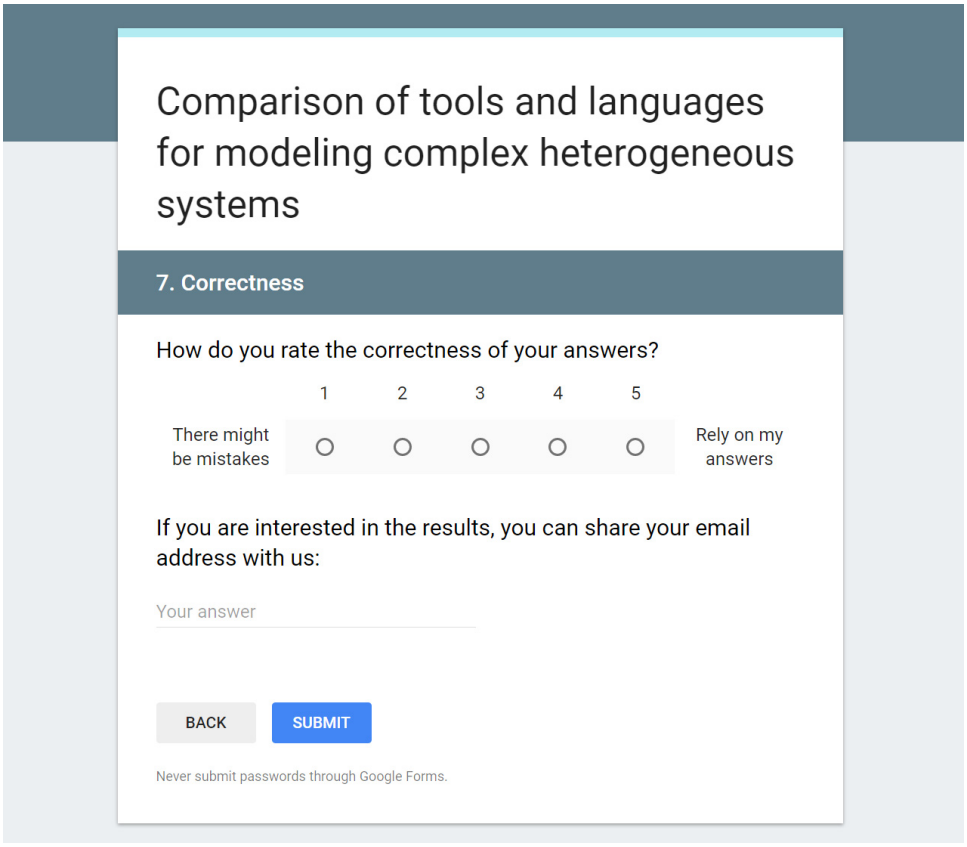
- [ ] Treating hardware, software and cyberware elements of the system in the same way
- [ ] Treating an array of components as one interrelated group
- [ ] Defining multiple, different, but simultaneously acting physical effects on the same component of the system model
- [ ] Replaceable sub-models and/or aggregates of components to modify the current model quickly, while preserving the integrity of and interoperability with the rest of the system model
- [ ] Pre-defined connectors, e.g. electrical pins/plugs, digital input/output signals, mechanical contact surfaces, etc.

BACK NEXT

Never submit passwords through Google Forms.

Page 10:

The last page, for submitting the responses.



APPENDIX 2: Glossary

Below, the most important terms used as *terminus technicus* in this thesis are listed and clarified. They may slightly or largely deviate from other interpretations that can be found in the related literature and may only be valid in the particular context they are used in the thesis and may not be well understood outside of it.

Aggregation: The term refers to a structural relationship that specifies that a compound thing constitutes of simple things and represents a "has-a" relationship.

Architectural domain: In our terminology, domain refers to an architectural area of a component in a system that performs a particular operation. A domain could have some 'entities'. An entity can be considered as a domain for lower lever entities.

Architecture: A general term to describe the logical and physical structure (components, their relations, the properties of both components and relations, and a set of rules and methods) that in turn describe the functionality, organization, and implementation of a non-trivial system. Architectures are unifying and coherent.

Artifact-service combination: When the services provided by an artifact are more important than the artifact itself, the outcome is not referred to as (traditional) product, but as artifact-service combination (e.g. physical hard drive vs. cloud storage).

Chunking: The term refers to and describes how engineering, like nature, constructs complex systems from the bottom up with building blocks (systems) that have proven themselves able to work on their own.

Closed system: The term refers to an isolated system that has no interaction with its external environment. The outputs of a closed system are knowable only thorough the outputs which are not dependent on the system being a closed or open system. Closed systems without any output are knowable only from within.

Complex system: System characterized by manifestation of complexity as opposed to simple, linear, and equilibrium-based systems. The characteristics of this kind of system are not reducible to one level of description and cannot be described by a single rule. Complex systems typically exhibit properties that emerge from the interaction of their parts and which cannot be predicted from the properties of the parts.

Complexity: A systemic characteristic that expresses having a large number of densely connected parts and multiple levels of embeddedness and entanglement. It is also description of the complex phenomena demonstrated in systems characterized by nonlinear interactive components, emergent phenomena, continuous and discontinuous change, and unpredictable outcomes.

Complicatedness: A systemic characteristic that denotes a situation or event that is not easy to understand due to the high cardinality of the actors, influences and interactions, regardless of its degree of complexity.

Component: A definitive, uniquely identifiable part, subassembly, assembly, subsystem or system that (i) is required to complete or finish an activity, item, or job, (ii) performs a distinctive and necessary function in the operation of a system, and (iii) is intended to be included as a part

of a finished, packaged, and labeled item. A component is typically aggregated-ware, which is a combination of hardware, software, and cyberware constituents. Components are usually removable in one piece and are considered indivisible for a particular purpose or use.

Composability: A characteristic of a system that its overall properties and operations can be synthesized as the sum of the properties and operations of its particular components that could be composed in infinite conditions. In fact, this is a bottom-up approach of creating a system by aggregating (usually) off-the-shelf components.

Compositionality: A characteristic of a system that its overall properties and operations cannot be synthesized as the sum of the properties and operations of its particular components.

Computational construct: Formal structures for the processing of SMFs containing a structured set of interrelated parameters, constraints, values, and semantic annotations.

Constituents: Units of a system which are homogeneous in nature, e.g. software, hardware or cyberware, and not any possible combination of them. If a domain of a system is made of a combination of hardware and software, it is not a constituent.

Constructs: The term refers to abstraction-based structural formations that are used to describe, represent, and examine phenomenon of theoretical or procedural interest. In the context of digital computation, constructs encapsulate variables and their relationships to implement particular computational concepts and may capture input, outcome, structural, transformational, logical, knowledge, storage, interaction, interoperation, etc. concepts. These information constructs can be static and dynamic, and determine the structure of a computational or informational model.

Decentralized problem solving: There is no one central components in charge of making decision for all components of the CPS. Instead, there are many components (fully or partially) in charge of making decision for their local problem, as well as participating in providing the overall operation of a CPS.

Domain: Logical and/or spatial extension and a form of physical manifestation of a component, elements of a component, or aggregation of components, which performs a particular operation in a system. A domain may be monolithic or composed of some 'entities'.

Environment: The whole of things, situations and context within which a system exists. It is composed of all things that are external to the system, and it includes everything that may affect the system in a given operation mode/state or at a moment/period in time, and may be affected by it at any given time.

Function: This term refers to the changes produced by any component (or subsystem and system) that performs the function, without considering the architectural domain that performs it. It is captured in the logical-linguistic space, rather than in the physical (spatiotemporal) space, and contrasts the notion of 'operation' that can only be considered together with the performer architectural domain.

Genotype: The term refers to the result of the first stage of SMF creation. Genotypes are differentiated by their different structural organization, which implies different and unique architectural and operational elements, parameters and values.

Heterarchy: An ordering of the components of a system in which there is no single top element, and in which the element that is dominant at a given time depends on the total situation.

Hierarchy: A vertical arrangement of entities (components, subsystems and systems), usually ordered from the top downwards rather than from the bottom upwards. The term is often used in contrast to heterarchy.

Holarchy: The term describes system behavior that is partly a function of individual nature and partly a function of the nature of the embedding system, generally operating in a bottom upwards fashion.

Information content: The entailment of signals, data and data structures that are (i) accurate and timely, (ii) specific and organized for a purpose, (iii) presented within a context that gives it meaning and relevance, and (iv) can lead to an increase in understanding and decrease in uncertainty.

Information schema constructs: computational concepts and entities to underpin the programming of a specific system manifestation feature orientated information management and composing system models.

Information theory: Basic data representation, processing, and communication theory that applies to the technical processes of capturing, formatting, encoding, transforming, communicating, and storing data and data/information structures.

Information: The term refers to the meaning of data and signals. In a technical sense, information referred to the bits of a message, as opposed to "noise" in a communication channel. Information also means the bits of data that are processed by the computer as information processor.

Instance: The term refers to the result of the final stage of SMF creation, when SMF phenotypes are coupled with other SMF phenotypes in the modeling space, and the interface, structural and operational parameters of the concerned phenotypes are jointly evaluated.

Interface of SMF: It forms a part of the SMF modeling building blocks and specifies several operational and architectural parameters and constraints to facilitate interoperation. (To couple SMFs, all constraints and protocols specified in their interfaces should be satisfied). The main interface specifications on the operation side are input and output 'streams', and the main interface specifications on the architecture side are 'contracts'.

Knowledge intensiveness: Ability of a system to gather information from several sources (e.g. knowledge bases, sensors, reasoning, learning, and peers) and to reason and make decisions based on it.

Knowledge structure: The term refers to various basic schemes under which knowledge may be organized: (i) Declarative knowledge structure (describes how and why the things work the way they do; (ii) Procedural knowledge structure (details steps or activities required to perform a task or job); and (iii) Conceptual knowledge structure (explains how and why a particular problem may get solved).

Level of aggregation: The term refers to the fact that, in a physical view, components can be aggregated both architecturally and operationally on various system levels to create higher level components. The same is true for operations.

Linear system: The term is applied to characterize any system in which the cause and effect relation shows a linearly proportional character, i.e. the change of values of the operational variables can be represented as a series of points suggesting a straight line on a coordinate plane. In a linear system, the components are isolated and are non-interactive.

Manifestation feature: The set and composition of observable characteristics of a specific system that could be manipulated by designers, unless it violates overall paradigmatic system features.

Mereology: The term refers to the formal theory of parts and associated concepts developed by Leśniewski. It literally means "science or theory of parts". Mereology captures how physical and conceptual parts relate and what it means for a part to be related to the whole and other parts.

Method: An established, habitual, logical, or prescribed practice or systematic route of achieving certain ends with accuracy and efficiency, usually in an ordered sequence of fixed steps. A method includes problem and task related techniques.

Methodology: The term refers to a system of broad principles or rules that support the achievement of complex objectives within the scope of a particular discipline. A methodology is underpinned by one or more explanatory theories, and provides definitive or alternative procedures, a set of methods, a pool of instruments, and quality criteria. Unlike an algorithm, a methodology is a set means and practices, rather than a comprehensive formula.

Modus-operandi: The term refers to the engineering theory that claims that a particular functionality can be realized by multiple alternative manifestations, which are based on different first principles or combinations of them. According to this theory, an operation can be delivered by an architectural domain that may work based on different physical and/or computational phenomena, and may have different morphologies and materializations.

Non-planned functional interaction: Ability of a system to interact with its environment, other components and technical systems via non-predefined or even unpredicted connections.

Open system: Open systems are those that exhibit the characteristics of openness and maintain their state and operation while their architecture and operation are (dynamically) changing. Openness is a form of organization of a system as well as a state and characteristics of that state in which a system continuously interacts with its environment and with other systems.

Operation: The term refers to a particular action (or a chain of actions) performed by a given architectural domain. The performer domain may be actual or imaginary. (Operations are defined in relation to particular components, while functions are defined as abstract way of operating).

Paradigmatic feature: This kind of features makes it possible to identify a category of systems and a set of characteristics of a category of systems that differentiates it from other comparable system categories. Paradigmatic features imply a set and arrangement of manifestation features.

Phenotype: The term refers to the result of the second stage of SMF creation. Phenotypes are derived from genotypes by specification of parameters and the range of values for them. In a phenotype, all specifications and constraints of its components and operations are defined.

Pre-embodiment design phase: The term indicates a phase, which is also called conceptual design in product development context, and in which a general concept of a product and service combination is delivered as a result. In software engineering, this phase called software configuration. In the context of this thesis, pre-embodiment design is the phase in which a heterogeneous system as a whole is being designed, before delegating the task of component design to the specific experts.

Procedure: A fixed, non-repetitive but repeatable step-by-step sequence of activities or course of action (with definite start and end points) that must be followed in the same order to correctly perform an operation. In the context of the thesis, the term identifies a composition of elementary operations of a system, which is specified as a self-contained unit of operation. The specification of procedures includes information about the dependencies of operations as well as the conditions and time stamps.

Reductionism: One kind of scientific orientation that seeks to understand phenomena by breaking them down into their smallest possible parts (a process known as analytic reductionism), or by conflating them conversely to a one-dimensional totality (a process known as holistic reductionism).

Smart products: The term refers to a wide group of products (e.g. smart phones) having a level of smartness due to their computational processing units and programmed algorithms or, recently, non-preprogrammed reasoning mechanisms. They could reason based on the collected information and decide to perform predicted functions accordingly.

Structure: The term refers to a construction or framework of identifiable elements and has defined boundaries within which (i) each element is physically or functionally connected to the other elements, and (ii) the elements themselves and their interrelationships are taken to be either fixed (permanent) or changing only occasionally or slowly.

System properties: All systems: (i) have inputs, outputs and feedback mechanisms, (ii) maintain an internal steady-state (called homeostasis) despite a changing external environment, (iii) display properties that are different than the whole (called emergent properties) but are not possessed by any of the individual elements, and (iv) have boundaries that are usually defined by the system observer.

System: The term indicates a purposeful organized structure that consists of interrelated and interdependent elements (components, entities, factors, members, parts etc.) and set of operations (which continually influence one another either directly or indirectly).

Topology: In mathematics, the term concerns the theory of continuous spaces that can be generated in a combinatorial manner. In its most abstract form, topology implies ontological laws pertaining to the boundaries and interiors of wholes and to relations of contact and connectedness. In its concrete form, topology handles structural relationships composed by linking discrete entities of continuous spaces.

Warehouse: In the context of the SMFs-based modeling tool, a warehouse is a composed of three parts (i) a database which includes several relational tables for storing the chunks of information, (ii) database management system, which is in charge of storing, removing, modifying pieces of information in the relational tables, and (iii) a meta-level knowledge-base.

This dissertation has been approved by the
promotor: Prof. dr. I. Horváth

Composition of the doctoral committee:

Rector Magnificus	chairperson
Prof. dr. I. Horváth	Technische Universiteit Delft, promotor

Independent members:

Prof. dr. J-P Pernot	Arts et Métiers ParisTech
Prof. dr. I.S. Sariyildiz	Technische Universiteit Delft
Prof. dr. F.J.A.M. van Houten	University of Twente
Prof. dr. G.W. Kortuem	Technische Universiteit Delft
Dr. S. Borgo	Consiglio Nazionale delle Ricerche
Dr. F. Derakhshan	University of Tabriz
Prof. dr. A.R. Balkenende	Technische Universiteit Delft