INTERNATIONAL DESIGN CONFERENCE - DESIGN 2010
Dubrovnik - Croatia, May 17 - 20, 2010.

DESIGN 2010

COMPUTER-AIDED CROSS-DOMAIN MODELING OF MECHATRONIC SYSTEMS

J. Gausemeier, R. Dorociak, S. Pook, A. Nyßen and A. Terfloth

Keywords: design methodology, mechatronics, principle solution, requirements tracing, computer aided conceptual design

1. Introduction

The products of mechanical engineering and related industrial sectors, such as the automobile industry, are increasingly based on the close interaction of mechanics, electronics and software engineering, which is aptly expressed by the term mechatronics. The development of such products is an interdisciplinary task due to the involvement of those different domains. As a consequence there is a need for an integrative and cross-domain cooperation between the developers during the development process. This concerns especially the early design phases of "planning and clarifying the task" and "conceptual design" which result in the so-called principle solution. Within the Collaborative Research Centre (CRC) 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" a specification technique for the cross-domain description of the principle solution of a self-optimizing mechatronic system has been developed [Gausemeier, et al., 2009a]. The principle solution describes the basic operation mode of the system and its desired behavior. The cross-domain description of the principle solution is divided into several aspects, which are mapped on computer by partial models. Combined they form a coherent system as all aspects correlate with each other. Software support is a necessary prerequisite for securing of the overall consistency of the principle solution. This was one of the reasons for the development of the so-called Mechatronic Modeller; a software tool which supports the usage of the specification technique mentioned above.
This paper begins with a brief overview over the specification technique for the description of the principle solution of a mechatronic system. Mechatronic Modeller will then be introduced. The focus lies on handling complex dependencies between different partial models and thus on ensuring the overall consistency of the principle solution. Finally we point out a case study from the development of a miniature robot. We show how to manage complex dependencies within the principle solution by using the example of requirements tracing.

2. Cross-domain specification of the principle solution

In the following, we present the specification technique for the description of the cross-domain principle solution of a self-optimizing system, which has been developed within the CRC 614. It includes mechatronic systems. Already at the beginning of the work, it became clear that a comprehensive description of the principle solution of a highly complex system needs to be divided into aspects. Those aspects are, according to Figure 1: environment, application scenarios, requirements, functions, active structure, shape, behavior and system of objectives. The mentioned aspects are mapped on computer by partial models. The principle solution consists of a coherent system of partial models because the aspects are in relationship with each other and ought to form a coherent system. The aspects correspond to the constituent steps of the procedure module for the

design of advanced mechatronic systems developed in the CRC 614 [Gausemeier, et al., 2009a]. In contrast to other system modeling approaches such as UML [Pilone/Pitman, 2005] or SysML [Friedenthal, et al., 2008] the specification technique is strongly interconnected with the underlying procedure model and focuses strongly on mechatronic systems. In the following, a short introduction of the partial models is given.

Environment: This model describes the environment of the system that has to be developed and its embedding into the environment. Relevant spheres of influence (such as weather, mechanical load, superior systems) and influences (such as thermal radiation, wind energy, information) will be identified. Disturbing influences on system's purpose will be marked as disturbance variables. Furthermore, the interplays between the influences will be examined. We consider a situation to be one consistent amount of collectively occurring influences, in which the system has to work properly. We mark influences that cause a state transition of the system as events.

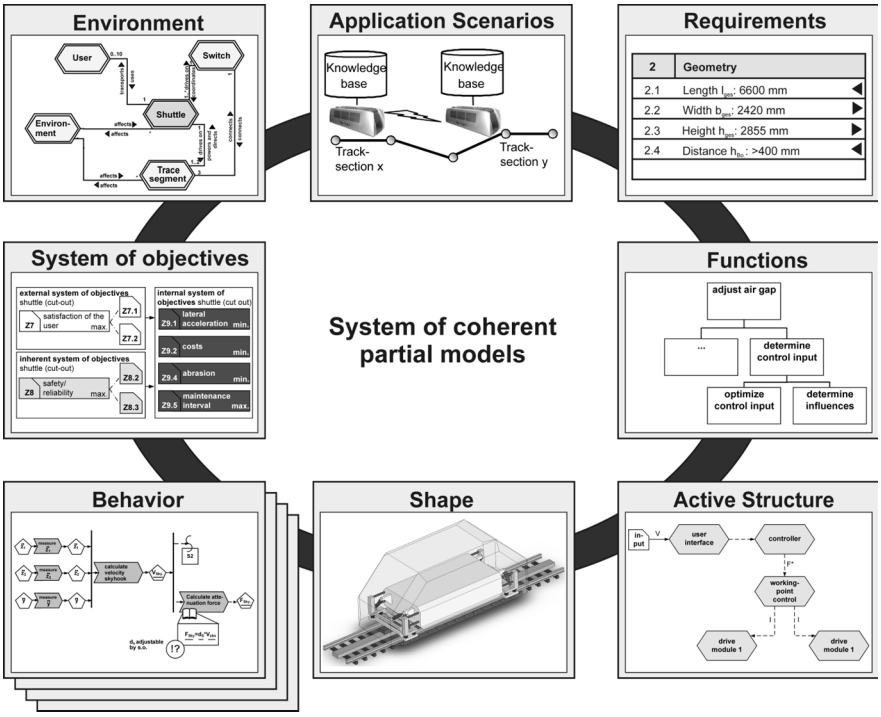

Figure 1. Aspects respectively partial models for the domain-spanning description of the principle solution of a self-optimizing mechatronic system

Application scenarios: Application scenarios form first concretizations of the system. They concretize the behavior of the system in a special state and a special situation and furthermore, what kinds of events initiate a certain state transition. Application scenarios characterize a problem, which needs to be solved in special cases, and also roughly describe the possible solution.

Requirements: This aspect considers the computer-internal representation of the requirements. The list of requirements sets up its basis. It presents an organized collection of requirements that need to be fulfilled during the product development (such as overall size, performance data). We distinguish between functional and non-functional requirements. Functional requirements describe the desired functionality of the system. Non-functional requirements describe properties of the system itself as well as of its behavior. Every requirement is textually described and, if possible, concretized by attributes and their characteristics. There are checklists that assist the setting up of requirements, see for example [Pahl, et al., 2007], [Roth, 2000].

System of objectives: Objectives in terms of self-optimization describe the behavior of the system to be achieved. The objectives are realized by the system itself during its operation. The objectives of a self-optimizing system build together a system of objectives. Further information regarding the system of objective and self-optimization can be found in [Gausemeier, et al., 2009a].

Functions: This aspect concerns the hierarchical subdivision of the functionality. A function is the general and required coherence between input and output parameters, aiming at fulfilling a task. For the setting up of these hierarchies, a catalogue of functions based on [Langlotz, 2000] was developed.
One leading point of any design methodology is the reuse of proven solutions in form of patterns. A pattern describes a recurring problem and the core of the solution to that problem. A solution pattern defines the characteristics of the elements of the system that is to be developed as well as interactions between those elements. Functions are realized by solution patterns and their concretizations. A subdivision into subfunctions is taking place until useful solution patterns have been found for the functions.
Active structure: The active structure describes the system elements, their attributes as well as the relation of the system elements. The active structure consists of system elements, such as drive and break modules, air gap adjustment, operating point control, tracking modules, spring- and tilt modules, energy management and their relations. Furthermore, incoming parameters are described, such as comfort, costs and time, which are external objectives of the user.
Shape: This aspect needs to be modeled because first definitions of the shape of the system have to be carried out already in the phase of the conceptual design. This especially concerns working surfaces, working places, surfaces and frames. The computer-aided modeling takes place by using 3D CAD systems.
Behavior: Two partial models are used to specify the behavior of the system. These are the partial models behavior – states and behavior – activities. The partial model behavior – states defines the states of the system and the state transitions. The state transitions describe the reactive behavior of the system towards incoming events. The partial model behavior – activities describes the logical sequence of activities in the system. Especially, activities executed in parallel and their synchronization can be described this way.

3. Computer-aided modeling of the principle solution

In the following, we introduce the Mechatronic Modeller, which is a software tool to describe mechatronic systems using the specification technique described previously. During the development of the specification technique it became clear, that software support is a necessity in order to be able to secure the overall consistency of the principle solution.
Mechatronic Modeller is a dedicated software solution, which is aligned with the specification technique described in section 2. It offers a separate editor for each partial model. Figure 2 illustrates the user interface of this software tool.
Within the model browser the elements of the principle solution are presented as a tree. This tree can be used to navigate within the principle solution. The currently processed partial model is shown on the right in the diagram view together with a tool palette. Within this diagram view the respective partial model can be modified. Using the tool palette new elements, for instance, can be added. In the bottom part of the GUI, there is a property window, in which the properties of the element selected in the model browser or in the diagram view can be modified. The outline view (located next to the property view) continuously shows the outline of the whole diagram. Within this view the user can navigate through the whole diagram. This allows the user to navigate to sections of the partial model which are currently not being shown in the diagram view.
A so-called metamodel has been defined for the specification technique. It defines [Stahl et al. 2006]:

- which model elements are available during the description of the principle solution as well as how they are related to each other (abstract syntax) and
- how a model element has to be linked in order to have a meaning (static semantics).

Mechatronic Modeller is based upon this metamodel. The principle solution is computer-internally represented as a data model, which is the instance of this metamodel. In addition to the metamodel, following aspects of the specification technique had to be formally defined during the development of the tool:

- the meaning of a model element or a relation in context of a particular principle solution (dynamic semantics), and
- how the models will be graphically represented (concrete syntax).

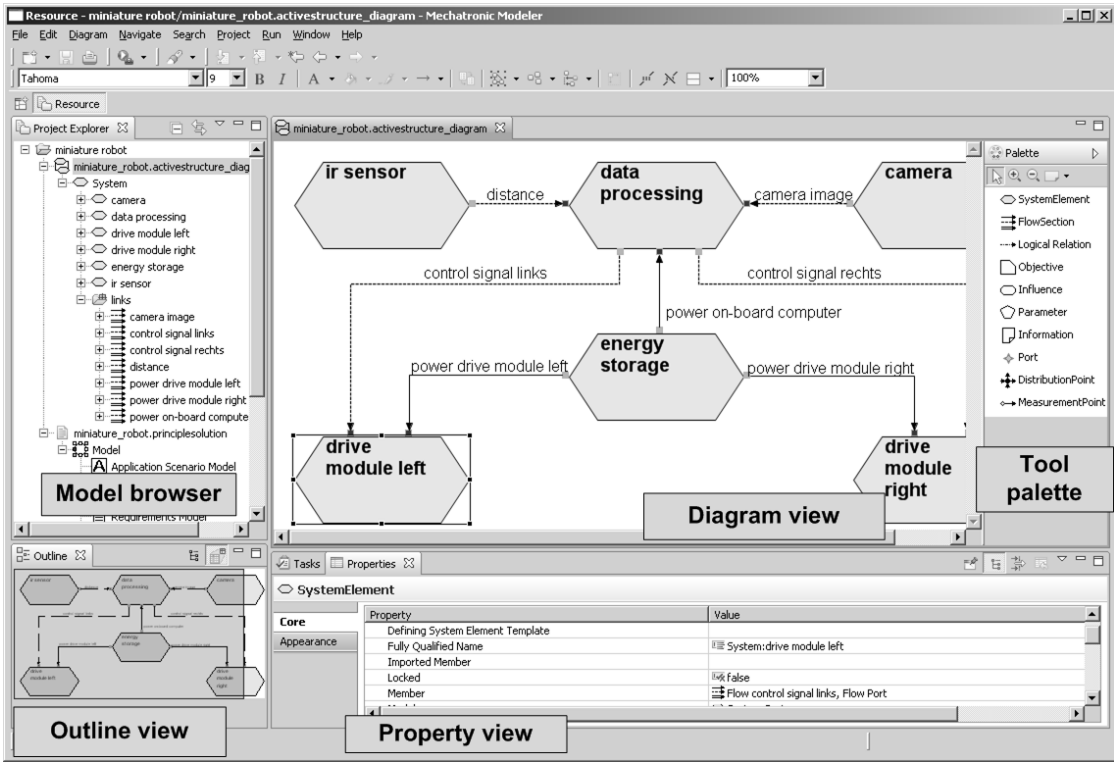

Figure 2. Screenshot from the Mechatronic Modeller showing the active structure editor

Since the Mechatronic-Modeller is based on a metamodel, the overall consistency of the principle solution model can easily be ensured. In order to check it, it is sufficient to examine if the data model representing the principle solution is compliant with the metamodel of the specification technique.
Furthermore, cross-references between elements of different partial models are stored in the data model. Thus, Mechatronic Modeller is capable of handling complex dependencies within the principle solution. In the following chapter, we will illustrate this capability by using the example of requirements tracing.

4. Requirements Tracing within the Principle Solution

The first step of each system development is the requirements gathering. Once requirements are set, developers initiate other technical works such as system design, development, verification and validation, etc. Changing the system design without validating the requirements can lead to huge problems in later development stages [Hull, et al., 2005]. To overcome this drawback requirements tracing is used. It provides means for accessing and maintaining requirements throughout the whole development process. In particular, it can be used to update the respective requirements in case of a change in the system design. This approach has been implemented in Mechatronic Modeller. However, the implementation details of this approach in the tool are not considered here. In the following, we describe which cross-references within the principle solution are focused in this paper. We show how these cross-references are created. Our approach of requirements tracing will then be introduced.

4.1 Cross-references in the data model considered for requirements tracing

There are several interrelations in the principle solution, which are substantial for the trace of requirements. These have been incorporated into the metamodel as cross-references. In this paper we consider three of them. These are the interrelations, which link requirements and functions, functions and system elements as well as requirements and system elements. Figure 3 shows a cut-out of the metamodel, which represents these cross-references.

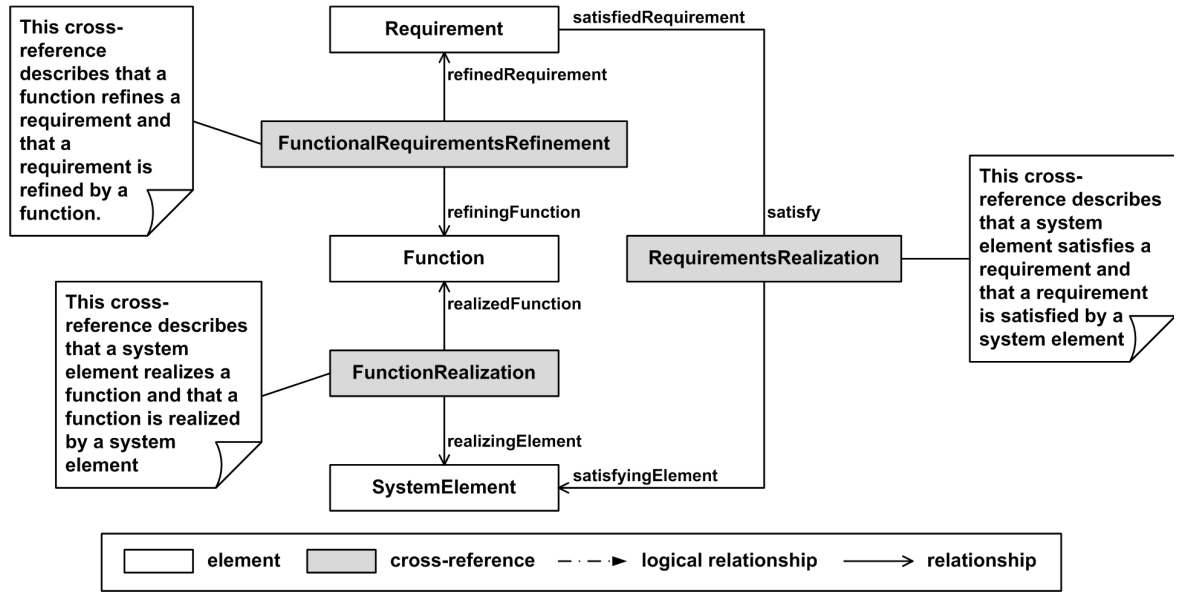

Figure 3. Cut-out of the metamodel of the specification technique, presenting the cross-references used during the tracing of requirements

4.2 Cross-references and hierarchical structures of partial models

The elements in the partial models requirements, functions and active structure are structured hierarchically. During requirements tracing these hierarchies have also to be considered. Therefore they will be explained shortly for requirements, functions and system elements of the active structure:

Requirements: Requirements are classified according to [Pahl, et al., 2007] in predefined classes such as geometry, energy, material, ergonomics, etc. Each requirement can be further refined by one or more requirements.

Functions: Functions are represented as a function tree (so called function hierarchy). Such a tree is created beginning by refining the overall function of the system into subfunctions.

System elements of the active structure: System elements also form a hierarchy. Each system element can be detailed by other subordinated system elements in the next hierarchical level.

In the following, we describe in more detail how such hierarchies are being build and interconnected (Figure 4). In particular, the creation of computer-internal cross-references will be explained.

At the beginning of the "conceptual design" phase the overall function is derived from the development task. This overall function is the starting point for the development of the function hierarchy (1). Functional requirements are then traced back, to fulfillment of which the overall function contributes. A cross-reference is created between them (2). The overall function is then divided into subfunctions, which emerge from functional requirements. In such a way the function hierarchy is being refined (3). In the course of this, cross-references between that subfunction and corresponding functional requirements are created (4). As solution patterns are being searched for, functions are further refined into subfunctions (5). This step-wise refinement continues, until for each of the new subfunctions a solution pattern has been found (6). Those solution patterns will be realized by suitable system elements and interrelations (7) [Gausemeier et al. 2009a]. Solution patterns are not an explicit part of the principle solution and as a result they are not present in the data model, either.

Therefore, the cross-reference is created when the system elements realizing the particular solution pattern have been found. The respective function and these system elements are then connected (8). The non-functional requirements, which had not been considered yet, are to be connected to the system elements, which realize them, provided that it is possible at this point of time (9).

By following this approach, all cross-references in the data model described had been set at the end. This enables requirements tracing within the principle solution.

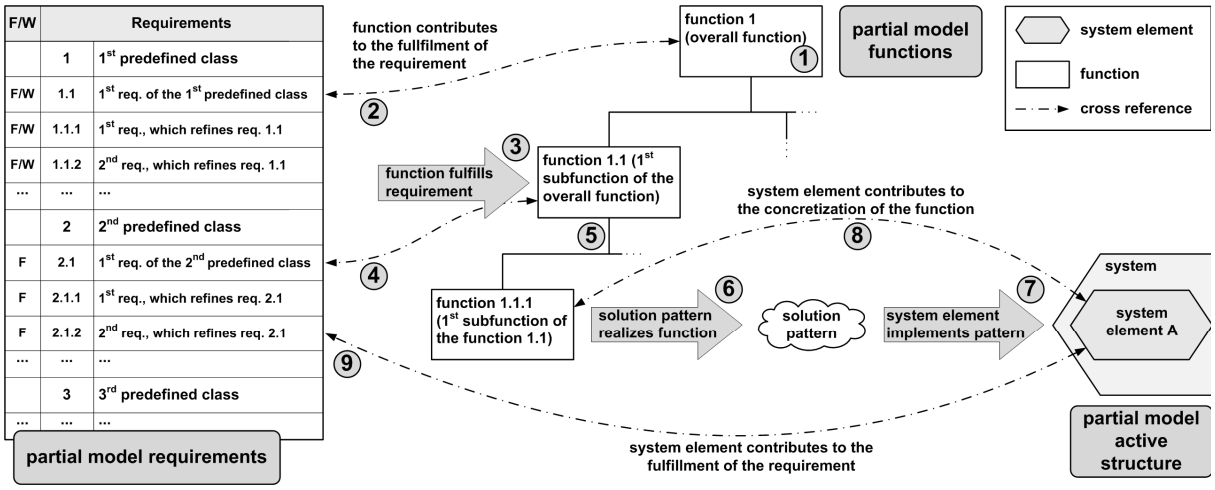

Figure 4. Creation of the cross-references that are needed for requirement tracing throughout the partial models requirements, functions and active structure

4.3 The approach of tracing of requirements within the principle solution

Using requirements tracing a developer can check whether the principle solution complies with all of the requirements. Requirements tracing can be realized in two different directions, which are depicted by Figure 5.

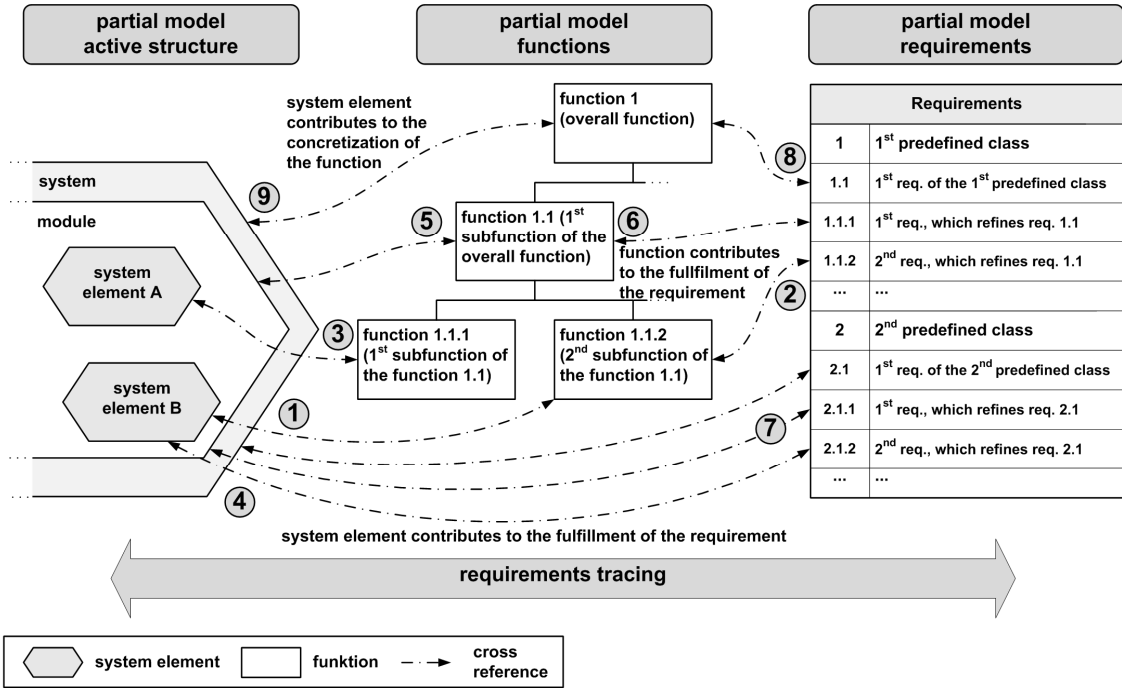

Figure 5. Requirement tracing throughout the partial models requirements, functions and active structure with regard to the hierarchical structure of these models

Tracing of requirements on a particular system element: The requirements concerning one particular system element have to be found. To accomplish this, functions of the system elements have to be traced back based on the cross-references between functions and system elements in the underlying data model (1). After the functions have been identified, the corresponding functional requirements have to be found (2). In the course of this, all cross-references between the subfunctions

of the function and requirements have to be evaluated as well. If there is no corresponding cross-reference between the respective function and requirements (3), then the superordinated function and its cross-references have to be traced back (6).
Non-functional requirements are traced back using cross-references between them and the system elements (4). If requirements on a superordinated system element ("module" in Figure 5) have been traced back, then the cross-references between this superordinated system element and functions are evaluated (5). After the respective function has been found, its corresponding requirements are traced back (6). In the course of this all cross-references connected to the subfunctions of the respective function have to be evaluated as well (2). This has then to be repeated for all subordinated system elements ("system element A" and "system element B" in Figure 5) of the considered system element.
Non-functional requirements are first traced back for the superordinated system element using the respective cross-references (7). This is then repeated for all of the subordinated system elements (4).
Search of system elements: System elements, which realize a particular requirement, have to be found. Is this requirement a functional one, then each cross-reference between this requirement and a function has to be evaluated. By doing so, all functions are found, which contribute to the fulfillment of this requirement (8). The cross-references between the functions found and system elements are then evaluated (9). This procedure is repeated for each subfunction of this function, if there are any (5), until all subfunctions in the function hierarchy have been covered (3 and 1). If a functional requirement has been refined by further functional requirements, then the search has to be repeated for each such subordinated functional requirement (6). The search process ends when the original functional requirements and all of its subordinated functional requirements have been considered (2).
If the original requirement is a non-functional one, it suffices to evaluate each cross-reference between this requirement and a system element. By doing so, system elements will be found, which fulfill this requirement.
Mechatronic Modeller supports the developer by handling the complex dependencies between elements of different partial models in its underlying data model. It provides user guidance for the sequence of steps associated it. The tool assists developers in evaluation of the relevant cross-references and displays the results of requirements tracing accordingly.

5. Requirements Tracing Case Study: the Miniature Robot

We illustrate requirements tracing by using the example of a miniature robot. The miniature robot BeBot[1] is being developed at the Heinz Nixdorf Institute of the University of Paderborn. It serves as a test carrier for swarm intelligence, and for multi-agent applications of the computer science. An overview about the architecture of the miniature robot (including sensors, actuators and information processing) can be found in [Deyter, et al., 2008]. In this case study the original development task has been enhanced. Thus, new requirements on the system arose. Figure 6 shows the process of changing the development tasks as well as the corresponding cut-out of the data model (compare Figure 3).
The original development task: Originally the development task was to develop a miniature robot vehicle, which is able to recognize obstacles and drive around them safely by itself.
The requirements list contains the corresponding requirements. In this example this are the requirement 6.1 "The vehicle drives around obstacles by itself", which is further refined by the requirements 6.1.1 "The vehicle determines the position of the obstacles" and 6.1.1.1 "The determined distance from the obstacle is accurate to 5 mm". These requirements are fulfilled by the function "Adjust the trajectory of the planned movement" and its subfunction "Convert the received signal into the obstacle position". The latter function has been refined in further subfunction in the course of choosing of the solution patterns. These subfunctions are "Convert the received signal into the distance" and "Convert distances into the obstacle position". These functions are realized by two system elements "ir sensor" and "data processing".
The enhanced development task: In the course of time, the development task has been enhanced. The miniature robot vehicle has now to able to move toward balls of a particular color by itself. As a result new requirements arise. These are the requirements 6.2 "The vehicle is able to move toward

[1] http://wwwhni.uni-paderborn.de/en/priority-projects/miniaturroboter/

balls by itself" and 6.2.1 "The vehicle distinguishes between balls of different colors" (1). In order for the system to fulfill those new requirements, the function hierarchy had to be extended. The new function "Compare recognized objects" is incorporated into the existing function hierarchy (2). Developers decided to refine it with some subfunctions. One of them is the subfunction "Digitalize the recorded image" (3). A solution has to be found, which fulfills these new requirements and concretizes the described functions. In the following, we examine this in more detail.

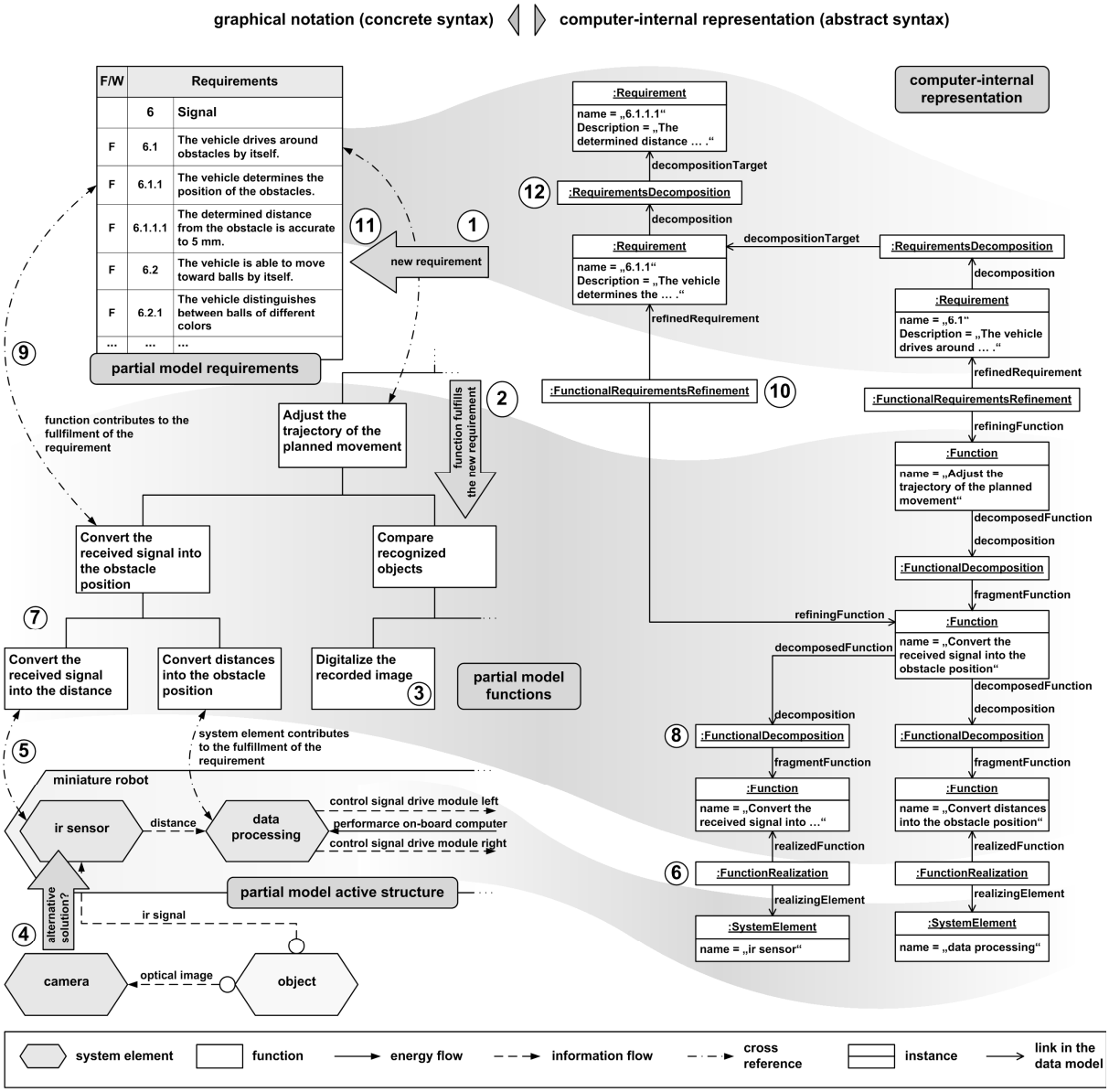

Figure 6. The change of the development task and its computer-internal representation (cut-off)

In order to find a suitable solution the existing active structure will be examined. All system elements, which gather information about the environment, are therefore inspected. In this example there is one such system element, the "ir sensor". This sensor, however, is not capable of distinguishing different colors. Thus, it does not satisfy the requirement 6.2.1. After examining possible solutions developers take the use of a camera into account. Two possibilities emerged, which have to be analyzed. These are the integration of an additional camera into the existing system, and the exchange of the ir sensor with the camera (4).

The latter possibility is preferred due to cost reasons and the need to keep the energy consumption of the robot at a low level.
It has to be assured, that the exchange of the ir sensor with the camera does not invalidate other system requirements. Camera fulfills the new requirements. Therefore it suffices to check whether it also fulfills the requirements, which were originally satisfied by the system element "ir sensor". These requirements have to be traced back. We use the requirements tracing approach described in chapter 4. In order to do that, functions have to be traced back, which are realized by the system element "ir sensor" (5). This interrelation is computer-internally described by the cross-reference "FunctionRealization" (6). These functions were originally created in the course of the search of solution patterns. As a result they are in no direct relation to the requirements. Thus, the search is continued starting with the next superordinated function (7). This function can be found by following the computer-internal cross-reference "FunctionalDecomposition" in the function hierarchy (8). In such manner the superordinated function "Convert the received signal into the obstacle position" has been found.
Requirements satisfied by this function were then traced back (9). The cross-references "Functional-RequirementsRefinement" have been evaluated in course of this (10). In this example the requirement 6.1.1 "The vehicle determines the position of the obstacles" was found. Also the subordinated requirements had to be considered. In this case, requirement 6.1.1.1 "The determined distance from the obstacle is accurate to 5 mm" has been found (11) by following the cross-reference "RequirementsDecomposition" (12).
The camera does not satisfy the requirements found. In addition, none of the existing system components fulfills them to the full extent. Therefore, although another solution was originally preferred, an additional camera has to be integrated into the existing system in this case.

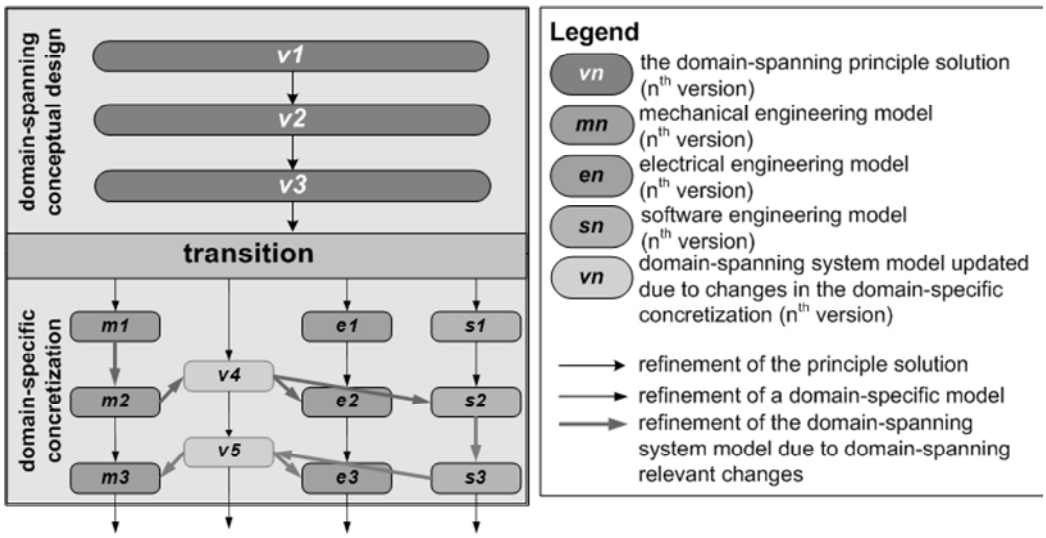

Figure 7. Synchronization of domain-spanning and domain-specific models within the development of mechatronic systems (adapted from [Gausemeier et al. 2009b])

6. Concluding remarks and outlook

Mechatronic Modeller is a dedicated software tool, which supports the specification of the principle solution of mechatronic systems. Due to being based on a formal model, Mechatronic Modeller assists with handling complex dependencies between partial models. It also provides new means of analyzing principle solutions. Thus, computer-aided analyses of production costs, robustness and reliability in the early development stages become possible. Furthermore, software supported impact-analyses of system design changes are also supported.
The domain-spanning principle solution is the result of the conceptual design. After this phase the domain-specific concretization takes place. All involved domains then work in parallel. Principle

solution is the common base for these domain-specific activities. The principle solution is further refined and extended and becomes the domain-spanning system model. Whenever a domain-spanning relevant change takes place, this system model has to be updated (see Figure 7).
It remains a challenge for future works to support the assurance of the consistency between the domain-spanning system model and the domain-specific models within the domain-specific concretization, when the involved domains work in parallel and domain-spanning relevant changes have to be propagated back in to the domain-spanning system model. A first approach of coupling domain-spanning with domain-specific models can be found in [Gausemeier et al. 2009b]. In the future this approach has to be extended to support the management of cross-domain model consistency across the complete development process.

Acknowledgement

This contribution was developed in the course of the Collaborative Research Centre 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" funded by the German Research Foundation (DFG).

References

Deyter, S., Gausemeier, J., Middendorf, A., Reichl, H., Steffen, D., Tsunezawa, K., Walachowicz, F., "Specifying Mechatronic Systems in early Design Phases for Analysing Sustainability Aspects", Proceedings of Electronic goes Green 2008+, Berlin, Germany, 2008.

Friedenthal, S., Steiner, R., Moore, A. C., "Practical Guide to SysML: The Systems Modeling Language", Elsevier Science, 2008.

Gausemeier, J., Frank, U., Donoth, J., Kahl.S., "Specification technique for the description of self-optimizing mechatronic systems", Research in Engineering Design, Vol. 20, No. 4, 2009, pp 201-223.

Gausemeier, J., Schäfer, W., Greenyer, J., Kahl, S., Pook, S., Rieke, J., "Management of Cross-Domain Model Consistency During the Development of Advanced Mechatronic Systems", Proceedings of the 17th International Conference on Engineering Design (ICED'09), Bergendahl, M.N., Grimheden, M., Leifer, L. (Ed.), Design Society, Stanford, 2009, pp. 1-12.

Hull, E., Jackson, K., Dick, J., "Requirements Engineering",2nd edition, Springer Verlag, London, 2005.

Langlotz, G., "Ein Beitrag zur Funktionsstrukturentwicklung innovativer Produkte", Dissertation, Institut für Rechenranwendung in Planung und Konstruktion, Universität Karlsruhe, Shaker-Verlag, Band 2/2000, Aachen.

Pahl, G., Beitz, W., Feldhusen, J., Grote, K.-H., "Engineering Design – A Systematic Approach", 3rd English edition, Springer Verlag, London, 2007.

Pilone, D., Pitman, N., "UML 2.0 in a Nutshell: A Desktop Quick Reference", O'Reilly, 2005.

Roth, K.-H., „Konstruieren mit Konstruktionskatalogen, Band 1, Konstruktionslehre", Springer Verlag Berlin, 2000.

Stahl, T., Voelter, M., "Model-Driven Software Development: Technology, Engineering, Management", John Wiley & Sons Ltd., 2006.

Prof. Dr.-Ing. Juergen Gausemeier
Heinz Nixdorf Institute, University of Paderborn,
Fürstenallee 11, 33102 Paderborn, Germany
Telephone: +49-5251-606267
Telefax: +49-5251-606268
Email: juergen.gausemeier@hni.upb.de
URL: http://wwwhni.upb.de/en/pe