INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 - 31 AUGUST 2007, CITÉ DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

TOWARDS THE DESIGN OF SELF-OPTIMIZING MECHATRONIC SYSTEMS: CONSISTENCY BETWEEN DOMAIN-SPANNING AND DOMAIN-SPECIFIC MODELS*

Jürgen Gausemeier [1], Holger Giese [2], Wilhelm Schäfer [2], Björn Axenath [2], Ursula Frank[1], Stefan Henkler [2], Sebastian Pook[1] ,and Matthias Tichy [2]

[1] Computer Integrated Manufacturing, University of Paderborn, Germany
[2] Software Engineering Group, University of Paderborn, Germany

ABSTRACT
Future mechanical engineering systems will consist of configurations of many highly distributed system elements with inherent partial intelligence. The complexity of self optimizing systems will grow enormously. The first major document of an engineering process is a set of models called the principle solution laying down the basic structure of the system. It is the foundation for the subsequent domain-specific refinement phases.

One purpose of this set of specifications is to give the engineers of the involved domains a common understanding about the system. The subsequent domain-specific development refines this domain-spanning model. We propose to use the principle solution also as a central point for keeping consistency between the refining models of the domain-specific development. Consequently, it is necessary to keep the principle solution consistent with the domain-specific refinement. Changes of the initial principle solution which result from iterative steps have to be propagated to the domain-specific models, as well as changes of the domain-specific models, which are relevant for the overall system design, have to be propagated.

As an example, we illustrate the consistency between the principle solution and software specific models. Therefore, we look in more detail at the views environment, functions and active structure of the principle solution and at the structural and behavioural models of the software design. We also explain how this procedure can be automated by applying triple graph grammars to maintain consistency.

Keywords: Consistency Management, Domain Spanning, Domain Specific

1 INTRODUCTION
Nowadays, most mechanical engineering products already rely on the close interaction of mechanics, electronics, control engineering and software engineering which is aptly expressed by the term mechatronics. The aim of mechatronics is to optimize the behavior of a technical system. Sensors collect information about the environment and the system itself. The system utilizes this information to derive optimal reactions. Future mechanical engineering systems will consist of configurations of system elements with inherent partial intelligence. The behavior of the overall system is characterized by the communication and cooperation between these intelligent system elements. From the point of view of information technology we consider these distributed systems to be cooperative software agents. This

* This contribution was developed in the course of the Collaborative Research Centre 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" (Speaker: Prof. Gausemeier) funded by the German Research Foundation (DFG) under grant number SFB 614 (see www.sfb614.de for further details).

opens fascinating possibilities for designing tomorrow's mechanical engineering products. The term self-optimization characterizes this perspective. Self-optimizing systems are able to react autonomously and flexible to changing environmental conditions. They are capable to learn and optimize their behavior in run-time.

The design of such systems is a challenge. The established development methodologies in the areas of classical mechanical engineering and mechatronics are not sufficient here, the VDI guideline 2206 "Design methodology for mechatronic systems" for example [1]. This concerns in particular the early phases "planning and clarifying the task" and "conceptual design". The result of the last phase is the principle solution. A new design methodology for self-optimizing systems is developed within the Collaborative Research Centre (SFB) 614 “Self-Optimizing Concepts and Structures in Mechanical Engineering” of the University of Paderborn. An important new component of this design method is a set of specification techniques for describing the principle solution of self-optimizing systems. With this set of specification techniques, specialists from the domains mechanics, electronics, software technology and control engineering are able to attain a common understanding over the system.

The principle solution is the starting point for the domain-specific concretization of the system and defines the interfaces between the domains. The domains derive the relevant information from the principle solution, and concretize, respectively refine the information. If changes for the whole system occur as a result of the domain-specific design, they will be transferred into the principle solution and their effects on the other domains will be examined. A fundamental goal is to protect the consistency of the models. Today, developers of different domains use personal communication to ensure the consistency of the models of the different domains. In this paper, we use the principle solution and an appropriate formalism to automatically ensure the consistency of the models of the different domains throughout the development process.

We describe first the paradigm of the self-optimization in this contribution. Afterwards we present the set of specification techniques for describing the principle solution of self-optimizing systems and demonstrate how to derive the domain-specific information from the principle solution exemplary for the domain software engineering. Subsequently, we describe a concept of the consistency protection of the models.

We clarify the points mentioned by an example of the project "Neue Bahntechnik Paderborn/RailCab" [http://nbp-www.nbp.de]. Autonomous, self-optimizing shuttles are developed in this project (Figure 1). The shuttles are powered by a linear motor and communicate with each other over wireless networks. The energy transfer is realised by a track stator, which can be added to a usual railway system.

Figure 1. Railcab Shuttles forming a convoy

In order to make the system energy efficient, the shuttles have the possibility to form convoys. The shuttles have to keep the balance of the distance minimization between the shuttles, in order to reduce air resistance and maintain the safe distance, in order to avoid collisions. In order to avoid collisions, the shuttles must coordinate their distance. Since the distance between the shuttles cannot be measured reliably, the coordination between the shuttles can be achieved only cooperatively. The shuttles exchange information with reference to velocity, position and driving direction during the coordination.

2 SELF OPTIMIZED SYSTEMS

The intelligent mechanical engineering systems of tomorrow we are considering here are based on mechatronics. We have therefore introduced a hierarchical structure of a complex mechatronic system (Figure 2, according to [2]). The basis of this is provided by what are called "mechatronic function modules" (MFMs), consisting of a basic mechanical structure, sensors, actuators and a local information processor containing the controller. A combination of MFMs, coupled by information technology and mechanical elements, constitute an autonomous mechatronic system (AMS). Such systems also possess a controller, which deals with higher-level tasks such as monitoring, fault diagnosis and maintenance decisions as well as generating parameters for the subordinated information processing systems of the MFMs. Similarly, a number of AMSs constitute what is called a networked mechatronic system (NMS), simply by coupling the associated AMSs via information processing. In the context of vehicle technology, a spring and tilt module would be a MFM, the shuttle would be an AMS, and a convoy would be a NMS. On each level the controller is enhanced by the functionality of self-optimization. Thus the previously mentioned system elements (that means MFM, AMS and NMS) receive an inherent partial intelligence.

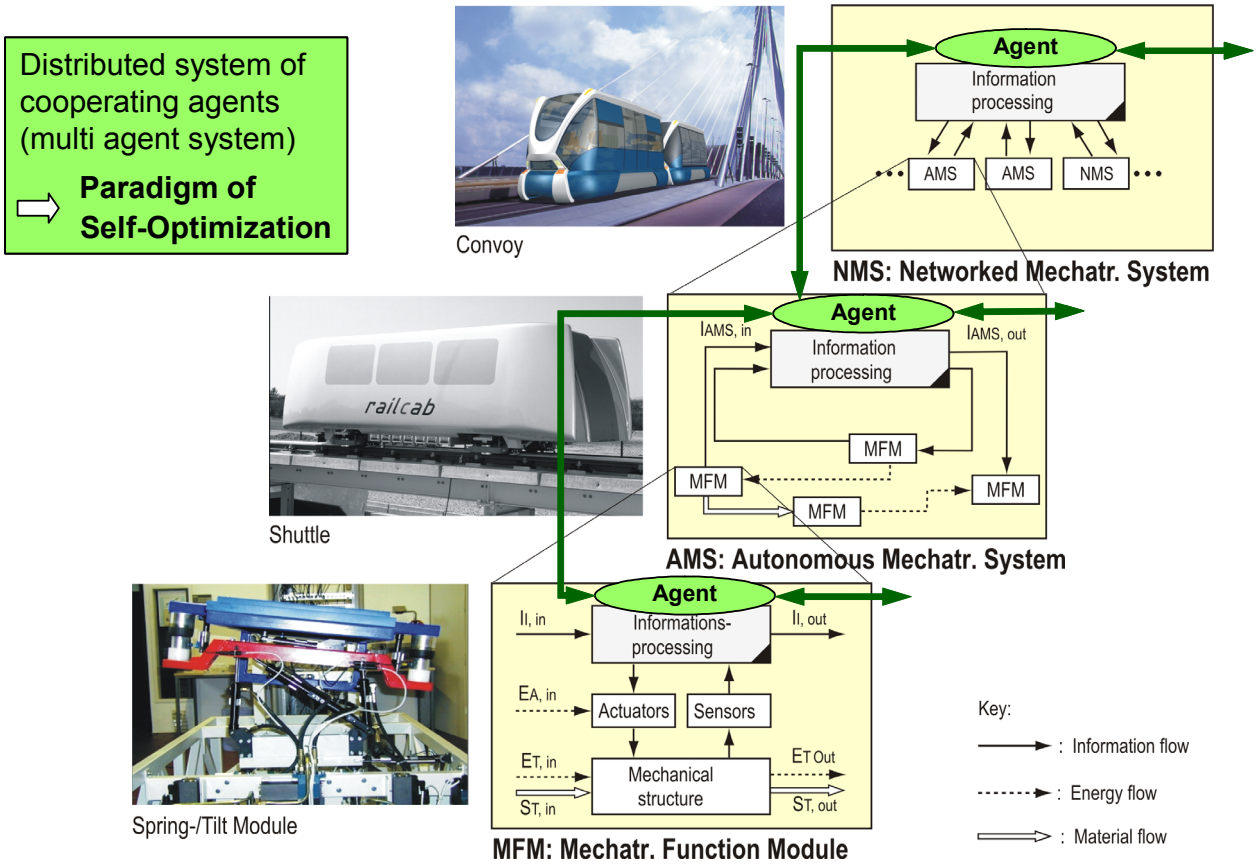

Figure 2. Hierarchical structure of mechatronic systems (according to Lückel [2])

The key aspects of self-optimization systems are reacting on changing influences of the environment, altering the system's objectives and adapting the system's behavior. The objectives pursued by the system play an important role. The self-optimizing system determines its currently active objectives on the basis of the encountered influences. For example, while a rail system is in normal operation mode the objectives include a high level of comfort and minimal power consumption.

The self-optimizing system is able to adapt its objectives autonomously. This means, for instance, that the relative weighting of the objectives is modified, new objectives are added or existing objectives are discarded and no longer pursued. Adapting the objectives in this way leads to adaptation of the system behavior. That is achieved by adapting the parameters and where necessary the structure of the system.

The term parameter adaptation means adapting a system parameter, for instance changing a control parameter. Structure adaptations affect the arrangement of the system elements and their relationships.
We express self-optimization as a series of three actions that are generally carried out repeatedly:

1) **Analyzing the current situation:** Here the current situation includes the state of the system itself and all the observations that have been made about its environment. Such observations may also be made indirectly by communicating with other systems.
2) **Determining the system of objectives.**
3) **Adapting the system behavior** according to the new objectives.

This sequence of actions is called a **self-optimization process.** From a given initial state, the self-optimization process passes, on the basis of specific influences, into a new state, i.e. the system undergoes a state transition.

3 PRINCIPLE SOLUTION

In order to describe the principle solution of a self-optimizing system we use a set of semi-formal specification techniques [3]. For a complete description we need several views on the self-optimizing system. The developed set of specification techniques allows describing these views and how they are interlinked. Each view is mapped by a computer onto a partial model. As shown in Figure 3, the principle solution is made up of the following eight views respectively partial models: requirements, environment, system of objectives, functions, active structure, shape, application scenarios and behavior. This last view is considered a group because there a various types of behavior (e.g. the logic behavior, the dynamic behavior of a multi-body system, the cooperation behavior of system components etc.).
There are also relationships between the partial models, leading to a coherent system of partial models that represents the principle solution of a self-optimizing system. Previously, in mechatronics, the focus was normally on the system's active structure, but here the system's states and state transitions are in the foreground, i.e. the self-optimization process and its effects on the active structure and the processes taking place within the system.

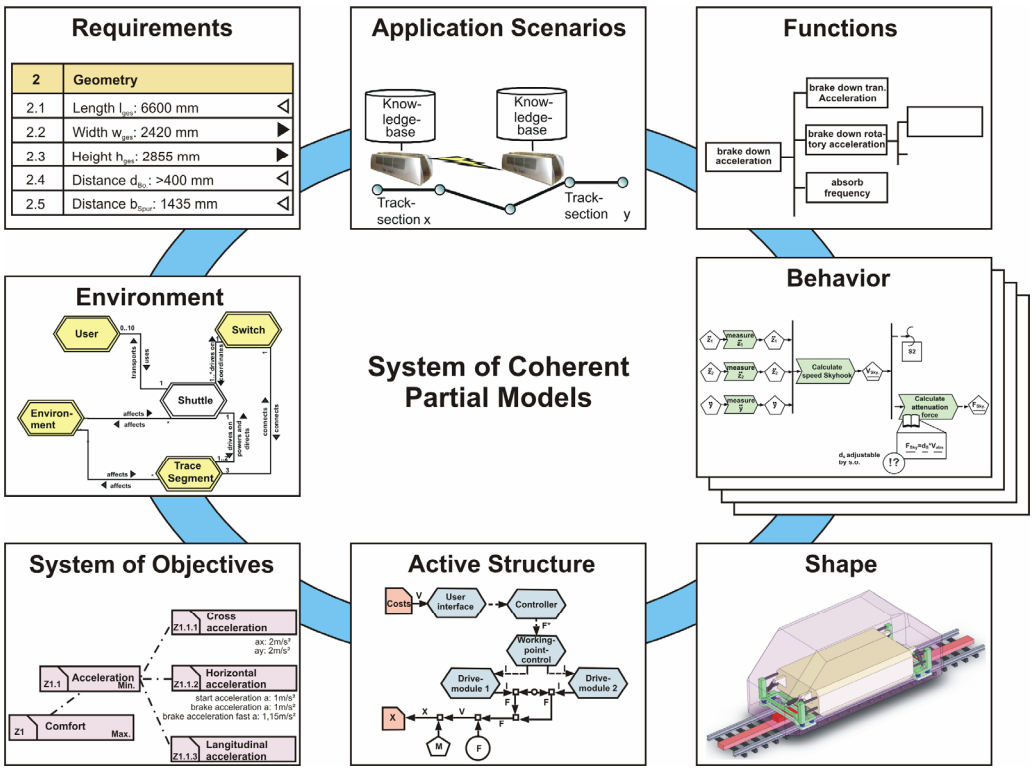

Figure 3. Coherent system of partial models for describing the principle solution of a self-optimizing system

The described partial models are not compiled in a predetermined sequence, but in a close interplay. Normally, you start with the partial models requirements, environment, application scenarios and system of objectives.

Figure 4 shows exemplarily a section of the active structure of a shuttle. The active structure illustrates the system elements, their characteristics as well as their relations to each other. The relations involve material -, information and energy flows as well as logical relations. The shuttle is illustrated as a system element on the hierarchy level of an AMS. The shuttles consist of two driving modules, which are specified here as logical groups. The driving modules contain the MFMs spring and tilt module, tracking module and drive and brake module. The drive and brake module is driven by a linear motor and is steered by a self-optimizing operating point controller. The rotor of the linear motor is in the shuttle and the stator is in the route section. A magnetic field is generated between the rotor and the stator, from which the driving power Fm results. The influences such as rolling resistance and wind force, as well as the weight of the shuttle reduce the driving power, so that the final current velocity V is present. The current velocity is measured and transmitted on to the velocity controller. The velocity controller compares the current velocity with the objective velocity, calculates a target driving power F* and passes this size on to the self-optimizing operating point controller. Depending on the situation, the velocity controller takes over the objective velocity from the objectives of the user or from the distance controller. This issue are presented by the alternative entrances A and B into the velocity controller. If the shuttle does not drive in a convoy, the velocity controller takes over the goal velocity from the objectives of the user. If the shuttle drives in a convoy or if it forms a convoy or the convoy dissolves, the velocity controller receives the objective velocity from the distance controller. The distance controller determines the goal velocity from the current distance of the shuttles to each other d and the distance demanded in the convoy d*. The distance controller receives the information concerning distance and demanded distance by communication with other shuttles. If the shuttle does not drive in a convoy, the distance controller will be inactive. Therefore different system structures are active depending on the current situation of the shuttle. Switching between these system structures leads to a behaviour adjustment of the system and this corresponds to the third step of the self-optimization process.

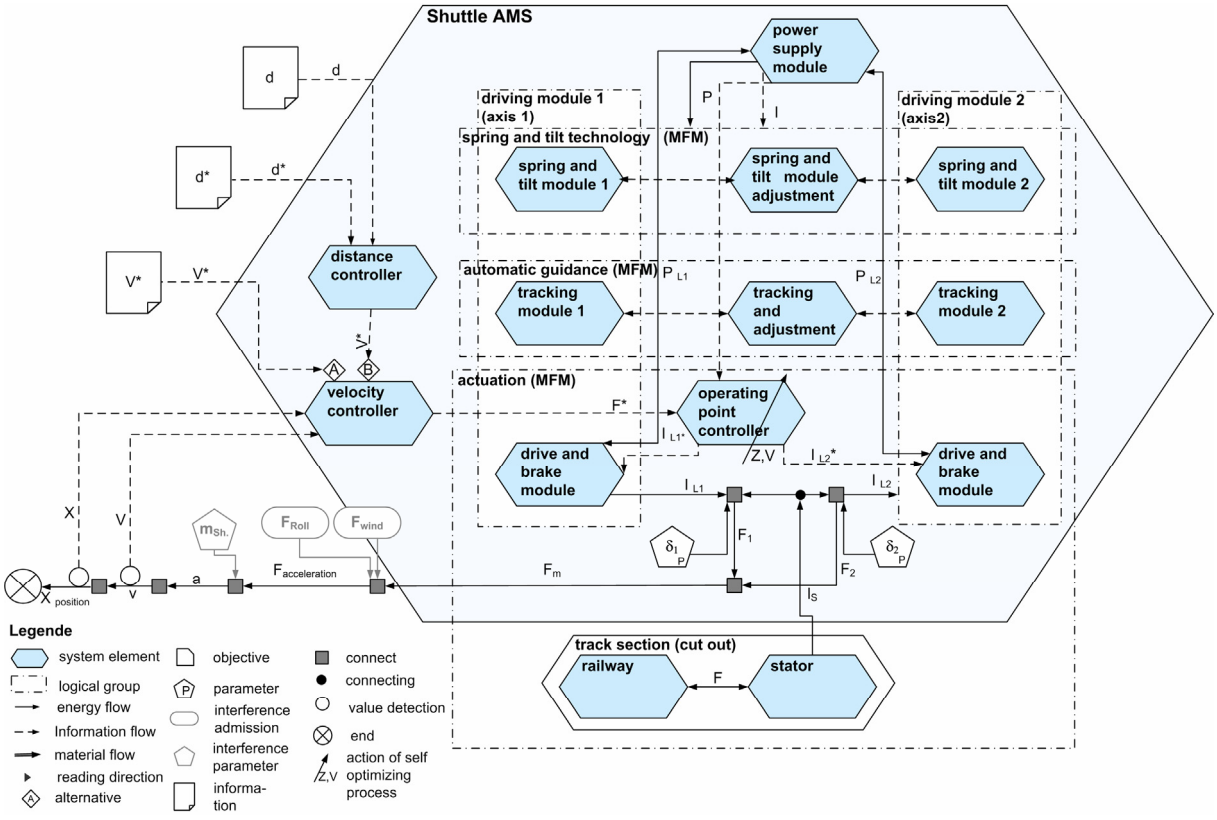

Figure 4. Active structure of a shuttle

4 SOFTWARE-SPECIFIC DESIGN

The data relevant for the domain-specific design, are derived from the principle solution. The partial model functions, active structure, target system and behaviour are in particular relevant for the software-technical design. The data relevant for the software specific design are specified with the Mechatronic UML [4]. In Mechatronic UML components are employed to model the structure. An embedding of continuous blocks into hierarchical component structures permits to integrate controllers into the component model. Discrete behavior of the components is specified by real-time statecharts or their hybrid extension hybrid reconfiguration charts.

4.1 Identification of software componentes

In the first step, the functions realised by the software technology are identified in the function hierarchy, e.g. the function "communicating. The functions are realized in the active structure by one or more system elements and their relations. The system elements and the associated relations are consequently extracted from the identified functions in the active structures. The system elements will be analyzed then. It has to be analyzed whether hierarchy exist between the system elements. Afterwards the system elements are transferred as software components into the component diagram with consideration of existing hierarchies. In the next step, the system elements are examined whether they pursue the optimization goals. If this is the case, the optimization objectives in the system of objectives must be identified and transferred in software components of different configurations. Finally it is to check, which behavioural models are relevant to the software solution. For example, the partial model “behaviour – states” specifies the fundamental operating states of the system. The states must be extracted from the partial model “behaviour – state”, in which the system elements identified above and their relations to each other are differently reflected. These states are transferred in state charts.

Figure 5 contains the data which are relevant for system concretization from the view of software technology for the example convoy. The data are a section of the principle solution from the project "Neue Bahntechnik Paderborn/RailCab", and thus also a section from the active structure represented in Figure 4. The partial model “behaviour – state” describes the fundamental operating state "single drive" and "drive in convoy". To the operating state „ single drive" and "drive in convoy", the active structures are assigned which are valid in these states. The active structures describe here only the section that is relevant for the software-technical design. The active structure on the left specifies two shuttles, which form a convoy and communicate about the current distance between them. The active structure on the right represents two communicating shuttles, which drive in convoy and exchange drive information about the current and the demanded distance. A shuttle is understood as a software component and must be accordingly transferred into the component diagram from the software-technical view.

The behaviour of the shuttles in the operating states "single drive" and "drive in the convoy", thus on NMS level, is determined operatively by the controller structures of the drive and brake module on MFM level. This aspect therefore must be considered in the software design. The velocity controller and the operating point control are active in the state "drive alone". These facts are marked in the active structure of the shuttle in the logical groups actuations MFM by dying the colour of the system elements: the velocity controller and the operating point controller. Furthermore, the distance controller is active in the state "drive in convoy".

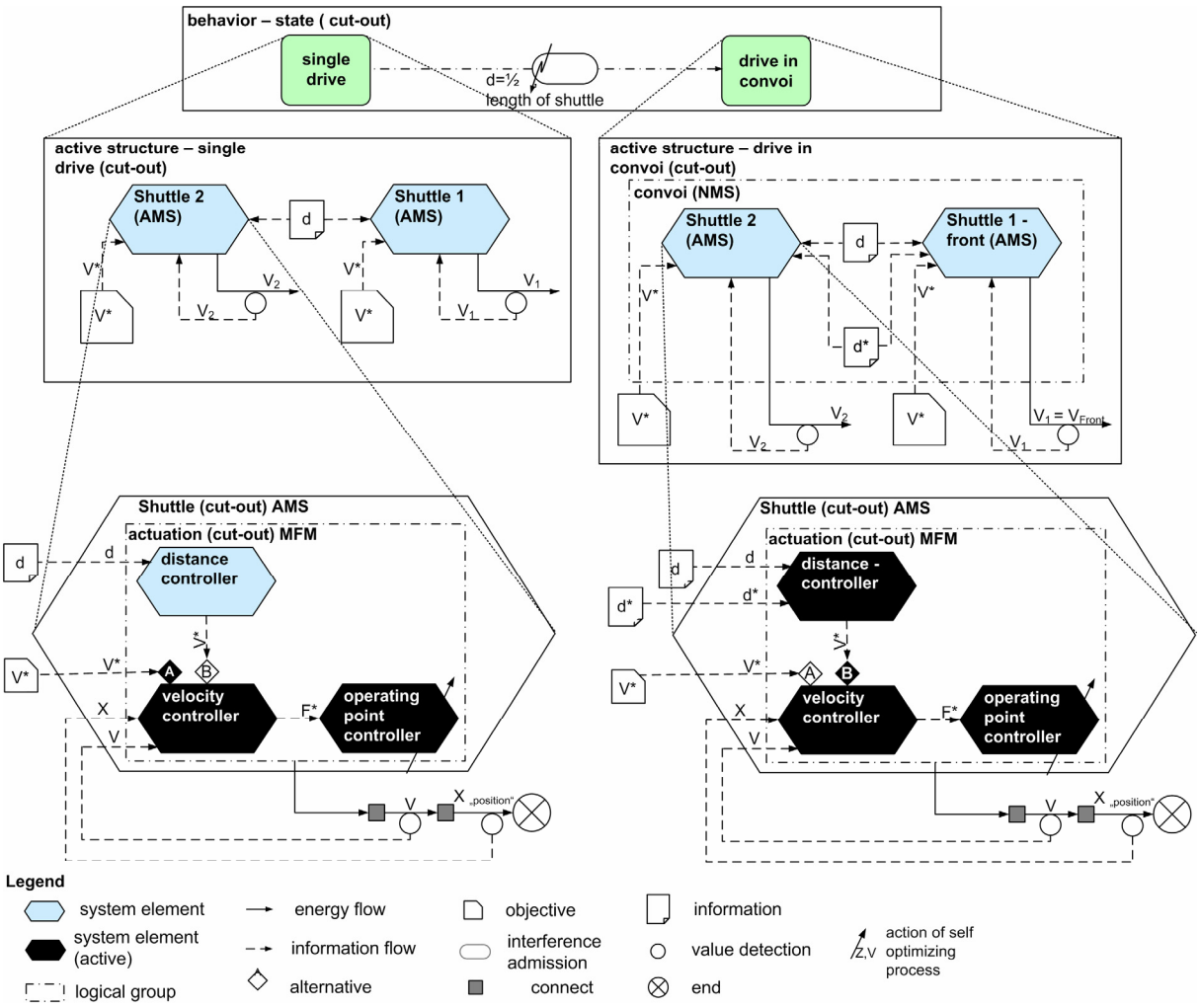

Figure 5. Relevant data in the principle solution for software design

4.2 Designing software components

For our example scenario, Figure 6 shows the component type for the shuttle, derived from the active structure in Figure 5. The Shuttle component contains a hybrid VelocityController (VC) component instance and a DistanceController (DC) derived from the distance and the velocity controller in Figure 5. The VC component computes the velocity needed to achieve a specific speed level. The VelocityController component has two incoming continuous ports for the current velocity V and the required velocity V* as well as the current position on the track X. Further, VC has one outgoing continuous port that sends the force value F* to the appropriate actuator devices. The DC component computes the required velocity. DC has three incoming continuous ports for the current distance d and the required distance d*. Further, DC has one outgoing continuous port that sends the required velocity value to following components (like velocity control, as we will see later).

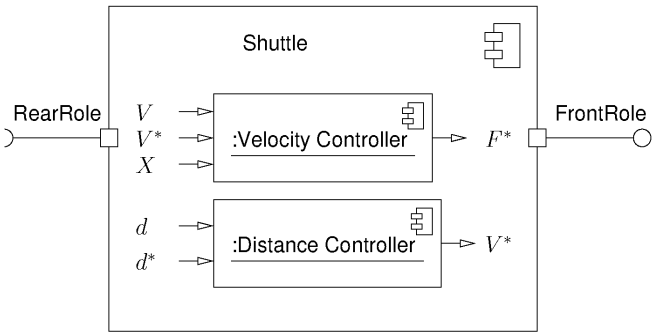

Figure 6. Structure of Shuttle

In Figure 7 the internal behavior of a Shuttle component is defined by a so-called Hybrid Reconfiguration Chart. This is an extension of a Realtime State Chart where a certain controller configuration can be assigned to a particular state. The figure shows the parallel state Synchronization is responsible for initiating and breaking convoys. The three sub-states of Synchronization represent whether the shuttle is in the convoy at the first position (convoyFront), at the last position (convoyRear), or whether no convoy is built at all (noConvoy). The two states convoyFront and convoyRear correspond to the state drive in convoy from the principle solution and noConvoy corresponds to the state drive alone. Note, that this is a simplified example in which the behaviour is restricted to convoys with a maximum of two shuttles.

Each sub-state of the parallel state Synchronization embeds different controller configurations. In Figure 7, it is specified that VC and DC has to be active when Synchronization is in state convoyRear. If Synchronization is in state noConvoy or convoyFront, only VelocityController is active. Thus, a change of a state induces a change in the control configuration and thus an adaption of system behaviour in accordance to step 3 of the self-optimizing process (see Section 2).

This kind of modeling has the advantage that it supports the decomposition into multiple components that is required to handle the complexity in mechatronic systems. Further the control engineering know-how is separated from the software engineering know-how.

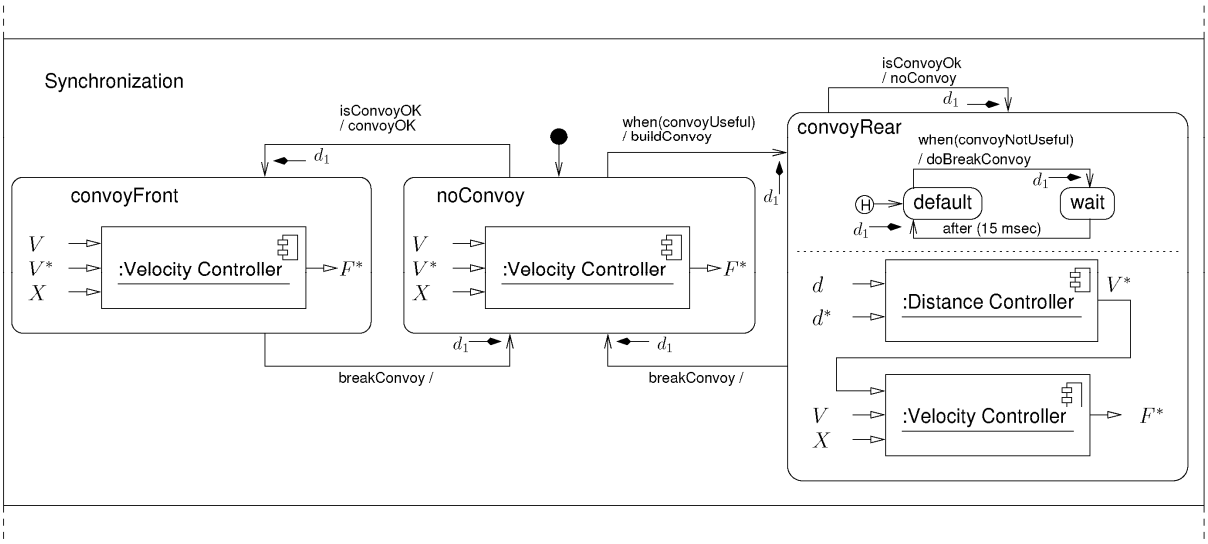

Figure 7. Extract of the Hybrid Reconfiguration Chart of the Shuttle software component

5 CONSISTENCY MANAGEMENT

During the domain-specific development, the models are constantly changed. Sometimes a change is not only relevant in one domain but in other domains as well. The idea is that relevant changes from the domain-specific models are propagated in the principle solution and then from there into the models of the other domains, since the principle solution integrates the different domain specific models. We use thus the principle solution as data model for all important data for the consistency management.

The following can occur referring to our example: When a change occurs between the discrete states, a discrete switch between the controllers could lead to a discontinuity in the output signal F*. This unsteadiness will cause additional stimulations which could lead to instability even when both controllers are stable on their own. In order to avoid these discontinuity, output cross fading is applied. This is specified by a fading function and a minimal and a maximal fading duration (d1) which is specified as an interval (Figure 8). These fading functions must be propagated into the principle solution, so that the controller developers can take them over in their design and implement them. This must be propagated over the principle solution, because the integration between the two domains, software engineering and control engineering based only on the interfaces.

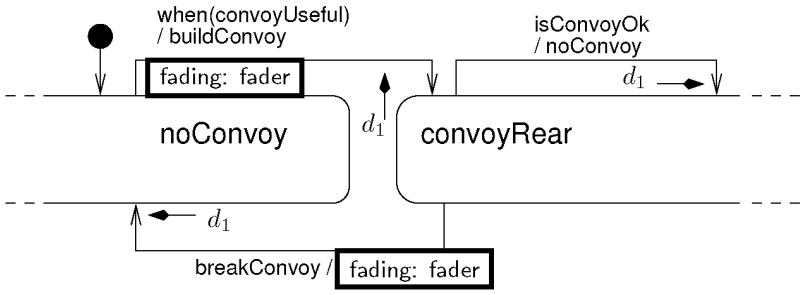

Figure 8. Extension of the Hybrid Reconfiguration Chart of Fig. 7 by two fading functions

The system element cross fade is introduced in the active structure between distance controllers and velocity controllers (Figure 9). The system element is completed with rules. These rules define the criteria to be over-dazzled. Additionally, there will be a reference that requests all other domains to examine possible effects of the change of the principle solution for their domain-specific design.

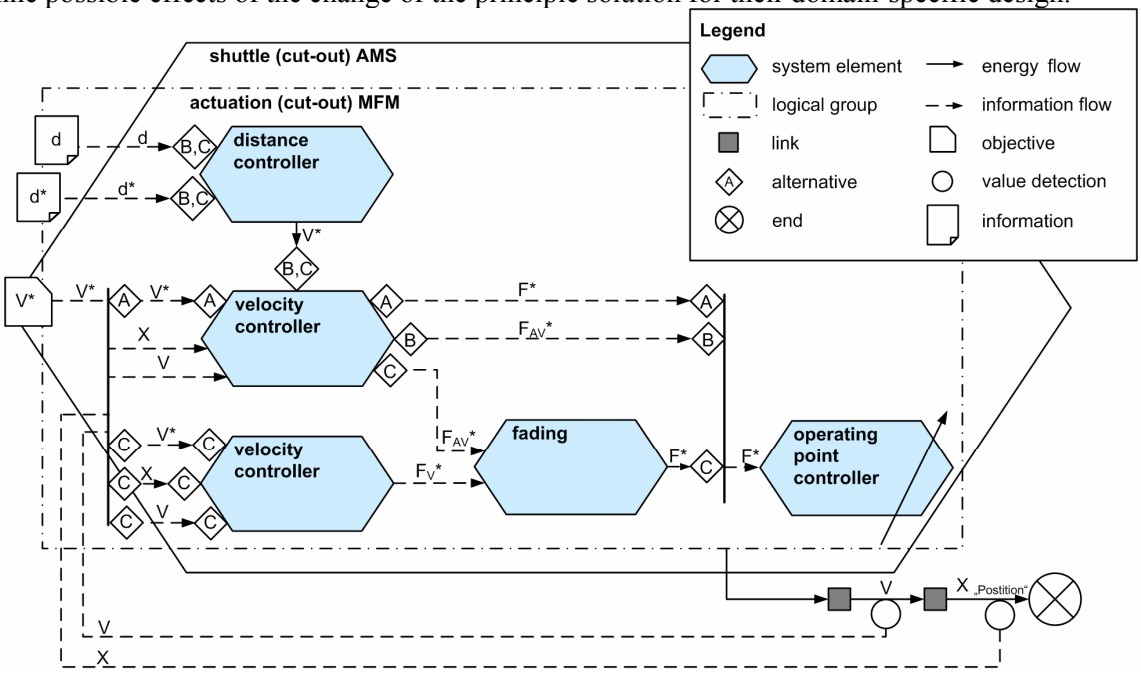

Figure 9. Addition of the fading functions in the active structure

The aim is to protect the consistency of the models automatically. The majority of the used models are based on graphs, so that approaches based on graphs for the consistency maintaining can be achieved. A formalism, which relates changes on two different models to each other, is triple graph grammars [5] (TGG). They connect the changes in two graphs by a third graph, the so called correspondence graph. As soon as the correspondence graph is created, it can be used to transfer changes of a model into the other model. TGG are used in our basic approach [6], in order to keep the consistency between more than two models. As described above, we connected model elements, respectively changes on model elements of different abstraction levels too each other. The TGG can be used to illustrate the refinement relations between model elements of different abstraction levels. The consistency-keeping action to the higher abstraction level, and the changes from the higher abstraction lever to concrete models should to consider during the change of a model.

Figure 10 shows a rule, which describes the consistency between objects of the active structure and objects of a reconfiguration diagram for the case that over survivor is involved. Two solution elements must already exist in the active structure, lying in different configurations for this situation. Each of these solution elements corresponds with a component, which is assigned to a state in the reconfiguration diagrams.

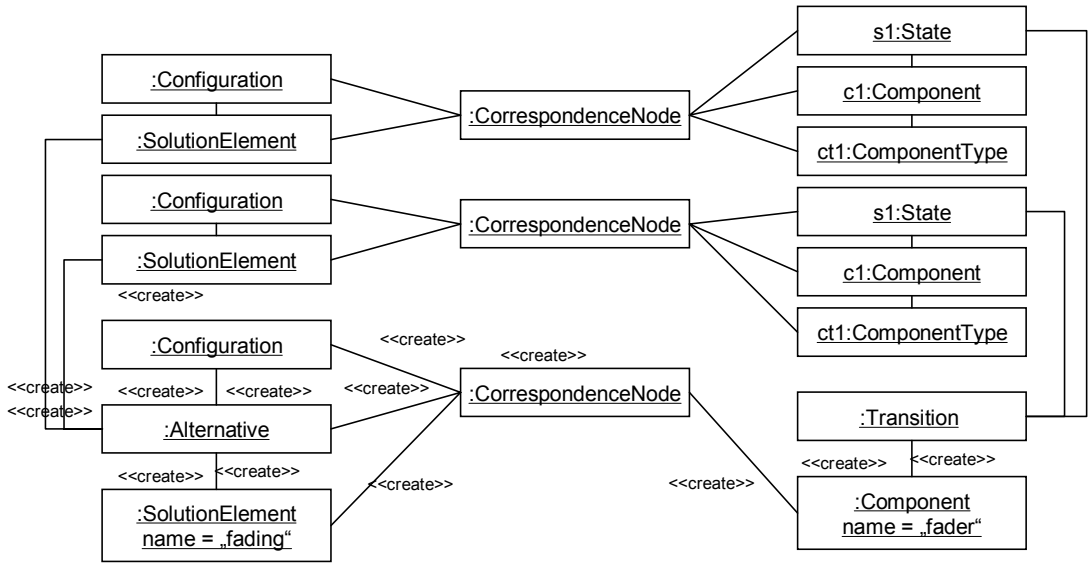

Figure 10. TGG-Rule for fading fuctions

In this example, it means that a new configuration with an alternative will be generated, if a new state is produced in the reconfiguration diagram. The components of the reconfiguration diagram have already two existing types in this case. This alternative consists of one solution element marked with "Fading" and two already existing solution elements.

Our approach these automatic modifications have to be proved by a developer, before the rules can be executed, which would modify further models. However, the rules can be calculated for further models a priori, so that the developer can see the effect of the original modification on further models.

6 RELATED WORK

In this contribution we introduced a new set of specification techniques for describing the principle solution of mechatronic and self-optimizing systems. This set of specification techniques extends existing techniques for modelling the concept of mechatronic systems. New are for example the views environment, application scenario, system of objectives, behavior activities and behaviour state. These views enable us to describe the self-optimizing process and its impact on the system. This can not be done by known approaches like:

ModCoDe – a system for the object-oriented modeling of mechatronic product concepts – models active elements and their interrelationships, including associated behavior models, described by means of block diagrams, state charts and bond graphs [7]. Buur specifies functions depending on system states and state transitions [8]. The specification technique for describing mechatronic systems developed by the project iViP focused on mapping models to describe requirements, functions, structures, constraints and shape and the cross-links between them [9]. Suh subdivides the description of technical systems into the domains: customer, function, physics and process [10]. An OMG consortium formulated SysML™ (System Modeling Language), which is a standard based on UML for the specification, analysis, verification and validation of technical systems. Here the emphasis is on modeling system structures, parameters, requirements and behavior (such as the system's activities, states and interactions) (http://www.sysml.org).

Furthermore, we have demonstrated how to describe the consistency between models with formal methods. And we described how to maintain consistency between domain-specific concretizing on the basis. There are two related works to our basic approach. The Viewpoint approach [11] describes, how to keep the consistency within and between different specifications through the so called "checks action". The approach uses conceptual graphs, graph matching, and causal dependencies to specify if a check action has to be performed. Our approach extends this idea as TGG can be used, in opposite to causal dependencies, in both directions. The MDI basic approach [12] is also based on the Triple graph grammars, focuses on the integration of tools and does not deal with the concretising of models and the consistency between domain-specific models.

7 CONCLUSION AND FUTURE WORK

This contribution shows, how the principle solution of self-optimizing systems can be domain-spanning described and how the domain-specific data can be extracted as basis for domain-specific concretizing of the system. For this purpose, we introduce a set of specification techniques developed in the SFB 614. Different views on the system and their networking are described with the specification techniques. The views requirements, environment, system of objectives, functions, active structure, shape, application scenarios and behaviour are necessary for the specification of the principle solution of self-optimizing systems.

We demonstrated the method exemplary on two model types in this work. We derived relevant data for software design from the partial models active structure and behaviour state and transferred them into software components and reconfiguration charts. In future work, we want to evaluate a generalisation on further specification techniques, which are used for the design of complex mechatronical systems. Specifically, we will work on a semi-automated transformation and consistency approach concerning the application scenarios of the principle solution and scenario description techniques used in software engineering [13,14]. This transformation cannot be fully automated since the application scenarios specified in the principle solution are to a great extend informal texts and figures. Consequently, the developer has to participate in the transformation and, subsequently, consistency processes. In addition, a tool that supports our basic approach will follow.

REFERENCES

[1] Verein Deutscher Ingenieure (VDI): *Design methodology for mechatronic systems.* VDI-Richtlinie 2206, Beuth Verlag, Berlin, 2004.

[2] Lückel, J.; Hestermeyer, T.; Liu-Henke, X.: *Generalization of the Cascade Principle in View of a Structured Form of Mechatronic Systems.*2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2001), Villa Olmo ; Como, Italy

[3] Gausemeier, J.; Frank, U.; Steffen, D.: *Specifying the principle solution of tomorrow's mechanical engineering products.* In: Proceedings of the DESIGN 2006, 9th International Design Conference, Dubrovnik, Croatia, 2006

[4] Giese, H.; Burmester, S.: Schäfer, S.; Oberschelp, O.: *Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration*. In: Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA, pp. 179--188, ACM Press, November 2004

[5] Schürr, A.: *Specification of graph translators with triple graph grammars.* In: Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG 1994. Volume 903 of LNCS., Herrsching, Germany (1994) 151-163

[6] Giese, H.; Wagner, R.: *Incremental Model Synchronization with Triple Graph Grammars.* In: Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS), Genoa, Italy (Oscar Nierstrasz, John Whittle, David Harel, and Gianna Reggio, eds.), vol. 4199 of Lecture Notes in Computer Science (LNCS), pp. 543--557, Springer Verlag, October 2006

[7] Welp, E.G.; Lippold, C.; Bludau, C.*: Ein System zur objektorientierten Modellierung mechatronischer Produktkonzepte (ModCoDe)*. In: VDI Mechatronik Tagung 2001. 12.-13. September 2001, Frankenthal. In: VDI-Berichte Nr. 1631, VDI-Verlag 2001

[8] Buur, J.: *A Theoretical Approach to Mechatronics Design.* Dissertation, Institute for Engineering Design, Technical University of Denmark, 1990

[9] Krause, F.-L.; Tang, T.; Ahle, U.: *iViP - Integrierte Virtuelle Produktentstehung - Abschlussbericht.* Carl Hanser Verlag, Stuttgart, 2002

[10] Suh, N. P.: *On functional periodicity as the basis for longterm stability of engineered and natural systems and its relationship to physical laws*. In: Research in Engineering Design, Springer-Verlag, London, Februar 2004

[11] Sunetnanta, T. & Finkelstein, A.: Automated consistency checking for multiperspective software applications; *Proceedings of the International Conference on Software Engineering Workshop on Advanced Separation of Concerns,* 2001, 1-12

[12] Königs, A. & Schürr, A.: MDI - a Rule-Based Multi-Document and Tool Integration Approach; *Special Section on Model-based Tool Integration in Journal of Software&System Modeling, Academic Press,* 2006

[13] Giese, H.; Klein, F.; Burmester, S.: Pattern Synthesis from Multiple Scenarios for Parameterized Real-Timed UML Models; *Scenarios: Models, Algorithms and Tools* (Stefan Leue and Tarja Systä, eds.), vol. 3466 of Lecture Notes in Computer Science (LNCS), 193-211, Springer Verlag, April 2005.

[14] Harel, D.; Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine, Springer Verlag, 2003.

Contact: Prof. J. Gausemeier
Heinz Nixdorf Institute, University Paderborn
Fürstenallee 11
33102 Paderborn
Germany
Phone +49 5251 606267
Fax +49 5251 606268
e-mail juergen.gausemeier@hni.upb.de