ICED13
INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED13
19-22 AUGUST 2013, SUNGKYUNKWAN UNIVERSITY, SEOUL, KOREA

AGENT-BASED CONSISTENCY CHECK IN EARLY MECHATRONIC DESIGN PHASE

Michael RAUSCHER, Peter GOEHNER
University of Stuttgart, Germany

ABSTRACT
Mechatronic design is a multidisciplinary design and has to face several challenges. One challenge is to provide a common understanding of the mechatronic system to be developed to all participating disciplines. This is done by sets of models which have many relations between them. If there are inconsistencies in the models, the whole mechatronic development project is endangered. This paper presents an agent-based concept for an automated consistency check of the models of the early design phase, which enables the providing of a reliable base for the further design process. After presenting the challenges of mechatronic design, the design models are abstracted to provide a broader application range of the presented concept. The structure of the consistency rules is presented and the agent-based concept is shown.

Keywords: agent-based system, consistency check, early mechatronic design,

Contact:
Michael Rauscher
University of Stuttgart
Institute of Industrial Automation and Software Engineering
Stuttgart
70569
Germany
michael.rauscher@ias.uni-stuttgart.de

1 INTRODUCTION

In mechatronic design multiple disciplines are involved. Depending on the definition of the disciplines, mechanical engineering, electrical or electronic engineering and information technology are integrated. Sometimes other disciplines like control engineering or fluid design are also involved. This multidisciplinarity can be a big benefit for engineering mechatronic systems. It enables the use of all advantages of the single disciplines. In the late 80s, when mechatronic systems began to emerge, the mechanical engineering was the main discipline and a mechanical basis was extended by electronic components. Due to the increasing importance of the information technology, this discipline was also integrated, but the unbalance between the disciplines persisted. The growing importance of the electrical engineering and the information technology equalized the emphasis of the involved disciplines more and more. Nowadays, the goal is to give all involved disciplines the same emphasis, to achieve better solutions. In order to achieve this goal, developers of mechatronic systems are exposed to various challenges, which are presented in the next chapter.

2 CHALLENGES OF MULTIDISCIPLINARY DESIGN

The involvement of multiple disciplines in mechatronic design is a first challenge. This means that developers from different disciplines have to incorporate their discipline and have to account for all of the constraints of it. The integration of these sometimes differing concerns would not be a special challenge, if the developers of the different disciplines had the same or a similar educational background. But due to the fact that for example mechanical engineers get another education than electrical engineers, with other educational contents and even differing development processes, the synchronization of the disciplines is not easy. Often same terms have different meanings for the different developers or they are talking about the same issue, but using different terms. To deal with these different terminologies, unified and often project specific terminologies can be used. In the VDI-guideline 2206 (VDI 2004), the V-model is presented as a common process model for the development of mechatronic systems to unify the different development processes.

Another challenge is the common understanding of the system to be developed (Svensson et al., 2003). Each developer has his specific view on the system and ignores information not relevant to him consciously or unconsciously. This can be seen often in a lack of mutual understanding or problem awareness for the other disciplines. An approach to deal with that is the Virtual Prototyping (VDI 2004). In early phases of the development virtual or sometimes real prototypes are created to have a vision of the common project goal and to understand the (partial) development of the other disciplines better.

A third challenge is that the different disciplines normally use different models to describe their development. Due to the fact that in each case other information is relevant, each discipline uses its special models. All these models have dependencies between each other, because in the end they describe the same mechatronic system. Gausemeier (2005) and Frank (2005) present a system of coherent partial models for describing the principle solution of self optimizing systems, which creates a common base in the early design phase to be used in the following discipline specific design phase. This discipline specific design phase was simplified by this common reference, but the challenge to keep the consistency of different dependent models is still present

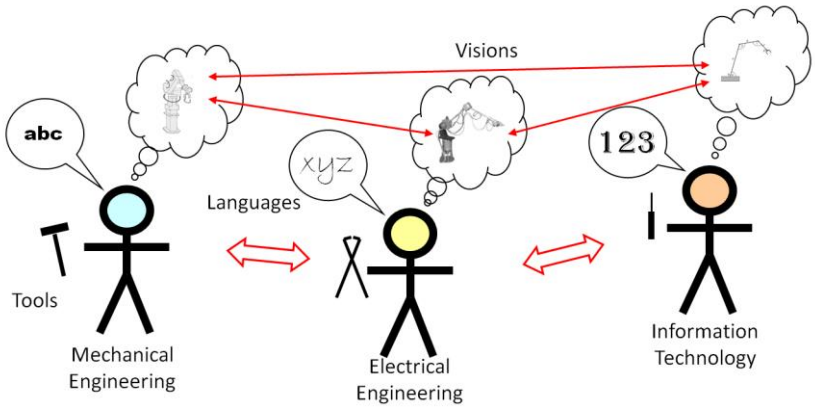

Figure 1: Challenges in mechatronic engineering

These challenges, visualized in Figure 1, make the mechatronic development or, more general, the multidisciplinary development difficult. Although if approaches using multiple common models in the early design phase are improving the development process, the risk of inconsistencies between the different models is still given. That means that if one model is changed and the change is not properly propagated to the other models, the developed mechatronic system might not work. If such a problem is not detected until the system is built, the removal of the problem is very expensive. An automated consistency check will help to reduce that risk and to improve the quality of the development. The check can be executed during the development process and will run in background. Due to the fact that the test is automated, the risk that a person might miss some issues is reduced. This will raise the quality of the mechatronic system and reduce the cost of the development. To enable such an automated consistency check, the models have to be formalized and abstracted to a more common form, which will be done in the next chapter.

3 ABSTRACTION OF THE DESIGN MODELS

One good approach for a common view in the early design phase is the system of eight coherent partial models which is presented by Frank (2005). Due to the fact that for some projects other or additional models are used, the partial models are abstracted to meta models. This enables the concept to be used also for other models than the partial models. Afterwards, the models have to be formalized and modeled in software, in order to be readable or interpretable for the automated consistency check.

3.1 Exemplary abstraction of the active structure model

As the model Active Structure is easy to understand, it is used as an example to present the abstracted model. In the original model of Frank (2005) there exist system elements, which have different interfaces. The system elements can be connected to each other by using these interfaces. A connection can symbolize a material, an energy or an information flow. Fig. 2 shows an extract of the active structure of the accelerating unit including a speed control of an automated soccer ball shooting machine.

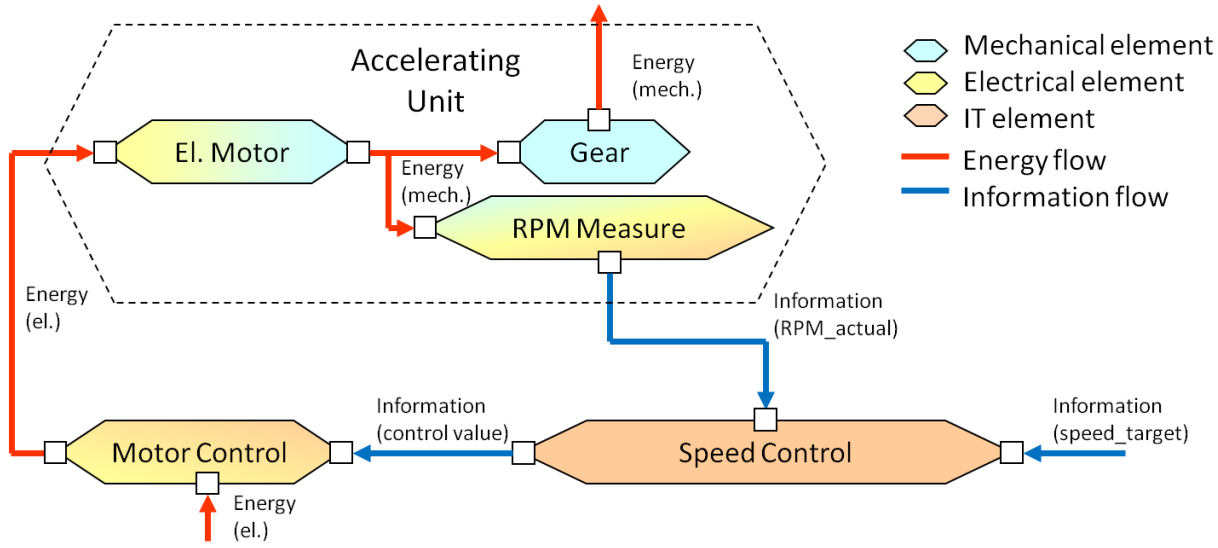

Figure 2: Exemplary active structure of an accelerating unit (representation according Gausemeier and Kahl (2010))

This machine is able to shoot soccer balls very precisely at a goal with a very high velocity. The electrical engine, the gear and the rpm-sensor were combined to the logical group “accelerating unit”. The flows between the single elements of the active structure (energy flow and information flow) have been specified more detailed, as the energy is specified as mechanical and electrical energy. Additionally to the original specification, in this example the differentiation between the different involved disciplines is realized by different colors of the system elements.

This representation of the active structure has to be abstracted to a meta active structure, to guarantee a wide applicability. In the abstraction the system elements become hardware/software components (HW/SW component), which can either be an elementary component or a composite component. The

connection between components, according to the active structure, can be a flow with the specialization of material, energy or information flow. Beside the flow, the dependency is introduced as an additional link between HW/SW components, to allow the applicability on other models. This dependency enables the modeling of logical relations such as “depends on” or “calls for”. The flows can be detailed during the design phase. An energy flow can be specified for example as electrical energy. Furthermore, electrical energy can be described in details as voltage and current values. Similarly, other flow types are being specified. The meta model of the active structure resulting from the abstraction is presented in Fig. 3 using UML.

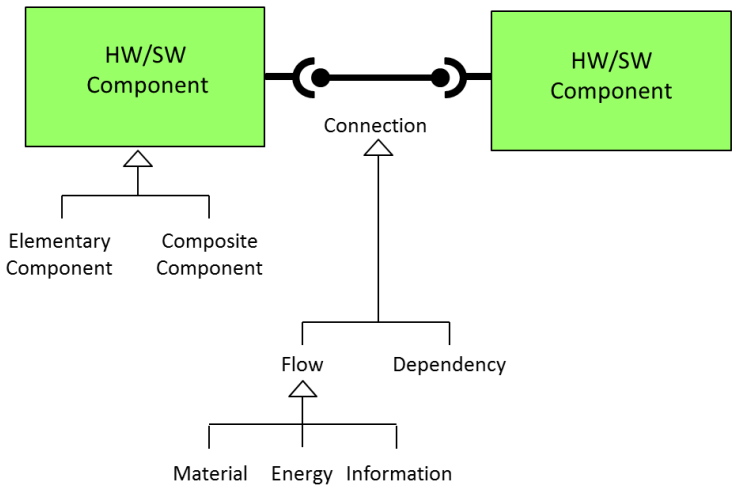

Figure 3: Meta model for the active structure

The meta model has to be represented in software to allow automated consistency checks. This means that the meta model has to be transformed in a class diagram in order to use it in software.

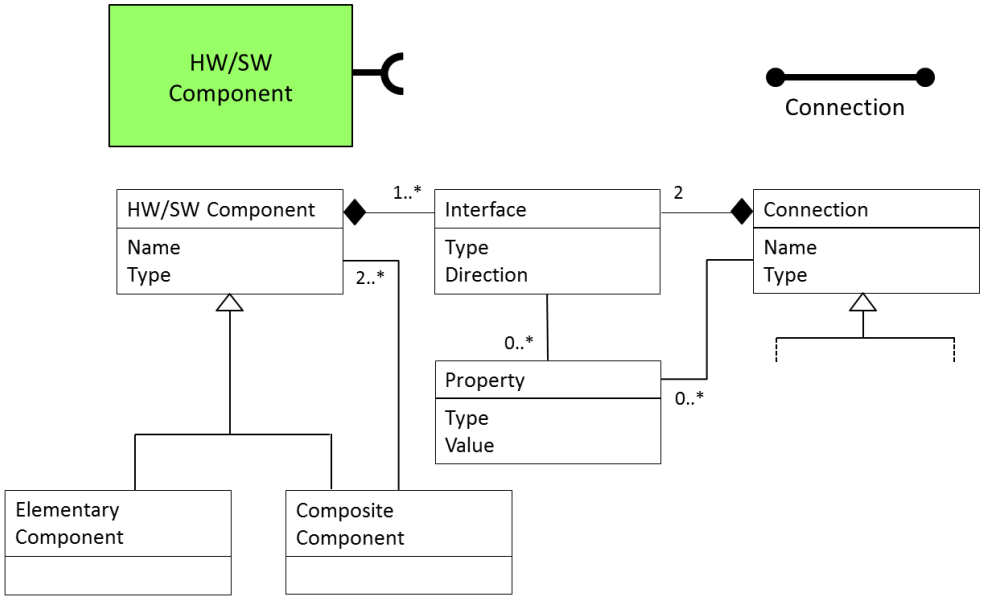

Figure 4: Extract of the class diagram for the meta model active structure

HW/SW components are represented as classes with the attributes name and type. An HW/SW component can have one or more interfaces. These possess the attributes type and direction (in/out). An interface can have refining properties. By adding properties, which can be done during the whole design process, the active structure can be detailed. Properties consist of a property type and a corresponding value. Connections are represented as discrete classes. They possess the attributes name and type. Due to the fact that connections have to represent the physical connection elements in the subsequent design they also possess interfaces. Connections have the same properties as their interfaces, which can be used to detail them. The difference between the various types of flow and their dependencies is realized by inheritance. This means that for example Flow is a subclass of

connection and the different types of flow are subclasses of Flow. An extract of the class diagram can be seen in Fig. 4.
The presented formalization in software is necessary for all models, which are to be used for the automated consistency check. After that, the rules which are applied for the check have to be modeled.

4 RULES FOR CONSISTENCY CHECK

The overall goal is to ensure the consistency of the design models. Therefore, rules, which check this consistency, are needed. There exist two types of rules: one type will guarantee the correct formal assembly of the models e.g. the definition of connection possibilities of elements. The other type is to check the content of the models, e.g. for assuring the correctness and the integrity of connections between elements. The first type of rules represents formal constraints and therefore is not in the focus here. The second type is much more interesting and will be presented in the following lines. These rules have to compare properties or attributes of the objects or the connections of the models. According to Wagner (2008), the rules should contain two conditions, a precondition and a main condition. The precondition qualifies the scope of the application of the rule. In other words, the precondition describes in which cases a rule has to be considered. An example for such a precondition is the existence of several energy sources in a model. The main condition contains the facts to be checked, e.g. the equality of properties of two interconnected objects. The conditions can exist in different forms. According to that, a condition can be the comparison of two values or the comparison of whole object structures. Additionally, a condition can be the existence of properties or of an object. Finally, a condition can be a logical combination of other conditions, which are associated e.g. with an AND or an OR. The representation of a rule can be seen in Fig. 5.

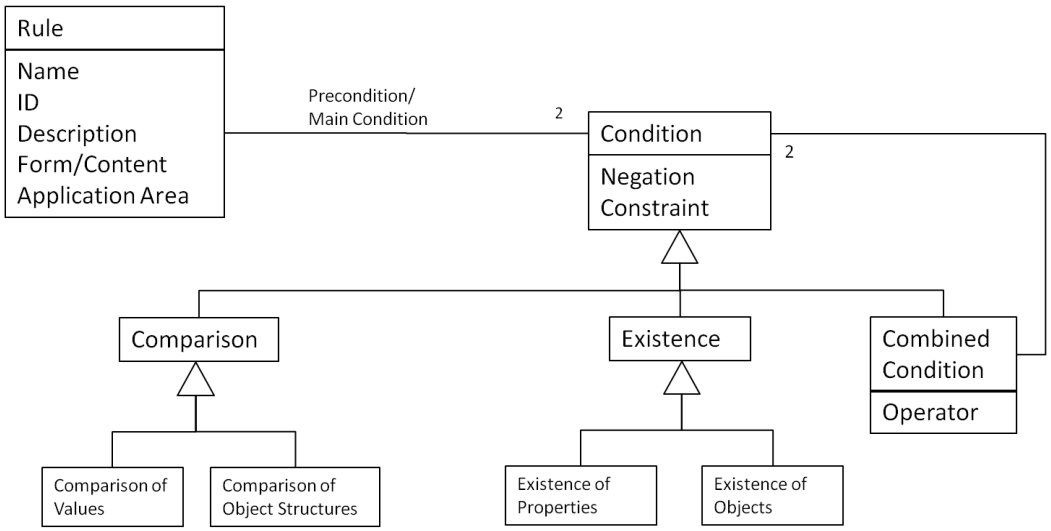

Figure 5: Representation of rules

Rules are used to check the consistency within a single model and within a system of several models. An example for the first case is that the interfaces of connected elements in the active structure have to have equal properties. An electrical energy flow, marked as an outgoing flow with a voltage of 5 volts, cannot be connected to an electrical energy flow, marked as an incoming flow with a voltage of 12 volts. Depending on the already reached detail level of the design process this rule is only applicable, if the necessary information is already modeled.
In the second case, rules are used to check the consistency between two or more models. An exemplary rule is, that the signals, which are defined in the behavior model (e.g. fire conditions of transitions), must also be defined as information flows in the active structure, if they have no other origin.
In order to realize a substantial consistency check, a large rule base is needed. For each relation between elements, several rules are necessary. For example for the connection of an electric motor to an energy source more than 5 rules have to be checked depending on the level of detail: electric energy flow, electric energy type for input and output interfaces, same voltage, same frequency, consuming power less than possible producing power etc.

The result of the application of these rules is a consistent model or system of models which describes the early design of a mechatronic system. The consistency check itself is a time-consuming process, which can be executed after having finished modeling or in parallel to the modeling. This article presents an approach where describes how the consistency check can be performed using software agents.

5 AGENT-BASED CONCEPT FOR CONSISTENCY CHECK

Due to the fact that the consistency check should run in background but provide an active behavior, e.g. active information of the developers about detected inconsistencies, software agents can be used to provide that functionality. Software agents or simply agents are encapsulated software units, which provide an active behavior and follow certain goals (Jennings, 2000). They are able to communicate with each other to reach their goals. Agents are often used for systems which are naturally distributed or need a high degree of flexibility. This enables a distributed development of a mechatronic system, because the agents can perform the consistency check even if the different models are distributed on different PCs. Another advantage is that not all situations have to be foreseen during the development of the agent system. This means that for example additional agents can be added to the system during runtime and they can communicate and work together without the need of changing the source code of the application. Agents can be classified in two main types: reactive and deliberative agents. A reactive agent reacts on changes of its environment with a predefined behavior. A deliberative agent also reacts on changes but has a proactive behavior and tries to plan its reactions. The agents presented here can be seen as reactive agents. Using agents has the advantage that a new mechatronic development model can be easily added to the consistency checking system without changing the system, only corresponding consistency rules have to be added.

In order to profit from that advantages, an entity-based approach is used, which means that each development model will be represented by a model specific agent. Inside some models there exist further agents. An overview of the architecture can be seen in Fig. 6.

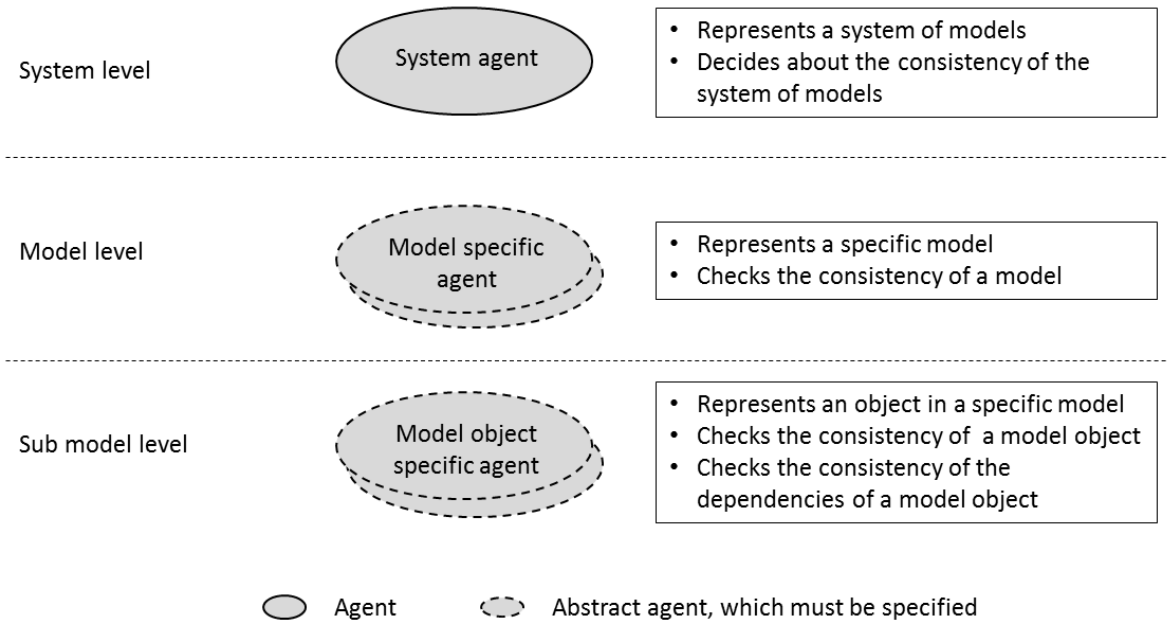

Figure 6: Architecture of the agent-based system

5.1 System agent

A system agent represents a system of models. It is the highest agent in the hierarchy of the agents. It will finally decide if the system of models is consistent or not. Therefore, it knows all contained models respectively their corresponding agents (model specific agents). It is able to communicate with those agents in order to get their inspection results. It will be informed by model specific agents if a change in one of the models was made and sets the system of models to an inconsistent state. If it gets a positive answer for the consistency check from each of those agents, it will decide that the system of models is consistent then.

5.2 Model specific agent

Each model specific agent represents one model. Due to the fact that there is a significant difference between the models, this agent is an abstract agent. In the realization there will be for example an active structure agent, a behavior agent etc. The term model specific agent will be used in the following, if these agents are meant in common. A model specific agent decides if its model and the dependencies to other models are consistent. If its corresponding model was changed by a developer, it will inform the system agent about that change and will check the consistency of its model. Therefore, it requests its sub agents, if they exist, to perform a consistency check on their level, checks the consistency of its model at a common level on its own and collects the consistency check results of the sub agents. Sub agents are model object specific agents, which are presented in the next chapter. Some models consist of several parts like the behavior model consists of multiple behaviors or the active structure models consists of HW/SW components. These parts are represented by separate agents as well.

5.3 Model object specific agent

A model object specific agent represents one modeled object like a HW/SW component in the active structure model. This agent is also an abstract agent and must be specified for the object it has to represent. It decides about the consistency of that object. Therefore, it receives the request to check the object and its dependencies from the corresponding model specific agent. The agent will perform the check and sends the results of its check to the requesting agent. In case of the component agent this is the active structure agent.

The agents are running in background and monitor their corresponding objects. If an agent detects a change in its object by a user, it will inform its directly related objects respectively the corresponding agents. They principally assume that there exists an inconsistency now and perform a consistency check applying the already mentioned rules. If they detect an inconsistency, they will inform the user about it. Else, they set the models to consistent again and wait for the next change by the user.

5.4 Exemplary implementation of the agents

In case of the active structure model, the model specific agent will be realized as an active structure agent. It is responsible for the active structure model and will check its consistency.

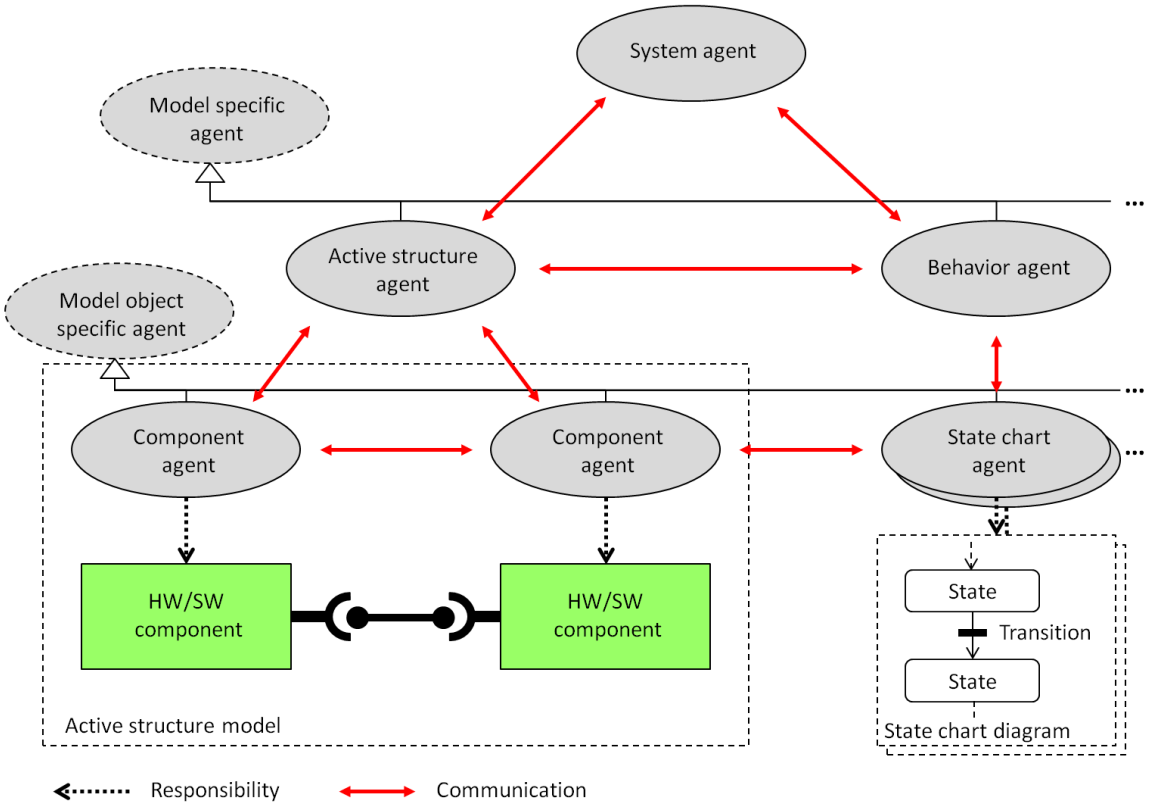

Figure 7: Exemplary agents for the active structure model and the behavior model

Therefore it knows all the HW/SW components of the model including their connections. By applying the corresponding rules, it will check the model. Additionally, there exist so called component agents, responsible for the HW/SW components. They will check the internal consistency of each component and its connections. For the behavior model there exists a behavior agent. Due to the fact that the behavior model consists of several parts, state chart agents represent the state charts.
These agents are able to communicate with each other and request information about the other models respectively objects. Each agent has a set of consistency rules it has to check after detecting a change. The results of the check are presented to the user so he can fix the inconsistencies found.

8 CONCLUSION AND OUTLOOK

Mechatronic design implies many challenges for the developers. Using common models and integrating all involved disciplines in the early design phase is one attempt to face these challenges. To reduce the number of problems in the later design phases, the consistency of the different models of the early design phase is very important. An automated consistency check eliminates human mistakes. Software agents provide an active and flexible behavior to perform this check. The results are reported to the developers to fix detected problems. Retrieving solutions for the inconsistencies by the agents themselves, like it is presented by Kratzer et al. (2011), is only possible in very few cases at this early state of the design. But, depending on the number of rules in the knowledgebase, this concept provides an extensive and automated consistency check of the design models. The abstraction of well-known models like the system of partial models of Gausemeier (2005) to meta models provides a broader range of application of the presented concept. The next step is to finish the implementation of the concept in a prototype in order to present the automated consistency check for a real mechatronic design. If the consistency of the early design models can be ensured, the concept can be extended to the later design phases, including the work of Kratzer et al. (2011). This will enable a design supporting system for the complete mechatronic design process, which will reduce the mistakes in mechatronic design and will reduce the costs for high quality mechatronic systems.

REFERENCES

Frank, U. (2005) ‚*Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme*‘. PhD thesis, Universität Paderborn, Paderborn Germany.

Gausemeier, J. (2005) ‘Designing tomorrow´s mechanical engineering products’. In Proceedings of the *ITI 3rd International Conference on Information & Communication Technology (ICICT 2005)*, Cairo, Egypt, December 2005. IEEE Computer Society.

Gausemeier, J., Kahl, S. (2010) ‘Architecture and Design Methodology of Self-Optimizing Mechatronic Systems’. In: Milella, Annalisa; Di Paola, Donato; Cicirelli, Grazia (eds): *Mechatronic Systems, Simulation, Modeling and Control*, pp. 255-282, Vucovar, 2010, InTech Open Access Publisher.

Howard, T. J.; Dong, A. (eds): *Proceedings of the 18th International Conference on Engineering Design (ICED)*, pp. 178-189, Copenhagen Denmark.

Jennings, N. R. (2000) ‘On agent-based software engineering‘. In *Artificial Intelligence* 117.

Kratzer, M., Rauscher, M., Binz, H., Göhner, P. (2011) ‘An Agent-based System for Supporting Design Engineers in the Embodiment Design Phase’. In: Culley, S. J.; Hicks, B. J.; McAloone, T. C.;

Svensson, D., Hallin, K., Zimmerman, T., Malmqvist, J. (2003) ‚An Information Model For Mechatronic Products Focusing On Early Development Phases‘. In: Martensen, N. H. (eds): *Proceedings of the 6th workshop on product structuring - application of product models*, pp. 111-130, Technical University of Denmark.

VDI (2004), *VDI2206: ‚Design methodology for mechatronic systems - Entwicklungsmethodik für mechatronische Systeme‘*, Düsseldorf Germany, Beuth Verlag GmbH.

Wagner, T. (2008) ‚*Agentenunterstütztes Engineering von Automatisierungsanlagen*‘. PhD thesis, Universität Stuttgart, Stuttgart Germany.