Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2010
August 15-18, 2010, Montreal, Quebec, Canada

IDTEC2010/28615

CONSISTENCY CHECKING OF MECHATRONIC DESIGN MODELS

Peter Hehenberger
Institute for Computer-Aided Methods in Mechanical Engineering
Johannes Kepler University
Linz, Austria
peter.hehenberger@jku.at

Alexander Egyed
Institute for Systems Engineering and Automation
Johannes Kepler University
Linz, Austria
alexander.egyed@jku.at

Klaus Zeman
Institute for Computer-Aided Methods in Mechanical Engineering
Johannes Kepler University
Linz, Austria
klaus.zeman@jku.at

ABSTRACT
During all phases of the design process there is a need to build models. Hierarchical models are very important tools for complex activities such as engineering design. In engineering of high performance products, mathematical modeling and simulation, i.e. experimenting with computer-based models, is an increasingly important technique for solving problems, evaluating solutions and making decisions. However, large design models may contain thousands of model elements. Designers easily get overwhelmed maintaining the correctness of such design models over time. Not only is it hard to detect new errors when the model changes but it is also hard to keep track of known errors. In the software engineering community this problem is known as a consistency problem and errors in models are known as inconsistencies. This paper presents an approach for consistency checking of mechatronic design models.

1. INTRODUCTION

1.1 Motivation
Today Time-to-Market and Rapid Development are very important aspects of the product development process. The evolution of the corresponding market requirements during the last years has deeply transformed the designer's way of thinking and acting during all stages of product development.

In fact, at present, time-to-market, quality standards, environmental impact, safety, and cost effectiveness have become essential demands which affect the whole life cycle of the product development. In order to improve the performance of new products, the different fields of mechatronics and their interactions are increasingly exploited which has led to higher product complexity. Today, the design activities take place in a multidisciplinary environment, which often involves engineers of different backgrounds working on a common product. Additionally, the enlarged use of computer aided tools and their continuous enhancement lead to more complexity of the product design process itself.

Hence, a new approach in Design and Engineering of products can be considered a key point for optimizing the design process. These methodologies are being used extensively to assist in decision-making during the product design process which starts with conceptual design and is followed by basic and detailed design. Conceptual design plays a central role in ensuring design quality and product innovation because many success-critical decisions are made during this phase.

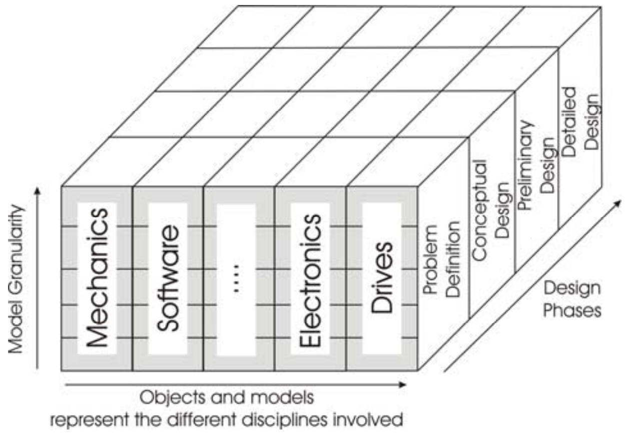

Figure 1. Viewpoint of product models and data

In the design process of mechatronic systems we have to analyze product models and data from different viewpoints – representing the different disciplines involved. Figure 1 lists several dimensions of viewpoints which include:

- objects and models from different disciplines, which are involved in the development process of a mechatronic system
- model granularity (which differs across disciplines), denoting the extent to which on object or model is broken down into smaller parts (e.g., sub-models with higher degree of detailing and lower level of abstraction, respectively)
- design phases: The development/design process can be structured into four phases, namely problem definition, conceptual design, preliminary design and detailed design.

Furthermore, there is also the need for consistency because if objects and models are independently created and maintained by the various disciplines then correctness is no longer guaranteed. The same is true for objects or models that are transferred from one discipline to another, from one abstraction level to another and from one design phase to the next one – if such objects or models are subsequently modified on both ends just as proposed in simultaneous engineering. This problem, of course, becomes increasingly pronounced with system size and complexity because likely more people and disciplines are involved. A proven strategy to handle this increasing complexity is to divide up the work as it is done in concurrent engineering. However, it is rarely possible to divide up a problem such that two people can work fully independently from one another and then, at the end, expect that the results of their work fit together without conflicts. Through good modularization, it is possible to achieve some degree of independence; however, it is never possible to avoid dependencies. The role of consistency is to characterize such dependencies – to identify the conditions that have to be met for objects and models to fit together.

1.2 Illustration and Problem

Consistency rules are simply conditions on a model which evaluate to true or false – depending on whether the rule is satisfied by the model or not. There are typically many such consistency rules and these rules need to be re-evaluated whenever the model changes. This re-evaluation is computationally expensive if done exhaustively with every model change. During a rule's evaluation, a consistency rule investigates a model - typically a portion of the model only. We define the accessed portion of a model as the scope of a rule. For example, we want to analyze different design models during the development process of a gripper. Figure 2 shows different simplified models from different design stages. A more detailed description of the application is outlined in chapter 4.

The function structure is used to describe product functionality because in engineering design, the final goal is the creation of an artifact, product, system, or process that performs a function or functions under certain constraints to fulfill customer need(s). The conceptual design model is used to select, define and fix the main parameters of the gripper. In detailed design we have a CAD-model which describes the geometry (and other properties) of the gripper.

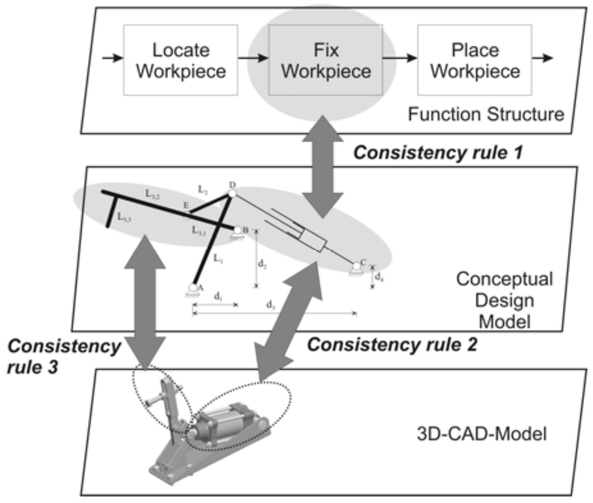

Figure 2. Consistency rules between different models

Consistency rules are written for a context element (a type of model element, e.g. a function) from where its evaluation starts. For consistency rule 1 (Fig. 2), the context element is a function and the interaction between the function structure and the conceptual design model is described as "applies to <fix workpiece> requires(this).contains(<pneumatic cylinder>)". This means that for the function "fix workpiece" a drive like a "pneumatic cylinder" is required. In the second rule the interaction between the model parameter "distance piston" and the geometric representation in the CAD-model is defined as equal. In the same way rule 3 is defined.

If we imagine that the different models in Figure 2 are created and maintained by different people – perhaps even people of different disciplines – then the benefit of consistency checking is the ability to identify conflicts that arise out of the independent work of these people or disciplines. From a implementation point of view, several mechanisms are necessary to automate this:

- integration of models and tools if separated so that they can be reasoned about uniformly
- online/offline mode if models, tools, people are distributed in time and space
- mechanisms for detecting and tracking constraints

The model/tool integration is not so much a research challenge but a standardization problem where tool vendors need to integrate their tools better. The distribution problem could be avoided by having one central server where all models are available. The third point on detection and tracking, however, has received the least attention. So while all three points are important, this paper deals solely with the third point and assumes that models are integrated and available in a central repository (not distributed) for simplicity.

2. BACKGROUND

2.1 Special Characteristics of Mechatronic Design

Mechatronics may be defined as an interdisciplinary field of engineering, which characterizes the interconnections between mechanical engineering, electrical engineering and computer science such that these interconnections are the basis for designing successful products [1, 2, 3].

Mechatronic systems are multidisciplinary products, therefore the knowledge required for developing such products/systems is broad. Currently, there is a lack of integrated development methodologies and tools for mechatronic product development. Traditionally, mechanical engineers develop their design with strong emphasis on the geometric domain, the electrical components are engineered separately in an electrical domain. These steps result in a given equipment for automation and control of the system which is usually developed in an automation department represented more or less by software engineers. All these persons have different views on the same mechatronic products.

When designing a mechatronic system, it is possible to design the mechanical equipment, before any of the control system design has been initiated. An obvious drawback of this sequential approach is the (predictable) lack of compatibility between the subsystems which results in additional efforts and (non conformance) costs to meet the specifications of the total (overall) system. Another drawback of this approach is that during the design process decisions have to be made about whether to use a mechatronical or just a mechanical solution concept. Designers have to navigate and coordinate between mechanical and electronic solutions.

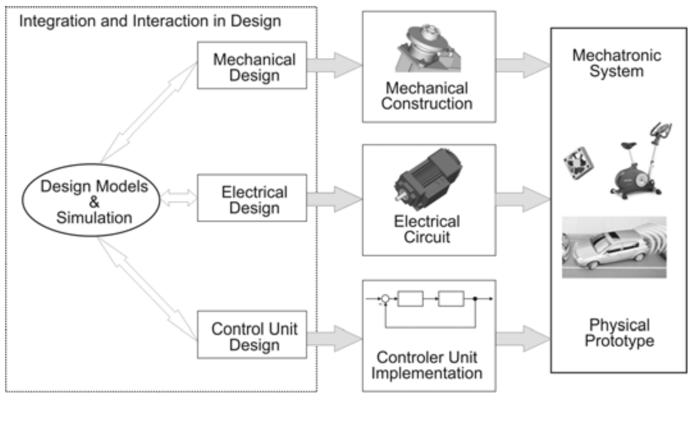

Figure 3. Integration in Mechatronic Design

Without coordination between the different disciplines it is difficult to find a superior solution. Not only the selection of materials and the knowledge about (process) constraints (e.g. with respect to the possible geometric properties of a part) play a role, but also the selection of completely different solution principles from different domains. It is clear that engineers need help for their increasingly complex and multidisciplinary design tasks. It is necessary to support the designers to rapidly review and evaluate alternative solutions during the design process and to identify the correct (i.e. optimum) implementation choice.

One of the key issues in the development of modern mechatronic systems is the strict integration of mechanical, control, electrical and electronic as well as software aspects from the beginning of the earliest design phases on, as it can be seen from Figure 3.

2.2 Consistency Checking of Models

Large design models contain thousands of model elements. Designers easily get overwhelmed maintaining the correctness of such design models over time. Not only is it hard to detect new errors when the model changes but it is also hard to keep track of known errors. In the software engineering community, this problem is known as the consistency problem and errors in models are known as inconsistencies.

Inconsistencies are detected through reasoning processes that require the existence of rules that describe correctness. Such consistency rules describe conditions that a model must satisfy for it to be considered a valid model (e.g., syntactic well-formedness, coherence between different diagrams, and even coherence between different models). Such consistency rules obviously vary for different models. That is, what is considered valid (correct) and invalid depends on the modeling notation (language) and its semantics. However, for many mainstream modeling languages (e.g., SysML, UML) such semantics are well-defined and engineers have been able to identify consistency rules. However, this paper will demonstrate that such rules should also be customizable for a domain or even application.

As with any error, the sooner it is detected the easier and cheaper it is to fix. Instant error feedback of any kind is thus a fundamental best practice in the engineering process. To date, many approaches exist that help designers detect inconsistencies but most of them are only capable of checking the consistency of design models as a whole (i.e., exhaustively in a batch process) where all consistency rules are evaluated on the entire model. Unfortunately, batch consistency checking does not scale because the checking of larger models takes hours to complete. Instant consistency checking requires an understanding when, where, and how the model changes. For this purpose, our approach, called the *Model/Analyzer Approach* [4, 5, 6], logs how the engineer uses the modeling tool. Figure 4 shows the architecture of our approach. It depicts the modeling tool on the lower-right corner. The modeling tool needs to be wrapped such that we can observe user activities – changes to the design model which is embedded inside the modeling tool. The Consistency Checker (top-left) evaluates the model and while doing so the Model Profiler (middle) monitors what model elements the Consistency Checker accesses. In particular, the profiler logs the accessed model elements in a database together with the knowledge what consistency rules accessed these model elements. We refer to this mapping between accessed model elements and consistency rules as a scope. The rule detector instructs the consistency checker on what rule to evaluate, when a model element changes. It determines this by observing model changes and looking up what rules previously accessed the changed model elements (scope).

The Model/Analyzer approach fully automatically, correctly, and efficiently decides which subset of consistency rules has to be evaluated when the model changes. The approach was also demonstrated to keep up with an engineer’s rate of model changes (even on very large industrial models).

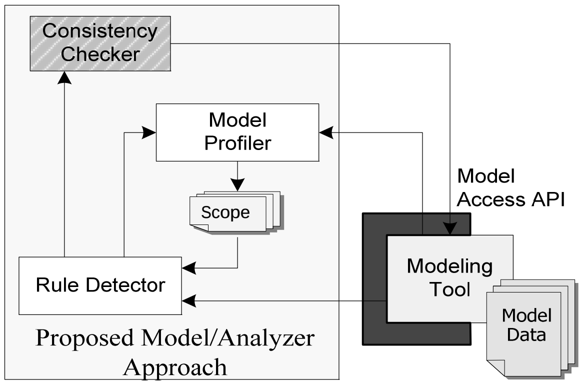

Figure 4. Architecture of Model/Analyzer

Even though our approach (or any approach) is not guaranteed to be instant for every consistency rule, we were able to show that it is easily able to keep up with an engineer’s rate of model changes for several software and systems modeling languages including the UML, a subset of Matlab/Stateflow, and a domain specific steel manufacturing product line model. Our empirical data showed, for example on the UML models, that the evaluation of a model after changes averaged to less than 1.4 ms - even on the largest industrial models we had available. This benefit comes at the expense of a linearly increasing memory cost to store the observed behavior of consistency rules.

3. MECHATRONIC DESIGN MODELS

3.1 Product Models

A model is the conceptual description of ideas, facts and processes that together represent the model of a design product [7]. A design object may be an assembly, a sub-assembly or a single part represented by the according product structure model. Mechatronic models are very important tools for complex activities such as engineering design [8, 9]. For high performance of the engineering of design tasks, mathematical modeling and simulation, i.e. experimenting with computer-based models, are increasingly important problem solving techniques.

In general, a model is devoted to the task of mapping reality onto a significant representation of reality in order to make valid predictions and conclusions for reality. It should include the relevant phenomena/effects of interest (“views of the object”, [10]) such as geometry, dynamics, stability, materials, electrodynamics, controllability, cycle time, maintenance, etc.

During all phases of the design process there is a need to establish models. If these models are simplified representations of the product to be built then we speak of product models. The purpose of these models changes depending on the phases of the product development. During the conceptual design phase, physical principles, functions, structures, etc have to be modeled and evaluated. In most cases, analytical (mathematical) and virtual models are less expensive and less time-consuming than physical prototypes. Virtual models can be implemented and used to simulate and evaluate (significant representations of) reality with the help of computers.

A principal role of engineers in each stage of product development is to make decisions [11, 12]. However, if the engineers have no overall overview of the multidisciplinary system under consideration, consultations with experts of the other fields are necessary. Decisions help to bridge the gap between an idea and reality. In general, decisions are controlled by information from many sources (and disciplines). As a rule, we have to accept that in most cases dealing with design not all of the information is available required to arrive at a fully correct decision. Some of the information may be “hard”, that is, based on well founded scientific principles and some information may be “soft”, that is, based on the designer's personal judgment, experience and instinct. Design is the process of converting information that characterizes the needs and requirements of a system into knowledge about the system itself [12]. It is clear that the decisions influence the number of iterations in the design process. The information flow in product development is controlled by the decision-makers in the organization of the product development process.

3.2 Hierarchical Mechatronic Conceptual Design Models

From the engineering design process viewpoint, models are containers of knowledge gained in the project, and simulations are activities producing information that may improve product knowledge and potentially also the quality of many analyses and decisions made in the design process [13]. The aim of a behavioral model, i.e. a model of the (physical) behavior of a system, is to serve as a tool to find an answer to a design question, i.e. models are unique and have a specific purpose. The modeling challenge may be addressed with a modular or an integrative model design. A modular subsystem has interfaces that are well defined and shared with only a few other subsystems. An integrative system has interfaces which may be more complex and are shared mainly inside the subsystem.

Design models are used for the evaluation of different solutions during the design process. At least for this step, quantitative models are indispensable, their parameters being the decisive elements providing an adequate quantification of design models and representing the properties of the original under consideration. The number of parameters increases from conceptual over preliminary to detailed design. According to the increasing degree of detailing during the design process, the granularity of the describing models becomes finer and finer, leading to a hierarchy of models as well as their describing parameters. For modeling and evaluation of solutions during all phases of the design process, we postulate models with different degrees of detailing (granularity of models) in correspondence with their describing parameters. The correspondence between models and parameters implies that the meaning of a parameter is well defined only via its related model.

Hierarchical models are very important tools for complex activities such as engineering design. Especially during the conceptual design phase there exists a high demand for models to describe the design concept with respect to the given requirements. The following section demonstrates the above experiences from an industrial case study.

We may define model conformance as the property, how well a model implements its intended objectives. Typical model conformance characteristics are its validity area, accuracy, speed and flexibility. In the modeling process, these characteristics must be judged in context of the model purpose. Non-routine simulations, which have a tendency to be more explorative than routine simulations, should be facilitated by flexible models, i.e. models which can be easily configured and adapted to slightly different purposes. Hence, a new approach in Life Cycle Design and Engineering of mechatronic products can be considered as a key point to optimize the design process (see Figure 5).

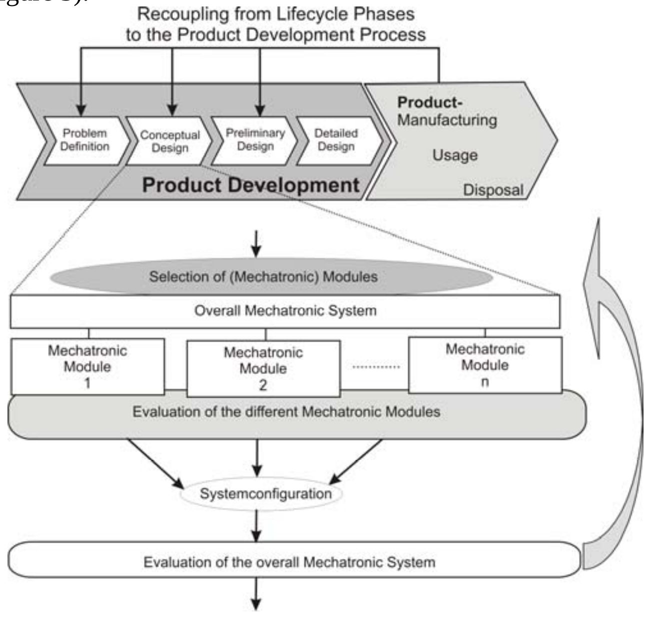
Recoupling from Lifecycle Phases to the Product Development Process
Problem Definition
Conceptual Design
Preliminary Design
Detailed Design
Product Development
Product-Manufacturing
Usage
Disposal
Selection of (Mechatronic) Modules
Overall Mechatronic System
Mechatronic Module 1
Mechatronic Module 2
Mechatronic Module n
Evaluation of the different Mechatronic Modules
Systemconfiguration
Evaluation of the overall Mechatronic System

Figure 5. Conceptual Design in Product Development

3.3 Definition of a Mechatronic Module

According to our definition [13, 14], a mechatronic module utilizes several (at least more than one) different domains (disciplines) of mechatronics (e.g. mechanics, automatic control etc.) merging the respective domain-specific components. This means that a mechatronic module can only be decomposed into domain-specific (non-mechatronic) components, but not into other mechatronic modules or mechatronic system components. A mechatronic module therefore designates the “smallest” indivisible mechatronic subsystem within the set of mechatronic sub-systems.

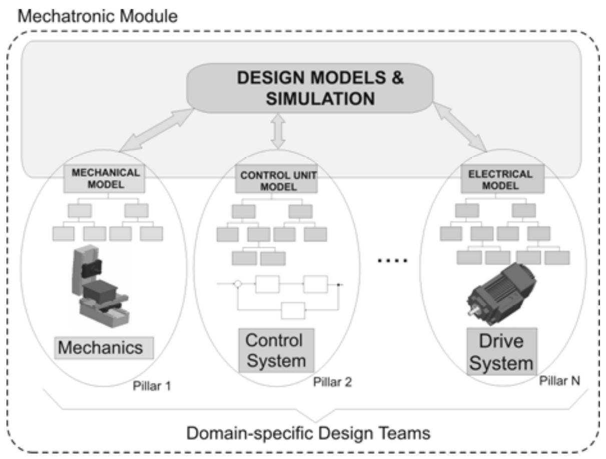

Figure 6. Mechatronic Module

Each pillar in Figure 6 characterizes the model of a domain-specific component, which is structured into several hierarchical levels corresponding to the proceeding degree of detailing. For example, the pillar “Control System” includes information about the structure of the controlled system as well as the control laws and their implementation. Only the first (highest) level has an interface to the other pillars (compare to object oriented programming) via the mechatronic coupling level. All couplings between the model pillars (e.g. design parameters and requirement parameters, which affect multiple disciplines) are captured and described at the mechatronic coupling level. The model structure has to be adapted if additional couplings between domain-specific components are detected during a design iteration (design, analysis, integration, performance check etc.). This is also true if new or additional domains (pillars in the model) come into consideration. If we consider the interaction between the different disciplines and models designed by different engineering teams, consistency checking is necessary to avoid conflicts in models, interpretation of parameters etc.

4. APPLICATION

4.1 Overall Mechatronic System

In Figure 7 the example of a hierarchical decomposition of an overall mechatronic system (mechatronic product) is sketched. The hierarchical structuring allows to recognize and to describe internal interactions and also the integration of all systems involved throughout the mechatronic coupling levels. In this way, all relevant interactions, interdependencies and interfaces are made transparent.

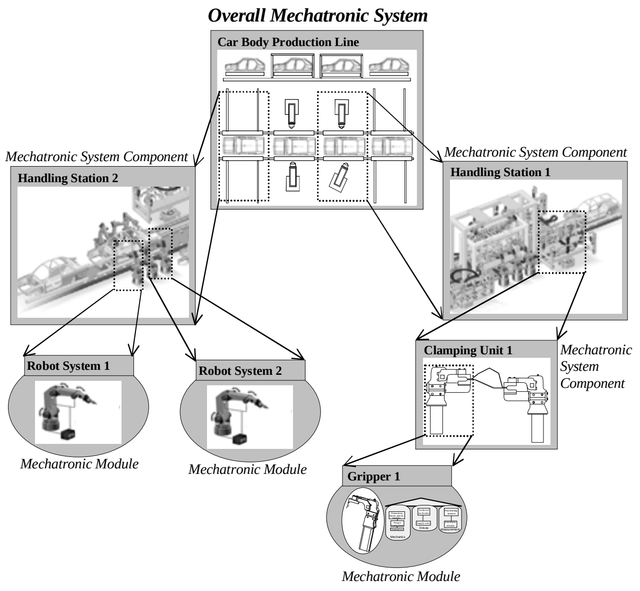

Figure 7. Hierarchical Structure of a complex Mechatronic System

4.2 Manipulator with Gripper Unit

Typical applications for robots are pick and place operations, screw driving, welding etc. A robot may consist of two main parts, a positioning system, which locates the functional part, sometimes called "end effector". A manipulator with prismatic joints is known as a cartesian manipulator, for which the joint variables are the cartesian coordinates of the end-effector with respect to the base. As might be expected, the kinematic description of this manipulator is the simplest of all manipulators. Cartesian manipulators are useful for table-top assembly applications and for transfer of material. The system can be decomposed into several sub-modules, which are to be designed simultaneously. The main mechatronic modules are

- a Main Drive for the movement from the start to the end position
- Robot Arms for the exact positioning of the parts (positioning unit)
- Grippers to establish the connection between the robot arms and the parts

Figure 8 represents the hierarchical model of one possible design concept. In the following, different solution concepts for the end-effector (gripper) are considered.

4.3 Mechatronic Ontology

When most of the knowledge has been acquired, it is still unstructured and needs to be organized and structured by using representations that both computers and humans can understand. Such representations are named "knowledge worksheets" [15, 16]. For the mechatronic ontology of the example above, six so-called "concepts", namely "mechatronic device", "environment", "material", "property", "function" and "manufacturing process" are defined. In general, these "concepts" are connected among one another through relationships.

For instance, a mechatronic device concept (e.g., Gripper) is related to the environment concept (e.g. connection to robot). The name of the relation is "interaction-with". The used definitions of the relationships are "has-part", "interaction-with", "has-material", "has-function", "has-process". The concepts and relationships among them describe the relevant knowledge of a new design concept.

When designing a mechatronic system, an experienced design engineer implicitly applies his aggregated knowledge to the new concept. Without being directly aware of, the design engineer considers design rules, functional requirements, economic and legal restrictions. Existing computer aided design tools (e.g. CAD systems) give only little support to this creative conceptual design stage. Whereas computer support and product models are well established for the process of embodiment design, the early phases of product development, such as conceptual design, even nowadays often are carried out isolatedly from the subsequent phases. Based on the presented method described in the previous section, a first prototype for such a computer aided semantic design tool for mechatronic systems was created by using the software tool OntoStudio™.

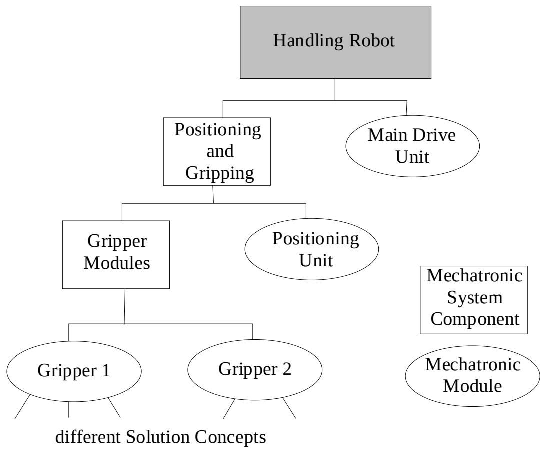

Figure 8. Hierarchical Model of a Handling Robot

Figure 9 shows the concept of mechatronic ontology by means of the above example. The arrangement and configuration of fixing devices used in various machines or stations of a production line for car bodies is an open question from the automotive engineering domain. Clamping devices are dedicated to fix different sheet metal parts quickly to one another at an exact position and with a pre-defined clamping force. After the parts are fixed, they are joined together e.g. by welding. The main design requirements for such fixing units are e.g. precision, closing speed, compact size, reliability, etc.

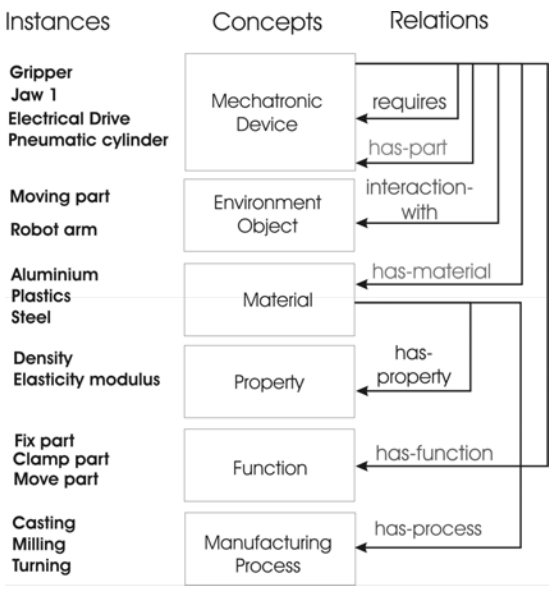

Figure 9. Mechatronic Ontology of the Gripper

4.4 Model Consistency

It is necessary to define rules for checking the consistency of the models. This is done with the help of the Model/Analyzer approach discussed in 2.2. Some examples are:

```
Consistency rule 1a: applies to <electrical drive>
    requires(this).contains(<power electronics>)
Consistency rule 1b: applies to <pneumatic cylinder>
    requires(this).contains(<air supply>)
Consistency rule 2: applies to <gripper>
    has-part(this).exists(part | part.name="electrical drive") xor
    has-part(this).exists(part | part.name="pneumatic
    cylinder")
```

For example, consistency rule 1a defines that an electrical drive requires a power electronics. The rule applies to any such electrical drive as there could be many. Hence, the rule applies to all instances (occurrences) of electrical drive. The "this" keyword refers to the given electrical drive and "requires(this)" refers to the ontological relation of this electrical drive as was discussed in Figure 9. The rule ensures then that one such required element is a power electronics by checking whether a power electronic is contained in the list of required elements. Rule 1b is largely analogous to Rule 1a and ensures that a pneumatic cylinder requires an air supply. Rule 2 is more complex. It makes use of quantifiers, such as "exists". The rule ensures that for a given "gripper" there exists one "has-part" relation (see Figure 9) whose name either equals "electrical drive" or "pneumatic cylinder" (with an exclusive or in between). If both exist or none exists then the condition returns false (i.e., an inconsistency).

The syntax of these rules closely follows predicate logic because a consistency rule is either satisfied or it is not (Boolean result). Many languages for describing consistency rules exist in the computer science domain (e.g [17]). These languages are often quite similar. The consistency rules can be written manually or, in certain cases, could be derived automatically (i.e., as in this case where the consistency rules are derivable from the above system). In large models, with tens of thousands of model elements, it is hard to detect new inconsistencies while the model changes and it is also hard to keep track of known inconsistencies. Our approach identifies inconsistencies instantly with design changes and it keeps track of all inconsistencies over time. It does not require consistency rules with special annotations. Instead, it treats consistency rules as black-box entities and observes their behavior during their evaluation. We demonstrated empirically [4], that our approach is capable of detecting and tracking inconsistencies quickly for many kinds of consistency rules.

To support the fast, incremental checking of design changes, the tool identifies all model elements that affect the truth value of any given consistency rule. A consistency rule needs to be re-evaluated if and only if one of these model elements changes. For example, if there are multiple "gripper" then the consistency rule 2 must be evaluated for each one of them separately. We thus instantiate consistency rule 2 for as many times as there are "grippers" in the model. Each instantiation would access different parts of the model. When the model changes then only those consistency rule instances are re-evaluated that accessed one of the changed model elements. According to the procedure pointed out in section 2.2, in this application the consistency rules have to be re-evaluated, if the variant electrical drive is modified to a pneumatic cylinder. We distinguish between instance, domain, and meta-domain consistency rules. Consistency rules 1 and 2 are domain rules which apply to specific elements of a certain domain.

```
Consistency rule 3: applies to an instance of <Mechatronic
Device>
    if (has-part(this) is empty) then not(has-material(this) is
    empty)
    has-part(<Mechatronic Device>) = forall (device:
    <Mechatronic Device> |
    device.outgoingDependencies("has-part")->client)
```

Consistency rule 3 applies to the meta-domain. For example, we define that mechanical devices need to be built out of a material, so every mechanical device should have a "has-material" relationship to a Material – or it should be composed out of parts that should have materials. Rather than writing this rule separately for every instance of a Mechatronic Device, we defined it once for any Mechatronic Device. This rule should then be instantiated – once for every instance of Mechatronic Device: gripper, jaw, electrical drive and power electronics. In the example in Figure 9, an inconsistency would be detected because, for example, electrical drive does not have a material, nor is it composed of parts that have materials.

The more generic a consistency rule (i.e., the more generally applicable), the more likely is a model error the cause for an inconsistency. Consistency rules 1-2, however, are less generic (domain and/or applicable dependent). Here, inconsistencies could also be the result of incorrect consistency rules. This assessment, on whether to trust the model or the rule is left to the engineer. Our approach does support both model and rule changes. Furthermore, the granularity of modeling information does affect the writing of consistency rules. It appears to be generally easier to write consistency rules for elements at the same level of granularity (level of abstraction). This aspect will be explored further in future work.

5. CONCLUSION AND OUTLOOK

In this paper, we discussed how to ensure consistency during the modeling of multi-disciplinary, mechatronical systems. We propose an approach for a mechatronic ontology, which allows the design engineer to describe different design concepts and their structure. During all phases of the design process there is a need to establish models which may be seen as simplified representations of the real world. However, since design models can be large and complex and since their creation and maintenance is often distributed among multiple designers and even disciplines, it is hard to ensure the correctness of such design models. There is the need for automatically checking the consistency of such distributed design models and this paper demonstrated how one such approach to consistency checking, developed in the software engineering domain, could be used to address the larger challenge of consistency checking among mechatronical design models.

ACKNOWLEDGMENT

We gratefully acknowledge that this work was supported in part by the Austrian Center of Competence in Mechatronics (ACCM), a K2-Center of the COMET/K2 program, which is aided by funds of the Austrian Republic and the Provincial Government of Upper Austria and in part by the Austrian FWF grant P21321-N15.

REFERENCES

[1] Tomizuka, M., 2000. “Mechatronics: From the 20th to 21st century”, In Proceedings of 1. IFAC Conference on Mechatronic Systems, Darmstadt, Germany.

[2] De Silva, C.W., 2005. *Mechatronics – an integrated approach*, CRC Press Boca Raton, London, New York, Washington DC.

[3] Isermann, R., 2005. *Mechatronic Systems. Fundamentals*, Springer Publishing Group, Berlin Heidelberg, Germany.

[4] Egyed, A., 2010. “Automatically Detecting and Tracking Inconsistencies in Software Design Models, *IEEE Transactions on Software Engineering (TSE)*, to appear

[5] Egyed, A., 2007. “UML/Analyzer: A Tool for the Instant Consistency Checking of UML Models,” Tool Demonstration, In Proceedings of the 29th International Conference on Software Engineering (ICSE), Minneapolis, MN, May 2007, pp. 793-796 (12 papers accepted out of 56 submitted).

[6] Balzer, R., 1991. “Tolerating Inconsistency”, In Proceedings of 13th International Conference on Software Engineering (ICSE), 1991, pp. 158-165.

[7] Avgoustinov, N., 2007. *Modelling in mechanical engineering and mechatronics*, Springer Publishing Group, London, UK.

[8] Pahl, G. and Beitz, W., 1999. *Engineering design – a systematic approach*. Springer Publishing Group, London, UK.

[9] Bishop, R.H., 2007. *Mechatronic Fundamentals and Modelling*, The Mechatronics Handbook. CRC Press Boca Raton, London, New York, Washington DC.

[10] Vajna, S., Weber, C., Bley, H., Zeman, K. and Hehenberger, P., 2009. *CAx für Ingenieure: Eine praxisbezogene Einführung*, Springer Publishing Group, Berlin Heidelberg, Germany.

[11] Herrmann, J.W. and Schmidt, L.C., 2002. “Viewing Product Development as a Decision Productions System“, In Proceedings of the ASME 2002 International Design Engineering Technical Conferences, Montreal, Canada.

[12] Wassenaar, H.J. and Wei, C., 2002. “An Approach to Decision-Based Design with Discrete Choice Analysis for Demand Modeling”, *Journal of Mechanical Design*, Vol. 125, pp. 490-497.

[13] Hehenberger, P. and Zeman, K., 2007. “Design Activities in the Development Process of Mechatronic Systems”, In Proceedings of AIM 2007, IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Zürich, Switzerland.

[14] Hehenberger, P. and Zeman, K., 2008. “The role of hierarchical design models in the mechatronic product development process”, In: Proceedings of TMCE International Symposium Series on Tools and Methods of Competitive Engineering, Izmir, Turkey.

[15] Li, Z., Raskin, V. and Ramani, K., 2007. “A Methodology of Engineering Ontology Development for Information Retrieval“, In Proceedings of International Conference on Engineering Design, Paris, France.

[16] Ahmed, S., Kim, S. and Wallace, K., 2005. “A Methodology for creating Ontologies for Engineering Design”, In Proceedings of IDETC/CIE 2005, ASME 2005 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, Long Beach, California USA.

[17] Warmer, J., Kleppe, A., 2003. *The Object Constraint Language: Getting Your Models Ready for MDA*. 2nd Edition, Addison-Wesley Professional,