Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2011
August 29-31, 2011, Washington, DC, USA

DETC2011-47924

A CONCEPTUAL FRAMEWORK FOR CONSISTENCY MANAGEMENT IN MODEL-BASED SYSTEMS ENGINEERING

Sebastian J. I. Herzig
Model-Based Systems Engineering Center
G. W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia, United States of America

Ahsan Qamar
Department of Machine Design
School of Industrial Engineering & Management
KTH Royal Institute of Technology
Stockholm, Sweden

Axel Reichwein
Model-Based Systems Engineering Center
G. W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia, United States of America

Christiaan J. J. Paredis
Model-Based Systems Engineering Center
G. W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia, United States of America

ABSTRACT

Developing complex engineering systems requires the consolidation of models from a variety of domains such as economics, mechanics and software engineering. These models are typically created using differing formalisms and by stakeholders that have varying views on the same problem statement. The challenging question is: what is needed to make sure that all of these different models remain consistent during the design process? A review of the related literature reveals that this is still an open challenge and has not yet been investigated at a fundamental level within the context of Model-Based Systems Engineering (MBSE). Therefore, this paper specifically focuses on examining the fundamentals of consistency management. We show that some inconsistencies cannot be detected and come to the conclusion that it is impossible to say whether or not a system is fully consistent. In this paper, we first introduce a mathematical foundation to define consistency in a formal manner. A decision-based approach to design is then studied and applied to the development of a real-world example. The research reveals several distinct types of inconsistencies that can occur during the design and development of a system. We show that these inconsistencies can be further classified into two groups: internal and external consistency. From these insights, the ontology of inconsistencies is constructed. Finally, requirements for possible tool support and methods to identify and manage specific types of consistency issues are proposed.

1 INTRODUCTION

Engineering systems have become increasingly complex due to a variety of stakeholders from different disciplines being involved in the design process. These stakeholders have different interests: some may only be concerned with structural aspects, while others are more interested in the behavior, cost or requirements. Such aspects of a system are expressed using differing formalisms. Structural aspects, for instance, are commonly represented as three-dimensional Computer Aided Design (CAD) models in tools such as SolidEdge [1], dynamics are represented in Modelica [2] and requirements in Rational DOORS [3].

Managing the large amount of information that is typically part of a specification is by no means a trivial task. Systems engineering is a field of engineering that focuses on this topic. It is a multidisciplinary approach towards developing and realizing balanced system solutions in response to diverse stakeholder needs and includes the application of both management and technical processes [4]. These include planning the technical effort, monitoring technical performance, managing risk, and specifying, designing and verifying the system to be built. Traditionally, a variety of documents that make up the specification are created. One of the major challenges is to ensure that this system specification does not contain any contradicting information. For example: if one part of the system specification reflects the use of English units, no other dependent part should be based on the use of the

International System of Units (SI units). Such contradictions can occur when the relations within and between documents are not clearly defined. They are the result of a lack of formal integration of the information. If no such contradictions are present, the specification of the system is said to be *consistent*.

The idea of only using computer-interpretable models to specify a system and thereby introducing a higher degree of formalism reflects the key principles behind *Model-Based Systems Engineering* (MBSE) as defined by the *International Council on Systems Engineering* (INCOSE) [4-5]. Using a repository to store these models enables relations to be drawn between these models more explicitly and formally. Theoretically, this increases traceability of model-related information across the system specification [4]. Ensuring that these models are consistent increases the chance of mission success, thereby decreasing risk. However, ensuring that all models within the repository are consistent, or *consistency management*, is still an unsolved and challenging problem.

To date the results of research within the field of consistency management mainly include ad-hoc solutions for detecting specific types of inconsistencies between specific types of models (e.g. [6-10]). Some work from the domain of software engineering focuses on developing methods that have a formal, mathematical foundation [11-17]. To the best of knowledge of the authors, consistency management in MBSE has, to the date of writing this paper, not yet been studied at a fundamental level. The primary scientific contribution of this paper is to provide a definition and ontology for consistency related to using models for the process of designing and developing systems. Additionally, a guideline (conceptual framework) on how and to what extent inconsistencies can be detected and managed is outlined. This topic of consistency management is discussed in sections 4 to 6. In section 3, the model-based design of a robot is discussed to illustrate the different types of inconsistencies that are identified in section 2. At the end of this paper, more specifically in section 7, the presented work is related to the status quo of consistency management. After a short summary of the most important aspects of the presented work, the conclusions that can be drawn from the research are outlined in section 8.

2 FUNDAMENTALS OF CONSISTENCY

Consistency implies an absence of contradictions. But what exactly are these contradictions? And how do they come about in the first place? To help answer these questions, it is imperative to study the theory behind constructing models and how these models can be integrated into a specification for a system in a formal manner. Understanding these fundamentals enables us to not only identify the different kinds of inconsistencies, but also to manage them.

2.1 Mathematical Foundation

Before looking more closely at consistency management in the context of MBSE, it is important to understand the underlying mathematical fundamentals. Similar to program code corresponding to a programming language, models are created using modeling languages. Such languages are part of *formal systems*.

A formal system consists of a formal language $\boldsymbol{L}$ and a set of *inference rules* $\boldsymbol{R}$ which are used to derive expressions from one or more premises that may either be *axioms*[1] ψ_i or previously derived statements, so called *theorems* [18-19]. A formal language is defined as a set containing words[2] $\boldsymbol{w_i}$. These words are assembled using *terminal symbols* from an *alphabet* $\boldsymbol{\Sigma}$. A grammar $\boldsymbol{G}$ uses such an alphabet, the mentioned inference rules and the axioms to define how words are constructed. Additionally, a finite set of *nonterminal symbols* $\boldsymbol{N}$ (disjoint from $\boldsymbol{\Sigma}$) and a distinguished start symbol $\boldsymbol{S} \in \boldsymbol{N}$ are required. Nonterminal symbols are used to specify the production rules of a grammar. To generate words one begins with a word consisting only of a special nonterminal symbol: the start symbol. The start symbol therefore acts as an entry point for the production of a word. In summary, a grammar is defined as a 4-tupel $\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{\Sigma}, \boldsymbol{R}, \boldsymbol{S})$ [18]. Elements within the set of all words that can be produced by a given grammar are considered to be *well-formed* and define the language. All axioms are part of this set and are therefore also well-formed words.

To verify whether or not a given statement is compatible with a specific formal system, it must be a theorem of it, i.e., there must be a way to derive the word using a *formal proof*. Formal proofs are sequences of well-formed words and are carried out by sequentially applying inference rules to previous well-formed words in the proof sequence. Axioms act as a starting point for these sequences [18].

Grammatical rules reflect two important properties of languages: syntax and semantics. The set of all syntactically correct words is the set of well-formed words. A formal system is consistent if all statements are theorems and therefore follow a valid syntax. The semantics of a language give the utterances of a language a specific meaning.

To elaborate on the concepts presented so far, an example formal system (borrowed from [20]) will be constructed in the following. This formal system is called the MIU-system. It is based on one axiom, five production rules and an alphabet with three terminal symbols:

$$
\begin{aligned}
\Psi_1 &= \text{MI} \\
\boldsymbol{\Sigma_{MIU}} &= \{\ \text{M, I, U}\ \} \\
\boldsymbol{N_{MIU}} &= \{\ \boldsymbol{S_{MIU}}, \text{x, y}\ \} \\
\boldsymbol{R_{MIU}} &= \{\ \text{“MxIy is a theorem”}, \\
&\quad\ \text{“If xI is a theorem then so is xIM”}, \\
&\quad\ \text{“If Mx is a theorem then so is Mxx”}, \\
&\quad\ \text{“If MxIIIy is a theorem then so is MxUy”}, \\
&\quad\ \text{“If MxUUy is a theorem then so is Mxy”}\}
\end{aligned}
$$

The above terminals, nonterminals and inference rules are then used to construct the grammar $\boldsymbol{G_{MIU}} = (\boldsymbol{N_{MIU}}, \boldsymbol{\Sigma_{MIU}}, \boldsymbol{R_{MIU}}, \boldsymbol{S_{MIU}})$. $\boldsymbol{S_{MIU}}$ is used as an entry point to build words, i.e. it

[1] Axioms are propositions that are not proved but are considered to be self-evident [18]

[2] In the related literature, words are sometimes also referred to as strings, formulas, statements, sentences or algorithms

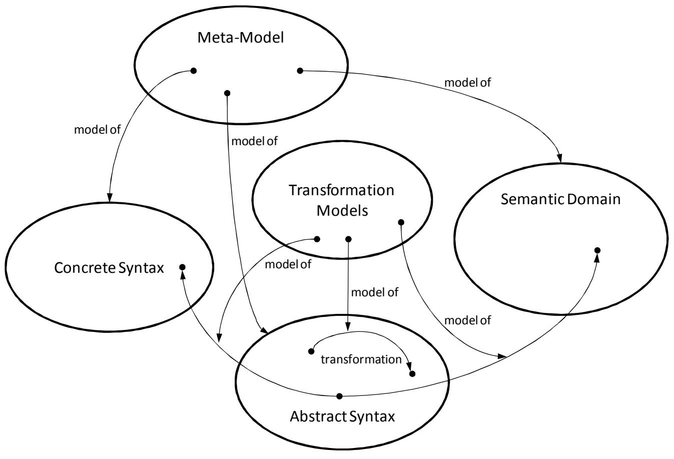

Figure 1: The structure of a modeling language (adapted from [21])

produces at least the axiom Ψ_1. By sequentially applying the inference rules in $\boldsymbol{R}_{MIU}$ one can construct the set of all words. The resulting language is then $\boldsymbol{L}_{MIU}$ = {MI, MII, MIU, MIUIU, MIIII, MIIIIU, …}. Using an appropriate algorithm, one can now decide whether the word MU, for instance, is part of the language and therefore a theorem.

Another, more complex example of a formal system, is a logical system, in which truth values are assigned to words. Other than the example system above, the logic system also includes a form of semantics that gives well-formed words a specific interpretable meaning.

2.2 Modeling Languages

So far we have only discussed languages that produce words or formulas consisting of symbols – but how does this relate to modeling languages? Analogous to the definition of languages in section 2.1, modeling languages have a formal grammar. The grammar is represented by a so-called meta-model (see e.g. [22]). A meta-model is a model itself and specifies the way utterances of the modeling language (i.e. models) may be constructed. Well-formed models are the equivalent to statements that can be expressed using the syntax of a formal modeling language. The syntax itself can have an abstract or a concrete form. Concrete syntax is used for easier user manipulation of the model. It contains non-essential information such as the graphical representation of the model on diagrams to enable a designer to manipulate a model more intuitively. The abstract syntax, on the other hand, contains only the essential information for the model to be interpreted by a digital computer. The transformation between the abstract and concrete syntax is done using a *model-transformation*. This model transformation defines how the meta-models of the abstract and the concrete syntax relate to each other. Well-formed models may also be given a semantic meaning. This is done by mapping these to a semantic domain, which is again done through the use of model transformations. Refer to Figure 1 for more details. New models that stakeholders wish to integrate into a system can be checked for consistency by using such a language definition. As previously, this is achieved through a formal proof, i.e. by checking whether the models are theorems of the given modeling language.

Rules of modeling languages must satisfy logical constraints and reflect the proper use of language constructs. Inheritance relationships, for example, should not be used in a looping fashion and must therefore possess transitive qualities: if *Apple* is a specialization of *Fruit*, *Fruit* cannot be a specialization of *Apple* at the same time. Such a circular inheritance relationship would lead to a logical contradiction. While most of these rules can be expressed using the meta-model alone, it is not always possible or practical to do so. Constraints may also be expressed using formal description logic.

Inconsistencies related to the construction of models result from incomplete models or models that are constructed using informally defined modeling languages. Not following a given formal grammar can lead to logical contradictions within the model. Hence we refer to this class of consistency problems as *logical inconsistencies*.

2.3 Types of Inconsistencies Related To Models

When using a formal modeling language to construct models, the question whether or not a given model is consistent with the formal system becomes a decidable problem through syntactical verification. It is therefore a logical inconsistency problem. However, whether or not a model is *correct*, i.e. whether it makes *sense* and therefore has *value* is not implied by the syntax alone. In order to give models a meaning, a semantic mapping from the model to a *reality* has to exist. It is only by using such a mapping that one can decide whether the model is compatible and therefore consistent with the given reality. For software, this reality takes on the form of the execution environment, which is typically based on a (virtually deterministic) logic system. A compiler performs a semantic mapping from the input (such as a model or a program) to the logic of the digital machine. The result is an isomorphism, that is, at least in theory, a bidirectional mapping between the software models and the execution environment.

Physical systems, on the other hand, are not bound to a reality in which the processes are well understood and relatively deterministic. The real world, or more precisely, reality as we conceive it, goes far beyond that. In order for us to say that a specification is truly consistent requires a corresponding formal system to contain sufficient information to form an isomorphism with nature. However, it is impossible to construct such a formal system for physical systems. The main reason is that we lack perfect knowledge about the processes and the phenomena in nature. For example, current production techniques do not allow for perfect precision in manufacturing. Models are used to approximate processes such as these, i.e. they are abstractions of reality and are therefore connected to *uncertainty*. The logical conclusion is that uncertainties must be taken into account when modeling a system. To accomplish this, one must include concepts from mathematics that go beyond logic systems. To consider the laws of probability, for instance, the model must adhere and be

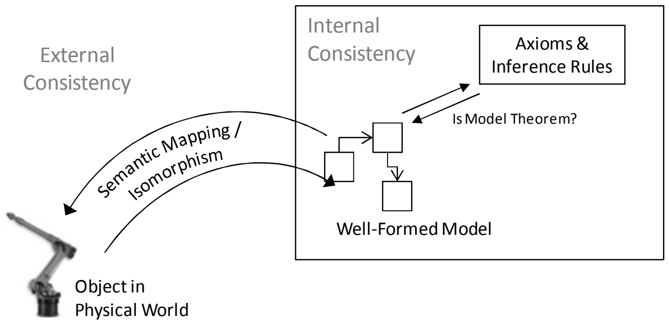

Figure 2: Internal and external consistency

compatible with Kolmogorov's axioms of probability. Failure to do so leads to an inconsistency with the laws of mathematics. We therefore call this class of consistency problems *mathematical inconsistencies*.

Models are, as we have already established, abstractions of reality. More precisely, models are, in fact, abstractions of our *beliefs*. Beliefs are reinforced by scientific data, in other words: by our observations of nature. These beliefs must be *rational* and consistent with the given scientific data. For this scientific data to be useful in the sense that one can derive adequate models of nature, the scientific data itself must be consistent. However, this is, as we have already mentioned, impossible since we lack perfect knowledge about nature. It is practical to differentiate the resulting class of *belief inconsistencies* from those related to mathematical inconsistencies and from those related to the *consistency with respect to nature*.

We conclude this section by introducing the notion of *internal* and *external consistency* (see Figure 2). Internal consistency problems relate to axiomatic systems that are well understood (e.g. logic systems and mathematics). Based on these systems we construct modeling languages. Models that are internally consistent do not violate the axioms and rules of the underlying formal system – they are theorems of the system. External consistency imposes an additional constraint, namely, that the model be true to reality. Grouping consistency issues into these two groups allows us to differentiate between inconsistencies that can occur within the bounds of well-understood systems and those that are not.

To elaborate on this notion of internal and external consistency, let us consider a simple example: a fair coin toss. If we were to assume that the probability of heads is 0.5 and that of tails is 0.6, we are in contradiction with Kolmogorov's axioms, since the combined probability is greater than 1. Our model is therefore internally inconsistent with the laws of mathematics. If we assume, on the other hand, that the probability of heads is 0.3 and that of tails 0.7, the model is internally consistent but externally inconsistent. This is due to our assumption of us observing a fair coin toss - the experiment is not compatible with the observations we make in the physical world (given that we are not using any controlled conditions). This example also shows the difference between designing a software system and a system with physical character: it is possible to write a computer program that determines the outcomes of experiments using any possible set of probabilities. The only rules that the program must adhere to are the laws of probability (i.e. Kolmogorov's axioms).

2.4 Forming Consistent Concepts in Decision-Based Engineering Design

Systems engineering can be thought of as a decision making process [23]. The decisions can be represented in the form of models, but these models need to reflect certain qualities. For one, models need to satisfy rationality constraints. Models also need to satisfy constraints on the expression of beliefs and preferences of the designer to make sure these expressions match observations of nature [24].

Over the past decade, many new approaches to engineering design have been proposed. Prominent examples include Design for Six Sigma [25], Taguchi's theory of robust design [26], concurrent engineering and design for manufacture [27]. Most of these methods are ad-hoc approaches that are not rooted in any fundamental theory, nor do they provide a basis for engineering design as a discipline.

Recently, design has come to be thought of as a decision-making process [23, 28]. A design theory that builds up on this foundation is Rational Design Theory (RDT) [24, 28]. The key idea is that a designer should act rationally, i.e., in a fashion that is consistent with his beliefs and preferences. The theory builds on probability theory and von Neumann-Morgenstern expected utility theory [29-30]. As part as their work on RDT, Thompson and Paredis introduce a conceptual model for design in which design concepts, concept predictions and decision criteria are defined. This conceptual model will be summarized here. For further details, refer to [24].

In the conceptual model for design, both concept specifications and concept predictions are defined in terms of properties. As illustrated in Figure 3, a property is a mapping from the set of all possible artifacts[3], ***S'***, to a range, $\boldsymbol{Y_i}$. There are potentially an infinite number of properties that could describe a system.

The Cartesian product of all property ranges is called the *property space* ***P***. When specifying a system, developers select a finite number of properties that are considered relevant and are therefore included in the specification. This finite dimensional space is referred to as the *property projection* ***P'***. In the context of robot design, this space could include properties such as "Cost", "Degrees of Freedom" or "Motor Torque". Other properties such as "Color" or "Number of 10N Thrusters" may not be relevant in the robot context and are therefore left completely open. This does not imply, however, that these properties may not become relevant at later stages of the design process, and can then be added as additional dimensions to the property projection.

[3] An artifact is anything produced through human intelligence or effort, including all human-produced physical objects and processes [24].

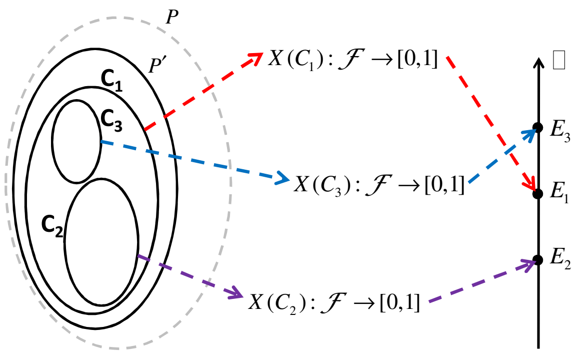

Figure 3: Principle of the concept selection criteria [24]

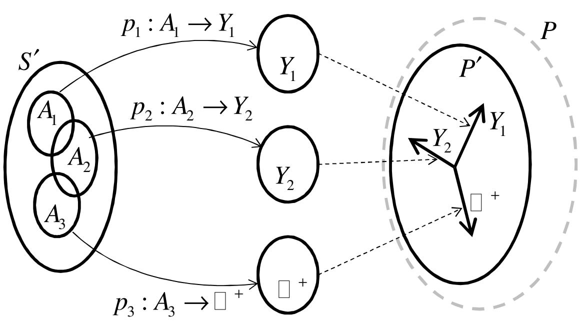

Figure 4: The property space and the mapping of properties from artifacts to concepts [24]

P' is the basis for defining concepts $\mathbf{C_i}$. A concept is a (partial) specification of an artifact. Typically, a concept is developed through a process of iterative refinement. In each refinement step, additional constraints are imposed, restricting properties to specific values. Refinement often begins with a functional specification, proceeds to behavior specification and ends with a structural specification. This sequence is, however, only a guideline and not a fixed order. For instance, when developing a physical system such as a robot, an initial decision may be that the movement should be restricted to a planar 2D-space. This decision could either result from an initial analysis of the environment in which the robot will perform its function or from a system-level requirement. Assume that this decision leads to a concept $\mathbf{C_1}$ as shown in Figure 3: concept $\mathbf{C_1}$ is the subset of ***P'*** containing only robots that work in a 2D-planar space. Assume that the next decision we make relates to choosing the number of degrees of freedom of the robot. Through analysis it may have been determined that values between two and three are feasible, thereby giving us two new sub-concepts $\mathbf{C_2}$ and $\mathbf{C_3}$, restricting the concepts to two or three degrees of freedom, respectively. Both sub-concepts are subsets of $\mathbf{C_1}$, meaning that both represent robots that move in a planar 2D-space.

Once several alternative concepts have been defined, a decision must be made to determine which one of the alternative sub-concepts to consider for further refinement. According to decision theory one should choose the alternative that leads to the most preferred outcome. The challenge is that the outcomes are uncertain and therefore need to be predicted by the decision maker. A *concept prediction* ***X*** is a mathematical characterization of the designer's beliefs about the properties of the artifact that will be realized when the concept $\mathbf{C_i}$ is used as a specification, i.e., is given to a manufacturer to be produced. Keeping in mind that a concept is only partially specified (i.e., many properties are left completely open), a concept represents a potentially very large set of alternative artifacts that a manufacturer could produce as a result of the interpretation of the specification. This introduces additional uncertainty.

Finally, based on the predictions, a *concept decision criterion* $\boldsymbol{E_i}$ is formulated for every concept $\mathbf{C_i}$. This criterion is a measure of the desirability of a concept and is, mathematically speaking, defined by an evaluation function that maps probability measures onto the real axis, as illustrated in Figure 4. By mapping onto the real axis, a complete (transitive) ordering of the concepts is guaranteed, so that the decision maker avoids internal *preference inconsistencies*. An example of a preference inconsistency would be the expression of intransitive preferences: the situation, for example, where one prefers apples over strawberries, strawberries over oranges and oranges over apples. This would lead to irrational behavior.

Both the concept predictions and concept decision criterion must also satisfy *external consistency* rules, namely, they must be accurate reflections of the beliefs and preferences of the decision maker. By adhering to all the consistency rules, one can be assured to arrive at a rational decision.

According to RDT, a design process should end when no additional design actions lead to an increase of the expected utility (including both the value of the artifact and the cost of the design actions). We call the concept specification that is the ideal end result of the refinement process the *optimal concept* **C*** (see Figure 5). Given that there always exist artifacts that are an improvement of the current state, **C*** is not the empty set. Therefore, if the concept at our current stage of refinement, $\mathbf{C_c}$, contains **C*** as a subset, we call the concept simply *consistent*. Consistency in this sense thus implies that through further refinement, we can still reach the optimal concept **C***. Should $\mathbf{C_c}$ equal **C***, the design process should stop because any further design action will lead to a less preferred outcome. All other cases (intersection, disjunction) lead to either over-specification or to concepts that are inconsistent with the designer's beliefs and preferences. The current concept $\mathbf{C_c}$ is

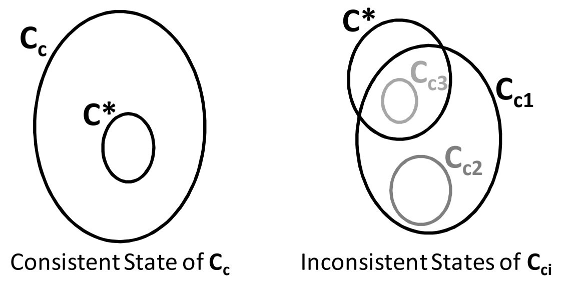

Figure 5: The optimal concept in relation to the current concept

then called inconsistent because **C*** can only be reached by loosening some constraints in its specifications. Note however, that **C*** is unknown (and generally unknowable). Hence, we again arrive at the conclusion that assuring (external) consistency is impossible. Instead, we will have to settle for the lesser goal of eliminating *certain types* of inconsistencies - but not *all* inconsistencies. We will come back to the different types of inconsistencies after introducing an illustrative example to show more clearly at what point in the design process inconsistencies can occur.

3 DESIGN AND DEVELOPMENT OF A ROBOT

In line with the ideas presented so far, this section aims at exemplifying the different kinds of consistency issues that can occur during an actual system design. In the following we attempt to answer the question how models are used to make decisions and hence how a particular concept is selected from a set of alternatives. As already mentioned in section 1, a robot was chosen as an example system. The multidisciplinary domain of mechatronics was chosen as an example domain due to it being a discipline in which models from a variety of domains (mechanics, electronics, control theory, and computer science) are used synergistically within product design and manufacturing to form a specification for a system. The robot was developed using the MBSE methodology outlined in [4], in which a system model is created using the Systems Modeling Language as defined by the Object Management Group (OMG SysML™) [31]. Components were refined using several domain-specific models.

Using OMG SysML™ has the advantage of being able to include multiple views within the same model. Furthermore, the resulting model can contain parts from multiple domains. Other modeling languages that were chosen to further refine the concept and to aid in the decision making process were MATLAB®/Simulink® [32] and Solid Edge [1].

The first step that was taken in the process of designing the robot was to specify an initial set of requirements to reflect the preferences of the stakeholders. To do so, an OMG SysML™ requirements diagram was created. Figure 6 shows an excerpt containing two top-level requirements: the constraint that the workspace is limited to 1 square meter and a functional requirement specifying that the robot should move within a

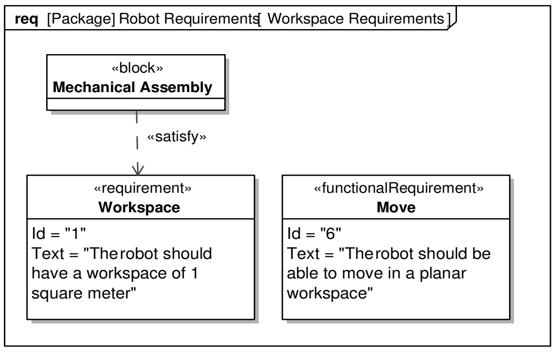
req [Package] Robot Requirements[Workspace Requirements]
«block»
Mechanical Assembly
«satisfy»
«requirement»
Workspace
Id = "1"
Text = "The robot should have a workspace of 1 square meter"
«functionalRequirement»
Move
Id = "6"
Text = "The robot should be able to move in a planar workspace"

Figure 6: An excerpt from the requirements diagram

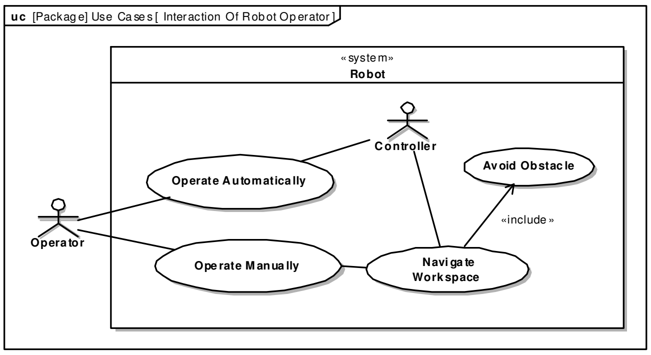

Figure 7: Functional view of the robot

planar space.

The decision to include these two requirements was made based on an informal analysis of the environment in which the robot would eventually be placed and are therefore a result of the robot problem definition. Further requirements are modeled based on the result of a discussion between different domain experts. In line with engineering design principles, the designers then create functional views that are based on the requirements (see Use Case Diagrams in Figure 7).

In order to satisfy these functions, a structure for the robot is chosen. In agreement with design principles by Tjalve [33], an initial discussion with domain experts provided a number of possible basic structures, followed by a selection of two quantified structures, which satisfy the requirements. Within the context of our example, the quantified structures differ in terms of the length of both arms: one uses an equal length for both arms; the other uses a combination of a long and a short arm. Both the alternatives were modeled as a tradeoff study in a weighted decision matrix (not shown here). The basic properties that were considered for this are reliability, manufacturing cost and controllability. Figure 8 shows an excerpt from the block definition diagram showing the basic structure of the mechanical assembly of the robot.

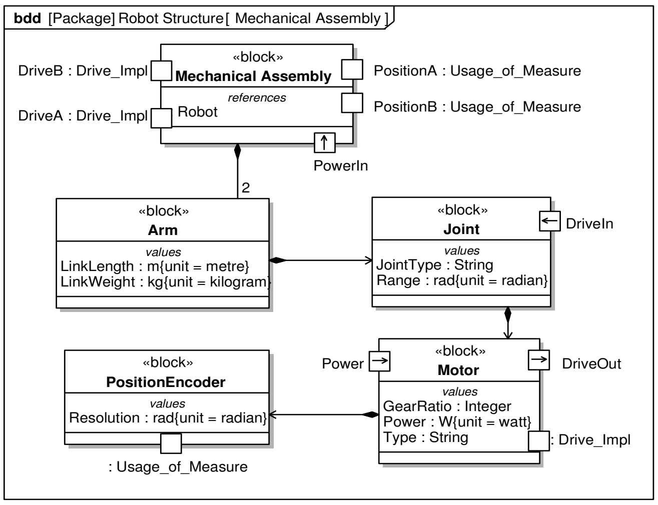

Figure 8: Basic structure of the robot with two arms

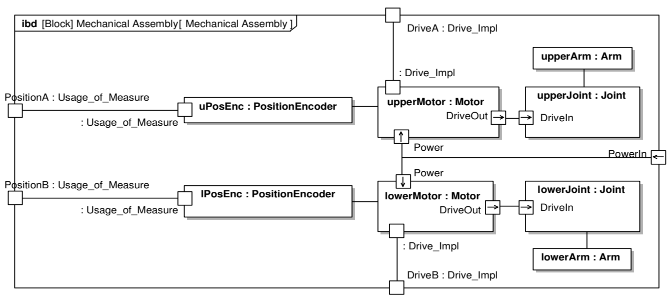

Figure 9: Selected alternative from the possible quantified structures

The reliability of the robot is predicted based on the reliability of each degree of freedom, which is a product of the reliability of the individual components: the arm, joint, motor, and position encoder. The motor and sensor are typically off-the-shelf components, of which reliability figures already exist. In the case of the arm and the joint, which, for the sake of this example, are manufactured, reliability test (stress and strain) can be performed to ultimately select a confidence level above 99%. However, for this simple example, it can be observed that the motor and the sensor are the most critical components per degree of freedom. For example: the failure of one motor not only leads to a malfunction in one degree of freedom, but also to the entire robot becoming unusable for further operations. When specifying characteristics such as the reliability, consistency with the axioms of Kolmogorov must be ensured. If one were to conclude, for example, that the probability of the robot not failing within the first year is 99.999% but the probability of it being unable to fulfill its function within the same time frame is 10% then clearly an inconsistency is present.

During the design activity, the quantified structure consisting of a long and a short arm was disregarded due to an uneven load balance. Hence the principle solution consists of a robot with two arms of equal length, driven by an actuator each. Figure 9 illustrates the chosen quantified structure by means of an internal block diagram.

As stated in the beginning of this section, one of the motivating factors to use OMG SysML™ is that different structural and behavioral aspects (see Figure 10) of the robot system can be expressed using the same modeling language. The multiple views inside the system model of the robot should be consistent with each other. This implies that the

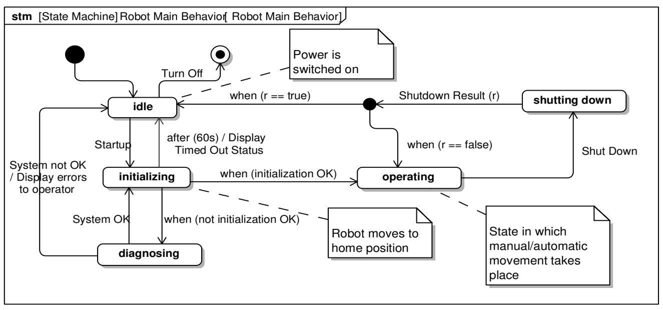
stm [State Machine] Robot Main Behavior[Robot Main Behavior]
Turn Off
Power is switched on
idle
when (r == true)
Shutdown Result (r)
shutting down
Startup
after (60s) / Display Timed Out Status
when (r == false)
Shut Down
System not OK / Display errors to operator
initializing
when (initialization OK)
operating
System OK
when (not initialization OK)
Robot moves to home position
State in which manual/automatic movement takes place
diagnosing

Figure 10: Excerpt from the robot behavior specification

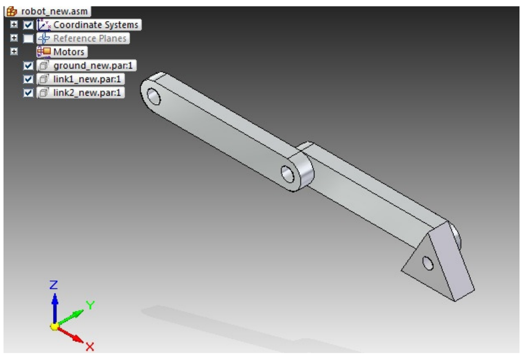

Figure 11: Mechanical view of the robot

requirements should be consistent with the functions, the functions with the structure, the structure with the behavior and the behavior with the requirements.

In order to design the embodiments, a mechanical CAD model providing a mechanical view was created using Solid Edge (see Figure 11). Furthermore, a MATLAB®/Simulink® model providing an analysis and controller design view was constructed. These different domain-specific models contain dependencies: the controllability is dependent upon the mechanical properties while the workspace is dependent upon the physical structure. A mechanical design can result in satisfying the workspace requirements of a robot possessing a bad controllability. A controller design can result in satisfying the controller requirements, but with a need to change the mechanical structure. In order for the design and development process to result in a consistent specification, the dependencies across different domain-specific views need to be identified and managed. This requires ensuring consistency not only within a particular model, but also in between different models that together affect a particular system property.

4 CHALLENGES IN DETECTING INCONSISTENCIES

The design and development process of the robot exemplified that inconsistencies can occur throughout the development process and detecting them is by no means a trivial process.

The challenge begins with modeling languages: as we have already established, models that are created using a formally defined modeling language can be checked for internal inconsistencies by means of syntactical verification. Two prominent modeling languages that are actively being used in the community today to construct system models are the Unified Modeling Language as defined by the Object Management Group (OMG UML™) [34] and SysML™. For historic reasons, both of these languages merely have semi-formal definitions [17]. The lack of formalism leads to supporting tools not being able to impose well-formedness and hence not all logical inconsistencies can be detected.

While the system model represents only one view on the system, there are many other models that are explicitly part of the system specification. These models represent the individual views on the system from the viewpoints of various stakeholders. Most of these models share common properties and attributes. Therefore, not only the models themselves need to be consistent, but also the information across the different kinds of models. When working with informal or semi-formal languages this leads to yet another challenge: the dependencies between the models may not be clearly definable. Ultimately, this leads to the intricacy that changes made to one model may have an impact on another, but the magnitude of the impact may not be quantifiable or obvious. To state an example: if software is developed in parallel to a corresponding circuit board and the software requires more memory than provided for, the models become inconsistent.

While these challenges are related to logical inconsistencies that can be detected through the use of an appropriate formal system, there are also other inconsistencies that are more difficult, in some cases even impossible to check for. Checking whether a given system is consistent with the processes of nature and not just with the observations thereof is impossible. While a designer has beliefs about these laws and expresses these using models and while these may represent the observed behavior of, e.g., physics fairly well, there is always a degree of uncertainty as to their validity.

Further inconsistencies related to beliefs occur when making decisions. All concept decision criteria are based on uncertainty, which, first and foremost, require adherence to Kolmogorov's axioms. Unfortunately this is still not enough to protect from inconsistencies occurring. It is still possible to be "dutch-booked", whereby a situation similar to the phenomenon of a *money-pump* may occur [23]. This is usually the result of an absence of transitive qualities in the preference ordering of the decision maker. If, for instance, A is preferred over B, B is preferred over C and C is preferred over A, one would end up with an irrational preference ordering. It is mathematically impossible to construct an objective function from this, hence optimization is also impossible. An individual could now, for instance, sell A for B plus a small amount ε, B for C plus ε and C for A plus ε. At the end of the trade, the individual possesses more than what he started off with. This is a result of the intransitive preference ordering of the buyer – his decisions were therefore *irrational*, i.e. inconsistent with rationality. Detecting such inconsistencies may not always be possible, since not all decisions in real-world scenarios are based on the definition of a mathematical model that can be checked for rationality.

As we have already established, formal systems are based on a set of axioms and inference rules that allow us to decide whether or not a given model is a theorem of the modeling language. However, Gödel has proven that, in mathematics, some true statements cannot be derived from the axioms of a given formal system. In the derivation of his first incompleteness theorem, he has shown this to be true for arithmetic [20]. Since our approach relies on using mathematics as a fundamental theory for modeling languages, we arrive at the conclusion that it is not possible to say with certainty whether or not a set of models is consistent. Instead, we can only detect specific types of inconsistencies within the bounds of a formal system.

5 ONTOLOGY OF INCONSISTENCIES

In the previous sections several types of inconsistencies were identified and elaborated on in detail. The question that remains to be answered is how these inconsistencies relate to one another.

As shown at the beginning of this paper, inconsistencies occur whenever contradictory information is present. When designing and developing systems using only models, such contradictions originate from modeling languages, the relations between different models, the beliefs and preferences of the decision maker and between nature and the observations of it. Figure 12 illustrates the ontology of inconsistencies that was identified as part of this research effort.

Apart from identifying different types of inconsistencies, a distinction between internal and external consistencies was made. Internal inconsistencies are those that occur if a model is not compatible with some *feasible* world, i.e. a world that is based on the axiomatic systems of logic and mathematics.

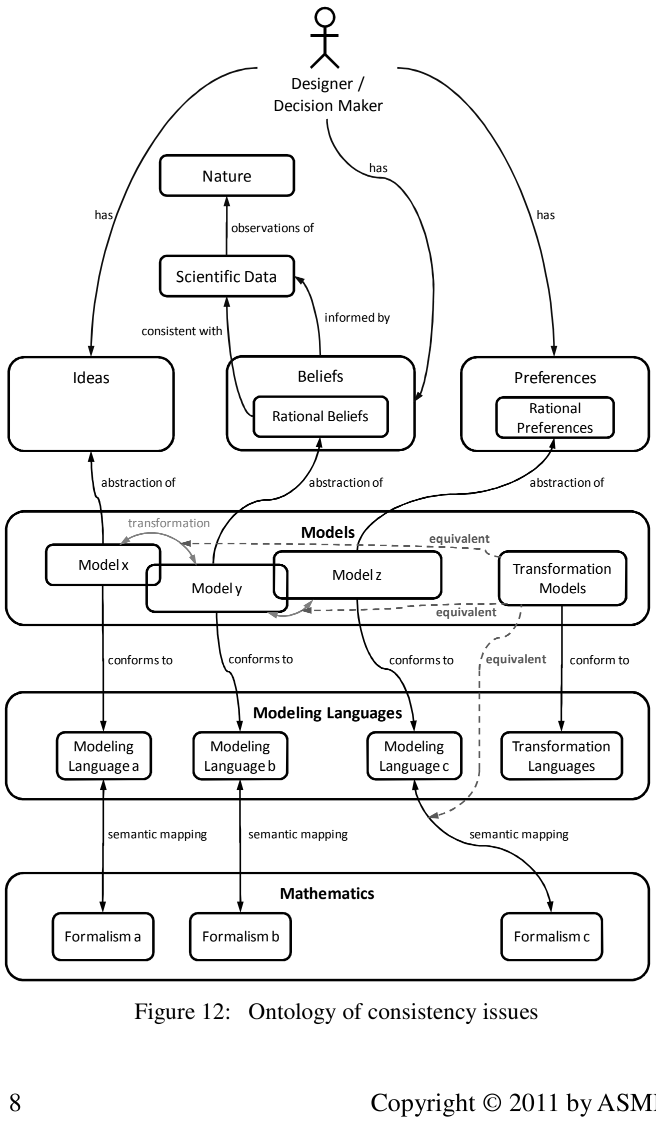

Figure 12: Ontology of consistency issues

External inconsistencies are present if the model is not consistent with reality.

6 REQUIREMENTS FOR CONSISTENCY MANAGEMENT AND TOOL SUPPORT

Now that the different types of inconsistencies have been identified and classified, we can determine requirements for consistency management methods and tools.

First and foremost, to verify the syntax of models, it is imperative to use only formal modeling languages. Without a formal language foundation, tools cannot check whether an individual model is well-formed. Therefore, commonly used modeling languages such as OMG UML™ and OMG SysML™ should be improved by removing any existing ambiguities (e.g., semantic variation points). This is already partially being fulfilled through a variety of research efforts such as those described in [12-13, 35-37].

Secondly, all models should be part of a formal system. This formal system should be based on the axiomatic systems of logic and mathematics and should have the form of a central repository. Within this repository, all models should ideally have a neutral, common data representation to make it easier for relations and dependencies between different models to be identified and satisfied.

Detecting inconsistencies outside the scope of logic and mathematics requires additional information to be present in the system. Therefore, additional, computer-interpretable rules must be defined in the form of, e.g., description logic or meta-models. These must include domain- or application-specific rules such as "there must be at least one actuator per degree of freedom". More rules may be necessary to define the relationship between two or more models affecting application or domain-specific properties. If, for instance, the amount of memory that is available to the controller software of a robot is smaller than the amount that is required, an inconsistency would be present between two distinct parameters from two domains ("available memory" from an electrical schematic and "memory required by the software" from the specification of the software). Relating this to the rational design theory, one could say that, if such an inconsistency occurs, the decision making process has lead to a specification that no longer contains a mapping to a realizable artifact, hence the optimal solution can no longer be reached unless the specification for the amount of available memory is increased. A common data representation, a system model showing the relationships between system components and ontologies of domain-specific concepts all aid in realizing a generally applicable mechanism to get rid of the possibility of such (internal) inconsistencies occurring during the design process.

7 RELATION TO PREVIOUS WORK

With the requirements for consistency management and the different types of inconsistencies having been identified, we now relate this work to the related literature. Most of the work that was analyzed originates from the domain of software engineering. Consistency management for systems with physical characteristics has mostly been dealt with in research projects from the domain of mechatronics, where the use of different domain-specific models for the purpose of specifying a system is common practice. The types of consistency issues that have been dealt with can be split up into two distinct categories: consistency of individual models that were created using a single modeling language and consistency across different, domain-specific models. Most of these papers present ad-hoc approaches, some, however, introduce concepts rooted in a fundamental theory.

The need to formalize OMG UML™ has been identified in several works, e.g. [15-16]. Unfortunately none of the research activities are complete as of yet. The formalization is done by using description logic or the derivation of a context-free grammar to satisfy the dependencies between different model elements contained in separate diagrams.

How two different modeling languages can be related to one another is treated in [7]. The authors use an approach where the meta-models of the individual languages are used to determine the dependencies of language utterances at the meta-level. Dependent parts are then connected using a meta-class that is used to perform a bidirectional transformation. Using this method, the authors were able to successfully build models in a CAD tool and synchronize the changes with a system model built using OMG UML™. Other approaches use Triple Graph Grammar (TGG) approaches to dynamically define the dependencies between a system model and several distinct domain-specific models that are used for refinement [10]. In this specific research effort, a modified version of OMG UML™ called *Mechatronic UML* is proposed by the authors to support the development of a system model that the authors refer to as the *principle solution*. A transition to embodiment and detailed design phase is supported by automated model transformations from the principle solution to several domain-specific models. The authors propose that consistency between domain-specific models and the principle solution can be maintained by use of TGG rules. Unfortunately, they were unable to offer a solution for the problem of potential inconsistencies occurring when modifying or deleting individual components.

Other approaches for checking the internal consistency of system models are discussed by Hehenberger at al. in [6]. The authors identify that processing the entire model at once may take hours to complete. They present an instant consistency checking method that is based on the Model/Analyzer approach. The Model/Analyzer observes changes to a system model performed by a user, and triggers a consistency checker that evaluates a relevant consistency rule for the performed changes. The authors also suggest utilizing a mechatronic ontology, letting the designer describe different design concepts and their structure. Later, consistency rules can be defined on top of an ontology model. The authors argue that this approach provides support for maintaining consistency between models developed and used across different domains during different design phases, especially in the conceptual design phase.

Among the works reviewed, there are some promising approaches that could potentially solve a subset of the consistency issues identified in this paper. Unfortunately most of the current work does not scale and is still incomplete.

8 CONCLUSIONS

The aim of the research presented in this paper is to explore the fundamentals of consistency and relate these to modeling and engineering design. In the process, several distinct types of inconsistencies were identified, classified and related to one another. We conclude that there are two major categories for inconsistencies: internal and external consistency issues. Internal consistency issues are inconsistencies with respect to some feasible reality that we can define using a formal system. This formal system is based on axioms and inference rules which map to a fundamental theory from mathematics (e.g. logic). External consistency applies to the reality in which the specified system will ultimately exist in, e.g. the real world in the case of technical systems with physical characteristics. Checking for external consistency issues is impossible to achieve, since it would require perfect knowledge about the processes occurring in nature. All one can do is express rational beliefs about such processes by constructing models that are based on scientific data. These models are, by definition, abstractions of reality and therefore introduce uncertainty. Therefore, they cannot guarantee consistency with reality.

Some types of inconsistencies can be managed. In order to detect internal inconsistencies, the modeling language that is being used to construct a given model needs to be part of a formal system, i.e. it needs to be based on a set of axioms and inference rules. Otherwise it is impossible to formally prove that a given model is consistent, i.e., well-formed and therefore a valid utterance of the language it is based on.

Inconsistencies related to the beliefs and preferences of the decision maker are not always detectable. Only rational preferences lead to consistent decisions. Detecting inconsistencies in the process of decision making requires all information that is being applied to be explicitly present. This is, however, not always the case.

Research efforts mentioned in the related literature discuss several methods to check for and potentially resolve inconsistencies that can occur within and across models from different domains. Unfortunately, most of the current work is incomplete and only considers specific types of inconsistencies. Many of the approaches are also ad-hoc solutions and are not rooted to any fundamental theory. This is one of the reasons why some of them do not scale.

In summary, it is only within a formally defined system, i.e., a system based on a set of axioms and inference rules that inconsistencies can be identified in. Even then it is not guaranteed that all syntactically correct models can be identified as such. Gödel has proved with his first incompleteness theorem that not all systems based on axioms are able to produce all possible valid utterances of its formal language, i.e. the system cannot assure well-formedness [20]. In his work, Gödel has shown this for arithmetic. Because of this and external consistency issues, we must conclude that it is impossible to say whether or not a system is consistent. It is only possible to detect inconsistencies of models within the bounds of an axiomatic system. A consequence of this is also that any attempt to try and fully validate models is fundamentally flawed.

ACKNOWLEDGMENTS

The principle author would like to thank Prof. Dr. Ulrich Walter (Institute of Astronautics, Technische Universität München) and the Heinrich & Lotte Muehlfenzl Foundation for making it possible to perform the presented research at the Georgia Institute of Technology. Furthermore, the authors would like to acknowledge the input provided by Dr. Markus Brandstätter (Institute of Astronautics, Technische Universität München) and Benjamin D. Lee (Model-Based Systems Engineering Center, Georgia Institute of Technology).

REFERENCES

[1] Siemens, "Solid Edge," http://www.solidedge.com/.

[2] Modelica Association, "The Modelica Language Specification Version 3.2," 2010. [Online]. Available: http://www.modelica.org/

[3] IBM, "Rational DOORS," http://www.ibm.com/-software/awdtools/doors/.

[4] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide To SysML: The Systems Modeling Language*. Morgan Kaufmann Publishers and OMG Press, August 2009.

[5] J. A. Estefan, "Survey Of Model-Based Systems Engineering (MBSE) Methodologies," Jet Propulsion Laboratory, California Institute of Technology, Tech. Rep., May 2008.

[6] P. Hehenberger, A. Egyed, and K. Zeman, "Consistency Checking Of Mechatronic Design Models," in *2008 IEEE Symposium On Visual Languages and Human-Centric Computing*, P. Hehenberger, A. Egyed, and K. Zeman, Eds., ASME. ASME, August 2010.

[7] C. Adourian and H. Vangheluwe, "Consistency Between Geometric And Dynamic Views Of A Mechanical System," in *Proceedings of the 2007 summer computer simulation conference*, San Diego, CA, USA, June 2007.

[8] X. Blanc, A. Mougenot, I. Mounier, and T. Mens, "Incremental Detection Of Model Inconsistencies Based On Model Operations," in *CAiSE '09 Proceedings of the 21st International Conference on Advanced Information Systems Engineering*, Amsterdam, The Netherlands, June 2009.

[9] M. Yoshioka and T. Tomiyama, "Pluggable Metamodel Mechanism: A Framework Of An Integrated Design Object Modeling Environment," in *Proceedings of the 1997 Lancaster International Workshop on Engineering Design CACD'97*, A. Bradshaw and J. Counsell, Eds., Lancaster University, Great Britain, 1997.

[10] J. Gausemeier, W. Schäfer, J. Greenyer, S. Kahl, S. Pook, and J. Rieke, "Management Of Cross-Domain Model Consistency During The Development Of Advanced

Mechatronic Systems," in *International Conference on Engineering Design*. ICED, August 2009.

[11] T. Mens, "Conditional Graph Rewriting As A Domain-Independent Formalism For Software Evolution," in *In the Applications of Graph Transformations with Industrial Relevance International Workshop, AGTIVE'99, p. 127, Kerkrade, The Netherlands. LNCS 1779*. Springer-Verlag, 2000, pp. 127–143.

[12] P. Andre, A. Romanczuk, and J. Royer, "Checking The Consistency Of UML Class Diagrams Using Larch Prover," in *Rigorous Object-Oriented Methods 2000*, York, UK, January 2000.

[13] J. Chanda, A. Kanjilal, and S. Sengupta, "UML-Compiler: A Framework For Syntactic And Semantic Verification of UML Diagrams," in *Distributed Computing and Internet Technology*, Eds. Springer Berlin / Heidelberg, 2010, vol. 5966, pp. 194–205.

[14] B. Schätz, P. Braun, F. Huber, and A. Wispeintner, "Consistency In Model-Based Development," in *Proceedings of ECBS 2003 10th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 2003, from the department of Systems & Software Engineering at TUM (Broy).

[15] T. Mens, R. Van Der Straeten, and J. Simmonds, "A Framework For Managing Consistency Of Evolving UML Models," in *Software Evolution with UML and XML*, H. Yang, Ed. Idea Group Publishing, 2005, pp. 1–31.

[16] J. Simmonds, R. Van Der Straeten, V. Jonckers, and T. Mens, "Maintaining Consistency Between UML Models Using Description Logic," in *RSTI série L'Objet, Langages et Modèles à Objets, LMO'04*, vol. 10, no. 2-3, pp. 231–244.

[17] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, "Seamless Model-based Development: From Isolated Tools To Integrated Model Engineering Environments," in *Proceedings of the IEEE, Special Issue on Aerospace and Automotive Software*, vol. 98, no. 4, April 2010.

[18] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction To Automata Theory, Languages, And Computation*, 2nd ed. Addison Wesley, November 2000.

[19] D. Kozen, *Automata And Computability*. New York: Springer-Verlag, 1997.

[20] D. R. Hofstadter, *Gödel, Escher, Bach*. Basic Books, February 1999.

[21] H. Giese, T. Levendovszky, and H. Vangheluwe. Summary of the Workshop on Multi-Paradigm Modeling: Concepts and Tools. *In Proceedings of the 2006 International Conference on Models in Software Engineering*, MoDELS'06, pages 252–262. Springer-Verlag, 2006.

[22] *Meta Object Facility (MOF) Version 2.0*, Object Management Group Std.

[23] G. A. Hazelrigg, "A Framework For Decision-Based Engineering Design," in *Journal of Mechanical Design*, vol. 120, December 1998, pp. 653–658.

[24] S. C. Thompson, 2011, *Rational Design Theory: A Decision-Based Foundation For Studying Design Methods*, Ph.D. Thesis, G.W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta.

[25] S. Chowdhury, *Design For Six Sigma*, 1st ed. Kaplan Business, March 2002.

[26] G. Taguchi, "On Robust Technology Development." ASME Press, NY, 1993.

[27] G. Boothroyd, "Design For Assembly - The Key To Design For Manufacture," *International Journal of Manufacturing Technology*, vol. 2, no. 9, pp. 3–11, January 1987.

[28] M. Tribus, *Rational Descriptions, Decisions And Designs*. Pergamon, 1969.

[29] J. von Neumann and O. Morgenstern, *Theory Of Games And Economic Behavior*. Princeton: Princeton University Press, 1947.

[30] P. C. Fishburn, *Utility Theory For Decision Making*. Wiley, 1970.

[31] *SysML, The Systems Modeling Language (OMG SysML™) Version 1.2*, Object Management Group Std., June 2010.

[32] Mathworks, "MATLAB®/Simulink®," http://-www.mathworks.com/.

[33] E. Tjalve, *Systematic Design Of Industrial Products*. Institute of Product Development, The Technical University of Denmark, 2003.

[34] OMG, "OMG Unified Modeling Language (OMG UML) Infrastructure Version 2.3," Tech. Rep. formal/2010-05-03, 2010. [Online]. Available: http://www.omg.org/spec/-UML/2.3/Infrastructure/PDF

[35] M. Broy, M. L. Crane, J. Dingel, A. Hartman, B. Rumpe, and B. Selic, "Formal Semantics For UML."

[36] *Semantics Of A Foundational Subset For Executable UML Models*, Object Management Group (OMG), May 2009.

[37] "UML 1.0 Semantics," Rational Software Corporation, Tech. Rep., 1997.