Patterns of inconsistency management in mechatronics – a survey

Technical report

István Dávid, Joachim Denil, Hans Vangheluwe

December 2015

Contents

Chapter 1

Introduction

Dealing with inconsistencies is a necessary burden in real-life MDE scenarios. In practice, we often face multi-model/multi-paradigm settings, but multi-user scenarios can give rise to inconsistencies as well.

What are inconsistencies after all? How do we formalize them? How do we deal with them? In the past years multiple contributions have been made to answer these question. In the current technical report, we attempt to collect and organize them to the best of our knowledge.

1.1 Goals of the survey

The main questions to be answered are the following ones.

- **What methods, techniques and tools are available for the management of consistency in a mechatronic and CPS design process?**

  Many of the contributions to the field have been made in the context of pure software systems. As opposed to this context, our main focus lies on the design of mechatronic and cyber-physical systems in particular.

- **What are limitations of the state-of-the-art?**

  Identifying limitations and rather, the shortcomings of the state-of-the-art is a vital part of this report. We rely on this information to fill the gaps in what we believe to be an efficient approach for handling inconsistencies.

- **What makes consistency management in a mechatronic and CPS context different from the software engineering industry in general?**

1.2 Selection method

Our selection method was based on the guidelines described in [1].

As shown in Figure 1.1, the process started with selecting the appropriate search terms (Section 1.2.1) and sources (Section 1.2.2). The Cartesian product of these inputs parametrized the search process. To successfully qualify to be a candidate, every paper went through the exclusion criteria (Section 1.2.3).

Selecting the appropriate papers took two rounds of reviewing (Section 1.2.4). Afterwards, the Google Scholar results were included by making sure not to introduce any duplicates.

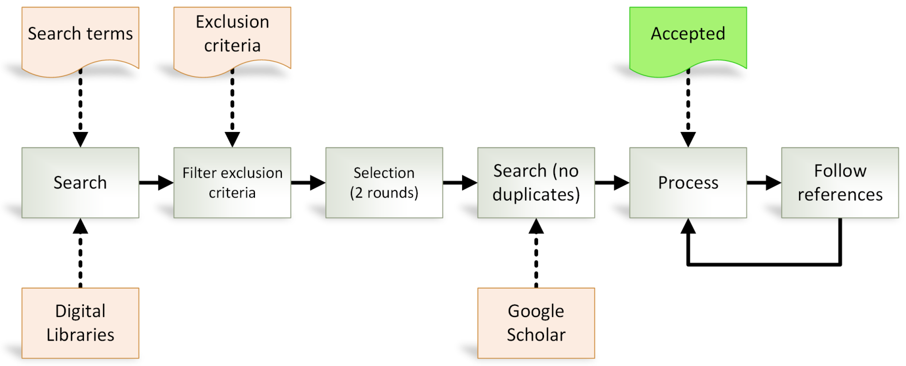


Figure 1.1: *High-level workflow of searching, selecting and processing*

Finally, we processed the selected papers and in parallel, followed the interesting references.

1.2.1 Search terms

To constraint scope of search for papers, the following search terms have been selected:

- consistency management, inconsistency management;
- overlap detection;
- inconsistency detection;
- inconsistency resolution.

Additionally, the following extra search terms have been also used:

- multi-paradigm model(l)ing, multi-view model(l)ing;
- mechatronic design;
- concurrent engineering;
- tool integration.

1.2.2 Sources

We used the search terms to look for relevant publications from the following sources:

- digital libraries: ACM, IEEE, Springer;
- Google Scholar;
- manual search based on top cited articles (authors and cited by).

(We plan to extend our current work by also including domain-specific sources, such as journals of software engineering, mechanical design, etc.)

First, the digital libraries have been searched through. The top 15 papers for every search term have been accepted as candidates for the further evaluation rounds. After selecting the top 15 publications, the year of publication has been constrained to the past two years as newer publications tend to sink lower in lists ordered by relevance. That makes a total set of candidates of 20 per source and per search term. The total amount of papers included for the fist round was approximately 200. (With duplicates.)

Additionally, we searched through Google Scholar after the review and selection step.

1.2.3 Exclusion criteria

We defined a set of exclusion criteria against the candidates: non-English and short papers got rejected immediately.

1.2.4 Review and selection

The detailed review process is shown in Figure 1.2. To review the approximately 200 papers, papers had been divided and assigned to one of the two reviewers. In the *1st round* of evaluation, three groups were formed.

Accept A paper found to be relevant based on its abstract, got immediately accepted for further processing.

Reject A paper found not to be relevant based on its abstract, got immediately rejected.

Maybe If the relevance of the paper could not be decided, the paper was assigned to this group and processed in the *2nd round* of evaluation.

To evaluate which group a paper belongs to, we investigated how many of the following properties a paper has.

- It gives an overview.
  - It's a survey / systematic review. (Strong accept)
  - Presents a classification of the types of (in)consistency problems...

    – in the domain of mechatronics or maybe avionics/automotive. (Immediate accept)

- Presents a technique…
    – for overlap detection.
    – for inconsistency detection.
    – for resolution.
    – the technique is supported via a (prototype) tooling.

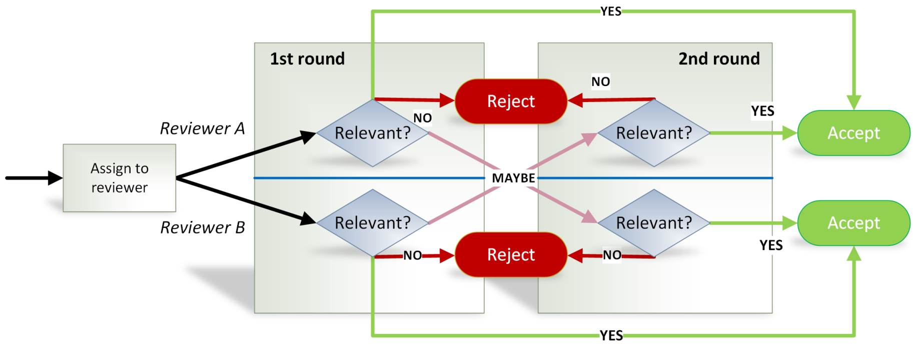


Figure 1.2: *Detailed selection process*

In the *2nd round* of evaluation the papers reviewed and assigned to the "Maybe" group, were evaluated by the other reviewer. This time, the evaluation was carried out based on not just the abstracts, but the introductions as well. Proving to be relevant, the paper got eventually accepted, otherwise rejected.

1.2.5 Results

At the of the selection process, approximately 70 papers have been identified to be interesting to be actually incorporated in our survey. This number has been extended by followed references during the processing phase.

1.2.6 Threats to validity

During the search and selection phase, we identified the following threats to the validity of this survey.

The overloaded meaning of "inconsistency" Inconsistencies are natural phenomena in several domains, not just in multi-model settings. Typical examples include: inconsistencies violating the ACID principle in relational database management systems, and well-formedness violations in software design. This overloaded nature of the term makes it hard to filter the works relevant to our scope.

Semantic gaps between different domains Even in closely related domains, the definition of inconsistencies is not clear. As a typical example, in pure software modeling, and especially in the case of UML modeling, inconsistencies are defined solely as well-formedness violations, i.e. a discrepancy arising from the linguistic properties of the domain. In a true MPM setting, although, semantic inconsistencies are also considered as discrepancies arising from the overlaps between various formalisms, domains.

Hidden domains Althouhg our survey focused on multi-model setting in general, this proposition also implicitly assumes the involved domain actually employs explicit modeling techniques. If there is no link made between a domain and explicit modeling, a domain may remain hidden. As a typical example, we reached as far as the rather exotic domains of "concurrent engineering", which is not linked directly to MPM, but we had a priori knowledge about potential links. We plan to gradually extend our technical report as hidden domains get discovered.

Chapter 2

The mechatronic design process

The mechatronic design process is characterized by an interplay among disciplines such as mechanical engineering, electronics, control engineering and software engineering, whilst designing and delivering complex products. Domains such as automotive, avionics and robotics are the prime examples employing such a design setting.

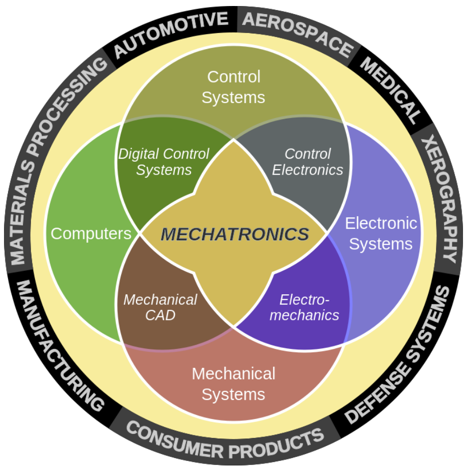


Figure 2.1: *Domains typically involved in a mechatronic design process. (Source: http://www.rpi.edu.)*

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. [2] Due to the conceptual distance between the involved engineering domains, facilitating an efficient mechatronic process is not trivial. The differing vocabularies, techniques, tools and practices make this goal very difficult to achieve. The need for integration of different engineering domains emerges already in early stages of mechatronic processes, as the eventually delivered complex systems are contemplated extensively through model-based approaches. [3]

Modeling various aspects of complex systems needs to be approached using the most appropriate formalisms, levels of abstraction, techniques and tools. Heterogeneous layouts like this inherently suffer from emerging inconsistencies among various models, which can lead to bad design, invalid products and eventually, enormous increase in associated costs and time-to-market.

In this report we (i) review consistency management approaches in the mechatronics domain and (ii) identify consistency management approaches in the software design domain potentially applicable for mechatronic processes.

Chapter 3

Overview

In this section, we give an overview on inconsistencies and the state-of-the-art management techniques. First, we briefly overview how the state of the art interprets the nature of inconsistencies (Section 3.1), then we define a taxonomy for discussing various inconsistency management techniques and approaches (Section 3.2).

3.1 On the nature of inconsistencies

What are inconsistencies?

There is no unified definition of "inconsistency", as it is always specific to the characteristics of the given domain, setting and task. According to the Merriam-Webster dictionary [4]:

> *"inconsistent": not always acting or behaving in the same way; having parts that disagree with each other: not in agreement with something.*

Spanoudakis et al [5] define inconsistencies specific to software models as

> *a state in which two or more overlapping elements of different software models make assertions about aspects of the system they describe which are not jointly satisfiable.*

Herzig et al [6] also add that

> *an inconsistency is present if two or more statements are made that are not jointly satisfiable,*

with the typical examples of

> *failure of an equivalence test, non-conformance to a standard or constraint and the violation of physical or mathematical principles.*

The origin of inconsistencies

Investigating their origin, gives a better understanding on the nature of inconsistencies. Some authors claim the **imprecise or vague semantics** of modeling languages being a potential source of inconsistencies:

> *Another source of inconsistency is the imprecise semantics of the UML.* [7]

More frequently, however, inconsistencies arise in settings featuring **multiple views** on the same virtual product and the mismanagement of the information between these.

> *(. . .) inconsistencies arise because the models overlap that is they incorporate elements which refer to common aspects of the system under development and make assertions about these aspects which are not jointly satisfiable as they stand, or under certain conditions.* [5]
>
> *Views conforming to these viewpoints are highly interrelated due to the concerns addressed overlapping. These interrelations and overlaps can lead to inconsistencies.* [6]
>
> *(. . .) if different views of a system have some degree of overlap, how can we guarantee that they are consistent, i.e., that they do not contradict each other?* [8]

Additionally, the **evolution** of single model artifacts can also lead to inconsistencies. Hehenberger et al [2] note that

> *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.*

Managing inconsistencies

Relational database management systems and version control systems overcome the problems of inconsistencies by *avoiding* them in the first place. In multi-view settings, however, this cannot be always guaranteed and the *allow-and-recover* style of management is more appropriate. Once inconsistencies are encountered, their root causes are required to be analyzed in order to manage them properly. Finkelstein et al [9] notes that

> *Rather than thinking about removing inconsistency we need to think about "managing consistency".*

In our view, however, it is more appropriate to reason about managing **in**consistencies, since we assume that inconsistencies cannot be avoided and the real challenge is to deal with them *after* they are encountered.

3.2 Features and patterns of inconsistency management techniques

In this section we define a structured taxonomy for discussing various inconsistency management (ICM) techniques and approaches. In our taxonomy, we combine those of Spanoudakis et al [5] and Hanmer et al [10] and characterize ICM techniques and by their *features* and the *patterns* used for of implementing those features. (For example, using *Pivot models* (Section 4.1.1.3) is a typical pattern of the *Representation* feature (Section 4.1.1).) While the usage of features assumes a closed-world, and thus gives a rigid structure to our view on ICM techniques in general, we assume an open-world in case of patterns of implementing those features, since it is not feasible to exhaustively enumerate and classify all of the patterns.

3.2.1 A feature model of ICM

Not every ICM technique addresses the same challenges in inconsistency management. While some of them focus on specific sub-problems (such as detecting inconsistencies), others approach inconsistency management from a holistic view and attempt to come up with a framework for ICM. Based on our survey of the state of the art, we compiled a feature model for ICM techniques, shown in Figure 3.1.

The two required features of ICM techniques are their *dimensions* and *activities*. Dimensions are used to investigate *what* an ICM technique attempts to tackle, while activities tell *how* they achieve that. Additionally, the extra-functional properties of an ICM techniques may be also interesting in some cases, as they tell *how good* a technique with respect to a specific metric (such as performance, precision, etc).

Feature	**Meaning**
Dimension	What?
Activity	How?
Extra-functional property	How good?

Table 3.1: *Features and their meaning*

3.2.2 Dimensions

Dimensions identify *what* a given ICM technique attempts to cover. More specifically, what types of inconsistencies it considers. Table 3.2 summarizes the dimensions considered at this place.

3.2.2.1 Space

Depending on whether the considered inconsistencies are restricted to just one single model or are allowed to extend over multiple models, intra-model and

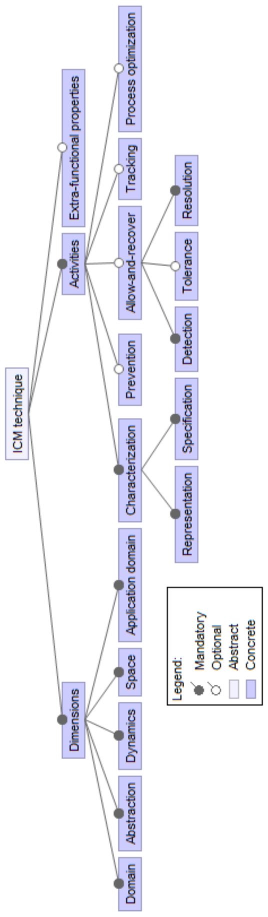


Figure 3.1: *Feature model of ICM techniques*

inter-model inconsistencies are distinguished. [11] Intra-model inconsistencies occur in one single model, while inter-model inconsistencies can be observed between different models.

Due to the prevalence of multi-view settings in the mechatronic domain, techniques feasible for inter-model (̃inter-view) inconsistency management are the preferred ones. Since ICM techniques suitable for inter-model situations are also suitable for intra-domain cases, we only consider approaches capable of inter-model ICM and do not investigate this dimension further.

3.2.2.2 Domain

The domain defines whether the considered inconsistencies are situated in the *linguistic* or the *semantic* domain of a model. Linguistic inconsistencies arise

Dimension	**Patterns**
Domain	Linguistic Semantic
Abstraction	Horizontal Vertical
Dynamics	Static Evolution
Space	Intra-model Inter-model

Table 3.2: *Dimensions and their patterns*

from syntactic discrepancies and are equivalent to well-formedness violations, as known in software modeling. Semantic inconsistencies, on the other hand, are results of overlaps between different domains and formalisms involved in the modeling process.

3.2.2.3 Abstraction

Depending on whether an inconsistency can extend over various levels of abstraction, Lucas et al [11] distinguish between *horizontal* and *vertical* inconsistencies. While horizontal inconsistencies occur between models on the same level of abstraction, vertical ones are associated with models on different levels of abstraction.

3.2.2.4 Dynamics

An ICM technique can static vs evolution] Depending on whether the notion of inconsistencies supports reasoning between different versions of the same model, inconsistencies may be defined either as static or evolutionary.

3.2.2.5 Discipline

Also known as the *application domain* of a technique, the discipline identifies the scope an ICM technique can be successfully applied within. The majority of the state of the art considers only software engineering as an application domain.

3.2.3 Activities

Activities define *how* an inconsistency management process is structured. We mainly follow the classification of Spanoudakis et al [5], but further extend it as presented later. Our classification of ICM processes is shown in Figure 3.2.

3.2.3.1 Characterization

Characterizing inconsistencies is the necessary first step to every ICM process. Its role is twofold. First, the inconsistency rules defined in this step serve as inputs

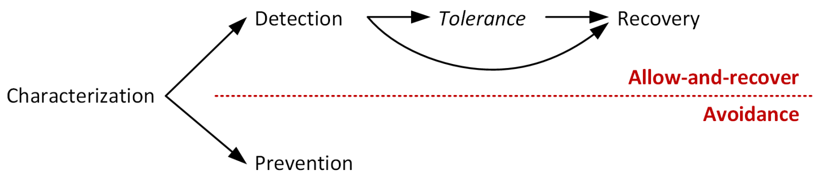


Figure 3.2: *Conceptual overview of ICM styles and activities*

for the subsequent activities (detection or avoidance). Second, the employed patterns of inconsistency characterization, heavily influence the choice on the patterns of the subsequent steps.

Different authors approach this process with various granularity and see the role of characterization (often referred as *inconsistency definition*) different. Spanoudakis et al [5] do not emphasize the role of proper inconsistency characterization, while Van Der Straeten [12] acknowledges characterization as the foundational first step towards a managed approach towards handling inconsistencies.

3.2.3.2 Detection

In non-prevention style techniques, inconsistent situations have to be detected based on the characterization of inconsistencies in order to execute the proper resolution actions. The patterns of characterization heavily influence the patterns of detection.

3.2.3.3 Resolution

Once an inconsistent state is detected, it is usually expected to be resolved. Although Finkelstein et al [9] notes, that managing consistency means not only removing the detected inconsistencies, but also

(...) preserving inconsistency where it is desirable to do so.

3.2.4 Summary of the state-of-the-art

Table 3.3 briefly summarizes the features supported by the state of the art techniques considered in our survey. The patterns of these techniques are classified and presented in Chapter 4.

	Activities			
	Characterization	Detection	Resolution	Prevention
Adourian	●	●	●	
Balzer	●	●		
Becker	●			
Bhave	●	●		
Bhave 2	●			
Blanc	●	●		
Dahman	●	●	●	
Easterbrook	●	●		
Egyed	●	●		
Engels	●	●		
Gausemeier	●	●	●	
Giese	●	●	●	
Giese 2				●
Hamlaoui	●	●		
Hehenberger	●	●		
Herzig	●			
Hessellund	●			●
Le Noire	●	●		
Lopez-Herrejon		●		
Lopez-Herrejon 2	●	●	◑	
Mens	●	●	●	
Nentwich			●	
Oka	●			●
Qamar	●	●		
Quinton		●		
Shah	●			
Stolz	●	●		
Van Der Straeten	●	●	●	
Van Der Straeten 2	●	●	●	

Table 3.3: *Overview on the state of the art. (Legend: ○: no support; ◑: possible support, not emphasized; ●: dedicated support)*

ID	**Reference**
Adourian	[13]
Balzer	[14]
Becker	[15]
Bhave	[16]
Bhave2	[17]
Dahman	[18]
Easterbrook	[19]
Egyed	[20]
Engels	[21]
Gausemeier	[22]
Giese	[23]
Giese 2	[24]
Hamlaoui	[25]
Hehenberger	[2]
Herzig	[26]
Hessellund	[27]
Le Noire	[28]
Lopez-Herrejon	[29]
Lopez-Herrejon 2	[30]
Mens	[31]
Nentwich	[32]
Oka	[33]
Qamar	[3]
Quinton	[34]
Shah	[35]
Stolz	[36]
Van Der Straeten	[37]
Van Der Straeten 2	[38]

Table 3.4: *IDs and their related bibliographic entries*

Chapter 4

Patterns of ICM activities

In this chapter, we overview the typical patterns of each activity presented in Section 3.2.3.

4.1 Characterization

In terms of characterization, we distinguish between two important aspects:

- representation, i.e. the form (in)consistency rules are defined in, and
- specification, i.e. the way (in)consistency rules are defined.

Typical representation patterns include using *naming conventions* (Section 4.1.1.1), *graph-like formalisms* (Section 4.1.1.2) and *pivot model based* techniques (Section 4.1.1.3). Other representation techniques are discussed in Section 4.1.1.4. Concerning specification, most of the available techniques are based on and involve *human inspection*, therefore we only mention those employing a semi-automated or automated technique.

4.1.1 Representational patterns

4.1.1.1 Naming conventions

According to Spanoudakis [5]:

> *the simplest and most common representation convention is to assume the existence of a total overlap between model elements with identical names and no overlap between any other pair of elements.*

Hessellund et al [27] investigate inconsistencies in a setting where multiple DSLs are involved in the design process. The technique aims to represent, check, maintain constraints, and that in an avoidance-like fashion. An EMF extension framework, SmartEMF is used which employs Prolog to achieve ICM. Referential integrity is mentioned as a typical (general) type of inconsistencies.

Oka et al [33] identify different types of relationships between (i) two components, (ii) composite and component objects, (iii) two composite objects. The technique has three main elements:

- update operation patterns - there are three of them: (a) not allowed update, (b) allowed update but no change propagation, (c) allow-and-propagate,
- modification rule matrix - specifies how the previous ones are applied: for every relationship type a pattern is chosen,
- modification algorithm - orchestrates the interplay between the matrix and the rules

The authors to not assess the performance of the framework and there is no known follow-up project or tool. The applicability of the technique seems to be very constrained.

4.1.1.2 Dependency graphs

The second group of characterization techniques by Spanoudakis [5] are the *shared ontologies*. This approach requires the authors of the models to tag the elements in them with items in a shared ontology. The tag of a model element is taken to denote its interpretation in the domain described by the ontology and, therefore, it is used to identify overlaps between elements of different models.

We found that this technique is hardly ever encountered in mechatronic settings. Instead, graph-like formalisms are used to depict overlaps between various models.

A number of techniques employs a specific subset of dependency models in form of TGGs. [39]

Becker et al [15] describe requirements and the characteristics of tools for inter-model consistency management. Most notably, they identify functionality (m–n relations between objects the tools needs to install), operation mode (incremental preferred over batch), direction (bidirectionalaty is preferred) and adaptability (support for different domains) among others and identify TGGs as a good fit with these requirements.

Adourian et al [13] use an explicit correspondence model to relate elements of different models to each other. Unidirectional change propagation can support *interleaved* evolution, where no conflicts are introduced in the separate views of the system. Bidirectional change propagation can support *parallel* evolution, where parallel and conflicting changes are allowed to be introduced. Triple-graph grammars (TGG) are used as a theoretical underpinning to the technique. It is thanks to the (computationally) non-causal nature of Modelica as well as its modularity that an almost one-to-one correspondence between geometry models and dynamics models can be found. Without non-causal models, we would have to associate many causal models with a single geometric model.

Gausemeier et al [22] propose using a cross-domain system model for consistency management, based on TGG. This view of the system is used in conceptual design. The conceptual view (active view), contains system elements and shows the information and energy flow in the system. After conceptual design, all views of the system (in siloed design for example) keep relations with this domain-spanning model. Model transformations are used to generate the domain-specific views from the conceptual model and to keep the models consistent.

Giese et al [23] focus on bidirectional model synchronization as an important technique in MDSD. They provide an algorithm based on the triple graph grammar formalism (TGG). While their previous work [40] optimizes for a single change, there might be compound changes to be synchronized. The work coins the problem of multiple transformation steps and the need for a formal framework for reasoning about algebraic-temporal structures of these.

Mechatronic/CPS settings.

Qamar et al [3] present a technique specifically for the mechatronics domain, which considers structural and parameter type dependencies between different domain-models in order to provide consistency between different views. This provides an ability to traverse between different views of the system, as well as maintaining consistency between those views. Inter-domain relationships are established via SysML as a pivot model, between a mechanical model (Solid Edge) and the dynamic analysis model (SimScape/SimMechanics).

Other authors focus on the operational side of dependency modeling.

Mens et al [31] expresses inconsistency detection and resolution as *graph transformation rules*. The dependency analysis of these enables appropriate ordering and refactoring of inconsistency rules. The analysis is carried out using the critical pair analysis algorithm. This analysis can be exploited to improve the inconsistency resolution process, for example, by facilitating the choice between mutually incompatible resolution strategies, by detecting possible cycles in the resolution process, by proposing a preferred order in which to apply certain resolution rules, etc. The approach focuses on structural inconsistencies and therefore, it falls short to tackle the problem of semantic inconsistencies.

Egyed [20] presents an *incremental technique* to detect and keep track of inconsistencies and a prototype tooling to support the approach. The author claims that even "very large industrial" models can be maintained efficiently. The technique identifies the consistency rule instances to be checked upon a model change, based on a *scopes of the changes*. The rules are defined manually and cover structural constraints only; this technique, therefore does not support reasoning over semantic inconsistencies.

Dahman et al [18] investigate the problem of consistency management in the domain of business processes. They propose incremental model transformations to mend evolutionary consistency issues. The case study is the consistency between a BPMN model and a Component-Based Model. Update mechanisms on one model are primitive updates: addNode, insertEdge, dropNode, deleteEdge, setLabel, setSource, setTarget, setIndex. A synchronization algorithm specifically for the BPMN to SCA model is specified in the paper. The characterization is achieved on meta-model level, and could be used for keeping analysis models up-to-date.

Apart from inconsistencies of static dynamics discussed above, evolutionary inconsistencies can be also supported by dependency graphs.

Hamlaoui et al [25] recognize the infeasibility of depicting virtual product by one huge model and propose a network of related models, which provides a global view of the system through a correspondence model. The approach is claimed to be domain independent. Two types of model evolution are presented: adaptation (between model and meta-model) and co-evolution (on the same level of abstraction). Changes (add, modify, delete) in one model trigger changes in other model(s). An Xtext-based textual DSL is used for defining the correspondence model. Domain specific and -independent structural changes can be detected; semantic changes can be handled via the suggestions provided to the human user, although the paper does not provide examples on this scenario.

4.1.1.3 Pivot models

A special case of the graph-based representation is the usage of pivot models, which is more frequently used in design of mechatronic and cyber-physical systems. Pivot models act as an intermediate language to transform models into each other.

Shah et al [35] present SysML as a pivot model for system engineering and the related concerns of tool integration. SysML is used as the pivot where also the other languages (abstract syntax) are included by using profiles. Model transformations are used to generate the domain-specific models in the different tools form the SysML model or vice versa. SysML parametric models are used to model the dependency relations within the pivot model. Because of structure changes (topological changes), this would have to be remade every time the structure changes. For this MAsCoMa [**?**] multi-aspect component models are used. These instantiate a correct component and parametric relations when the topology changes. This information is domain-specific. The paper also provides a case study where the approach is applied to a log splitting machine.

Bhave et al [17] describe an architectural approach to reasoning about relations between heterogeneous system models. A pivot model (the "run-time base architecture") is used to associate related models. Models are related to the pivot model through architectural views, which capture structural and semantic correspondences between model elements and system entities. The component-and-connector (C&C) perspective is used to define the architecture. The C&C perspective is extended to efficiently address not just software and computational infrastructures, but also the physical parts of a CPS - this is the base architecture (BA). The BA of a CPS is an instance of the CPS architecture style, which contains all the cyber and physical components and connectors that constitute the complete system at runtime. Architectural views for a modeling formalism are defined as a tuple of (i) the C&C configuration of the view (with types, semantics and constraints); (ii) associations of model elements with elements of CV; and (iii) associations of elements in CV with elements in BA, respectively. The technique is demonstrated through a multi-domain case study on engineering a quadrotor system. The only type of inconsistencies considered is the one arising when model elements are mapped to the BA in a many-to-many style, which is clearly a structural one. The authors claim that current C&C architectural styles, which focus primarily on software and computational infrastructures, are not comprehensive enough to describe a complete CPS. Although the aim of this paper is not ICM, as this question is addressed in Bhave et al [16] introduce the notion of structural consistency management for CPS. Views of a single system are kept consistent using an architectural base model (the pivot model). This base architecture relates the different views of the system to each other. Well defined mappings between a view and a base architecture manage the different consistencies. Defines "weak consistency" as (a) every component in the view is accounted for in the architectural base model, (b) every communication pathway and physical connection in view should be allowed in the architectural base model. Defines "strong consistency" as weak consistency, plus: every element in the architectural base model must be represented in the view.

4.1.1.4 Other approaches

Operation-based representation
Blanc et al [41] propose to represent models by sequences of elementary construction operations, rather than by the set of model elements they contain. The key idea of the approach, to uniformly detect structural and methodological inconsistencies is, that it relies on elementary model construction operations instead of the model elements themselves. Change operations are motivated by MOF: create, delete, setProperty, setReference. Structural and methodological consistency rules can then be expressed uniformly as logical constraints on such sequences. Structural and "methodological" inconsistencies can be detected. Methodological rules constrain the overall construction process and the authors claim this being the core contribution of the paper. The approach supports multi-model and multi-level modeling. For structural constraints, they define

operation pairs, where the second operation cancels out first one. For example, creating a model element and subsequently deleting it.

Le Noire et al [28] propose an approach that represents models as a sequence of operations. The Praxis tool is used as a (meta-model independent) tool for operation based model management, with and PraxisRules for consistency constraint definition. The textual DSL expresses consistency constraints in terms of operation primitives and also allows complex change definitions. The search for inconsistencies is performed on the subset of the model that was modified since the same inconsistency check was run last time. The provided use case builds on the ARCADIA [**?**] framework of Thales. The approach allows modeling both structural and semantic inconsistencies.

Stolz et al [36] present a notion of potentially re-orderable model transformations to track the semantic dependencies of the different modeling steps. The technique assumes model evolution on various meta-levels and that specific model elements may contain different data on different levels. Both structural and semantic issues are considered when change propagation fails. The method starts from an empty model which evolves via primitive operations: add, update, delete. Prerequisites are assumed to be captured in first order logic (e.g. OCL in UML settings), which imply a set of model elements as dependencies. These can be mapped to other parts of the model in the time and scope of a transformation. "Proof obligations" are used to keep models in a semantically consistent state: transformations generate new proof obligations and the user must prove these. These are stored as the part of the dependency model.

Ontologies
Hehenberger et al [2] present an approach for consistency checking of mechatronic design models by using domain ontologies. Ontologies are meant to be a "structured and organized" way to represent domain knowledge and therefore, enable reasoning over multiple domains. The approach explicitly aims to managing semantic inconsistencies.

Rule based representation
Van Der Straeten et al [38] argue that inconsistency resolution is the key enabler of refactoring as a software engineering technique. Inconsistencies emerge as intermediate states are traversed during a complex refactoring step; the final state is reached via gradually eliminating these inconsistencies. The idea is highlighted by the step-by-step execution of the Move Operation refactoring activity. The paper also presents a rule-based technique for rule-based inconsistency resolution. The authors observe that the same inconsistencies occur in multiple refactoring scenarios and therefore, reusability of resolution strategies is acknowledged as a key factor. Resolution strategies are user-guided. The rule-based approach translates to LHS-RHS structures where the user-specified patterns (LHS) trigger the user-defined resolution strategy (RHS). As the motivation of the paper comes

from model/code refactoring, the takeaways are really syntax-oriented; semantic inconsistencies are not addressed. “Object-oriented modeling languages” (and UML, in particular) are the application domain of the presented resolution technique.

Quinton et al [34] supports the evolution of cardinality-based feature models. Specifically, range inconsistencies are handled, i.e. the cases when no product exists for some values of a range of a feature cardinality. The authors identify key scenarios upon add/remove/update/move operations which can lead to range inconsistencies.

Formal methods
Engels et al [21] present a prototype framework for modeling consistency constraints, specifically in UML-based software development. The idea behind the approach is to capture consistency rules by arbitrary formal techniques and then evaluate those over input models. The user has to choose the appropriate formal technique and provide a mapping from and to the input models. As a case study is presented where CSP serves as the formal underpinning. Mappings are captured via graph transformation rules. An extensible catalogue of various consistency problems is provided by the framework to support reusability of (in)consistency concepts.

Giese et al [24] presents an inconsistency *avoidance* technique in the context of mechatronic system design by safe composition of systems.

Logic-based approaches
Van de Straeten et al [37] use description logic to characterize and evaluate inconsistent model states. The approach mainly targets inconsistencies arising during model evolution. Introduces horizontal and evolution consistency; horizontal defined as: consistency between different models within the same version; evolution consistency defined as: consistency between different versions of the same model. A classification of inconsistencies is provided. Description logic is used to depict ontological relationships among model elements. Loom is used as a description logic tool. ICM is achieved by a UML profile. Unfortunately, the technique has a strong focus on UML and its applicability in a broader modeling domain is questionable.

Lopez-Herrejon et al [30] pose the research question “where should the fixes be placed?” rather then when and how to fix. Possible configurations are mapped to formulas of propositional logic. Consistency rules are captured as well-formedness rules using logical formulas. Inconsistencies arise due to changes in the feature models, although removing elements is not permitted. A metric called “pair-wise commonality” is used to express how many configurations a feature appears in. Given a faulty feature F, the algorithm selects the features with the highest pwc value, since these are most likely to suffer from inconsistencies in this case. The efficiency of the heuristic is demonstrated via measurements. The repair actions, although, are constrained to adding missing features.

4.1.2 Specification patterns

The most common approach to specify inconsistency rules is to manually inspect models and make links between the appropriate model elements. Spanoudakis [5] refers to this approach as *human inspection*. The vast majority of the state-of-the-art considered in this report relies on human inspection.

As an attempt for automation, Herzig et al [26] presents a characterization technique based on pattern matching and **similarity metrics** defined between different (graph-like) models. Model characteristics, such as entity and property names, property units, relationship names and cardinalities, etc are used to calculate similarity between sets of different models. This enables identifying overlapping elements which are not necessarily connected via syntactic link, but still represent the same or related information. It builds on the premise that models of similar meaning are represented similarly in terms of naming, structure, dimensions, etc. The characterization rules are derived automatically, although the intervention of a human domain expert is inevitable in this case also. This technique, however, is the most advanced automation technique in terms of specification in the state of the art.

4.2 Detection

In non-prevention style techniques, inconsistent situations have to be detected based on the characterization of inconsistencies in order to execute the proper resolution actions. Typical patterns are the *human-centered collaborative exploration* and the *specialized forms of automated analysis* [5].

In general, the patterns of characterization heavily influence the operational patterns of detection, i.e. the means the detection is executed by.

Techniques employing the dependency graph representational pattern (Section 4.1.1.2), typically rely on the related techniques of **graph reasoning**, such as graph querying, transformations, pattern matching, etc. Hamlaoui et al [25] use EMFCompare [42] to enrich the correspondence model by change deltas among various models. Giese et al [23] employ the FUJABA framework to transform TGGs in consistency-preserving bidirectional synchronization scenarios. The majority of techniques relying on a graph-based representation pattern also uses some form of model transformations, such as Qamar et al [3] and Adourian et al [13] for change propagation, and Mens et al [31] for model-dependency analysis.

Rule-based detection is used in combination with dependency graphs by Becker et al [15]. While consistency rules are defined via TGGs among the related models, the detection achieved by the PROGRESS rule engine [**?**].

Numerous techniques rely on the **solver-based detection** pattern, typically using a variant of Prolog for tooling. [27] employ this patten in combination with naming conventions (Section 4.1.1.1), using SmartEMF, an EMF extension framework, which is actually a Prolog fact base. This fact base captures models in EMF (MOF) terms. Basic EMF operations are extended by propagating changes into the fact base. Constraints are captured by Prolog terms. The evaluation is implemented via higher-order queries (using call predicate).

Both Blanc et al [41] and Le Noire et al [28] use Prolog in combination with operation-based representation (Section 4.1.1.4). In the former case, edit operations are recorded and traces are fed to a Prolog engine, resulting in an on-demand, batch-like inconsistency check approach. The latter approach builds on SWI-Prolog. The authors compare the time- and space complexity of the detection phase in operation-based and standard representation patterns: no difference in batch mode has been identified, while in incremental mode "some" advantages have been found. (E.g. batch rules can be used without rewriting them).

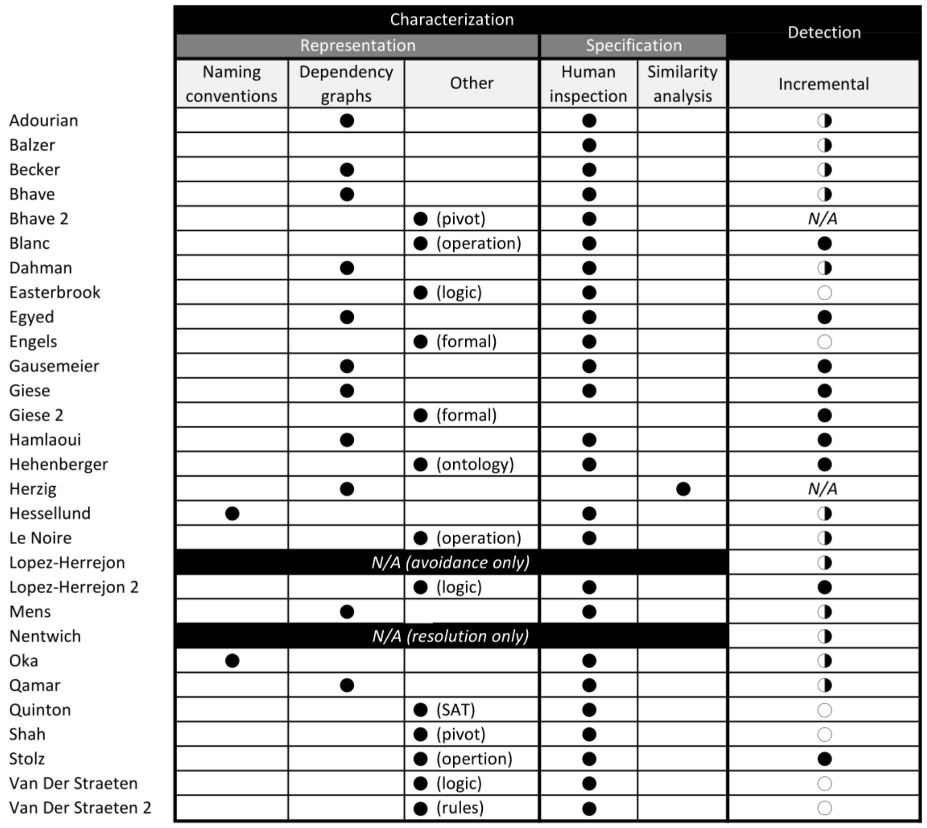
Quinton et al [34] use SAT solvers to evaluate sequences of edit operations in operation-based representation scenarios. Model changes are translated to the BR4CP/Aralia language [43] and are fed to a SAT solver.

While there is not much work on it, the **ontological reasoning** detection pattern fits well with ontology-based inconsistency characterization (Section 4.1.1.4).

Although this pattern is not found in the state of the art, techniques such as Hehenberger et al [2] could employ it.

The **incremental evaluation** pattern is one of the typical requirements for ICM techniques described by Becker et al [15]. Some authors explicitly aim to exploit the benefits of such an approach (Egyed et al [20], Giese et al [23], Gausemeier et al [22]). While not always emphasized, incremental evaluation is usually handled as a desirable property of ICM techniques.

In Table 4.1, we summarize the patterns supported by the ICM techniques investigated in this paper.



	Characterization					Detection
	Representation			Specification		
	Naming conventions	Dependency graphs	Other	Human inspection	Similarity analysis	Incremental
Adourian		●		●		◑
Balzer				●		◑
Becker		●		●		◑
Bhave		●		●		◑
Bhave 2			● (pivot)	●		*N/A*
Blanc			● (operation)	●		●
Dahman		●		●		◑
Easterbrook			● (logic)	●		○
Egyed		●		●		●
Engels			● (formal)	●		○
Gausemeier		●		●		●
Giese		●		●		●
Giese 2			● (formal)			●
Hamlaoui		●		●		●
Hehenberger			● (ontology)	●		●
Herzig		●			●	*N/A*
Hessellund	●			●		◑
Le Noire			● (operation)	●		◑
Lopez-Herrejon	*N/A (avoidance only)*					◑
Lopez-Herrejon 2			● (logic)	●		●
Mens		●		●		◑
Nentwich	*N/A (resolution only)*					◑
Oka	●			●		◑
Qamar		●		●		◑
Quinton			● (SAT)	●		○
Shah			● (pivot)	●		○
Stolz			● (opertion)	●		●
Van Der Straeten			● (logic)	●		○
Van Der Straeten 2			● (rules)	●		○

Table 4.1: *ICM patterns in the state of the art. (Legend: ○: no support; ◑: possible support, not emphasized; ●: dedicated support)*

4.3 Resolution

Resolution patterns address detected inconsistencies. Depending on their intent, changing and non-changing actions are distinguished [5], i.e. the ones aiming to bring the models back to a consistent state, and the ones aiming to notify stakeholders about inconsistencies and trigger further (manual) evaluation, respectively. From an automation point of view, manual and semi-automated patterns can be distinguished. (We argue that full automation cannot be achieved in general cases and human intervention - especially from the domain experts and stakeholders - is inevitable.) Additionally, Gausemeier et al [22] distinguish between operations that required user interaction and the ones that are straightforward to execute, i.e. no further inconsistency is introduced by the resolution.

Model synchronization and **change propagation** are commonly used patterns for automating inconsistency resolution. [22, 18, 13, 23, 32, 3, 31] As a prerequisite, appropriate detection algorithms are required to be deployed, which are capable to signal inconsistent states between the participating models. (E.g. by graph reasoning (Section 4.2).)

Editing hints are also a commonly used pattern, although requiring more human participation. During the resolution phase, the user is provided by resolution/editing hints by the modeling environment. The environment is responsible only for generating and visualizing the hints; it is the user who decides which one of the hints to apply. Hessellund et al [27] present a prototype tool that queries a Prolog fact base for valid model change operations and these are presented in a pop-up view. Hegedus et al [44] present a technique for generating quick fixes for DSLs.

Design-space exploration is another pattern that can be used in combination with editing hints and explicitly model resolution processes. The modeling environment is integrated with an appropriately configured DSE engine, which is responsible to generate an exhaustive list of resolution strategies. An example is provided by David et al [45].

The logic-based representational pattern (Section 4.1.1.4) is often used in conjunction with similarly **logic-based** resolution patterns, such as in Van der Straeten et al [37].

4.4 Optional activities

As Figure 3.1 shows, apart from the mandatory activities discussed previously, there are optional activities an ICM technique may incorporate. At this place, we focus on tolerance, tracking and process optimization.

4.4.1 Tolerance

Finkelstein et al [9] hint that instead of just removing an inconsistency, we have to reason about *managing* them. In our view, this also includes *tolerating* detected inconsistencies before executing potentially costly resolution actions, i.e. postponing the resolution activity as much as viable in order to allow the potential resolution of *transient* inconsistencies without intervention.

Blanc et al [41] propose to represent models by sequences of elementary construction operations. To detect structural constraints, they define pairs of operations, where the second operation cancels out first one. E.g. creating and subsequently deleting a model element. Although the authors do not investigate it, the approach provides foundations for **temporal inconsistency tolerance**. In these scenarios, inconsistencies are tolerated based on temporal and timing relations of elementary or compound model changes, or operations.

Balzer et al [14] focus on augmenting instances of inconsistencies with *state*. The authors propose an approach which keeps track of violated constraints (by using "pollution markers") and instead of instantly initiating a resolution action, concerned processes (and stakeholders) can be notified about the inconsistency. Easterbrook et al [19] propose a similar technique.

4.4.2 Tracking and process optimization

The management of detected inconsistency rules also may involve storing relevant information upon detection, such as the affected model element, timestamp, etc. This data can be later reused in various ways, for example

- revising inconsistency rules,
- fine-tuning the tolerance and resolution approach,
- optimizing the engineering process in order to minimize additional costs arising from inconsistencies.

These patterns, however, are not properly addressed by the state of the art.

Chapter 5

Conclusions

After investigating the broad topic of inconsistencies in multi-model mechatornic/CPS settings, we draw the conclusions in this section and answer the questions posed in Chapter 1.

What methods, techniques and tools are available for the management of consistency in a mechatronic and CPS design process?

In Chapter 3 we addressed this question in greater details. We found that there are only a few ICM techniques efficiently supporting the design of CPS/mechatronic systems. These foundational works focus mainly on characterizing inconsistencies as a necessary first step towards ICM. The majority of the ICM techniques of the state-of-the-art have their roots in software system modeling and their applicability in CPS/mechatronic design processes is questionable.

What are limitations of the state-of-the-art?

Focus on software system modeling and UML in particular. As mentioned earlier, the main limitation of the state-of-the-art is the focus on software system modeling and the lack of techniques to reason about physics in terms of inconsistencies. A majority of the publications considered in this survey assume UML as the de facto modeling language and implicitly tie the techniques and approaches to UML. Even though UML is supposed to be a general-purpose modeling language, it's applicability in CPS/mechatronic design is marginal, as only a small set of design tasks are suitable to be efficiently addressed by UML or one of its variants (SysML, MARTE, etc). This strong relation to UML makes most of the state-of-the-art techniques unfeasible to apply in CPS/mechatronic design.

Tolerance is an optional activity in allow-and-recover style of ICM techniques, situated between detection and resolution. Its various forms (temporal, design, spatial) enable better runtime performance and better scalability of the overall design process. Inconsistency tolerance has been addressed sporadically by the state of the art and there is no comprehensive, structured approach on the topic.

Explicitly modeled resolution processes. While many authors propose semi-automated human-guided resolution techniques (Hessellund et al [27], Hegedus et al [44]), the appropriate resolution actions typically require complex sequences of changes. Explicitly modeling resolution actions is a missing link in the state of the art that.

Process optimization. By reusing the data on encountered (i.e. detected and resolved) inconsistencies, the overall engineering process can be optimized. For this purpose, appropriately designed databases and smart analysis techniques are required. This activity, however, is not addressed by the state of the art.

Figure 5.1 summarizes our view on efficient ICM techniques in the mechatronic domain, which is based on our survey of the state of the art and extended by our suggestions.

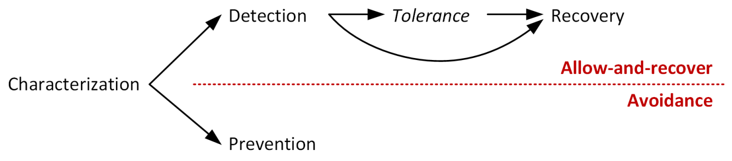


Figure 5.1: *Extended view on ICM styles and techniques*

What makes consistency management in a mechatronic and CPS context different from the software engineering industry in general?

The main difference between modeling pure software systems and mechatronic systems is the involvement of models of physics in the latter case. Incorporating physics in the design of a virtual product links the whole design process to a belief system with inherited semantics. In contrary, the semantics of a pure software system are fully to be defined. Reasoning about inherited semantics, and especially: modeling the behavior of inherited semantics is what makes mechatronic design radically different from software design. The involvement of physics is also the reason of many software- and UML-oriented ICM techniques being unfeasible to apply in mechatronics.

Bibliography

[1] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.

[2] P. Hehenberger, A. Egyed, and K. Zeman, “Consistency checking of mechatronic design models,” in *ASME 2010 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 1141–1148, American Society of Mechanical Engineers, 2010.

[3] A. Qamar, J. Wikander, and C. During, “A mechatronic design infrastructure integrating heterogeneous models,” in *Mechatronics (ICM), 2011 IEEE International Conference on*, pp. 212–217, April 2011.

[4] “Merriam-Webster Dictionary and Thesaurus.” `http://merriam-webster.com`. Accessed: 2015-11-20.

[5] G. Spanoudakis and A. Zisman, “Inconsistency management in software engineering: Survey and open research issues,” in *in Handbook of Software Engineering and Knowledge Engineering*, pp. 329–380, World Scientific, 2001.

[6] S. J. Herzig and C. J. Paredis, “A conceptual basis for inconsistency management in model-based systems engineering,” *Procedia {CIRP}*, vol. 21, pp. 52 – 57, 2014. 24th {CIRP} Design Conference.

[7] Z. Huzar, L. Kuzniarz, G. Reggio, and J. L. Sourrouille, “Consistency problems in uml-based software development,” in *Proceedings of the 2004 International Conference on UML Modeling Languages and Applications*, UML’04, (Berlin, Heidelberg), pp. 1–12, Springer-Verlag, 2005.

[8] J. Reineke and S. Tripakis, “Basic problems in multi-view modeling,” in *Tools and Algorithms for the Construction and Analysis of Systems* (E. brahm and K. Havelund, eds.), vol. 8413 of *Lecture Notes in Computer Science*, pp. 217–232, Springer Berlin Heidelberg, 2014.

[9] A. Finkelstein, “A foolish consistency: Technical challenges in consistency management,” in *Database and Expert Systems Applications* (M. Ibrahim, J. Kng, and N. Revell, eds.), vol. 1873 of *Lecture Notes in Computer Science*, pp. 1–5, Springer Berlin Heidelberg, 2000.

[10] R. Hanmer, *Patterns for Fault Tolerant Software.* Wiley Publishing, 2007.

[11] F. J. Lucas, F. Molina, and A. Toval, "A systematic review of uml model consistency management," *Inf. Softw. Technol.*, vol. 51, pp. 1631–1645, Dec. 2009.

[12] R. Van Der Straeten, *Inconsistency Management in Model-Driven Engineering: An Approach using Description Logics.* PhD thesis, Vrije Universiteit Brussel, Software Languages Lab, 2005.

[13] C. Adourian and H. Vangheluwe, "Consistency between geometric and dynamic views of a mechanical system," in *Proceedings of the 2007 Summer Computer Simulation Conference*, SCSC '07, (San Diego, CA, USA), pp. 31:1–31:6, Society for Computer Simulation International, 2007.

[14] R. Balzer, "Tolerating inconsistency," *[1991 Proceedings] 13th International Conference on Software Engineering*, pp. 158–165, 1991.

[15] S. M. Becker and A.-T. Körtgen, "Integration tools for consistency management between design documents in development processes," in *Graph Transformations and Model-driven Engineering* (G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, and B. Westfechtel, eds.), pp. 683–718, Berlin, Heidelberg: Springer-Verlag, 2010.

[16] A. Bhave, B. Krogh, D. Garlan, and B. Schmerl, "View consistency in architectures for cyber-physical systems," in *Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on*, pp. 151–160, April 2011.

[17] A. Bhave, B. Krogh, D. Garlan, and B. Schmerl, "Multi-domain modeling of cyber-physical systems using architectural views," *AVICPS 2010*, p. 43, 2010.

[18] K. Dahman, F. Charoy, and C. Godart, "Towards consistency management for a business-driven development of soa," in *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pp. 267–275, Aug 2011.

[19] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh, "Coordinating distributed viewpoints: the anatomy of a consistency check," *Concurrent Engineering*, vol. 2, no. 3, pp. 209–222, 1994.

[20] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *Software Engineering, IEEE Transactions on*, vol. 37, pp. 188–204, March 2011.

[21] G. Engels, R. Heckel, and J. Kster, "The consistency workbench: A tool for consistency management in uml-based development," in *UML 2003 - The Unified Modeling Language. Modeling Languages and Applications* (P. Stevens, J. Whittle, and G. Booch, eds.), vol. 2863 of *Lecture Notes in Computer Science*, pp. 356–359, Springer Berlin Heidelberg, 2003.

[22] J. Gausemeier, W. Schäfer, J. Greenyer, S. Kahl, S. Pook, J. Rieke, *et al.*, "Management of cross-domain model consistency during the development of advanced mechatronic systems," *DS 58-6: Proceedings of ICED 09, the 17th International Conference on Engineering Design, Vol. 6, Design Methods and Tools (pt. 2), Palo Alto, CA, USA, 24.-27.08. 2009*, 2009.

[23] H. Giese and S. Hildebrandt, "Incremental model synchronization for multiple updates," in *Proceedings of the Third International Workshop on Graph and Model Transformations*, GRaMoT '08, (New York, NY, USA), pp. 1–8, ACM, 2008.

[24] H. Giese, S. Burmester, W. Schäfer, and O. Oberschelp, "Modular design and verification of component-based mechatronic systems with online-reconfiguration," in *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, SIGSOFT '04/FSE-12, (New York, NY, USA), pp. 179–188, ACM, 2004.

[25] M. El Hamlaoui, S. Ebersold, B. Coulette, M. Nassar, and A. Anwar, "Heterogeneous models matching for consistency management," in *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*, pp. 1–12, May 2014.

[26] S. J. Herzig and C. J. Paredis, "Bayesian reasoning over models," in *11th Workshop on Model Driven Engineering, Verification and Validation MoDeVVa 2014*, p. 69, 2014.

[27] A. Hessellund, K. Czarnecki, and A. Wsowski, "Guided development with multiple domain-specific languages," in *Model Driven Engineering Languages and Systems* (G. Engels, B. Opdyke, D. Schmidt, and F. Weil, eds.), vol. 4735 of *Lecture Notes in Computer Science*, pp. 46–60, Springer Berlin Heidelberg, 2007.

[28] J. Le Noir, O. Delande, D. Exertier, M. da Silva, and X. Blanc, "Operation based model representation: Experiences on inconsistency detection," in *Modelling Foundations and Applications* (R. France, J. Kuester, B. Bordbar, and R. Paige, eds.), vol. 6698 of *Lecture Notes in Computer Science*, pp. 85–96, Springer Berlin Heidelberg, 2011.

[29] R. Lopez-Herrejon and A. Egyed, "Detecting inconsistencies in multi-view models with variability," in *Modelling Foundations and Applications* (T. Khne, B. Selic, M.-P. Gervais, and F. Terrier, eds.), vol. 6138 of *Lecture Notes in Computer Science*, pp. 217–232, Springer Berlin Heidelberg, 2010.

[30] R. E. Lopez-Herrejon and A. Egyed, "Towards fixing inconsistencies in models with variability," in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, (New York, NY, USA), pp. 93–100, ACM, 2012.

[31] T. Mens, R. Van Der Straeten, and M. DHondt, “Detecting and resolving model inconsistencies using transformation dependency analysis,” in *Model Driven Engineering Languages and Systems* (O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, eds.), vol. 4199 of *Lecture Notes in Computer Science*, pp. 200–214, Springer Berlin Heidelberg, 2006.

[32] C. Nentwich, W. Emmerich, and A. Finkelstein, “Consistency management with repair actions,” in *Software Engineering, 2003. Proceedings. 25th International Conference on*, pp. 455–464, May 2003.

[33] A. Oka, S. Yamamoto, and S. Isoda, “Consistency management for software design information repository,” in *Computing and Information, 1993. Proceedings ICCI ’93., Fifth International Conference on*, pp. 579–585, May 1993.

[34] C. Quinton, A. Pleuss, D. L. Berre, L. Duchien, and G. Botterweck, “Consistency checking for the evolution of cardinality-based feature models,” in *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC ’14, (New York, NY, USA), pp. 122–131, ACM, 2014.

[35] A. A. Shah, A. A. Kerzhner, D. Schaefer, and C. J. J. Paredis, “Multi-view modeling to support embedded systems engineering in sysml,” in *Graph Transformations and Model-driven Engineering* (G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, and B. Westfechtel, eds.), pp. 580–601, Berlin, Heidelberg: Springer-Verlag, 2010.

[36] V. Stolz, “An integrated multi-view model evolution framework,” *Innovations in Systems and Software Engineering*, vol. 6, no. 1-2, pp. 13–20, 2010.

[37] R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers, “Using description logic to maintain consistency between uml models,” in *UML 2003 - The Unified Modeling Language. Modeling Languages and Applications* (P. Stevens, J. Whittle, and G. Booch, eds.), vol. 2863 of *Lecture Notes in Computer Science*, pp. 326–340, Springer Berlin Heidelberg, 2003.

[38] R. Van Der Straeten and M. D’Hondt, “Model refactorings through rule-based inconsistency resolution,” in *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC ’06, (New York, NY, USA), pp. 1210–1217, ACM, 2006.

[39] A. Schürr, “Specification of graph translators with triple graph grammars,” in *Graph-Theoretic Concepts in Computer Science*, pp. 151–163, Springer, 1995.

[40] H. Giese and R. Wagner, “Incremental model synchronization with triple graph grammars,” in *Model Driven Engineering Languages and Systems* (O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, eds.), vol. 4199 of *Lecture Notes in Computer Science*, pp. 543–557, Springer Berlin Heidelberg, 2006.

[41] X. Blanc, I. Mounier, A. Mougenot, and T. Mens, "Detecting model inconsistency through operation-based model construction," in *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, pp. 511–520, May 2008.

[42] Eclipse Foundation, "EMFCompare." `https://www.eclipse.org/emf/compare/`. Accessed: 2015-11-20.

[43] A. Rauzy and Y. Dutuit, "Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within aralia," *Reliability Engineering & System Safety*, vol. 58, no. 2, pp. 127 – 144, 1997. {ESREL} '95.

[44] A. Hegedus, A. Horvath, I. Rath, M. Branco, and D. Varro, "Quick fix generation for dsmls," in *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pp. 17–24, Sept 2011.

[45] I. Dávid, I. Ráth, and D. Varr, "Foundations for Streaming Model Transformations by Complex Event Processing," *Software and Systems Modeling.*