Towards inconsistency tolerance by quantification of semantic inconsistencies

Reference:
David Istvan, Syriani Eugene, Verbrugge Clark, Buchs Didier, Blouin Dominique, Cicchetti Antonio, Vanherpen Ken.- Towards inconsistency tolerance by quantification of semantic inconsistencies
Proceedings of the 1st International Workshop on Collaborative Modelling in MDE (COMMitMDE 2016) co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2016), October 4, 2016, St. Malo, France / Muccini, Henry [edit.]; et al. - ISSN 1613-0073 - 2016, p. 35-44

Towards Inconsistency Tolerance by Quantification of Semantic Inconsistencies

István Dávid
University of Antwerp - Flanders' Make, Belgium
istvan.david@uantwerpen.be

Eugene Syriani
University of Montreal, Canada
syriani@iro.umontreal.ca

Clark Verbrugge
McGill University, Canada
clump@cs.mcgill.ca

Didier Buchs
University of Geneva, Switzerland
didier.buchs@unige.ch

Dominique Blouin
HPI, University of Potsdam, Germany
Telecom ParisTech School, France
dominique.blouin@telecom-paristech.fr

Antonio Cicchetti
Mälardalen University, Sweden
antonio.cicchetti@mdh.se

Ken Vanherpen
University of Antwerp - Flanders' Make, Belgium
ken.vanherpen@uantwerpen.be

ABSTRACT

Due to the increase of their complexity, currently engineered systems cannot be developed by one individual, but are a product of a collaboration between multiple stakeholders who develop the system from different domain-specific views. Inconsistencies between views, however, hinder collaboration and therefore, must be managed. Since the encountered inconsistencies may potentially disappear as the natural consequence of a design workflow, tolerating them to a given extent may be desirable and can lead to a more efficient collaboration. A key to reason about tolerance is the quantification of the impact of single inconsistencies on the overall system design.

In this paper we present a quantification model for semantic inconsistencies in discrete and continuous systems. We investigate characteristic behavioral patterns of inconsistencies based on this model and identify the links with various forms of tolerance. Finally, we discuss the directions of further expanding the approach required for a comprehensive inconsistency tolerance technique.

Keywords

inconsistency management, inconsistency tolerance, temporal tolerance, multi-view modeling, collaborative modeling

1. INTRODUCTION

The complexity of current engineered systems has increased drastically over the last decades. Pertinent examples are today's mechatronic and cyber-physical systems (CPS). Due to their complexity, these systems are no longer engineered by a single individual, but rather by the collaboration of experts. Such collaborative endeavors involve stakeholders from different domains, who bring their point of view on the system to be built, resulting in typical settings of multi-tool and multi-view modeling (MVM) [37, 47]. Collaborative environments are further characterized by multi-formalism and multi-abstraction, meaning, different formalisms and languages are used to support each team of stakeholders.

Models of different views must be kept consistent in order to develop a correct system. However, as stakeholders develop the respective domain-specific parts of the system independently from each other, models from these domains can become inconsistent. Following the definition of Herzig et al. [36], this means two or more statements about the system are made that are not jointly satisfiable.

While dealing with inconsistencies has been a well-researched topic, most approaches focus on maintaining consistency in terms of syntactic relationships between models [23, 25, 26, 40]. These approaches, however, fall short of properly addressing the problem of *semantic inconsistencies*, which are typical in collaborative settings where disparate formalisms of various domains are involved, e.g. when engineering mechanical and control parts of a CPS.

Finkelstein [21] suggests that instead of simply removing inconsistencies from the system from time to time, one should manage consistency. This entails characterization and detection of inconsistencies before resolving them, and the subsequent analysis. Inconsistencies are, however, stateful entities that might occur, evolve and later potentially disappear as the natural consequence of a design workflow. This gives room for temporarily *tolerating* them [4], i.e. allowing inconsistencies to exist for a period of time, which promises lower resolution costs by (i) postponing resolution to a more appropriate phase of the design process; or, in some cases, even (ii) completely avoiding resolution when specific inconsistencies get resolved on their own.

Deciding whether or not to intervene in an inconsistent situation requires information about (i) how severe the inconsistency of the whole model space is; and (ii) what are the chances that the inconsistency gets resolved without intervention. In this paper, we focus on the first question and present a quantitative approach for assessing the severity of semantic inconsistencies. For that, we formalize the notion of (in)consistency in multi-view modeling by defining a relation between a property of the whole system, the satisfaction of the property by individual views, and the consistency between views. We identify characteristic behavioral patterns of properties in terms of their consistency and use these patterns for guiding the tolerance. The results of this paper serve as formal foundations to implement a consistency management framework for collaborative modeling scenarios with multi-paradigm and multi-abstraction characteristics.

In Section 2, we give an overview on multi-view modeling along with an illustrative example used throughout the paper. In Section 3, we provide a quantitative model for assessing the severity of semantic inconsistencies. In Section 4, we show how the previously defined quantitative model can be extended to support tol-

erance of parameter inconsistencies and temporal tolerance of inconsistencies. We also discuss the directions of further expanding the approach required for a comprehensive inconsistency tolerance technique. Finally, we discuss the related work in Section 5 and conclude our paper with Section 6.

2. OVERVIEW ON MULTI-VIEW MODELING

In this section, we give an overview on the context of our work and provide a motivating example to illustrate the contributions of the paper.

2.1 Typical Scenarios in MVM

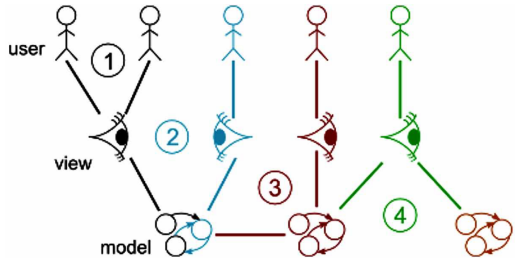

Figure 1: Scenarios in multi-view modeling.

Corley et al. [13] identify four typical collaborative modeling scenarios shown in Figure 1. In *Multi-User Single-View* settings (Scenario 1), users work on the same view of the model and observe the exact same data in the same concrete syntax. The other three scenarios are characteristically different as they feature multiple views or multiple models, which are the primary cause of emerging inconsistencies. In *Multi-View Single-Model* settings (Scenario 2), users work on different views of the model. The two views may use different concrete syntax. The views are projections of the same single underlying model (SUM) and therefore conform to the same modeling language. Changes in the abstract syntax of one view have to be reflected in the other view in order to retain consistency. The *Multi-View Multi-Model* scenario (Scenario 3) does not assume a SUM, but multiple separated underlying models, which are connected via their semantic domains. Manipulating the models throught the respective views causes mismatches in the models themselves and thus, inconsistencies may arise. Finally, the *Single-View Multi-Model* scenario (Scenario 4) is a special form of the Multi-View Multi-Model scenario, in which one view corresponds to multiple underlying models. In practice, collaborative modeling settings typically feature combinations of these scenarios.

The views can belong to different domains, i.e. they may represent various aspects of the SUM in different formalisms and on different levels of abstraction. A typical example is the mechanical design of a complex system where multi-body models and more detailed finite element models are used to reason about different aspects of the same virtual product.

Inconsistencies arise due to the *shared elements* between views: as one view changes an overlapping element, the change has to be propagated to the other views that share the same element, otherwise an inconsistency will occur [22, 27]. These related elements are not necessarily of syntactic nature, but they may exist in the *semantic domain* of the SUM as the *ontological properties* of the system, e.g., safety, energy-efficiency, autonomy, etc. The conformance of the system to such properties can be evaluated by simulations or model checking.

To reason about characteristics of the semantic domain of models (such as overlaps and linked properties), an explicit representation of the semantic domain has to be provided. In this paper, we consider models M expressed with an operational semantics $[\![M]\!]$, consisting of *traces* on given states.

2.2 Motivating Example

2.2.1 Collaboration models

Our example concerns the development of complex and heterogenous systems such as CPS involving the collaboration of several development teams. This collaboration can be supported by cloud-based model-based development environments such as the one presented in [13]. Such environments allow the definition of viewpoints specifying how different views of the system under development can be constructed to be used by the various teams to interact with the models of the system. These environments can typically support two different working modes. For the *simultaneous* mode, any update performed on a view is immediately sent to the server for being applied to the model and changes are reflected on all other depending views. This is the way well known collaborative tools work, such as Google Docs or Overleaf, which were used to write this paper collaboratively.

However this simultaneous mode may not meet the needs of all development organizations. For example, large and complex systems are often decomposed into subsystems developed by different entities (e.g., manufacturers, suppliers, subcontractors, etc.). Furthermore, due to intellectual property protection concerns, the models may never be allowed to cross the boundaries of an enterprise local network. Then, a system integrator needs to plan ahead of time a series of virtual integration steps where the models developed by all external entities are integrated and analyzed for various properties. Consistency of the system as a whole then needs to be checked as the models have evolved independently for a given period of time. According to the analysis results, decisions and updates will then be made to restore consistency. We call this mode *delayed commit*. The frequency of such virtual integration steps may vary but has to be carefully considered, e.g., to avoid the large costs potentially induced by the rework of highly inconsistent models. In this mode, providing a measure that quantitatively determines how inconsistent the models are is highly relevant.

2.2.2 The Automated Guided Vehicle

Our running motivational example is the collaborative model-based development of an Automated Guided Vehicle (AGV) (Figure 2). According to the stakeholder requirements specification, the AGV needs to be able to transport a number of objects by following a given trajectory with a given maximal tracking error. Furthermore, it needs to have a given autonomy as defined by the number of trajectories that can be followed before needing to recharge the AGV's empowering component. Finally, the mass of the AGV must not exceed a given value because, for example, it will be too heavy for the setting in which it is planned to operate.

To meet these requirements, an initial design is provided consisting of a custom mechanical frame (Figure 2), an off-the-shelf electrical battery, an off-the-shelf electrical motor, a drive train and a drive train control system. A set of viewpoints for this design is then defined in the collaborative environment to be used for design activities such as requirements specification, electric motor selection, battery selection, mechanical frame design, drive train design and drive train controller design. These are listed in the header row of Table 1.

	Requirements (V_R)	Mechanical Frame (V_{MF})	Battery (V_B)	Motor (V_M)	Drive Train (V_{DT})	Drive Train Control (V_{DTC})	Integrated Vehicle (V_{IV})
Battery Support Size		$=$ Battery Support Size	$\geq$ Battery Size				
Electrical Current Capacity			$\leq$ Battery Current Capacity	$\geq$ Minimum Current Capacity Required by Motor			
Power				$\leq$ Motor Power		$\geq$ Maximum Motor Power Required by Drive Train Control System (e.g. obtained from simulation)	
Mass	$\leq$ Maximum Vehicle Mass (obtained from a Requirement)	$>$ Frame Mass	$>$ Battery Mass	$>$ Motor Mass	$>$ Drive Train Mass	$\leq$ Maximum Vehicle Mass Supported by Drive Train Control System (e.g. obtained from simulation)	$=$ Vehicle Mass (computed from the sum of its constituents)

Table 1: Example consistency properties and statements about their values for each relevant viewpoint.

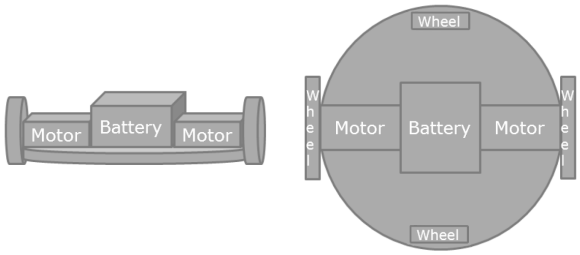

Figure 2: Overview of the Automated Guided Vehicle example.

2.3 System Properties and Consistency Statements

Herzig et al [36] define inconsistency as two or more statements that are not jointly satisfiable. In our approach, we provide such set of statements per system *property* and concerned *viewpoint* as defined in the cells of Table 1 for the AGV example. Overall consistency then consists in ensuring consistency for each given property of interest of the system. For a given property, consistency can be checked for each pair of viewpoints providing a *statement* about the property. From the two statements of the viewpoints, a compound statement can be derived about the consistency of both viewpoints.

For example, consider property $P = BatterySupportSize$ for the AGV (first row of Table 1). Let us assume that there is a single underlying model for the mechanical frame viewpoint and a separate model for the battery viewpoint. The battery support size provided by the mechanical frame view is then simply the value of the corresponding property in the underlying mechanical design model. On the battery view however, a less precise information is provided as all that can be stated for a consistent design is that it should be greater or equal to the size of the battery. Therefore, only a range of values can be inferred from the battery viewpoint. Composing these two statements, we derive the compoud statement $BatterySupportSize \geq BatterySize$, which can be evaluated at design time from the involved views V_{MF} and V_B to evaluate consistency. In other words, a value from the battery view outside the range provided by the mechanical frame view will indicate an inconsistent system.

Generalizing this example, we can say that for a given property P of the system under design, it is possible to automaticaly derive for each pair of viewpoints providing a statement about P a compound statement about the consistency of both views with respect to P. This compound statement will involve all properties involved in the composed statements. For example, for $P = BatterySupportSize$, we obtain a consistency statement about the pair of properties $BatterySupportSize$ and $BatterySize$. In Section 3, we further show how such statement can be adapted to provide a quantification of inconsistency.

It should be noted that such considerations are independent from the source of inconsistency. For example, it can originate from the actual values of the underlying models used by the views, but also simply from the fact that the views are outdated due to network latency or delayed commit between the views in the cloud-based development environment. In order to explore different types of inconsistency tolerance patterns as proposed in Section 4, we will therefore consider that the two different modes (simultaneous and delayed commit) for the cloud-based multi-view modeling environment are used for two different development processes. Example use cases for this are provided in Section 4.

2.4 Other Applicable Cases

While our motivating example concerns collaborative development, we emphasize here that what we provide is applicable to many other cases, thus making this proposal even more relevant. One of them is that of *runtime* models and views such as those employed in computer games implying a variety of consistency concerns. These are most apparent in network-based, multiplayer games, which are both realtime environments and distributed systems, and as such bring a number of consistency concerns due to network latency, lossy communication protocols, and bandwidth limitations. Even single player games induce consistency issues, with modern designs organized through a separation of domains in order to provide good, domain-specific performance and behaviour, a distinction quite visually evident in, for example, the way collision domains are simplified abstractions of the representation domain [20]. Games make a further interesting and motivational subject due to the way consistency is tolerated - the perception-oriented, entertainment focus of game software means subtle inconsistency is not necessarily a problem, and weaker or heuristic consistency guarantees are often viewed as an acceptable trade-off

for increased performance or broader scope in implementation.

3. A QUANTITATIVE MODEL FOR ASSESSING INCONSISTENCIES

Based on our running example, we can now provide a model for quantifying inconsistency. This can be achieved as explained previously for a given system property such as those listed in Table 1. We first consider consistency for a given system property P and two given views V_1 and V_2 each providing a statement about P. The consistency statement composed from the statements of V_1 and V_2 about P then provides a set of two properties that must be evaluated to evaluate the consistency statement. We then propose a metric combining these two properties to quantify the degree of inconsistency. Such measure can later be used for assessing the importance of the inconsistency and support its management.

3.1 Formal Definitions

The definition of consistency is based on a *distance function* that takes the two views and provides a measure value. In this simple setting we need a simple domain for measures that can be used in a compositional and associative way.

Definition 1. (Measure) A given set of values $\mathbb{M}$ with operations $0 :\rightarrow \mathbb{M}, + : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{M}$ (0 neutral, + associative and commutative) and an order relation $\leq$ on $\mathbb{M}$ is called a measure.

Definition 2. (State distance) Given two state domains Σ_1 and Σ_2, a distance is a function $\delta : \Sigma_1 \times \Sigma_2 \mapsto \mathbb{M}$.

From the state distance we can define by extension distances on finite prefix of traces. By trace, we mean a sequence of performance values [49] a semantic property exhibits as the underlying model changes. This distance is a natural extension that can express the observation on a trace with a finite window starting from the beginning of the trace. The size of the window λ is a parameter of such distance.

Definition 3. (Sequence of ordered points) A sequence of ordered points (or index on a trace) is a word $I = i_1, i_2, ..., i_n, ... \in \mathbb{N}$ s.t. $\forall k, i_k < i_{k+1}$.

Definition 4. (Window) Given a sequence of observations $O = o_1, o_2, \ldots, o_n$, a window w of length λ is defined as a prefix of length λ of O.

Definition 5. (Trace distance) Given two set of traces Σ_1^ω and Σ_2^ω generated by two properties ρ_1 and ρ_2, respectively, a distance on trace for a given window of lenght λ is a function $\delta_\lambda : \Sigma_1^\omega \times \Sigma_2^\omega \mapsto \mathbb{M}$ defined as:

$$\delta_\lambda(\rho_1, \rho_2) = \sum_{i=0}^{\lambda-1} \delta(\rho_1(i), \rho_2(i)),$$

where $\rho(i)$ denotes the ith observation of ρ.

The above definition can be extended to continuous systems as follows:

$$\delta_\lambda(\rho_1, \rho_2) = \int_{i=0}^{\lambda} \delta(\rho_1(i), \rho_2(i)) d_i.$$

Definition 6. (Consistency) Two views V_1 and V_2 are said to be consistent iff

$$\forall \rho \in [\![M]\!]_{V_1 \times V_2} : \delta_\lambda(\rho|_{V_1}, \rho|_{V_2}) = 0.$$

That is, V_1 and V_2 are consistent iff every property ρ has the same value in both views.

3.2 Measuring Distances

Inconsistency management requires providing a meaningful consistency measure given the two properties and the states for evaluating them. Such measure should take into account the amount of development work required to fix the inconsistency. This depends on several factors such as which activities must be performed to fix the design, which in turn depends on which part(s) of the design will be changed to resolve the inconsistency (e.g. for our example battery selection or battery support from mechanical design or both). One way to evaluate this is to take into account the cost of the involved development activities that will need to be performed. This set of activities can be determined by the graph of dependencies of the involved properties and their dependency to other properties. However, this complex task is left for future work. For now, we only provide illustrative simplified examples showing how simple measures can be derived for the different properties and viewpoints of Table 1.

For our AGV example, the aforementioned state trace is defined by the successive model states corresponding to the versions of the system design evolving through the activities performed along a development time axis.

3.2.1 Battery Support Size

For the battery support size property, let us assume that the sections of the battery and its support are of circular shape and can therefore be characterized by a single numerical value of the section diameter. We compose the statements of viewpoints V_{MF} and V_B of Table 1 to obtain a Boolean consistency statement about the pair of properties $BatterySupportSize$ and $BatterySize$ as:

$$\rho_{BatterySize} \leq \rho_{BatterySupportSize}$$

where $\rho_{BatterySupportSize}$ and $\rho_{BatterySize}$ are the property values obtained from V_{MF} and V_B respectively. In such case, a simple measure of inconsistency can be given assuming that the amount of rework required for changing the battery support size of the mechanical design to account for the battery size is proportional to the difference in the diameters of the batttery and battery support. Thus, the consistency statement can be transformed to provide a metric for the amount of inconsistency as the difference between the battery and battery support diameters:

$$\delta = |\rho_{BatterySize} - \rho_{BatterySupportSize}|$$

3.2.2 Vehicle Mass

For the AGV mass property, we first consider the statements from the viewpoint of the drive train control system V_{DTC} and the integrated vehicle mechanical design V_{IV}. In a similar process as for the battery support size we derive the consistency statement:

$$\rho_{VehicleMass} \leq \rho_{MaxVehicleMassDriveTrainControl}$$

And with similar simplification as for the battery support size property that the amount of rework to account for fixing an inconsistent design is proportional to the difference between the masses, we derive the measure:

$$\delta = |\rho_{VehicleMass} - \rho_{MaxVehicleMassDriveTrainControl}|$$

4. TOWARDS TOLERANCE OF INCONSISTENCIES

In this section we discuss how the quantitative model in Section 3 can be used to tolerate semantic inconsistencies. We support the notion of two types of inconsistency tolerance.

In *parameter tolerance*, the definition of inconsistency is weakened to allow slight deviations from desired values of parameters, i.e. performance values of semantic properties [49]. In *temporal tolerance* scenarios, semantic inconsistencies are allowed for a certain amount of time before intervening and resolving inconsistencies, making use of the fact that inconsistencies might occur, evolve and later potentially disappear as the natural consequence of a design workflow.

4.1 Tolerating Parameter Inconsistencies

The idea of parameter tolerance embraces the presence of slight deviations from desired values of properties while still considering the situation a consistent one. In our specific case this means the distance between two views (properties) is not exactly 0, but stays within a parameter β.

Parameter tolerance enables reasoning about *how much* deviation is to be tolerated. For this purpose, Definition 6 can be weakened as follows.

Definition 7. (Bounded consistency) V_1 and V_2 are β-bounded consistent for M iff

$$\forall \rho \in [\![M]\!]_{V_1 \times V_2} : \delta_\lambda(\rho|_{V_1}, \rho|_{V_2}) \leq \beta,$$

where β is a measure. (See Definition 1.) The above definition allows the distance of two properties being within β, instead of restricting it to 0.

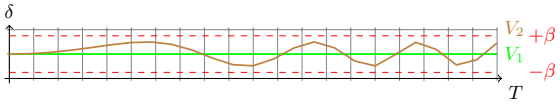

Figure 3: β-bounded consistency of two views.

From our running example, this occurs for the mass property when a high level of abstraction model of the drive train control system is used that, when executed, does not meet real-time deadlines for the given inertia due to the mass of the AGV. While the high level of abstraction model is functionally correct, it does not react fast enough due to poor execution time. The inconsistency with the mass obtained from the integrated vehicle view as evaluated according to the equation of Section 3.2.2 giving the measure of inconsistency may be tolerated at this level of abstraction up to a given bound β, otherwise it is well known by an experienced designer that no implemented system can ever execute fast enough for the given AGV inertia.

Such parameter inconsistency tolerance is a frequently encountered technique in engineering practice, albeit without explicitly reasoning about it. It is the tacit domain knowledge of stakeholders (engineers) that allows tolerating parameter inconsistencies. The caveat is, however, that without proper formalizaion, the automation of tolerance mechanisms is not possible.

Contract-Based Design (CBD) [6, 17] can be seen as an enabling methodology to formalize an agreement upon the allowed parameter tolerances. By defintion, an agreement consists of a set of assumptions and guarantees, called a contract, defining under which conditions a system promises to operate satisfying desired parameters. Typically, a contract and its content, e.g. the boundaries of certain parameters, is defined prior to the design phase. When the relation between parameters is known, e.g. using an ontology [49], one can reason about the tolerated (in)consistency of parameters during design.

4.2 Temporal Tolereance of Inconsistencies

When inconsistencies emerge it may be also desirable to temporarily tolerate them, even if the extent of inconsistencies (i.e. the distance between views) exceeds the boundary β as defined previously. Temporal tolerance enables reasoning about *how long* a deviation is to be tolerated. For this purpose, we investigate characteristic behavioral patterns of inconsistencies in the temporal dimension. The patterns can be used in a *prescriptive* manner, i.e. specifying the expected behavior of properties and consider intervention when the expected behavior is not followed. In each pattern, we assume β-bounded consistency. (See Definition 7.)

4.2.1 Exact Consistency

The most simplistic behavioral pattern of properties wrt their consistency is when the consistency is maintained constantly.

Definition 8. (Exact consistency) V_1 and V_2 are exact consistent for M iff

$$\forall \rho \in [\![M]\!]_{V_1 \times V_2}, \forall t \in T : \delta^t_\lambda(\rho|_{V_1}, \rho|_{V_2}) \leq \beta.$$

That is, V_1 and V_2 are exact consistent iff the consistency is preserved constantly, i.e. for every timestamp t∈T.

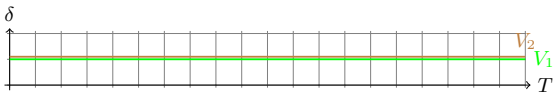

Figure 4: Exact consistency of two views with β = 0.

For our AGV example, this corresponds to the case where the cloud-based multi-view development environment is operating in simultaneous mode. In such case, the consistency of the properties defined in Table 1 is consistently monitored for each pair of views as the design of the system evolved and immediate resolving actions must be performed in case of any detected inconsistency.

In practice, it is however often not efficient to enforce exact consistency during system design, especially in complex multi-paradigm and multi-abstraction settings. The following patterns formalize cases more likely to be encountered.

4.2.2 Eventual Consistency

Eventual consistency guarantees that at some point in the future the two views become consistent.

Definition 9. (Eventual consistency) V_1 and V_2 are eventually consistent for M iff

$$\forall \rho \in [\![M]\!]_{V_1 \times V_2}, \exists t \in T : \delta^t_\lambda(\rho|_{V_1}, \rho|_{V_2}) \leq \beta.$$

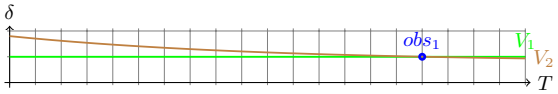

Figure 5: Eventual consistency of two views with β = 0.

For our AGV example, eventual consistency with $\beta = 0$ will be required when the drive train controller design is refined into implementation code and the combined code and execution platform result in meeting the real-time deadlines for the total mass of the AGV.

Demanding eventual consistency is also a typical scenario when parallel branches of design are to be integrated at some point in the design process. This leads to the definition of the repetitive and regular consistency patterns presented below.

4.2.3 Repetitive and Regular Consistency

Similarly to the eventual case, consistency of views may form a repetitive pattern, i.e. views may be consistent at given points in time, but may deviate in between.

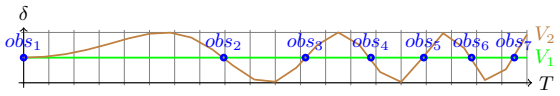

Figure 6: Repetitive consistency of two views.

Definition 10. (Repetitive consistency) V_1 and V_2 are repetitive consistent for M wrt a sequence of ordered points I (see Definition 3) iff

$$\forall \rho \in [\![M]\!]_{V_1 \times V_2}, \wedge_{i \in I} \delta_\lambda(\rho^i|_{V_1}, \rho^i|_{V_2}) \leq \beta.$$

For our AGV example, this corresponds to the case where the development of the different subsystems is subcontracted to external entities and virtual integration steps, at which consistency must be restored, are planned in advance in an aperiodic manner.

The repetitive pattern can be more regular w.r.t. to the number in the sequence, in this case it means that the index I is such that $\forall i \in \mathbb{N}, I(i) = \tau * i$.

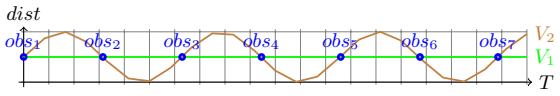

Figure 7: Regular consistency of two views.

For our AGV example, this corresponds to the same case as for repetitive consistency but with periodically planned virtual integration phases.

4.3 Discussion

The quantification approach of semantic inconsistencies described in this paper serves as foundations for a comprehensive inconsistency tolerance approach that can be adopted in state-of-the-art consistency management frameworks. Our work focuses on two types of inconsistency tolerance: parameter inconsistency tolerance and temporal tolerance in general.

Toleration methods for parameter inconsistencies can be directly implemented based on the theory described in Section 4.1. A requirement to such methods is the explicit notion of semantic traces, which can be achieved by explicit modeling of operational semantics, described e.g. in [16, 51].

In the following, we discuss the required extensions and future directions to the current results to make the foundations of temporal tolerance described in Section 4.2 applicable in complex collaborative modeling settings.

4.3.1 Explicitly Modeled Processes

Explicitly modeled processes enable various analyses required for reasoning about temporarily tolerating inconsistencies. First, they depict how models *evolve* throughout the development. The precedence relationships between various activities modifying the models are made clear, which helps understanding where and how inconsistencies occur exactly.

Deciding whether to resolve an inconsistency or further tolerate it, eventually depends on the added value a fixed inconsistency carries. If the cost of resolving an inconsistency is high, it may be more appropriate not to intervene and let inconsistencies exist for a while. The efficiency of resolution depends on numerous factors, one of them being the *formalisms* (i.e. domain-specific languages) used throughout the process. Some formalisms may support more efficient resolution of a specific type of inconsistencies than others and therefore, the analysis of the formalisms used in specific design steps may help identifying when it is most efficient to address the issue of detected inconsistencies. Richer process models enable explicit modeling of this information [14], along with the notions of actors and resources of the process [3].

4.3.2 Specification and Evaluation of Temporal Tolerance Rules

Specifying temporal tolerance rules makes only sense if those rules can be evaluated at runtime. Reasoning about temporal structures is traditionally addressed by various forms of temporal logic [24, 2, 41], which enable concise proofs and automated theorem proving with off-the-shelf software [29]. In nowadays' engineering practice, however, more high-level and domain-specific specification languages are desired. Such languages can be designed in accordance with the Dwyer specification patterns [18] for finite-state verification that occur commonly in the specification of concurrent and reactive systems. For evaluating temporal tolerance patterns, the techniques of trace matching and complex event processing can be used, as presented in previous work [15].

4.3.3 Automation of Tolerance Specification

Tolerance rules applied to single properties usually influence tolerance rules of other, related properties. For example, in the running example, the "battery size" is related to the "battery mass" and inconsistencies between "size" could be also imply inconsistencies in "mass", which may not be tolerated in the same manner. Manual reasoning about the influence relationships of tolerance rules will result in scalability issues in real, industrial-sized models. Automation of specifying tolerance rules is, therefore, a critical requirement for an efficiently tolerating inconsistencies. Combinatorial optimization techniques and design-space exploration can aid the process of tolerance specification, and can provide a mechanism for exploring incompatible tolerance rules.

4.3.4 Predictive Impact Analysis

The real challenge of answering whether or not to intervene at a specific point of the design process to resolve inconsistencies, is to understand how the inconsistencies can evolve in the subsequent steps of the process. Inconsistency resolution should be carried out only if the views are potential subjects of further diverging, but it should be postponed if the views can converge as a result of subsequent activities. With respect to the quantified distance metric, this raises the need for predictive techniques to be incorporated in our distance models to estimate how the impact of an inconsistency can evolve. In its most simplistic form, techniques of time series analysis can be employed [11]. In more complex scenarios, the propagation of inconsistencies may be required to be predicted too.

4.3.5 Resolution Scheduling

Another important cost-related factor is the resolution scheduling strategy of inconsistencies. Mens et al [35] investigated how a small number of inconsistencies can lead to hard resolution scheduling problems, including cyclic dependencies between inconsistencies, which make naïve resolution techniques fail. Consequently, the analysis of resolution plans is important to understand the feasibility of resolution and provides essential information on whether or not to execute resolution at the given time of the process.

4.3.6 Proving Global (In)Consistency

Proving the consistent (or inconsistent) state of the models is a natural extension of the quantification model. By evaluating local consistency and defining the algebraic rules of *composition* of views [6], information about the global consistency can be inferred in an automated fashion. Apart from the composition rules of views, proving global (in)consistency depends on how the semantic domains of views overlap as discussed in [37].

5. RELATED WORK

5.1 Inconsistency Management

Managing inconsistencies has been a well-studied topic in the domain of software engineering [46]. With the advent of complex engineered systems that require collaborative model development, addressing the topic of inconsistencies in a multi-view context became more relevant. Persson et al [37] identify consistency between the various views of cyber-physical system design as one of the main challenges in design of such complex systems. The outlook to the near-future of multi-view modeling by Hanxleden et al [50] further emphasizes the need for inconsistency management techniques in distributed, agile development settings with globalized DSLs and actor-oriented cloud based modeling tools.

Traditional approaches focus on maintaining consistency on the linguistic level, typically employing some form of correspondence models, such as dependency graphs [25, 26] and pivot models, e.g. by using SysML [44, 43] or EAST-ADL [8]. Qamar et al [38] introduce an inconsistency management approach by explicit modeling of inter- and intra-model dependencies. Dependencies are direct results of semantic overlaps and are used to notify stakeholders about possible inconsistencies when dependent properties change. Adourian et al [1] use triple-graph grammars (TGG) for maintaining consistency between geometric and dynamic properties of mechanical systems. Bhave et al [9] use an architectural base model as a pivot model and a set of model transformations for bi-directional mapping between various views and the base model.

As opposed to our work, the above techniques do not consider inconsistencies of purely semantic nature, i.e. the ones that do not manifest on the syntactic level. Multiple works focus on efficiently expressing semantic and ontological properties. Hehenberger et al [28] relate semantic concepts to the linguistic concepts of modeling by organizing structural design elements and their relations into a domain ontology to identify inconsistencies. Chechik et al [12] introduce the notion of approximate properties: linguistic properties expressed as graph patterns which are accurate enough to appropriately approximate a semantic property. These works can serve as complementary techniques to our work, to provide the semantic properties our technique can be applied on.

5.2 Measuring Inconsistency

Measuring inconsistency has been a topic of interest in the ontology and knowledge engineering. Hunter and Konieczny [30] approach measuring the level of inconsistency through the notion of minimal inconsistency sets. They show how their inconsistency metric is a special Shapley Inconsistency Value, which enables using SAT techniques for proving consistency and inconsistency. Ma et al [34] propose a technique to measure the degree of inconsistency in description logic based ontologies. As compared to these approaches, we mainly focus on collaborative modeling of complex engineered systems where the semantic inconsistencies of the models are not that obvious as in ontologies.

Inconsistency measurement approaches in software engineering settings typically focus on inconsistencies on linguistic level. Lange et al [33] use the number of detected instances of various inconsistency rules between UML diagrams as an inconsistency metric between views. Inconsistency rules are syntactic, such as messages in sequence diagrams without names.

Similar to our approach is the one presented by Barragáns-Martínez et al [5], who provide a formal framework to assess the significance of inconsistencies in requirement specifications. The scope of our work is more general as our technique is not constrained to requirement specifications but arbitrary models in a collaborative setting.

5.3 Temporal Tolerance

Balzer et al [4] introduces the notion of temporal tolerance by deconstructing inconsistency rules to two derived rules, the appearance and disappearance rule which span a temporal interval of the model(s) being in an inconsistent state, hence making inconsistencies stateful entities. By allowing further engineering activities to be executed during the inconsistent interval, the better parallelization of the design workflow can be achieved and ultimately, these may lead to the inconsistencies to be resolved without interrupting the design process for further reconciliation. As a limitation, the technique only deals with the most simplistic version of temporal consistency relations, in which a pair of subsequent operations form an identity transformation. In practice, more complex structures of operations have to be supported.

Easterbrook et al [19] propose a framework for temporal inconsistency tolerance in the context of multi-view modeling. Tolerating inconsistencies decouples the viewpoints and introduces flexibility in the design process as deciding upon *when to resolve inconsistencies* is the responsibility of the owner of the view. The authors provide a formal approach for guiding the decision in form of pairs of pre- and postconditions. Our approach extends this model by using a quantifiable distance metric to evaluate the divergence of views (and viewpoints). The distance metric also helps understanding the impact of unresolved inconsistencies and reason over their accumulation and evolution.

5.4 Consistency in Other Contexts

Consistency issues are of course well known in distributed and multiprocessor contexts, where program correctness is strongly dependent on understanding precise conditions under which the memories of different processors may differ. Lamport's seminal work on parallel computers, for instance, developed core notions of consistency as preserving *causality* between events [31], or in terms of interleaving sequential streams in the well known *Sequential Consistency* model [32]. A wide variety of less strict, or "relaxed" consistency models have since been defined, although they are often quite specific to the design decisions made in the underlying hardware [42]; Sorin et al.'s book provides a good overview [45] of memory model designs and issues.

A number of approaches to categorizing consistency have also been attempted. For distributed, virtual environments, Bouillot and Gressier-Soudan decompose consistency into elements of causality, concurrency, simultaneity, and instantaneity [10]. The latter 2 are difficult to achieve, and typically imply sacrificing the former 2 and living with short-term inconsistencies. Liu et al.'s survey paper organizes models based on their focus on *ultimate consistency* (systems which become eventually consistent), or on being *deadline-based* (given an event at time t, consistency is achieved at $t + \delta$). The former includes Lamport's basic models, as well as various similar forms of *serializability* [7], while the latter can be further broken down into perceptive (or absolute) consistency [10], where every process executes events at the same absolute time, *delayed consistency* [39], which imposes a fixed, maximum pair-wise delay

of $\delta(i, j)$ between processes i and j, and *timed consistency* [48], which requires a fixed global bound of δ on all processes. In this terminology our definition of eventual consistency is an instance of ultimate consistency, exact consistency a kind of absolute consistency, and regular consistency is deadline-based. Repetitive consistency and bounded consistency introduce new ideas based on a flexible notion of delay, and a formal distance metric for relative consistency.

6. CONCLUSION

In this paper, we presented a quantification model for semantic inconsistencies in multi-view settings of collaborative system design. The model serves as the foundation of a more comprehensive inconsistency tolerance methodology. By elaborating on characteristic examples we concluded that the model is well-suited for supporting the notion of parameter inconsistency tolerance and also tolerating inconsistencies in the temporal aspect.

This paper reported the results of a work in progress and therefore, the links to the state-of-the-art have been explored, as well as the further directions to be investigated. One particularly important direction concerns the measure of level of inconsistency. Our simplified example measures need to be extended to include a study on how to relate a measure of inconsistency with the cost of fixing it given an actual design, the other dependent consistency properties and their dependency graph, and given an inconsistency resolution plan expressed as a sequence of activities to be performed on the involved views. Adequate modeling of all these artifacts and corresponding analysis methods will be required to better support this task.

Acknowledgments

This work has been partially carried out within the MBSE4Mechatronics project (grant nr. 130013) of the Flanders Innovation & Entrepreneurship agency (VLAIO). This research was partially supported by Flanders Make vzw.

The authors would like to express their gratitude to the organizers of the Computer Automated Multi-Paradigm Modelling workshop (CAMPaM) where the foundations of this work have been carried out.

7. REFERENCES

[1] 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, pages 31:1–31:6, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[3] C. Artigues, S. Demassey, and E. Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. ISTE, 2007.

[4] R. Balzer. Tolerating inconsistency. In *13th International Conference on Software Engineering*, pages 158–165, 1991.

[5] B. Barragáns-Martínez, J. Pazos-Arias, and A. Fernández-Vilas. On measuring levels of inconsistency in multi-perspective requirements specifications. In *Proceedings of the 1st Conference on the Principles of Software Engineering (PRISE'04)*, pages 21–30, 2004.

[6] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen. Contracts for Systems Design : Theory. Technical Report RR-8759, INRIA, 2015.

[7] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[8] A. Bhave, B. Krogh, D. Garlan, and B. Schmerl. Multi-domain modeling of cyber-physical systems using architectural views. *The First Analytic Virtual Integration of Cyber-Physical Systems Workshop*, pages 43–50, 2010.

[9] A. Bhave, B. H. Krogh, D. Garlan, and B. Schmerl. View consistency in architectures for cyber-physical systems. In *Proceedings - 2011 IEEE/ACM 2nd International Conference on Cyber-Physical Systems*, pages 151–160, 2011.

[10] N. Bouillot and E. Gressier-Soudan. Consistency models for distributed interactive multimedia applications. *SIGOPS Operating Systems Review*, 38(4):20–32, Oct. 2004.

[11] G. Box, G. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, 2008.

[12] M. Chechik, F. Dalpiaz, C. Debreceni, J. Horkoff, I. Ráth, R. Salay, and D. Varró. Property-Based Methods for Collaborative Model Development. In *Proc. of 3rd Int. Workshop on The Globalization of Modeling Languages (GEMOC 2015)*, 2015.

[13] J. Corley, E. Syriani, H. Ergin, and S. Van Mierlo. Cloud-based multi-view modeling environments. In *Modern Software Engineering Methodologies for Mobile and Cloud Environments*. IGI Global, 2015.

[14] I. Dávid, J. Denil, and H. Vangheluwe. Towards inconsistency management by process-oriented dependency modeling. In *Proceedings of 9th International Workshop on Multi-Paradigm Modeling*, pages 32–41, 2015.

[15] I. Dávid, I. Ráth, and D. Varró. Foundations for streaming model transformations by complex event processing. *Software & Systems Modeling*, pages 1–28, 2016.

[16] J. de Lara and H. Vangheluwe. AToM3: A Tool for Multi-formalism and Meta-modelling. In R.-D. Kutsche and H. Weber, editors, *Fundamental Approaches to Software Engineering: 5th International Conference, FASE 2002 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8–12, 2002 Proceedings*, pages 174–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[17] P. Derler, E. A. Lee, S. Tripakis, and M. Törngren. Cyber-Physical System Design Contracts. In *ICCPS '13*, pages 109–118. ACM, 2013.

[18] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *Proceedings of the second workshop on Formal methods in software practice*, pages 7–15. ACM, 1998.

[19] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh. Coordinating distributed viewpoints: the anatomy of a consistency check. *Concurrent Engineering*, 2(3):209–222, 1994.

[20] C. Ericson. *Real-time collision detection*. CRC Press, 2004.

[21] A. Finkelstein. A foolish consistency: Technical challenges in consistency management. In M. Ibrahim, J. Küng, and N. Revell, editors, *Database and Expert Systems Applications*, volume 1873 of *Lecture Notes in Computer Science*, pages 1–5. Springer Berlin Heidelberg, 2000.

[22] A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, Aug 1994.

[23] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In L. Briand and A. L. Wolf, editors, *Future of Software Engineering*, pages 37–54, Minneapolis, may 2007. IEEE Computer Society.

[24] D. M. Gabbay, I. Hodkinson, and M. Reynolds. Temporal logic mathematical foundations and computational aspects. 1994.

[25] 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 *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*, pages 1–12, 2009.

[26] 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, pages 1–8, New York, NY, USA, 2008. ACM.

[27] J. Grundy, J. Hosking, and W. B. Mugridge. Inconsistency management for multiple-view software development environments. *IEEE Transactions on Software Engineering*, 24(11):960–981, Nov 1998.

[28] P. Hehenberger, A. Egyed, and K. Zeman. Consistency checking of mechatronic design models. In *30th Computers and Information in Engineering Conference*, volume 3: Parts A and B, pages 1141–1148. ASME, 2010.

[29] P. Höfner. How to find algebraic semantics for modal and temporal logics. https://www.informatik.uni-augsburg.de/lehrstuehle/dbis/pmi/alumni/hoefner/talks/09_relmics/. Invited talk at the 11th International Conference on Relational Methods in Computer Science / 6th International Conference on Applications of Kleene Algebra (RelMiCS11/AKA6). Accessed: 2016-08-30.

[30] A. Hunter, S. Konieczny, et al. Measuring inconsistency through minimal inconsistent sets. *KR*, 8:358–366, 2008.

[31] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[32] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, Sept. 1979.

[33] C. Lange, M. R. V. Chaudron, J. Muskens, L. J. Somers, and H. M. Dortmans. An empirical investigation in quantifying inconsistency and incompleteness of uml designs. In *Incompleteness of UML Designs, Proc. Workshop on Consistency Problems in UML-based Software Development, 6 th International Conference on Unified Modeling Language, UML 2003*, 2003.

[34] Y. Ma, G. Qi, P. Hitzler, and Z. Lin. *Measuring Inconsistency for Description Logics Based on Paraconsistent Semantics*, pages 30–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[35] T. Mens and R. Van Der Straeten. Incremental resolution of model inconsistencies. In J. L. Fiadeiro and P.-Y. Schobbens, editors, *Recent Trends in Algebraic Development Techniques: 18th International Workshop, WADT 2006, La Roche en Ardenne, Belgium, June 1-3, 2006, Revised Selected Papers*, pages 111–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[36] G. Moroni, T. Tolio, S. J. I. Herzig, and C. J. J. Paredis. A conceptual basis for inconsistency management in model-based systems engineering. *Procedia CIRP*, 21:52–57, 2014.

[37] M. Persson, M. Törngren, A. Qamar, J. Westman, M. Biehl, S. Tripakis, H. Vangheluwe, and J. Denil. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *Proceedings of the International Conference on Embedded Software*, 2013.

[38] A. Qamar, C. J. Paredis, J. Wikander, and C. During. Dependency modeling and model management in mechatronic design. *Journal of Computing and Information Science in Engineering*, 12(4), 2012.

[39] X. Qin. Delayed consistency model for distributed interactive systems with real-time continuous media. *Journal of Software*, 6(13):1029–1039, 2002.

[40] J. Reineke and S. Tripakis. Basic problems in multi-view modeling. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *LNCS*, pages 217–232. Springer, 2014.

[41] G. Roşu and S. Bensalem. Allen linear (interval) temporal logic–translation to ltl and monitor synthesis. In *International Conference on Computer Aided Verification*, pages 263–277. Springer, 2006.

[42] P. Sewell, S. Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen. x86-TSO: A rigorous and usable programmer's model for x86 multiprocessors. *Communications of the ACM*, 53(7):89–97, 2010.

[43] A. A. Shah, A. A. Kerzhner, D. Schaefer, and C. J. J. Paredis. Multi-view modeling to support embedded systems engineering in sysml. In G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, and B. Westfechtel, editors, *Graph Transformations and Model-driven Engineering*, pages 580–601. Springer-Verlag, Berlin, Heidelberg, 2010.

[44] A. A. Shah, D. Schaefer, and C. J. J. Paredis. Enabling multi-view modeling with sysml profiles and model transformations. In *International Conference on Product Lifecycle Management*, pages 527–538, 2009.

[45] D. J. Sorin, M. D. Hill, and D. A. Wood. *A primer on memory consistency and cache coherence*. Number 16 in Synthesis Lectures on Computer Architecture. Morgan & Claypool, 2011.

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

[47] M. Törngren, A. Qamar, M. Biehl, F. Loiret, and J. El-khoury. Integrating viewpoints in the development of mechatronic products. *Mechatronics*, 24(7):745 – 762, 2014. 1. Model-Based Mechatronic System Design 2. Model Based Engineering.

[48] F. J. Torres-Rojas, M. Ahamad, and M. Raynal. Timed consistency for shared distributed objects. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '99, pages 163–172, New York, NY, USA, 1999. ACM.

[49] K. Vanherpen, J. Denil, I. Dávid, P. De Meulenaere, P. J. Mosterman, M. Törngren, A. Qamar, and H. Vangheluwe. Ontological reasoning for consistency in the design of cyber-physical systems. In *2016 1st International Workshop*

on Cyber-Physical Production Systems (CPPS), pages 1–8, April 2016.

[50] R. von Hanxleden, E. A. Lee, C. Motika, and H. Fuhrmann. Multi-view modeling and pragmatics in 2020. In R. Calinescu and D. Garlan, editors, *Large-Scale Complex IT Systems. Development, Operation and Management*, volume 7539 of *Lecture Notes in Computer Science*, pages 209–223. Springer Berlin Heidelberg, 2012.

[51] G. Wachsmuth. Modelling the operational semantics of domain-specific modelling languages. In R. Lämmel, J. Visser, and J. Saraiva, editors, *Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007. Revised Papers*, pages 506–520. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.