Foundation for "Problem Statement" and "Motivation"
=> Provides challenges that new SysML v2 capabilities can be positioned to solve

J. Michael, J. Pfeiffer, A. Wortmann (Hrsg.): Modellierung 2022 Satellite Events,
Digital Library, Gesellschaft für Informatik e.V. 77

Challenges in Multi-View Model Consistency Management for Systems Engineering

Sebastian Bergemann[1]

Abstract: A way to handle the complexity of cyber-physical systems is model-based systems engineering (MBSE) with multiple viewpoints. These viewpoints satisfy different concerns, but they likely have information dependencies and overlaps among each other. Inconsistencies can be introduced whenever there are changes in only some of the views without consistent synchronization in other dependent views. In this paper, we motivate why consistency management is important in multi-view MBSE and define requirements for it. By analyzing the State of the Art, we identify limitations in (multi-view) consistency management approaches, especially for inconsistency detection. Besides general performance issues, we notice primarily that most approaches are limited to or at least tested on only very specific views and tools with homogeneous models and few specific predefined consistency rules. Furthermore, in most approaches we cannot find solutions regarding subsequent updates of consistency rules by the user, allowance of tolerating inconsistency, and handling confidentiality. These literature gaps pose open research challenges for making multi-view consistency management more applicable in the industry.

Keywords: consistency management; multi-view modeling; model-based systems engineering

1 Introduction

The development of cyber-physical systems (CPSs) is much more difficult than of usual software systems since CPSs combine software with hardware, embedded in the physical world [KS08, TS18]. An approach to reduce the resulting higher complexity of CPSs is model-based systems engineering (MBSE) with multiple viewpoints. The key aspect of MBSE is the usage of models instead of or in combination with textural documents to design a system. A model is an abstraction of something real that holds enough information to represent this real subject regarding a specific purpose but nothing beyond to simplify the handling of it. Related to this abstraction concept are views and viewpoints, which play an important role in this paper. We are following their definition in ISO 42010 [In11]. A view is like a specific perspective on the system. It is a model representing the system regarding specific concerns. Therefore, it contains or represents only the information needed for these concerns while hiding everything else, resulting in a lower complexity of the system (for these concerns). A viewpoint is like the instructions for a view where it is specified how to construct it, meaning that a view is an instance of a viewpoint. Although views often represent different information due to different concerns, they might still share some

[1] fortiss GmbH – Research Institute of the Free State of Bavaria, Munich, Germany; bergemann@fortiss.org

information of the system or are in other ways related, which is why dependencies and overlaps between them exist. Therefore, it is likely that inconsistencies between views will occur if several engineers are working on the same system in different views (of different viewpoints). We want to clarify that we do not mean different views of a single model, i.e., a single source of truth, because in this case it is much less likely or even impossible to have inconsistencies between the views. We are referring to views without a single source of truth, meaning that we have different models or reused parts of models in other models. Unsolved inconsistencies between such views can cause problems at the latest when the system should be realized based on the combined information of the inconsistent model(s). Therefore, multi-view model inconsistencies need to be detected as early as possible so that it can then be decided if the inconsistency can be tolerated or if it needs to be resolved.

Multi-view consistency is much harder to identify than consistency within a view due to the many, often hidden, relations and dependencies between views. We provide in Sect. 2 our own requirements for this specialization. The importance of automatic (multi-view) consistency management in the industry is also mentioned by Jongeling et al. [Jo22], who provide a current State of Practice analysis regarding agile model-based development with consistency management in several industrial settings. Besides giving an overview about current research in this field, the authors point out the struggles the industrial companies have with consistency management resulting in often insufficient and/or manual solutions.

In this paper, we are continuously using the term consistency, but to be precise we are meaning model consistency, which is about consistency within and between models during MBSE. Therefore, the context is always MBSE including for example SpesML [Mod22], where modeling in multiple viewpoints plays a major role.

1.1 Contribution

We want to highlight challenges in the field of multi-view consistency, which can mostly be applied to general consistency management as well. The challenges are derived from limitations of current approaches for managing multi-view consistency, mainly detecting inconsistencies between multiple views and viewpoints. The limitations are based on our current literature analysis, which we will present in an overview, and on requirements we define for applicable multi-view consistency management in the industry.

1.2 Outline

According to our contribution, we first provide the basis in Sect. 2 with requirements and supporting examples for multi-view consistency management in the industrial setting, and then continue in Sect. 3 with the literature analysis overview including current limitations. This results in the challenges presented in Sect. 4. We conclude in Sect. 5 with a summary and an outlook of our next steps.

2 Requirements for Multi-View Consistency Management

We first provide the basis with some requirements for a good applicability of multi-view consistency management in the industry. We do not limit this to the software engineering domain but keep it open to systems engineering, where cross-domain consistency management is often needed. The upcoming requirements in Sect. 2.2 might not be complete, but based on our experience they are the most important ones while not being trivial. We have collected the requirements mainly by listening to feedback from industrial partners in previous research projects, where they told us with what problems they have to deal and what they still need to have an efficient workflow. Before we show the requirements in Sect. 2.2, we first want to give a short example of a setting in which the requirements are important. This should help to better understand the requirements, although we will stay abstract with the example since a detailed example would be out of scope for this paper.

2.1 Introducing Example

Several users, usually engineers, from different companies are working in different views on the same product, e. g., an autonomous vehicle. These views are heterogeneous: they separately model requirements, the logic, the technical implementation, the 3D appearance (via CAD), and the simulation of the system. Relations exist between elements (intra- and inter-view) like references to or from other elements (e. g., requirements tracing or allocation/deployment) or shared information (e. g., identical/related names). Some relations are important to exist/persist in a specific way, which is why consistency conditions are defined to specify which properties these relations need to fulfill.

Within this general setting, we create the example scenario that a user wants to reuse a part of a sensor from a library or another system. The part is reused inside the logic view, but it has also modeled information inside the technical and the CAD view. Due to the reuse, information in these views has changed, which means that probably some relations between the views have changed as well. Therefore, it needs to be checked if they are still according to the consistency conditions, because otherwise the change could have created mismatches between views, resulting in (future) failures. Defining and checking of consistency conditions are usually easy for simple relations, like related names, but it becomes challenging in our example due to the following four aspects:

A1) Not explicitly modeled information. The following consistency condition exists: *The weight of a product part must be identical over all views that are referring to this product part*. The sensor has an explicitly annotated weight inside the technical view, but inside the CAD view the weight is not explicitly stored. Nevertheless, the weight information (indirectly) exists also in the CAD view, because the geometry and material is modeled there and based on this data, a weight can be deduced in the CAD view. Therefore, the consistency check needs to consider this not explicitly modeled information as well, since not only the annotated weight in the technical view could have changed due to the reuse

but also the geometry or material in the CAD view, resulting in a new deduced weight. This means that the consistency check needs to know about this not explicitly modeled information, and it needs a deduction mechanism to get the weight information from the CAD view based on the modeled geometry and the material.

A2) Semantic relevance. Another consistency condition exists: *The step size in the simulation view needs to fit to all simulated logical and technical components modeled in their views*. The updated sensor in the logic view has a worst-case execution time of 0.005 s, in the technical view it has a sampling rate of 100 Hz and the step size in the simulation view is 1 ms. A consistency check solely based on the syntactical value would find that the value of the worst-case execution time is less than the value of the step size in the simulation view and consider the consistency condition as fulfilled. However, if the unit is taken into account, it can be seen that the condition is actually violated, because a worst-case execution time of 0.005 s is 5 ms, which is larger than the step size of 1 ms. Therefore, the consistency check needs to be aware of the semantics, otherwise it will not work. It also needs to know that 100 Hz is 0.01 s or 10 ms to compare it with the step size. Hard coding this for every consistency check is not efficient and error-prone.

A3) Changing situation. Due to the reuse and the resulting new situation, the engineers gain new insights like new technical annotations they need to add and respect in the model. Therefore, the engineers want to add new consistency conditions or update existing ones for future consistency checks. This can be difficult or impossible, because their consistency tools have only predefined consistency conditions that cannot be changed or only by the developers of the tools and not the engineers as end-users. This means that consistency tools/concepts without the possibility of easy extension will be inefficient and, in some cases, unusable for modern industrial projects, especially in the agile context.

A4) Confidential information. The internal logic of the reused sensor part is confidential because it was developed by another vendor. Usually, requirements will make sure that the confidential parts are kind of consistent with the remaining parts, e. g., which ASIL level can be guaranteed, but especially in agile development the situation can often change, e. g., a higher ASIL level might be required after the reuse. An option would be to exchange again every time the requirements with the other partners, but especially for a first quick overview it would be more efficient if consistency conditions could be checked also in combination with confidential parts (without violating confidentiality).

These challenging aspects in our example will be converted in the next section into our requirements for multi-view consistency management, with which we will continue during the remaining paper.

2.2 Derived Requirements

The given examples of challenging aspects while applying multi-view consistency management in the previous section can be converted to these requirements (R1 is based on A1, R2 on A2, etc.):

R1) It should be possible that consistency can also be checked based on not explicitly modeled information. Information might not always be modeled and stored explicitly in the view but can exist hidden behind a combination of other information. This not explicitly modeled information is for consistency as important as explicitly modeled information and needs to be deduced for the consistency check. Therefore, the consistency checking mechanism not only needs to be aware of such hidden information, but also needs to know how to access it via a deduction mechanism.

R2) Inconsistency detection should also use semantic information. As shown in the example in Sect. 2.1, just comparing the syntax and pure values will not lead every time to a correct consistency statement. Consistency checks need to be aware of the semantics as well, which is usually more difficult than syntax checks.

R3) Consistency specifications should be extensible at any time. In the context of agile development but also beyond, it happens often that new insights occur, which needs to be applied as soon as possible to the system model. This can also result in the need to update the relations and consistency conditions. Therefore, the end user, and not the developer of the consistency tool, needs to have the possibility to easily extend the conditions.

R4) Inconsistency detection should be possible even with confidential model parts. In industrial projects it is common that model parts are confidential, especially if different manufacturer are involved. In this case, it will become difficult to get all information needed for the consistency checks. Outdated requirements after agile iterations and a continuous back and forth with the partners of the confidential parts about requirement changes will consume much time. A way to check for consistency with even confidential parts can be beneficial.

We will use these four requirements to analyze approaches in the literature (see Sect. 3) and to identify open challenges (see Sect. 4).

3 Literature Analysis

In this section we analyze the State of the Art in literature for how well it can fulfill the requirements of Sect. 2.2. Based on these findings we will identify open challenges later on in Sect. 4.

For the State of the Art analysis we have analyzed 124 papers. These papers are not directly the result of a systematic literature review but were continuously collected during our overall

research and orientation in the field of consistency management. We used the literature collections of studies and reviews like [CCP19, Pe13, TvBS20, UTZ17, KM18, LMT09] as foundation and filtered them for multi-view consistency management approaches. During further research in this area, we found more interesting multi-view consistency management papers, which we added to build our current paper base. One of our next plans is to extend this literature analysis and provide a systematic literature review to have a more structured base.

3.1 Current Approaches

As a short overview, we will first briefly discuss the categorization of consistency approaches and afterwards present the three most interesting approaches regarding the requirements of Sect. 2.2.

Categorization. The literature does not provide a universal categorization of consistency approaches. Spanoudakis et al. [SZ01] differentiate between logic-based approaches, model checking approaches, specialized model analysis approaches, and human-centered collaborative exploration approaches. Torres et al. [TvBS20] provide not directly categories but list strategies to keep consistency between different domains, e. g., parameter or constraint management, or ontology. Other categorizations exist as well, but they are often focused on UML approaches [KM18, LMT09, Hu05]. A valuable contribution is the categorization by Feldmann et al. [Fe15]. They distinguish the approaches in proof-theory-based, rule-based, and synchronization-based (e. g., via model transformations). Similarities exist to Haesen et al. [HS05], where the approaches are labeled by how they generally manage consistency: by construction, by monitoring or by analysis. Consistency by construction matches to Feldmann's synchronization-based category, where an approach generates a new consistent model based on a changed one. Consistency by monitoring means simple rule checking that is executed after every new change and is similar to Feldmann's rule-based category. Consistency by analysis means one algorithm is run once in the end to check for all defined possible inconsistencies, but in our opinion, this is very similar to the monitoring category, because both are basically executed checks based on rules where only the time and frequency of the checks are different. What we did not find as explicit category is consistency by design, where the viewpoints and thus the resulting views are already designed from the beginning on in a way that inconsistencies cannot occur due to the precisely defined relations between the viewpoints. In summary, we think that all categories and their approaches can be fitted into the following three general categories: consistency be design (everything where the concept and the system of viewpoints are already designed to prevent inconsistency), consistency by rule checking (everything where models are checked explicitly for inconsistency based on rules), and consistency by construction/generation (everything where models are (newly) generated to preserve consistency).

1) Consistency by design. The *SPES* methodology [Po12] aims on consistency by design, because the underlying concept is defining the different viewpoints and their relations in a

consistent way and predefined well-formedness rules are preventing basic inconsistencies like in the currently ongoing SpesML project [Mod22]. However, the concept does not enforce consistency in all aspects but rather suggests specific designs and leaves it to the user to ensure final consistency. Therefore, inconsistency between views can still occur and they can only be detected if they are covered by the predefined well-formedness rules, otherwise the inconsistency remains hidden.
Regarding requirements of Sect. 2.2: Although it is theoretically possible to hard code the consideration of not explicitly modeled information (R1) and semantics (R2) inside the well-formedness rules, it is not really supported with this approach. The well-formedness rules are predefined and not intended to be modified and extended by the end user (R3). Confidentiality in connection with consistency (R4) is not yet a topic within *SPES*.

2) Consistency by rule checking. A rule-based approach is the *Model/Analyzer*, which was first proposed for solely UML models [Eg06] and was then generally extended [Eg11]. The approach is not directly about how to check consistency and define consistency rules. It rather sees the consistency rules as black boxes to check which model elements are used for each rule. This way the scope of a rule is determined, and it provides a fast and automatic way to identify which rules need to be evaluated again after a certain model change instead of just checking everything each time or a typed-based scope checking. Although this is an important improvement and provides flexibility, it mainly leaves all the difficulties to the developer of the consistency checker and rules, and the engineer as end user. The approach ignores the semantics of rules and model information and does not specify how to handle different heterogeneous views and models. However, this is also not directly the scope of the paper. We also assume that the consistency checker needs to have direct access to the whole model (set), which is critical regarding confidentiality.
Regarding requirements of Sect. 2.2: As already mentioned, the requirements regarding not explicitly modeled information (R1), semantics (R2) and confidentiality (R4) are not met. However, an advantage is definitely the extensibility (R3), because this is one of the few approaches where the end user can freely extend the consistency checks by any language.

3) Consistency by construction/generation. As a third major direction of research we want to present a consistency by construction or synchronization-based approach, which is *Vitruvius*. It was developed over several dissertations and papers, e. g., [Bu13, Kr17, La17, Kl18, Kl21], and proposed with a virtual single underlying meta-model (V-SUMM). Instead of explicitly collecting all information in one central model (like in Orthographic Software Modeling (OSM) [ASB09]), the idea is to combine all meta-models virtually to one V-SUMM, coupled by model transformations. These transformations are based on consistency conditions and will generate again consistent models of the other meta-models after a model change. This has many benefits, but we still see several limitations. 1) V-SUMM is currently only focused on software engineering and not generally on systems engineering. 2) Model transformations themselves have limitations and they are difficult to define for heterogeneous views. 3) They also hinder easy consistency modifications by the later engineers because for example adding a new consistency specification would mean

updating the whole V-SUMM and model transformations. 4) Only greenfield development is well feasible because of the assumption/requirement of V-SUMM that the whole model structure is in the beginning already consistent [An18]. 5) V-SUMM does not have the possibility to tolerate inconsistency. The generation is executed automatically and will always enforce consistency preservation.
Regarding requirements of Sect. 2.2: We could not find any source stating that this approach is dealing with the confidentiality issue (R4). As already mentioned, it is difficult to update the model transformations regarding new or modified consistency conditions and it is not intended for the end user to do this (R3). Due to the model transformations it might also be difficult to consider semantics (R2) and not explicitly modeled information (R1).

3.2 Current Limitations

Based on the previously mentioned literature research and to the best of our knowledge, we see following limitations that exist in the literature on consistency (management). None of the approaches in literature can fulfill all requirements from Sect. 2. We will divide them into 1) the group of missing features, 2) the group of too high restrictions and 3) the group of general performance problems.

Missing features. As many others [Ba91, We18, Eg11], we argue that inconsistency is tolerable in some cases, either temporarily for tests or even permanently depending on the inconsistency type. Therefore, a consistency approach should enable this option. However, many approaches force automatic consistency preservation either by direct (re-)construction without separate inconsistency detection or by forced inconsistency repair after detection. The next feature we miss in many approaches is the possibility to update the consistency specifications at any time after the actual consistency approach was implemented. Most approaches can only take the predefined consistency rules into account or provide a poor extension interface. However, consistency rules might change and evolve over time, especially during agile development, which is why the engineer should always be able to modify them. The last major feature is about confidentiality. In the industry this is a critical issue and approaches should at least consider this in some way during consistency checking but based on our research this is nearly never the case.

Not generalizable. We are sure that it is not possible to have a completely generic approach for consistency management across domains, but the level of restrictions and specialization in current approaches is often too high and thus not suitable for industrial applicability. This starts with the usage of very specific consistency rules that cannot consider, e. g., semantics or not explicitly modeled information. The next level is limiting oneself to only few specific views if different views are included at all. Finally, consistency can in many approaches only be checked for specific models that are usually solely in the software domain and/or very homogeneous, where defining consistency conditions is not complicated. As an example, many approaches, especially the older ones, are completely focused on only UML models.

General performance problems. Some approaches are not scalable for large models and projects, especially if they hold high complexity. In addition, consistency approaches should be fast and executable at any time, or at least after each model change, to have the possibility of immediate feedback, but this is not to case for several reviewed approaches. We do not want to focus on these performance limitations, but to ignore them is also not an option regarding industrial applicability.

In sum, all these limitations make it difficult to apply current consistency approaches in the industry, where project models are usually heterogeneous, based on multiple views and tools, often confidential, large in scale, and should be easy to handle by engineers who are usually not experts in programming. This leads to the open challenges presented next.

4 Challenges

The previously mentioned limitations in the literature show that the stated requirements in Sect. 2.2 are not yet covered. Therefore, we can derive from them open challenges that are beneficial to solve for a good applicability of multi-view consistency management in the industry. For all following challenges it should be considered to enable the possibility of inconsistency toleration, because always directly forcing automatic consistency preservation is a limitation (see Sect. 3.2).

Consistency between heterogeneous views. Inconsistency detection across multiple views is already difficult because of the many dependencies between them, which are not always easily identifiable. However, it becomes even more complex when the views are heterogeneous and not only limited to the software domain. In addition, they might even be distributed among different tools, requiring tool interoperability. Overall, this makes it more difficult to compare them and finding as well as checking dependencies between them.

Not explicitly modeled information. Checking consistency relations is becoming more challenging when the information on which the relation is based is not explicitly modeled. With explicitly modeled information we mean directly accessible/stored information in the model. If it needs to be checked because of a consistency condition, it can easily be received. However, if the information is not explicitly modeled, we need to deduce the needed information from other explicitly modeled information (with a deduction mechanism). Such deduction needs to be possible during consistency checks.

Semantic-related consistency. We consider consistency checking as challenging when it is semantic-related. This is already mentioned as missing feature for many approaches in the literature [CCP19, TvBS20]. It is even more important for multi-view consistency because a system information can be interpreted and represented differently by different views. If we only consider syntax and not semantics during the comparison, we might easily get false positives or false negatives during inconsistency checks.

Continuous extensibility. Another challenge we see, especially in the context of agile development but also beyond, is the extensibility possibility of the consistency management system by the engineer. It is easy for developers of consistency management approaches to predefine consistency rules and only consider them for the consistency checks. However, if the engineer as end-user gets new insights during agile development or has very specific project-related consistency rules, it should be possible to extend and update the consistency rules at any time, especially during the actual modeling. This extensibility gets even more challenging when the extension and modification of consistency conditions should not be very restricted but more user-friendly, i.e., more understandable for end-users like engineers.

Confidentiality. The confidentiality issue is broadly factored out in the field of consistency, but especially for large CPSs it is very common that different companies are working together, and they do not want to expose all of their own intellectual property. However, this information hiding makes it difficult to check consistency. A possibility is to only check consistency for the parts that are accessible and everything else is based on pure requirements that are given to the other companies for their black box models. For agile development this is getting more difficult and costly because new insights might result in changes of own models and to ensure that they are still consistent with the external black box models, requirements might need to be updated or the suppliers need to be asked directly. This could be saved when consistency checks would be possible while still respecting confidentiality.

5 Conclusion

As part of this challenge paper, we have shown that consistency management between views plays an important as well as challenging role for MBSE. For this purpose, we have provided requirements for a good industrial applicability of multi-view consistency management.

Based on our literature analysis, we have stated that the combination of these requirements is not yet fulfilled by state-of-the-art approaches. Therefore, we have derived several open challenges for multi-view consistency management in industrial systems engineering. The presented list of requirements does not aim to be exhaustive. Hence, there can very well be further challenges not mentioned in this paper.

The derived challenges are based on the major difficulties of not explicitly modeled information, the relevance of semantics, the heterogeneity of occurring consistency relations, the need for continuous extensibility of them, and handling confidentiality. All of this is even more complex when consistency is managed between multiple tools. However, overcoming these challenges will make consistency management more applicable and efficient for the industry, especially with respect to agile development.

As a next step, we are planning a systematic literature review of specific multi-view consistency management and comparing it with the State of Practice to confirm our current research as well as refine it.

Acknowledgement

This work was in parts supported by the German Ministry for Education and Research (BMBF) in the SpesML project (https://spesml.github.io/index.html/; grant number 01IS20092G).

Bibliography

[An18] Ananieva, Sofia; Klare, Heiko; Burger, Erik; Reussner, Ralf: Variants and Versions Management for Models with Integrated Consistency Preservation. In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems. pp. 3–10, 2018.

[ASB09] Atkinson, Colin; Stoll, Dietmar; Bostan, Philipp: Orthographic Software Modeling: A Practical Approach to View-Based Development. In: Evaluation of Novel Approaches to Software Engineering. Springer, pp. 206–219, 2009.

[Ba91] Balzer, Robert: Tolerating inconsistency. In: Proceedings of the 13th International Conference on Software Engineering. pp. 158–165, 1991.

[Bu13] Burger, Erik: Flexible views for view-based model-driven development. In: WCOP 2013 - Proceedings of the International Doctoral Symposium on Components and Architecture. pp. 25–30, 2013.

[CCP19] Cicchetti, Antonio; Ciccozzi, Federico; Pierantonio, Alfonso: Multi-view approaches for software and system modelling: a systematic literature review. Software and Systems Modeling, 18(6):3207–3233, 2019.

[Eg06] Egyed, Alexander: Instant consistency checking for the UML. In: Proceedings of the 28th International Conference on Software Engineering. pp. 381–390, 2006.

[Eg11] Egyed, Alexander: Automatically detecting and tracking inconsistencies in software design models. IEEE Transactions on Software Engineering, 37(2):188–203, 2011.

[Fe15] Feldmann, Stefan; Herzig, Sebastian J.I.; Kernschmidt, Konstantin; Wolfenstetter, Thomas; Kammerl, Daniel; Qamar, Ahsan; Lindemann, Udo; Krcmar, Helmut; Paredis, Christiaan J.J.; Vogel-Heuser, Birgit: A Comparison of Inconsistency Management Approaches Using a Mechatronic Manufacturing System Design Case Study. In: IEEE International Conference on Automation Science and Engineering. IEEE Computer Society, pp. 158–165, 2015.

[HS05] Haesen, Raf; Snoeck, Monique: Implementing Consistency Management Techniques for Conceptual Modeling. In: Proceedings of the 3rd International Workshop on Consistency Problems in UML-based Software Development. Springer, 2005.

[Hu05] Huzar, Zbigniew; Kuzniarz, Ludwik; Reggio, Gianna; Sourrouille, Jean Louis: Consistency Problems in UML-Based Software Development. In: Lecture Notes in Computer Science. volume 3297. Springer Verlag, pp. 1–12, 2005.

[In11] International Organization Of Standardization: , ISO/IEC/IEEE 42010:2011 - Systems and Software Engineering – Architecture Description, 2011.

[Jo22] Jongeling, Robbert; Ciccozzi, Federico; Carlson, Jan; Cicchetti, Antonio: Consistency management in industrial continuous model-based development settings: a reality check. Software and Systems Modeling, 2022.

[Kl18] Klare, Heiko: Multi-model Consistency Preservation. In: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. pp. 156–161, 2018.

[Kl21] Klare, Heiko; Kramer, Max E.; Langhammer, Michael; Werle, Dominik; Burger, Erik; Reussner, Ralf: Enabling consistency in view-based system development — The VITRUVIUS approach. Journal of Systems and Software, 171(110815), 2021.

[KM18] Knapp, Alexander; Mossakowski, Till: Multi-view consistency in UML: A survey. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 10800, pp. 37–60. Springer, 2018.

[Kr17] Kramer, Max E.: Specification Languages for Preserving Consistency between Models of Different Languages. Phd thesis, Karlsruhe Institute of Technology (KIT), 2017.

[KS08] Karsai, Gabor; Sztipanovits, Janos: Model-Integrated Development of Cyber-Physical Systems. In: Software Technologies for Embedded and Ubiquitous Systems. volume 5287. Springer Berlin Heidelberg, pp. 46–54, 2008.

[La17] Langhammer, Michael: Automated Coevolution of Source Code and Software Architecture Models. Phd thesis, Karlsruhe Institute of Technology (KIT), 2017.

[LMT09] Lucas, Francisco J.; Molina, Fernando; Toval, Ambrosio: A systematic review of UML model consistency management. Information and Software Technology, 51(12):1631–1645, 2009.

[Mod22] Model-based system engineering with SysML and SPES methodology. `https://spesml.github.io/index.html/`, 2022. Online; accessed 16 April 2022.

[Pe13] Persson, Magnus; Törngren, Martin; Qamar, Ahsan; Westman, Jonas; Biehl, Matthias; Tripakis, Stavros; Vangheluwe, Hans; Denil, Joachim: 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. IEEE Computer Society, 2013.

[Po12] Pohl, Klaus; Hönninger, Harald; Achatz, Reinhold; Broy, Manfred: Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology. Springer, 2012.

[SZ01] Spanoudakis, George; Zisman, Andrea: Inconsistency Management in Software Engineering: Survey and Open Research Issues. In: Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals, pp. 329–380. World Scientific, 2001.

[TS18] Törngren, Martin; Sellgren, Ulf: Complexity Challenges in Development of Cyber-Physical Systems. In: Principles of Modeling, pp. 478–503. Springer International Publishing, 2018.

[TvBS20] Torres, Weslley; van den Brand, Mark G.J.; Serebrenik, Alexander: A systematic literature review of cross-domain model consistency checking by model management tools. Software and Systems Modeling, 20(3):897–916, 2020.

[UTZ17] Ul Muram, Faiz; Tran, Huy; Zdun, Uwe: Systematic review of software behavioral model consistency checking. ACM Computing Surveys, 50(2), 2017.

[We18] Weidmann, Nils: Tolerant Consistency Management in Model-Driven Engineering. In: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. 2018.