How to live with inconsistencies in industrial model-based development practice

Robbert Jongeling
Mälardalen University, Västerås, Sweden
robbert.jongeling@mdh.se

***Abstract*—Modern development of complex embedded systems utilizes models to describe multiple different views on the same system. Consistency between these models is essential to successful development but ensuring it is in current practice often a manual effort. In this research project, we aim to develop a methodology that helps developers to maintain consistency in industrial model-based development projects by identifying inconsistencies throughout the development and maintenance of the system. For such support to be applicable in industrial practice, it should fit in with current development, i.e., should be able to identify inconsistencies between models expressed in different modeling languages and created in different modeling tools. Furthermore, the required user interaction to defining consistency checks should be minimal. This paper sketches an approach meeting these requirements, initial results towards it and discusses future research plans towards a doctoral dissertation.**

***Index Terms*—Model-based development, Consistency checking,**

I. Problem

Models are being used as core development artifacts in the development of embedded systems [1], we consider this practice model-based development (MBD) [2]. The use of models promotes communication and re-use, and they are used as a basis for the development of other artifacts (being other models, code, or documents). In industrial practice, multiple modelling languages and modelling tools are being used to create these models [3]. We refer to them as being *heterogeneous*. Software systems are in practice modelled using multi-view modelling, i.e., using multiple models, representing multiple views of the same system [4]. The views are thus not completely independent and relationships exist between them and their elements [5]. Modelling systems through multiple views and multiple heterogeneous models incurs a need to ensure consistency across those views and models [6]. We focus on the problem of bringing about consistency between heterogeneous models in an industrial MBD context.

A. Consistency

We adopt the definition of consistency from the 24765 ISO/IEC/IEEE international standard as: *"degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component"* [7]. This is closely related to the definition of traceability from the same standard: "degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another" [7]. In contrast, we consider consistency between development artifacts regardless of the existence of these predecessor-successor or master-subordinate relationships. More specifically, following the ISO/IEC/IEEE 42010 standard, relationships between development artifacts are defined by correspondences, that are themselves governed by correspondence rules [8].

Consistency between models is required to have confidence in the created product and to prevent late changes or incorrect development [8]. Nevertheless, complete consistency cannot continuously exist, since inconsistency must be allowed at some times during development to allow for asynchronous changes to artifacts [9]. To prevent extra work on late changes to development artifacts, or worse, incorrect development, it is in the interest of developers to be timely aware of introduced inconsistency between development artifacts. Waiting for manual reviews is then not sufficient and automated support for *consistency checking* is required and desired in industrial practice.

B. Industrial context

A consistency checking approach aimed at detecting inconsistencies and maintaining consistency, is required throughout the entire development cycle, since inconsistencies can be introduced at any time during system specification, design, implementation, and maintenance. For instance, when a developer adds functionality to the system design (in e.g. an internal block diagram of a SysML model) that makes it inconsistent with other existing artifacts (e.g. a Simulink model), that latter model needs to be updated to reflect the additional information now captured in the former. This can be done at a later stage of development, and possibly by a different developer, but the inconsistency should be resolved before other artifacts start relying on the current (non-updated) version of the Simulink model. Additionally, when models are re-used in other development projects, it is important for developers to be aware of inconsistencies with existing artifacts. Some approaches to consistency checking provide automatic resolution of detected inconsistencies. This is not feasible in these use-cases, firstly because added functionality cannot be automatically implemented. Secondly, in cases where changes are made to existing model elements, automatically propagating those changes to related artifacts is possible, but not always desirable, since this depends on the development phase

This work is funded by Software Center: http://www.software-center.se

and the intentions of the particular changes. Therefore, we argue that a consistency checking approach should focus on identifying inconsistencies throughout all development phases, but not automatically resolving them.

The considered industrial context of consistency checking is in MBD development contexts, potentially involving many different tools and developers. In our industrial partners, development usually focuses around a system model as the main, leading development artifact. Managing the consistency between this and other artifacts is an ongoing challenge [10]. Regardless of the exact implementation of MBD, created artifacts are commonly expressed in different formalisms (e.g. modelling language or code) and likely created in different tools. Those artifacts, particularly for example system models, can grow to contain huge amounts of elements. Furthermore, these models may be created in a collaborative setting, where several developers are contributing to them and to other development artifacts. These factors complicate consistency checking and underscore the need for supporting it in industrial contexts, which is currently lacking [11].

Indeed, interoperability between different modelling tools and by extension consistency checking are shortcomings in current industrial MBD [1], [12], [13]. Industrial adoption of consistency checking is thus low, despite the potential improvements in interoperability and synchronization between heterogeneous models. We hypothesize that one of the main reasons for this is the complexity of consistency checking approaches for the targeted end-users.

C. Defining consistency checks

Checking for consistency between models requires correspondence rules between them. These might be implicitly derived, for example through name similarity, but are typically explicitly defined by users [4]. A consistency check then requires two pieces of information, 1) a correspondence rule identifying which elements should be consistent, and 2) what constitutes consistency between those elements. These two may be combined in one formalism, expressing how to transform a specific model element into another model element, e.g. through model synchronization transformations [6]. Such approaches are very expressive in the sense that they can potentially detect and automatically resolve many different types of inconsistencies. An example of an often-used formalism to express such correspondence rules is Triple Graph Grammars [14]. Correspondence rules are then themselves becoming complex expressions or models, thus requiring a large time investment to set-up as well as requiring subsequent time-intensive maintenance.

This level of expressiveness is not always required, particularly in our case, since we aim only to identify inconsistencies and notify developers of them. A trade-off can be made between expressiveness and the level of automation in identifying and correcting inconsistencies [15]. Indeed, less expressive rules could be easier to define and therefore more suitable for industrial adoption. In addition, lowering the required complexity in user interactions with the consistency checking methodology promotes the likeliness of industrial adoption, especially when the approach also considers the industrial development process and tooling context. Thus, a consistency checking methodology applicable in industrial practice requires a the amount of user interaction for defining and maintaining correspondence rules to be minimal.

D. Summary

In summary: inconsistency between models should be identified early in the development process to prevent late changes or possibly incorrect system implementation. Current adoption is low due to the inherent complexity of industrial MBD and the complexity in defining correspondence rules. The main research question is thus: how can we create a consistency checking mechanism suitable for industrial MBD practice? In particular, we aim to research how to support:

1) industrially most relevant types of inconsistencies; i.e., what types of inconsistencies between what type of models should developers be notified about?
2) heterogeneous models; i.e., how to check consistency between models, and parts of models, created in different tools and expressed in different formalisms (modelling languages, code languages, etc.)?
3) lightweight correspondence rule definitions; i.e., how to define and maintain consistency rules requiring minimal user interaction?
4) industrial MBD contexts; i.e., how to integrate consistency checking such that it complements existing development processes and is able to handle the pluriformity of industrial MBD tooling.

II. Related work

There is a large body of work describing approaches to consistency checking [4]. Nevertheless, a recent review of the literature finds that few modelling tools provide consistency checking support between heterogeneous models, and that available support does not suffice for industrial practice [12]. In this section, we discuss work most closely related to our proposed approach, thereby excluding several approaches aiming specifically at consistency checking between different UML diagrams [16]. Papers on model evolution [17] are also excluded since we do not consider the co-evolution of metamodels and the models conforming to them. Indeed, in our considered industrial cases, the metamodels (e.g. SysML) typically rarely evolve.

A. Consistency checking approaches

Early approaches are based on the declaration of rules that must hold over a set of models, for example expressed in first-order logic [18]. Following these steps, the Epsilon Object Language was proposed, which embodies the ideas of defining rules over models [19]. In contrast, our approach requires direct correspondence links to be defined between model elements, which allows correspondence rules to address more specific model elements. Further built on EOL, the Epsilon

Validation Language (EVL) can be used to express consistency checks as well as ways to resolve them when detected [20].

An established approach in the literature is to express correspondence rules using Triple Graph Grammars (TGGs), which are bidirectional transformations allowing for automatically updating related artifacts [14]. The correspondence rules are defined as a transformation between source and target meta-model elements. A similar amount of expressiveness is provided by link models, which also captures the way of translating model elements, but as models conforming to a link language [21]. In our approach, we are not aiming at automatically resolving inconsistency and argue that simpler correspondence rule definitions can still sufficiently detect relevant inconsistencies.

Specifically aimed at MBSE is an approach comparing graph representations of models, where inconsistencies in the graph are indicative of inconsistencies in the underlying models [22]. This approach requires an automatic or manual mapping of elements between different models to appropriately compare them. Automatically matching elements from different models is a difficult problem for which name comparisons might be used as a simple but ineffective approximation. In our approach, we require that this mapping is made manually, through defining the correspondence rules.

Egyed has proposed an incremental consistency checking approach agnostic of the language of expressing consistency rules [23]. The approach furthermore runs consistency checks only when needed, thus reducing total execution times compared to batch-based approaches. Stevens has a similar goal and proposes bidirectional transformations to repair inconsistencies when they are introduced, or at least before code is generated from models [24]. Here, the bidirectional transformations are executed only when models are changed in a relevant way, improving over the conventional use of relying on time stamps to detect changes between versions. Our approach can benefit from these ideas to be made less batch like and run only those consistency checks that are required before a build.

Diskin et al. have proposed an approach for checking global consistency of heterogeneous models [25]. Similar to our approach, comparisons between heterogeneous models are simplified to comparisons in homogeneous models. In their approach, this is achieved through the merging of metamodels and models, such that constraints can be checked globally, on the set of models. In our approach, we propose not to merge metamodels, but instead to lift relevant aspects of metamodels into a common metamodel. König and Diskin have then proposed further improvements to this idea of merging models for consistency detection [26], [27].

A more comprehensive comparison of types of different consistency checking approaches is provided by Feldmann et al. [28]. They divide approaches into categories of theory-based, rule-based, or synchronization-based. Where approaches in the latter category are aimed at automatically resolving inconsistencies. The first two categories are concerned with checking provided rules.

B. Other related work

Tangentially related to our work is an approach that links XML-based artifacts in a software ecosystem [29]. There are similarities with our approach, since we also aim to link model elements but in addition also maintain a definition of what consistency means. Similarly, there is a clear relation between our work and work on mega-modelling, which promotes creating models to capture inter-model links [30]. This has been also applied in the context of consistency checking. For example, MegaL/Forge allows users to describe relationships between EMF based models in a project and describe tactics to attempt automatic recovery of inconsistencies [31].

III. Proposed Approach

A. Overview

In this section, we sketch our proposed approach using a running example. We consider a simple evolution scenario in an MBD project consisting of one big SysML system model and several Simulink models refining it by implementing software functions. Specifically, let three SysML diagrams (an internal block diagram, an activity diagram, and a parametric diagram) describe the anti-lock braking system of a car, which is further refined and described in a Simulink model. Note that this is a simple example and potentially we could have additional models involved in describing this part or related parts of the software system, such as UML state machines or sequence diagrams.

Our proposed consistency checking mechanism simplifies the definition of correspondence rules through a separation of concerns. The two concerns inherent to a consistency rule are 1) to connect two elements, and 2) to declare those elements consistent or not. In our approach, a global set of rules dictates the meaning of consistency between metamodel elements. We refer to this as a *language consistency mapping*. In the example, this would constitute a set of rules dictating how the SysML diagrams and the Simulink model should be expressed in a model conforming to the common metamodel. Then, the user-defined correspondence rules are simplified to linking model elements between which consistency should be checked. We call this a *model consistency mapping*.

Using these two rules, executable consistency checks can be automatically generated. The passing or failing of these consistency checks is determined by comparing the representations of the models. This is possible since both representations are conforming to the same common metamodel. The mappings are discussed in more detail in the remainder of this section. An abstract overview of the approach is shown in Figure 1, where we show two models, M_a and M_b conforming to metamodels MM_a and MM_b respectively. To check consistency, M_A and M_B are transposed to representations CM_A and CM_B respectively. In turn CM_A and CM_B conform to the CMM metamodel.

B. Consistency mappings

The language consistency mappings allow for the comparison of heterogeneous models by expressing them in a common

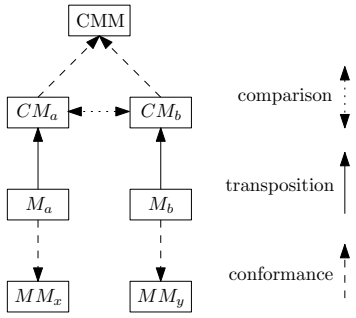

Fig. 1. Abstract overview of the approach, models are represented in a common way, conforming to a comparison metamodel (CMM)

format. They are thus model transformations, transforming an input model into an instance of the comparison metamodel. In our running example, we would have two language consistency mappings, one to transform the SysML diagrams to a model in the common modeling language and one to transform the Simulink model to a model in the common modeling language. An implementation would rely on the file representations of these models to serve as the input source of these transformations. These mappings are global and reused for any consistency check involving model elements of these types. Note that these mappings are not one-to-one between metamodel elements of e.g. SysML and Simulink but rather describe how to express a model in the common format. This potentially allows consistency checks between n model elements, depending on the comparison algorithm. Furthermore, the complexity of the language consistency mappings can be varied in case more or less expressiveness is desired. In the simplest case, the language consistency mapping can be omitted entirely, marking any consistency check a failure and thus being useful only for a simple form of change impact analysis. One of the aspects of this mapping in the running example would be a mapping detailing which Simulink block types to transpose into the common representation. For example, the most detailed algorithmic blocks in a Simulink model are probably not of interest in a comparison between SysML and Simulink models. Rather, the mapping would detail that e.g. subsystems and ports should be transposed to the comparison model.

To generate consistency checks, a correspondence rule is required to define which elements should be compared to each other. Since this is the part of the consistency check definition that the user interacts with directly, we aim to keep the required input as minimal as possible. Our approach requires as input for this model consistency mapping only pointers to two model elements. In addition, the user will be allowed to configure some parts of the consistency check, such as the strictness and the type of comparison algorithm executed. In the example, this could relate e.g. the parametric diagram "ABS" and the Simulink model "Anti-lock braking." At a more detailed level, we can imagine a relation between a Simulink subsystem and a block in the SysML internal block diagram. Note that this mapping can be performed at other abstraction levels as well. For instance, in a different case we might relate only a block inside a diagram to an entire Simulink model. Furthermore, the mapping is not necessarily constrained to one-to-one relations.

The final part of the approach is not a mapping, but a set of comparison algorithms that are executed on the comparison models. The algorithms are provided in the implementation, but the definition of the consistency checks contains the choice of the comparison algorithm. In this way, different types of consistency can be checked between model elements. For example, we can check for equivalence across model elements, or just for some common values. In our example, we could for example detect if the Simulink model is missing a subsystem that was expected in the SysML block diagram, indicating a probable inconsistency between the two models.

C. Process view

Besides the mechanism by which to define correspondence rules, an industrially applicable consistency checking approach should, from its inception, be designed to fit in with the development process and environment in which it will be used. Here we briefly discuss three aspects of this process view.

Egyed [23] proposes instantaneous checking of consistency, a frequency similar to syntax highlighting in modern IDEs. This frequency may get annoying in a normal development scenario, since during the task of changing a model many trivial inconsistencies will be marked. Therefore, we propose to run the consistency checks at a lower frequency, at a time when developers have completed their tasks and consistency should hold. A good example of such a time is before developers integrate their work into a shared repository.

Since we are considering multi-view modeling and inconsistencies between models in those views, it is relevant to ask who in the development process defines and maintains the consistency rules, as well as who should be involved in resolving identified inconsistencies. Since the answers to these questions depend are specific to a particular development setup, on specific development scenarios, we assume that the implementation should not be limiting to only a small set of users. It is often argued that consistency checking and similar types of supporting tools should be integrated in tools that are already in use by developers. Although a new separate tool is undesirable, since it would be easily overlooked if not already in the development process, integration consistency checking in one of the development tools is also undesirable, since, it can then only be used by users of that tool.

For these reasons, we have, in our prototype implementation, integrated consistency checking in the continuous integration pipeline [32]. Of course, the assumption of the presence of such a pipeline does not hold in all MBD projects. Nevertheless, we do consider this a likely future direction for industrial MBD and notably, one in which inter-model consistency becomes more important due to its short development cycles.

D. Summary and Limitations

In summary, our proposal is to transpose relevant parts of models (i.e. relevant elements as specified in the metamodel

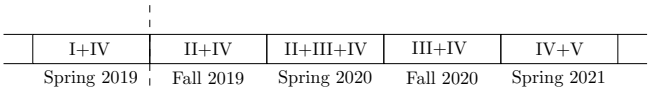

Fig. 2. Timeline for planned activities in the next phase of the research, roman numerals refer to planned items as detailed in Section IV.

level language consistency mappings) to a common format. When the user then creates a specific mapping between model elements, consistency checks can be generated. Then, comparison rules are applied on the models in common formats to detect inconsistencies according to the language consistency mapping. The approach is thus lightweight in usage and can check consistency between heterogeneous models.

The main trade-off of our approach is between expressiveness and ease of use. We argue that in different scenarios in industrial practice, lower levels of expressiveness can suffice. For example, only a mapping between model elements can be enough for an indication of potential inconsistency. With more complex language consistency mappings, the precision of the checks can be increased. Our approach can be applied along this spectrum, but is aimed specifically at providing a lightweight way of defining consistency checks.

One of the other limitations is that we currently focus on local consistency checking, disregarding for now global consistency checking [25]. However, the current approach does support more than one-to-one correspondence rules, depending on the comparison algorithm and the common metamodel. Global consistency rules can then be defined as a sequence of these local consistency checks.

IV. Current status and future work

This section describes the current status and future of the work, as well as plans for its evaluation and validation. Furthermore, a list of planned contributions and a timeline for the next phase of the work is presented. Figure 2 shows an overview of the other planned contributions as discussed below and the planned timeline for them. In it, part I of our research marks the initial problem definition and proof-of-concept implementation of the approach, as described in Section I and Section III respectively. The included plans cover the next two years of the research towards a doctoral dissertation. That timeline roughly corresponds to licentiate, which is a common intermediate degree towards a doctoral degree in Sweden.

The current status is that we have identified a need for a lightweight inter-model consistency checking approach, which is described in the problem description in this paper. Secondly, we have created a proof-of-concept implementation for a narrow application of our approach, namely structural consistency [32]. This is a first step towards a more general approach that is applicable to the industrially most relevant cases.

The next step (II) in the research is to perform empirical research in the target industrial context to uncover the industrially most relevant consistency checking scenarios. This includes the types of languages and diagrams used, the types of inconsistencies that are introduced, and the types of inconsistencies that are most valuable to be notified about (for example because they go often unnoticed). We plan to perform this research in two stages. In the first part, we aim to identify the most relevant types of inconsistencies through interviews with practitioners. The second part consists of a user study in which we present the developers with some emulated but realistic examples of inconsistency and several alternative versions of consistency checks to uncover which of them would be considered most helpful in practice.

When we have determined the exact scope of the consistency checks the next step (III) is to simplify also the definitions of the language consistency mappings and to define an appropriate comparison metamodel to express model elements in. We plan to research ways to allow users to define them as easily as is done for model consistency mappings, for example by deriving language consistency mappings from user-provided examples.

In parallel (IV), we aim to develop the existing proof-of-concept implementation further. When matured, we plan an initial evaluation by MBD practitioners in realistic scenarios with respect to its ability to detect the required types of inconsistencies and the ability of engineers to define the required consistency checks (V).

These steps roughly correspond to one of the following planned contributions:

1) A set of industrially most relevant consistency checking scenarios and the expressiveness of correspondence rules that is required to automatically identify inconsistencies in those scenarios. (II)
2) An approach to defining correspondence rules using minimal user input such that consistency checks can be automatically generated from them. (III)
3) A consistency checking tool implementing this approach such that it can be applied in industrial practice. (IV)

V. Summary

Inter-model consistency is vital in model-based development, but inconsistencies are inevitable in industrial multi-view modeling scenarios. Therefore, we focus on supporting developers by providing simple means of defining correspondence rules that identify inconsistencies during model evolution. We argue that simplifying the consistency check definitions as made by the user is required to promote the usage of a consistency checking approach in industrial practice, which also includes heterogeneous models and complex tool environments.

Our proposed approach thus differs from existing approaches in that it trades expressiveness for required user input, which is minimal. The research is in an early stage and the plans for evaluation and validation are reflecting that, since they include also plans for narrowing the scope of this work. Currently, we have early result on a few of the contributions, showing a proof of concept of a specific scenario within the industrial context we address.

References

[1] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software & Systems Modeling*, vol. 17, no. 1, pp. 91–113, 2018.

[2] D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, p. 25, 2006.

[3] H. Bruneliere, E. Burger, J. Cabot, and M. Wimmer, "A feature-based survey of model view approaches," *Software & Systems Modeling*, Sep 2017. [Online]. Available: https://doi.org/10.1007/s10270-017-0622-9

[4] A. Cicchetti, F. Ciccozzi, and A. Pierantonio, "Multi-view approaches for software and system modelling: a systematic literature review," *Software & Systems Modeling*, pp. 1–27, 2019.

[5] M. Persson, M. Torngren, 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 *Embedded Software (EMSOFT), Proceedings of the International Conference on*. IEEE, 2013, pp. 1–10.

[6] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 37–54.

[7] "ISO/IEC/IEEE International Standard Systems and software engineering — Vocabulary," *ISO/IEC/IEEE 24765:2017(E) (Revision of ISO/IEC/IEEE 24765:2010)*, pp. 1–522, Sept 2017.

[8] "ISO/IEC/IEEE Systems and software engineering – Architecture description," *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pp. 1–46, Dec 2011.

[9] A. C. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh, "Inconsistency handling in multiperspective specifications," *IEEE Transactions on Software Engineering*, vol. 20, no. 8, pp. 569–578, 1994.

[10] A. M. Madni and M. Sievers, "Model-based systems engineering: motivation, current status, and needed advances," in *Disciplinary Convergence in Systems Engineering Research*. Springer, 2018, pp. 311–325.

[11] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill *et al.*, "The relevance of model-driven engineering thirty years from now," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2014, pp. 183–200.

[12] S. Torres, M. van den Brand, and A. Serebrenik, "Model management tools for models of different domains: a systematic literature review," in *International Systems Conference*, 2019.

[13] R. Jongeling, J. Carlson, and A. Cicchetti, "Impediments to Introducing Continuous Integration for Model-Based Development in Industry," in *Technical Track Model-Driven Engineering and Modeling Languages (MDEML) at 45th EUROMICRO Software Engineering Advanced Applications (SEAA) Conference, (Accepted)*, 2019.

[14] H. Giese and R. Wagner, "Incremental Model Synchronization with Triple Graph Grammars," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2006, pp. 543–557.

[15] R. Paige, P. Brooke, and J. Ostroff, "Metamodel-based model conformance and multiview consistency checking," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 16, no. 3, p. 11, 2007.

[16] F. Lucas, F. Molina, and A. Toval, "A systematic review of UML model consistency management," *Information and Software Technology*, vol. 51, no. 12, pp. 1631–1645, 2009.

[17] R. F. Paige, N. Matragkas, and L. M. Rose, "Evolving models in Model-Driven Engineering: State-of-the-art and future challenges," *Journal of Systems and Software*, vol. 111, pp. 272–280, 2016.

[18] C. Gryce, A. Finkelstein, and C. Nentwich, "Lightweight checking for UML based software development," in *Workshop on Consistency Problems in UML-based Software Development., Dresden, Germany*, 2002.

[19] D. S. Kolovos, R. F. Paige, and F. A. Polack, "The Epsilon Object Language (EOL)," in *European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 2006, pp. 128–142.

[20] D. Kolovos, R. Paige, and F. Polack, "Detecting and repairing inconsistencies across heterogeneous models," in *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE, 2008, pp. 356–364.

[21] S. Feldmann, K. Kernschmidt, M. Wimmer, and B. Vogel-Heuser, "Managing Inter-Model Inconsistencies in Model-based Systems Engineering: Application in Automated Production Systems Engineering," *Journal of Systems and Software*, 2019.

[22] S. Herzig, A. Qamar, and C. Paredis, "An approach to identifying inconsistencies in model-based systems engineering," *Procedia Computer Science*, vol. 28, pp. 354–362, 2014.

[23] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 188–204, 2010.

[24] P. Stevens, "Towards sound, optimal, and flexible building from megamodels," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2018, pp. 301–311.

[25] Z. Diskin, Y. Xiong, and K. Czarnecki, "Specifying overlaps of heterogeneous models for global consistency checking," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2010, pp. 165–179.

[26] H. König and Z. Diskin, "Advanced local checking of global consistency in heterogeneous multimodeling," in *Modelling Foundations and Applications*. Springer International Publishing, 2016, pp. 19–35.

[27] ——, "Efficient consistency checking of interrelated models," in *Modelling Foundations and Applications*. Springer International Publishing, 2017, pp. 161–178.

[28] S. Feldmann, S. Herzig, K. Kernschmidt, T. Wolfenstetter, D. Kammerl, A. Qamar, U. Lindemann, H. Krcmar, C. Paredis, and B. Vogel-Heuser, "A comparison of inconsistency management approaches using a mechatronic manufacturing system design case study," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2015, pp. 158–165.

[29] C. Nentwich, W. Emmerich, A. Finkelsteiin, and E. Ellmer, "Flexible consistency checking," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 12, no. 1, pp. 28–63, 2003.

[30] J.-M. Favre, "Towards a Basic Theory to Model Model Driven Engineering," in *3rd Workshop in Software Model Engineering, WiSME*. Citeseer, 2004, pp. 262–271.

[31] J. Di Rocco, D. Di Ruscio, M. Heinz, L. Iovino, R. Lämmel, and A. Pierantonio, "Consistency Recovery in Interactive Modeling," in *MODELS (Satellite Events)*, 2017, pp. 116–122.

[32] R. Jongeling, F. Ciccozzi, A. Cicchetti, and J. Carlson, "Lightweight Consistency Checking for Agile Model-Based Development in Practice," in *15th European Conference on Modelling Foundations and Applications (ECMFA)*, 2019.