Model Management Tools for Models of Different Domains: A Systematic Literature Review

Wesley Torres, Mark van den Brand, Alexander Serebrenik
Eindhoven University of Technology
Eindhoven, The Netherlands
\{w.silva.torres, m.g.j.v.d.brand, a.serebrenik\}@tue.nl

Abstract—Objective: The goal of this study is to present an overview of industrial and academic approaches to cross-domain model management. We aim at identifying industrial and academic tools for cross-domain model management and describing the inconsistency types addressed by them as well as strategies the users of the tools employ to keep consistency between models of different domains. Method: We conducted a systematic literature review. Using the keyword-based search on Google Scholar we have analyzed 515 potentially relevant studies; after applying inclusion and exclusion criteria 88 papers have been selected for further analysis. Results: The main findings/contributions are: (i) a list of available tools used to support the model management; (ii) approximately 31% of the tools can provide consistency model checking on models of different domains and approximately 24% on the same domain; (iii) available strategies to keep the consistency between models of different domains are not common in the industry. However, it is considered as a requirement by academia if one wishes to manage inconsistency between models of different domains. Conclusion: This study presents an overview of industrial practices and academic approaches about the cross-domain model management. The results presented in this study can be used as a starting point for future research on model management topics, and also for further improvement of actual model management tools.

Index Terms—Model Management, Systems Engineering, Model-Based Systems Engineering, Systematic Literature Review

I. INTRODUCTION

As in any discipline, inconsistencies can be found in several stages of the system development life cycle. In earlier stages, when engineers are eliciting requirements, they might misunderstand the stakeholders’ needs. Thus, the stakeholders’ needs might be modeled wrongly, resulting in a product that does not match their expectations. Another inconsistency can arise when the models (e.g., class diagram, activity diagram) are correct but the software developers misunderstand them, resulting in a source code that does not represent the stakeholders’ needs. The crucial point here is that the faster the inconsistency is found, the less it will cost [1].

In the previous examples, only one domain was involved, i.e., the software engineering domain. In these scenarios, identifying and managing inconsistencies is already difficult. However, this task can become even more complicated when it involves multiple domains.

Furthermore, systems are becoming increasingly complex to manage. The complexity of design systems increases, especially when these systems are heterogeneous and there is a need to combine models created by engineers from different expertise and different domains [2]–[4]. One example of such a complex system is a mechatronic component: to develop it, one might need to combine expertise from different domains such as mechanics, electronics and software [5].

Due to sheer complexity of modern systems and presence of multiple authors, inconsistencies between the models might be inadvertently introduced, e.g., one model might assume the presence of a certain feature while another one might assume its absence. This problem might be further amplified by differences in terminology used in different domains: e.g., for a software engineer, a feature is a functionality provided by the system, but for system engineer, a feature is an aspect of the system, like the color. This kind of misunderstanding can affect the consistency of the models. Therefore, the terms have to be well described in order to simplify the process of maintaining consistency between models.

Maintaining consistency between models is known to be a challenging task, especially because it is difficult to predict the effects of changes introduced in one model on other models [6]. While maintaining consistency between models is imperative [7], in practice, it can never be fully ensured [8], and the system engineer is responsible to define what has to be consistent and when. The process to manage these models can be expensive. Thus, we believe that the consistency should only be managed when the costs to maintain the consistency are lower than the costs that an eventual inconsistency can cause. The process of organizing and maintaining models ensuring consistency is known as model management.

The goal of this study is to present an overview of industrial practices and academic approaches to cross-domain model management. We aim at identifying industrial and academic tools for cross-domain model management and describing the inconsistency types addressed by them as well as strategies the users of tools employ to ensure consistency between models of different domains. The Research Questions (RQ)s used to guide our study are the following:

- **RQ1:** What model life cycle management tools are available?
- **RQ2:** What inconsistency types are addressed by the model life cycle management tools?
- **RQ3:** Which strategies have been used to keep the consistency between models of different domains?
To achieve this goal we have conducted a systematic literature review. Taking into consideration the fact that scientific publications do not always reflect industrial practices, we have decided to include white papers, such as technical reports. Thus, we have covered both industrial and academic sources.

The main contributions of this study are:
- List of model management tools;
- Classification of inconsistency management approaches;
- Identification of gaps and future work.

II. BACKGROUND

We introduce the concepts related to model management.

INCOSE\(^{1}\) defines **model-based systems engineering** (MBSE) as “the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.” [9]. According to Friedenthal et al. [10] MBSE was proposed to facilitate systems engineering activities: following MBSE the system engineers would use models instead of documents, and this is expected to improve quality of system specification and design, as well as the communication among the development team.

A **model** is a representation of reality, an abstraction of something relevant to the stakeholder described using well-defined and unambiguous languages [11].

**Model management** emerged with the need to organize and maintain models, ensuring consistency. Franzago et al. [12] state that the infrastructure for the model management may include a model repository and modeling tools. This infrastructure is responsible for managing the life cycle of the models such as creating, editing, and deleting.

**Product lifecycle management** (PLM) [13]–[17] is an environment, infrastructure, a system of methods, processes, and practices that cover the entire product lifecycle, from requirements definition, design, to late stages such as maintenance and recycling of the product. While model management is focused on the models of the product, PLM includes every artifact related to the product. Since PLM can include model management features, we have decided to include it.

III. METHODOLOGY

A. Selecting the Literature Review Technique

In order to answer the Research Questions (RQ1–RQ3), we conducted a literature review. Several literature review techniques have been proposed in the scientific literature, e.g., snowballing [18], [19], systematic literature review (SLR) [20], and systematic mapping review (SMR) [21].

We opted for SLR because of the SLR characteristics that identify, analyze and interpret the data related to specific RQs. In contrast, SMR aims to answer general research questions and snowballing can be labor intensive. Thus, we believe that this approach is the most appropriate to answer our RQs. To circumvent the inherent SLR limitations implied by the choice of the search strings, we have combined different keywords obtaining 600 different search strings. This process is more extensively explained in the next section.

The SLR consists of the creation of research questions (RQs), the queries on electronic sources (data source) having the RQs as a guide, and the use pre-determined criteria for eligibility and relevance to form the set of accepted papers to be used in the study. As the data source, Kitchenham and Charters [20] recommend the use of a search engine that offers a wide coverage of sources. Thus, we have chosen Google Scholar: it offers a wide coverage of electronic sources of different research areas, and it has been used in multiple software engineering studies [18], [22]–[24].

B. Data Extraction

Since we are mainly interested in **tools** (product life cycle management tools, and model management tools) we create search strings to query Google Scholar, based on **PICO** [25]. Thus, we have selected and organized keywords into four categories: process supported by tools, model, consistency, and multiple domains. For each category, we have selected keywords related to the Research Questions, as presented in Table I.

The first two categories are the base for answering all RQs, and are more specific for answering RQ2 and RQ3. For example, the reasoning behind choosing the keyword “Dependency” in the category “Consistency” was that in our initial research we found that dependency modeling has been used to maintain consistency. Thus, this keyword could help find more results that could answer RQ3.

We have combined the keywords from different categories to create queries to be executed in Google Scholar. For instance, for the first query we have used the following keywords: ”Model Management, MBSE, Consistency, Multidomain Model Integration”. For the second query we used ”Model Management, MBSE, Consistency, Multi Domains”, and so on. In total we have \(600 = 4 \times 10 \times 3 \times 5\) combinations.

Due to the similarity of the queries, some papers have been retrieved multiple times. We have automatically excluded these duplicates prior to the manual inspection. In total we have obtained 3222 hits, but only 515 of them were unique.

C. Manual Inspection

The selection criteria was defined in order to avoid bias and to reduce subjectivity. The inclusion (I) and exclusion (E) criteria were designed to answer the RQs, as proposed by [26]. The following are the inclusion and exclusion criteria we used:
- **I1**: Studies written in English and available in full-text.
- **I2**: Studies review or propose of a new technique, approach, method, or tool (prototype) that support model management.
- **I3**: Studies mention tools related to Product Lifecycle Management (PLM) and Model lifecycle management.
- **E1**: Studies do not mention model (in)consistency.

\(^{1}\)The International Council on Systems Engineering (INCOSE) is a non-profit membership organization dedicated to the advancement of systems engineering and to raise the professional stature of systems engineers. https://www.incose.org
### TABLE I

<table>
<thead>
<tr>
<th>Process Supported by Tools</th>
<th>Model</th>
<th>Consistency</th>
<th>Multiple domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Management</td>
<td>Model-Driven Systems Engineering</td>
<td>Consistency</td>
<td>Multidomain Model Integration</td>
</tr>
<tr>
<td>Model Lifecycle Management</td>
<td>Model-Driven System Engineering</td>
<td>Inconsistency</td>
<td>Multi Domain</td>
</tr>
<tr>
<td>Product Lifecycle Management</td>
<td>Model-Driven Systems Engineering</td>
<td>Dependency</td>
<td>Multiple Domains</td>
</tr>
<tr>
<td>Systems Lifecycle Management</td>
<td>Model-Driven Systems Engineering</td>
<td></td>
<td>Domains</td>
</tr>
<tr>
<td></td>
<td>Model-Driven Systems Engineering</td>
<td></td>
<td>Heterogeneous</td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>The $\kappa$ Value Obtained in Each Iteration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>First Review Round</td>
</tr>
<tr>
<td>Second Review Round</td>
</tr>
<tr>
<td>Third Review Round</td>
</tr>
<tr>
<td>Fourth Review Round</td>
</tr>
</tbody>
</table>

• E2: CVs, PhD and Master theses, and books or book chapters. Although we excluded all PhD theses, we considered publications related to the PhD theses and applied the inclusion and exclusion criteria to them. We decided to check for derived papers because we chose to be as conservative as possible, and we did not want to exclude PhD theses without checking for derived papers.

In the end of this process, we included 20 derived papers.

In order to identify the relevance of the paper, we read the title, abstract and conclusion of 515 papers. Relevance assessment was performed iteratively. At each iteration we used 15 papers randomly selected from the list of papers we had downloaded. The first and the last authors of this paper, individually read the title, abstract and conclusion to label the relevance of the paper. Both of the raters were software engineering researchers having at least a Master’s degree in Computer Science. At the end of each iteration, we computed Cohen’s $\kappa$ [27] to measure the agreement between the raters and discussed the disagreements. According to Cohen [27] and Landis et al. [28] the Kappa coefficients in the range of 0.61–0.80 is interpreted as substantial agreement and this range was used in previous studies [29]–[31].

As presented in Table II, four review rounds were needed to reach the $\kappa$ value greater than 0.6. In the first review round we obtained the lowest $\kappa$ value, because the interpretation of the inclusion and exclusion criteria were not clear among the raters. We improved the agreement level in every subsequent review round. In the second and third review rounds we obtained 0.29 and 0.33 respectively. We finalized the process in the fourth round with the agreement level of 0.61.

When we reached the acceptable agreement level, the first author continued the selection procedure independently. After reading the title, abstract and conclusion of 515 papers, we labeled 168 as possibly relevant. To finalize, the first author completely read these papers and selected 88 papers to answer the Research Questions. Figure 1 presents the summary of the process to select the papers.

### D. Gathering information about tools

The previous section describes how to identify relevant papers supporting us to find the answers for the RQs. Although the papers provide a list of tools, the information regarding to the tools is not necessarily well presented or detailed. As a consequence, we decided to use additional data sources, for instance the website of each tool. One option to gather information about the tools would be to install and to try all the tools. However, this option was not feasible, mainly because of the need to know how to use them, but also because most of the tools are commercial, requiring the license to try them.

We selected the closed card sorting technique [32] to categorize the type of consistency the selected tools address. Taylor et al. [33] describe consistency as “an internal property of an architectural model, which is intended to ensure that different elements of that model do not contradict one another” and distinguish the following four inconsistency types.

a) Name inconsistency: components, connectors or services have the same name.

b) Interface inconsistencies: connected interface elements have mismatching values, terminologies or schemes [34]. Name inconsistencies are interface inconsistencies but not vice versa.

c) Behavioral inconsistencies: Taylor et al. [33] explain that these inconsistencies “occur between components that request and provide services whose names and interfaces match,
but whose behaviors do not.” This kind of inconsistency can happen when the behavior of the element is not the expected one. An example of behavioral inconsistency would be if the service provider assumes that the distance is expressed in kilometers and the requester assumes it to be in miles.

d) Interaction inconsistencies: These inconsistencies “occur when a component’s provided operations are accessed in a manner that violates certain interaction constraints, such as the order in which the component’s operations are to be accessed.” [33].

e) Refinement inconsistencies: These inconsistencies occur between models of different abstraction levels.

We used the consistency types identified by Taylor et al. [33], to label the selected tools. We opted to use the consistency types provided by the selected tools, in those cases which it was not possible to match the consistency types identified by Taylor et al. with the description of the consistency type provided by the tools. We labeled DNF (data not found), those tools that we did not find information about.

IV. RESULTS

A. RQ1 - Which model life cycle management tools are available?

We analyzed the descriptions of the tools that were mentioned in the selected papers.

a) Provide consistency model checking on models of different domains: ADES [35], [36], CADOM [37], Capella (Arcadia) [38], [39], CATIA v5 [40]–[44], CoDeMo [42], COLIBRI [35], Dassault Systèmes PLM platform [41], [43]–[51], Dymola [34], [41], [52], IntePLM [53], Isight [50], [54], LOTAR [44], Melody [55], [56], Modelica Development Tooling (MDT) [57], OntoSTEP (Protégé plug-in) [40], [58], [59]–[60], OpenModelica [57], ParaMagic [56], [61], PDES [44], ProSTEP [44], Siemens PLM Tools [40], [43], [47], [49], [51], [62]–[65], Simulink [52], Synedia (SLIM) [55], [61], [66], [67], SysDice [45], SysML4Modelica [7], [68], [69], VE4PD [70], as well as the academic prototypes of István Dávid [4], [71], and Diana Penciu [72].

b) Provide consistency model checking but on models of the same domain: Agile PLM (Oracle) [49] Artisan Studio [61], [73], Autodesk Vault [43], Cameo [63], ControlBuild [52], EPLAN [68], EUCLID [43], Magic Draw [6], [7], [50], [61]–[63], [67], [73]–[75], Mechatsoft [69], Mechatronics Concept Designer (Siemens) [45], ModelCenter (Phoenix Integration) [50], [54], MOFLO [69], [73], [76], [77], Windchill (PTC) [36], [43], [49], [78], ParaSolver [56], Rational Rhapsody [50], [61], [73], SCAD [52], SimMobLi [79], Simscape [45], [69], SolidWorks 2010 [40].

c) Do not provide any consistency checking: We assumed that tools that do not explicitly claim that they provide consistency check, do not have this functionality. Acceleo [76], Athena Project [36], ATL [76], ATOM3 [76], Comet Workbench [54], DXF [80], E2KS tool [35], EA Parametrics (Enterprise Architect) - Sparx systems [56], EAST-ADL2 [5].

Entime [81], Epsilon [76], FUJABA [69], [76], GAM framework [82], REAt [76], IBM’s Jazz Collaborative Lifecycle Management (CLM) suite [83], iFES project [54], IGES [80], Interdisciplinary Communication Medium (ICM) [37], KerMeta [76], modeFRONTIER [54], ModelHeiX framework [74] Modelcrear consortium/FMI: Functional Mock-up Interface [46], [84], OBIIS [85], OntoPLM framework [72], Open Service for Lifecycle Collaboration (OSLC) [45], Pro/Engineer Wildfire 4.0 [40], Product Design Graphics System (PDGS) [43], ProOne [86], [87], Rosetta [74], SAP Product Life Cycle Management [51], Share-A-space (Eurostep) [49], SPML [88], VIATRA [76], Vitezcor [50], VW Surf [43].

B. RQ2 - What consistency types are addressed by the model life cycle management tools?

In order to answer RQ2, we classified the consistency types addressed by the tools identified in the previous subsection. We focus on those tools that could provide model consistency check at some level, more specifically regarding to what type of consistency those tools provide. However, not all tools described the consistency check.

Table III presents the list of tools and the consistency types they address. Additional description of the consistency check those tools address are presented bellow:

- Requirement Consistency - It checks whether the requirements from a requirement list are related to some model element and if this relationship is valid.
- Information Consistency - It checks if the data that can be presented on different media, remain the same regardless of how they are presented [89]. An example of Information inconsistency would be when the distance is presented in different units without respecting the conversion calculation.

C. RQ3 - Which strategies have been used to keep the consistency between models of different domains?

We have selected papers that cited tools that manage consistency between models of different domains. We selected 56 papers; however, only half described how they check and keep the consistency between models.

Qamar et al. [62] believe that the way to manage inconsistency is through interoperability between tools like MagicDraw, TeamCenter, and Simulink. Wegner [90] defines interoperability as “the ability of two or more software components to cooperate despite differences in language, interface, and execution platform”. On one hand, standard file formats as [91], [92] and XML [93] are used to maintain interoperability in engineering and software domains. On the other hand, the use of these standard files to maintaining consistency could be problematic due the data loss [94], since the data would be transiting between different tools and domains.

We organize the papers into categories according to the approach they use to keep consistency between models of different domains.

1https://nl.mathworks.com/help/srl/multitool/ug/requirements-consistency-checks.html
**TABLE III**

**TOOLS AND KIND OF CONSISTENCY. Data not found means that, although these tools claim to provide consistency check, they do not describe what kind of consistency check they provide and at which level.**

<table>
<thead>
<tr>
<th>Consistency Type Checking</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Consistency</td>
<td>Modelica Development Tooling, Open Modelica, CATIA v5, SysML4Modelica, Prototype [72], ADES</td>
</tr>
<tr>
<td>Behavioral and Interaction consistency</td>
<td>OntoSTEP (Protegè plug-in)</td>
</tr>
<tr>
<td>Interaction Consistency</td>
<td>CADOM, COLIBRI, SysDice, Prototype [82]</td>
</tr>
<tr>
<td>Interface and Requirement consistency</td>
<td>Simulink</td>
</tr>
<tr>
<td>Information Consistency</td>
<td>VEAPD</td>
</tr>
<tr>
<td>Data Not Found</td>
<td>Dassault Systèmes PLM platform, Insight, Syntedia/SLIM, CoDeMo, IntePLM, Dymola, Capella (Arcadia), Siemens PLM Tools, Prototype [4], [71]</td>
</tr>
</tbody>
</table>

a) **Inconsistency Patterns:** in order to manage inconsistency, this approach recommends selecting the appropriate technique from an extensible catalogue of inconsistency patterns, and apply it in an unmanaged process to achieve a managed one [4], [71].

b) **Modeling dependencies explicitly:** some researchers believe that making the inter/intra-model dependencies explicit will facilitate the model management. Such dependencies can be identified between properties or between structural elements of two models, in such a way that the properties or elements can affect each other. This dependency modeling can be done using any technology that explicitly map dependency, such as SysML, DSLs, and DSM (Design Structure Matrix) [6], [54], [62], [63], [65], [95]. Specifically, DSM is used to make the direct and indirect dependencies explicit. It is a representation of the components and their relations, in order to make the shared information more precise and less ambiguous [5], [53], [96]. DSM consists of a matrix with properties mapped horizontally and vertically. Each marked box inside a cell of a DSM indicates a dependency between the corresponding properties. A dependency loop occurs when there is a dependency marked above the main diagonal on the DSM. In order to avoid these loops, a reorganization of the DSM is needed [54].

c) **Parameters or constraints management:** This approach proposes using parameters or constraints to check the model consistency within a multi-disciplinary development team. If these parameters or constraints are violated, the inconsistency can be detected and managed. According to [47], to implement this approach, it is necessary to have a well-designed data model. [37], [47], [48], [67], [70], [82]

d) **Ontology:** Ontology is an explicit specification of a conceptualization and consists of several components, e.g.: concepts, relations, attributes, instances and axioms [97]. This approach allows engineers to independently develop partial descriptions of the same product and check consistency when the descriptions are combined [98]–[100].

e) **STEP:** Standard for the Exchange of Product model data (ISO 10303) [101]. STEP consists of a number of components, called application protocols (APs), which define data models on which translators for CAD data exchange are based. The International Organization for Standardization (ISO) developed STEP in order to cover a wide range of application areas, such as automotive, aerospace, and architecture [92]. In our systematic literature review, we have not found papers that only use STEP to check consistency between models of different domain; instead, all papers use an extension of STEP, or a combination of STEP and other technologies [40], [58], [60], [92].

f) **KCModel:** This approach is organized basically into “Information Core Entity” (ICE) and “Configuration Entity” (CE). The former is the smallest information entity used, responsible for storing parameters and rules, and represents a generic multi-domain baseline. In order to use the parameters and rules in a specific context (3d, thermal calculations excel files, etc.), it is necessary to create a Configuration Entity instantiating ICE. This approach allows engineers to create their own models, trace parameters and rules, and check consistency. [35], [72], [102], [103]

V. **DISCUSSION**

The results described in this paper can serve as a starting point for future research on model management topics. We provide a list of available tools used to support the model management. We group them according to the functionality they offer related to the consistency model checking on models of different or same domains. We observe that approximately 31% of the tools we found can provide consistency model checking on models of different domains, approximately 24% on the models of the same domain, and approximately 45% do not provide any consistency model checking.

Regarding commercial tools, we have found that they do not fully describe the kind of inconsistency they can address (Section 4.2). This lack of information makes it difficult to map the inconsistencies theses tools can handle, since these tools are commercial and we would need the licenses and the expertise to use them. Further evaluation on commercial tools is necessary, and this should be done with the help of specialists of each tool or at least a full description of all features should be provided. For those tools that describe the consistency type they address, we have found that the majority of them can perform the interface consistency check, checking whether the connected interface elements have mismatching values. We expected that these tools could address more than one kind of consistency type. However, this was not what we observed. We observed that most of them address only one.

Our study reveals (Section 4.3) that the strategies to keep the consistency between models of different domains are not mature enough, because most of them are based on prototypes or methodologies/approaches.
We found that explicit dependency modeling between models is not commonly used in the industry [54]; however, this is regarded as a requirement by academia if one wishes to manage inconsistency between models of different domains. Qamar et al. [54] claim that “Capturing dependencies formally and explicitly is currently not supported by available methods and tools in MBSE, and having no explicit knowledge of dependencies is a main cause of inconsistencies and potential failures”. The explicit dependency modeling between models can be done using design struct matrix, ontology, and it can be followed by the use of standards as STEP.

VI. Threats to Validity

Wohlin et al. [104] provide a list of threats that researchers can face during a scientific research.

External validity concerns about how the results and findings can be generalized. We only accepted studies written in English, and this can represent a threat despite the fact that English is the most widely used language for scientific papers. As one of the goals of this study is to understand what the industrial practices are we decided to accept gray literature (white papers and technical report).

The fact that we could not try all the tools and we could not find the full description of the kind of inconsistency they address can represent a threat. We believe that this threat can be minimized with the help of specialists of each tool or at least a full description of all features should be provided.

Internal validity: Google scholar continuously indexes new papers. Hence running the queries at different moments of time might lead to different results. However, it is not possible to run all queries simultaneously due to limitations of Google scholar. We do not think that a considerable amount of papers was missed, since all queries were similar to each other and more than half of our query results were duplicated hits.

VII. Conclusions

We presented a systematic literature review intending to give an overview of industrial practices and academic approaches to cross-domain model management. We started with 515 potentially relevant studies and after a rigorous selection criteria we concluded the process with 88 papers.

We provide a list of available tools used to support the model management. We observed that approximately 31% of the tools can provide consistency model checking on models of different domains, approximately 24% on the same domain, and approximately 45% do not provide any consistency model checking.

Our study reveals that the strategies to keep the consistency between models of different domains are not mature enough because most of them are based on prototypes or methodologies/approaches, and the majority of these tools address only one kind of consistency.

We found that explicit dependency modeling between models is not commonly used in the industry. However, it is considered as a requirement by academia if one wishes to manage inconsistency between models of different domains. Due to the lack of details about the kind of inconsistency that commercial tools address, we suggest that a further evaluation on commercial tools is needed. This should be done with the help of specialists of each tool or at least a full description of all features should be provided.

To conclude, we observe that more research has to be done to improve the quality of the approaches and tools used to ensure consistency. There is no silver bullet, but at least we have a set of strategies that together can provide consistency.

REFERENCES


