Management of Inconsistencies in Domain-Spanning Models – An Interactive Visualization Approach

Stefan Feldmann[1(✉)], Florian Hauer[2], Dorothea Pantförder[1], Frieder Pankratz[3], Gudrun Klinker[3], and Birgit Vogel-Heuser[1]

[1] Institute of Automation and Information Systems, Technical University of Munich, Munich, Germany
stefan.feldmann@tum.de
[2] Chair of Software Engineering, Technical University of Munich, Munich, Germany
[3] Chair for Computer Aided Medical Procedures & Augmented Reality, Technical University of Munich, Munich, Germany

Abstract. The complexity of automated production systems increases steadily – especially due to the rising customer demand to manufacture individualized goods. To stay competitive, companies in this domain need to adapt their engineering to deliver machines and plants with higher quality in shorter time. Hence, to reduce design errors and identify problems already in early engineering stages, it is essential to ensure that the disparate engineering models – e.g., from mechanical, electrical and software engineering – are free from inconsistencies. This paper presents a concept for inter-model inconsistency management. In particular, the proposed concept provides an interactive visualization approach that captures the dependencies between the different engineering models explicitly and visualizes them to the involved stakeholders. By that, the location of and cause for inconsistencies can be identified more easily; dependencies between the different engineering disciplines can be visualized in a comprehensive manner.

Keywords: Model-based systems engineering · Automated production systems · Inconsistency management · Semantic web technologies

1 Introduction

Engineering in the automated production systems domain (aPS) is an interdisciplinary process and incorporates a multitude of heterogeneous, domain-specific models [9] – e.g., requirements models, engineering models as well as analysis models. Although these disparate models consider different aspects of the system under investigation, they are not completely disjoint [3]. Consequently, overlaps – that is, information being introduced redundantly – are present in between the different models [21] and, hence, inconsistencies are likely to occur [10]. Such inconsistencies reflect a state of conflict between the involved

S. Yamamoto (Ed.): HIMI 2017, Part I, LNCS 10273, pp. 71–87, 2017.
DOI: 10.1007/978-3-319-58521-5_5

models that results from conflicting information [21] being introduced in the different, domain-specific views. Furthermore, inconsistencies do not necessarily lead to engineering errors, but may rather also indicate aspects that need to be further elaborated by the involved stakeholders [21]. Hence, to ensure a high quality engineering process, it is essential to continuously check for and handle these inconsistencies.

Especially within large-scale mechatronic systems, complex dependencies exist between the different models. The reasons for these complex dependencies are manifold: For one, the enormous system complexity that results, e.g., from the increasing customer demand for lower costs and better quality, is one essential reason for complexity. In addition, the variety of stakeholders from different domains and with individual mental models that are involved in the engineering of aPS is an essential factor that leads to numerous dependencies in between different engineering models. Hence, inconsistencies cannot be regarded individually – rather, they must also be regarded from an overview of the entire system. Especially as the various models also differ in the used modelling language and level of detail [9], system designers must be able to switch between different levels of detail to maintain this overview. Hence, this paper aims at extending our existing inconsistency management framework [7] by an interactive, configurable visualization approach that envisions three central objectives:

- an extensible framework based on a *standardized representational formalism* using Semantic Web Technologies – namely, the Resource Description Framework (RDF) [25] and the SPARQL Protocol and RDF Query Language (SPARQL) [23,24],
- an interactive, *graph-based visualization* capturing model *entities* and *dependencies*, and
- a *rule-based configuration* to configure and adapt the visualization regarding different levels of abstraction.

By means of such an interactive inconsistency management approach, we envision the basis for an adaptable and extensible framework that can be applied easily to a variety of applications. The remainder of this paper is structured as follows: From a comparison of the related research works (Sect. 2), we derive the requirements that must be solved by an interactive and configurable visualization approach for the purpose of inconsistency management (Sect. 3). Based on these requirements, we introduce our concept in Sect. 4. In Sect. 5, we show how our concept can be applied to a typical scenario in the aPS domain at the hand of a prototypical software implementation. The results obtained from our research as well as an outlook on further research potential are given in Sect. 6.

2 Related Work

Within this section, our inconsistency management approach is compared to the related work. In particular, we discuss related research in the field of model-based (systems) engineering (see Sect. 2.1), followed by inconsistency management approaches (see Sect. 2.2) and different visualization approaches that aim

at visualizing the interconnections between heterogeneous engineering models (see Sect. 2.3). Finally, our findings are summarized in Sect. 2.4.

2.1 Model-Based (Systems) Engineering

Model-based engineering approaches are especially important to the software engineering domain [18], as they allow for problem and solution abstraction. A variety of description formalisms are available to the software engineering domain, e.g., modelling languages such as the Unified Modelling Language (UML). Such modelling languages are more and more applied in the aPS engineering domain. Especially model-based systems engineering (MBSE) is part of a trend from document-centric to model-centric approaches [12] and entails the specification of many other types of models like the Systems Modelling Language (SysML), which enables engineers to focus on an overarching system's perspective. However, the different disciplines involved in the development process of aPS use their domain specific models [6], e.g., CAD drawings in mechanical engineering or contact planes in electrical engineering. Especially due to semantic overlaps [21] that arise among these disparate models, inconsistencies are likely to occur and, hence, it is complex to handle this complexity and heterogeneity of the distinct model types [3]. Even standardized export formats such as the Extensible Markup Language (XML), which try to detach the abstract information from the model implementation and try to make the models tool independent, are not sufficient, as the tools and models are loosely coupled [15] and implementations differ, leading to compatibility issues [7].

2.2 Inconsistency Management Approaches

For the purpose of being able to address inconsistencies within the engineering process of aPS, the specification, diagnosis and handling of inconsistencies are essential steps [17]. In previous works [6], we identified three distinct approaches for inconsistency management – namely proof theory-based, rule-based and synchronization-based inconsistency management approaches. Our main findings indicate that, due to their flexibility, rule-based inconsistency management approaches are beneficial, as rules can be easily extended, adapted and configured according to company-, project- or stakeholder-specific needs.

Besides the means to specify, diagnose and handle inconsistencies, the incorporation of dependencies between the models – that is: model links – is another important requirement for inconsistency management. In the literature, a multitude of approaches, e.g. for model weaving [13] and model-merging [14], can be found. However, none of the approaches provides the appropriate means to interactively support stakeholders in verifying such links.

2.3 Visualization of Heterogeneous Models

Potentially, thousands of inconsistencies can be found in engineering models, so that a solely textual representation of diagnosed inconsistencies is not sufficient

to adequately support the stakeholders. Additionally, in many cases, stakeholders in the aPS domain must be supported in better understanding the dependencies that lead to an inconsistency.

In information visualization, especially in graph visualization, many approaches are dealing with the representation of large amounts of information and their relationship and dependencies among each other. These approaches provide techniques and principles, not only to visualize the information, but also to navigate and modify it. *Focus and Context* [16] is one of the main principles to support the users in the recognition of relations and dependencies and in decisions making. It allows the user to focus on a special area of interest in a very detailed view, without losing the context of the information. The *Visual Information Seeking Mantra* [19] provides an approach of "overview first, zoom and filter, then details-on-demand" to get detailed information from a given visualization. Hence, visualization can be also an essential part of inconsistency management [7]. A first visualization approach is given in [2], where the requirement is stated that such an visualization approach has to be interactive and has to offer the possibility to change how information is visualized. It is noted in [2] that there is currently no such visual analytic tool available to system engineers. Although the approach is promising and provides first insights into visualization of model information, inter-model dependencies and inconsistencies, it does not provide the means of visualizing any input model type, since information is read in directly from the models. It is argued that by use of an abstraction mechanism, together with a hierarchy and filtering mechanism for complexity reduction, is flexible and extensible enough to visualize any kind of graph-structured information. An approach that aims at better describing the dependencies between engineering models is presented in [8]. Therein, templates are used to describe the relationships between models – however, visualization in the context of inconsistency management is not addressed by that approach.

2.4 Summary

As can be seen from the related work, a multitude of research works can be found in and around the field of inconsistency management in the aPS domain. However, to the best knowledge of the authors, there exists currently no approach that combines the distinct fields – namely model-based systems engineering, inconsistency management and visualisation – to a holistic approach that aids stakeholders in specifying, diagnosing and handling inter-model inconsistencies in a comprehensive and interactive manner.

3 Requirements

As a basis to develop an interactive visualization approach for the purpose of inconsistency management, a variety of requirements need to be fulfilled. These requirements mostly stem from a review of the related research, extended by our experiences gained throughout developing our inconsistency management framework as described in [7].

Requirement 1: Extensible and adaptable approach

Although a multitude of inconsistency management frameworks has been presented in the related literature [6], it is essential that such an approach is tailored to the specific application domain – i.e., the aPS domain. However, as the concrete model types to be investigated and the inconsistencies to be managed highly depend on the specific set-up – e.g., the respective company and its guidelines or the specific project – we argue that an inconsistency management approach must be *extensible and adaptable for company- and/or project-specific purposes*. Such an extensibility and adaptability can, e.g., be achieved by relying on standards and by providing reference implementations that are comprehensive to the respective domain experts. Besides, it is essential that, by means of configuration, tailoring to the specific set-up is enabled.

Requirement 2: Interactive approach

Inherently, inconsistency management is an interactive process. Mostly, critical inconsistencies stem from overlaps between engineering models – that is, from a set of distinct models being in conflict due to similar or redundant, but non-consistent information. If such an inter-model inconsistency is diagnosed automatically, stakeholders must decide, which handling action should be taken – e.g., to ignore, tolerate or resolve the inconsistency [17]. Whereas in some cases, inconsistency resolution proposals can be generated automatically, this is certainly not the normal case. Rather, stakeholders must negotiate with respective stakeholders from other disciplines to identify the cause for an inconsistency and to evaluate handling alternatives. In addition, if a respective handling action is performed, stakeholders must be supported in evaluating whether an appropriate result was achieved by the chosen handling alternative. Consequently, an *interactive inconsistency management approach* is essential for engineering in the aPS domain.

Requirement 3: Visual representation of dependencies

Especially in industrial settings, complexity is an essential issue to be addressed by inconsistency management. This complexity is the consequence of three essential reasons: First, this complexity results from the systems' complexity – models that are created during engineering of aPS often consist of thousands of entities. Second, the multitude of stakeholders from different domains with their individual mental models that are involved during engineering causes the introduction of a vast number of dependencies in between these domain. Third, and resulting from the first two reasons, the number of inconsistencies that are likely to occur is enormous. As a consequence, in order to address this enormous complexity, it is essential that stakeholders are supported visually. Therefore, we believe that a *visual representation of dependencies* between the different engineering models is a key requirement for engineering projects in the aPS domain.

Requirement 4: Comprehensive support in finding inconsistencies

Given an extensible (cf. Requirement 1) and interactive inconsistency management approach (cf. Requirement 2) as well as a visual representation of model dependencies (cf. Requirement 3), the basis for an interactive inconsistency management approach in the aPS domain is laid. However, in order to support the management – that is, the diagnosis and handling – of inconsistencies, it is essential that stakeholders are supported in *finding relevant inconsistencies* in the heterogeneous engineering model landscape. Thus, it is essential that the interactive visualization approach is able to support engineers in determining, which parts of the entire model landscape is interesting to them.

4 Concept

Based on the previously introduced requirements (cf. Sect. 3), this section introduces our conceptual interactive inconsistency management framework (see Fig. 1). The *model management* part of the inconsistency management framework (cf. Sect. 4.1) is responsible for transforming models into a knowledge base. Inconsistency *diagnosis and handling rules* are applied to diagnose and handle inconsistencies (cf. Sect. 4.2). *Visualization rules* define and configure the interactive *visualization*, which is displayed to the user as a directed graph. This rule-based concept for configuring the visualization to company-, project- or stakeholder-specific needs is discussed in Sect. 4.3. By means of these rules, different levels of abstraction can be incorporated and displayed to support users in finding the critical model aspects.

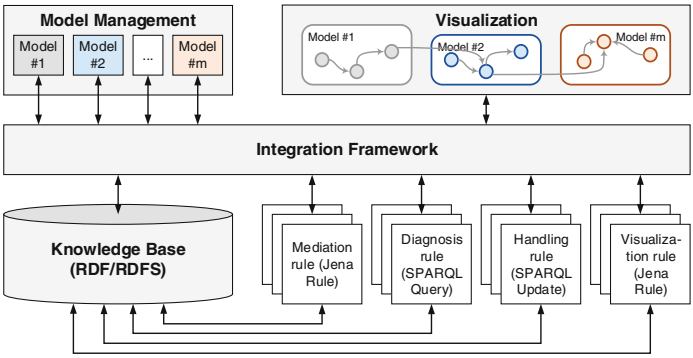


Fig. 1. Overview of the inconsistency management framework based on [7]

4.1 RDF(S) as Representational Formalism

As a basis for inconsistency management, it is essential to ensure a common representational formalism for all involved engineering models [7]. Within our inconsistency management framework, we, hence, make use of the Resource Description Framework (RDF). RDF is a highly flexible "framework for representing information in the Web" [25]. Knowledge is represented in form of RDF graphs, which are sets of triples, consisting of a *subject – predicate – object* statement. Especially as engineering models are, basically, a collection of such predicated statements (e.g., *sensor – is a – component*), such graphs are appropriate for capturing the knowledge modelled in the distinct engineering models. Hence, by means of RDF, a common representational formalism for the different engineering models is provided. More details on how RDF is applied for inconsistency management can be found in [7].

4.2 SPARQL for Diagnosing and Handling Inconsistencies

Given that such a common, representational formalism is available, respective concepts for diagnosing and handling inconsistencies can be put in place. In particular, we make use of the SPARQL Protocol and RDF Query Language (SPARQL). SPARQL defines a set of specifications that contain, among others, the set-based SPARQL Query Language [23] for querying against RDF graphs (i.e., for diagnosing inconsistencies) as well as the SPARQL Update Language [24] for manipulating RDF graphs (i.e., for handling inconsistencies). Especially as inconsistencies can mainly be regarded as patterns to be matched against the engineering models, we define each inconsistency type as a SPARQL pattern by means of a SPARQL query. Each pattern match found in the RDF graph is then a diagnosed inconsistency that is represented to the respective user. Analogously, any resolution rule that can be used to resolve the inconsistent pattern is represented through a SPARQL update action, which replaces the inconsistent pattern by a consistent one. More details on how SPARQL is applied for inconsistency management can be found in [7].

4.3 Rule-Based Interactive Graph Visualization to Support the Different Stakeholders in Diagnosing and Handling Inconsistencies

Although a multitude of different inconsistency diagnosis and handling rules can be specified by our inconsistency management framework, it is essential that stakeholders are supported in identifying the inconsistent parts of their models and in selecting the appropriate handling action to be taken. Especially as we use a graph-based representational formalism for the distinct engineering models, it is obvious that we use graph visualization technique for the respective stakeholders. Additionally, for an optimal stakeholder support, we envision a flexible visualization of the model and inconsistency information as well as the

inter-model dependencies. In order to provide maximum flexibility and the possibility to visualize any model type, the visualization framework makes use of rules. By means of these rules, stakeholders can configure their specific visualizations on different levels of abstraction, thereby aggregating information and easing comprehensibility.

Additionally to providing the means to define and execute visualization rules, two essential concepts are necessary for visualization to the respective stakeholders: a *clustered graph structure* to allow for aggregating information and, hence, easing comprehensibility as well as a mechanism for finding the appropriate *hierarchical clusters* for visualization. These concepts are presented in the following.

Clustered Graph Structure. Given that all modelled information is present in RDF, the respective graph can be visualized. However, as especially industry models can contain thousands of entities, mechanisms to enhance comprehensibility are mandatory. Consequently, some sort of information filtering, aggregation and/or abstraction is required for our visualization approach. For various problems from the information visualization domain, the so-called *Overview and Detail* [4,19] paradigm is used. Following that, a top-level view is created (see Fig. 2(a)), consisting of a single vertex for each model instance (M#1 to M#6). If there is at least one link between an entity from one model instance and another entity from a distinct model instance, an edge in that top-level graph exists between the corresponding vertices. For the detailed view (see Fig. 2(b)), one or more model instances and, with that, their graph representations are

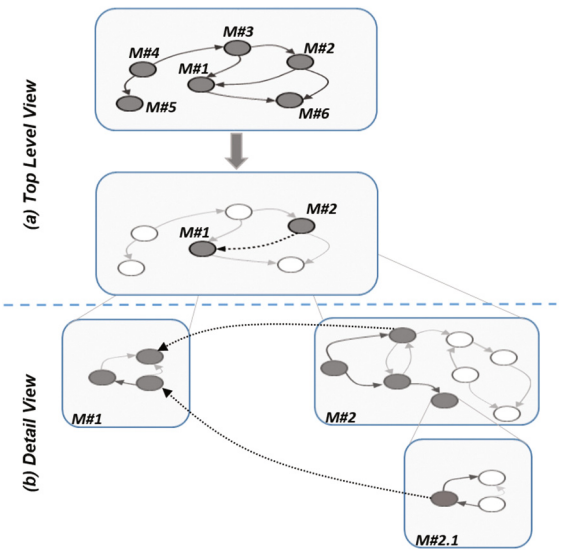


Fig. 2. Concept of an overview and detail visualization approach for an interactive inconsistency management framework

chosen from the top-level view and visualized in the detailed view. The vertices of each model instance are layouted inside a respective cluster, which get layouted on cluster level, similarly to the top-level graph. However, edges between clusters have vertices as endpoints inside those clusters. The clustering mechanism emerged as the best way of keeping the information of each model instance together, separated from other model instances. The graph of the detail view is layouted in two phases to suit the two level hierarchy, embodied by the clusters and their contained graphs. First, the coordinates of the clusters are computed. Second, for each cluster, the respective nodes are layouted. By that, a clustered graph structure is visualized to the stakeholder. Depending on what kind of level of detail is chosen, different hierarchy levels can be used to display the necessary information to the stakeholder.

Hierarchy Levels. By means of the clustered graph structure, different levels of detail can be displayed to the stakeholder. However, it is essential that stakeholders are supported in choosing for different levels of detail depending on their specific tasks. Therefore, an approach for finding and filtering according to different hierarchy levels is required. These hierarchy levels describe the different levels of aggregation and composition of the entities in the models.

In order to find these hierarchy levels, an algorithm was developed (see Fig. 3). As first step, the metamodel is analysed for aggregations and compositions for which hierarchical statements can be made. Solely from these and their related endpoints, a graph is created. For further steps, a hierarchy level has to be chosen, either by the user based on his specific task or by some heuristic, e.g. maximum path length of 2. Then, like for the metamodel, a graph is built for each model instance of the respective metamodel. All visited vertices up to there are contained in hierarchy levels up to the chosen depths. From this result in

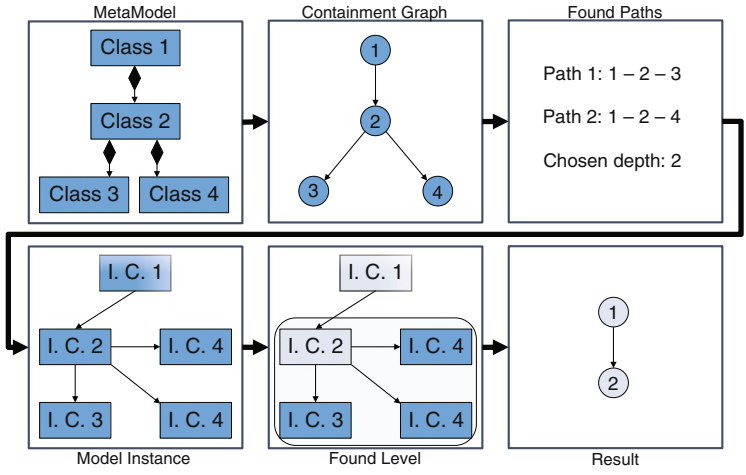


Fig. 3. Overview on the algorithmic steps for determining hierarchy levels

addition to the paths, a rule set is generated, which itself forms the visualized graph from the knowledge in the triple store.

Filtering. Independent from the used hierarchy level, filtering may be performed on the visualized information to reduce the complexity of the visualization. Based on the specific task of the user, only special, desired entities, relations and links may be chosen to be visualized. To achieve this, the configuration rules, which infer the necessary information for the visualization, are adapted by the user (see Fig. 4).

For the task of inconsistency management, the top level view represents the whole map of models, where each vertex represents a single model and each edge represents the links, respectively the information flows between the models. If there is an inconsistency between two models (M#1 and M#2), the edge between the representative vertices is marked. In Fig. 2(a) this is visualized by a dotted edge. The user is now able to select the vertices/models he wants to see on a detailed level. In this level, the graph representation of the chosen model instance and the inconsistencies between the single elements of the models are shown (see Fig. 2(b)).

Vertices and edges can have different colours to give additional information to the user (e.g. red for errors, orange for warnings or yellow for information). To reduce complexity the user is able to hide all vertices and edges, which are not involved in the inconsistency management task.

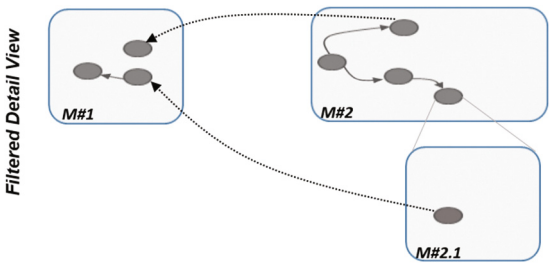


Fig. 4. Reducing complexity by applying filter rules in the detailed view of the visualization

In Sect. 5 the approach is evaluated by means of a case study from the aPS domain and the interconnection between three engineering models.

5 Implementation and Evaluation

This section is devoted to evaluating the presented inconsistency management framework with its according visualization approach. In order to validate the applicability of the framework, a software prototype is provided, which is introduced in Sect. 5.1. Subsequently, an inconsistency management case study is

provided as an excerpt from [6] (see Sect. 5.2). Accordingly, the approach to diagnose and handle inconsistencies by means of RDF and SPARQL is discussed at the hand of this case study (see Sect. 5.3) and it is shown how the visualization approach supports stakeholders throughout this process (see Sect. 5.4).

It has to be noted that, within this publication, we do not aim at a full usability evaluation of our software prototype, but rather at a principle evaluation that shows the feasibility of our presented approach. A full evaluation together with experts from the aPS domain is subject to future work.

5.1 Software Prototype

As a basis to evaluate the proposed framework, a software prototype was implemented. Therein, the inconsistency management framework is designed as a Java-based Eclipse plugin. The Eclipse Modeling Framework (EMF) [5] is therein used as the main component, as it allows to specify metamodels and create model instances as well as to automatically generate texts from the models. The Apache Jena Framework, especially the included RDF triple store Fuseki [1], was used for RDF handling. The transformation between the EMF and RDF was implemented by means of the EMF Triple API [11]. Moreover, the visualization framework is implemented in C♯ as a Windows Presentation Foundation (WPF) application. The dotNetRDF API [22] serves as the RDF backend and access to the triple store on the Fuseki server. For the visualization and the graph drawing purposes, the WPF-based GraphX API [20] is used.

5.2 Inconsistency Management Case Study

A second basis for the purpose of evaluating the feasibility of our approach is an appropriate inconsistency management case study. We use an excerpt of a case study from the aPS domain, which was initially presented in [6].

Within our case study (see Fig. 5), three distinct model types are used: A *planning model*, which is used to evaluate different alternatives to realize the required engineering solution, a *SysML model*, which is intended to describe the logical system architecture, as well as a *MATLAB/Simulink model* to predict, whether the respective system architecture is capable of fulfilling the demanded properties. Although these three model types only represent a small excerpt of typical models in the aPS domain, we argue that they are representative for a multitude of potential engineering models.

As visualized in Fig. 5, the involved engineering models are overlapping, i.e., there exist links in between these different models. In particular, three link groups are specified within our case study, one for every combination of two model instances from the three models in the case study: Between the planning and the SysML models, so-called *refines* links are introduced to denote that modules in the planning model are refined by blocks in the SysML model. Accordingly, *equivalentTo* links are established to denote, e.g., that requirements in both planning and SysML models are equivalent to each other. In order to denote that the output of the MATLAB/Simulink is used to verify, whether certain

properties in the planning model are fulfilled, *satisfies* links are introduced. We use these different link types in order to explicitly capture the dependencies between the different engineering models.

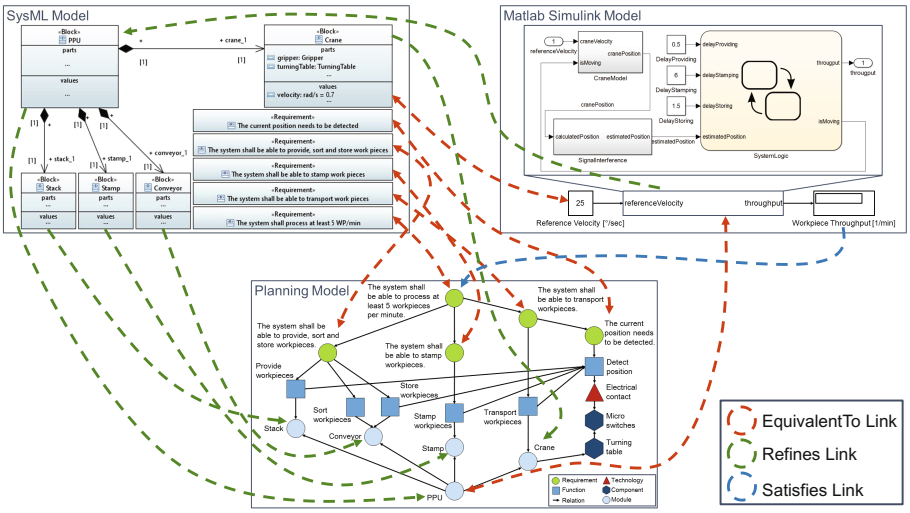


Fig. 5. Exemplary inconsistency management case study, excerpt from [6]

5.3 Evaluation of the Inconsistency Management Framework

In order to illustrate the feasibility of the presented concept for the purpose of diagnosing and handling inconsistencies, disparate types of inconsistencies were defined by means of the SPARQL query and update languages. Due to spatial restrictions, only an excerpt of these different inconsistency types can be illustrated in the following. In particular, we focus on a naming inconsistency in the following, which demands that, for each entity within our engineering models, an appropriate name in *UpperCamelCase* style is specified.

As can be seen from Fig. 6, an inconsistency diagnosis rule is specified by means of the SPARQL Query Language to check, whether all entities are named in *UpperCamelCase* style (Fig. 6a). Therein, the rule body includes three essential parts: First, the pattern identifies all named entities (`?x base:name ?xName`). Second, the entity's name `xName` is compared with a pre-defined regular expression that identifies, whether the name meets the *UpperCamelCase* style or not – the result of the comparison is bound to the variable `isInconsistent`, which denotes, whether an inconsistency occurs (`true`) or not (`false`). Third and finally, the original property is retrieved, which shall later be replaced by a custom name. An according inconsistency handling rule is specified by means of the SPARQL Update Language (Fig. 6b). Within the rule, placeholders are used to replace the name `xName` with the new name `customName`[1].

[1] For simplicity reasons, the new name is not checked for (in-)consistency.

Properties

Property	*Value*
Name	Naming convention for all entities
Severity	Warning
Message Pattern	Entity x is not correctly named (name was $xName$, but should be named in UpperCamelCase style).

Rule Body

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix base: <http://example.org/base#>
SELECT * WHERE {
  ?x base:name ?xName .
  BIND (!regex(?xName, "^([A-Z]{1}[a-z]+)+$.")
    AS ?isInconsistent) .
  ?x ?oldname ?xName .
  ?oldname rdfs:label "origin" .
} ORDER BY DESC(?isInconsistent)
```

(a) Inconsistency diagnosis rule

Properties

Property	*Value*
Name	Replace name by custom name
Type	Solution
Message Pattern	Replace name of entity x (name was $xName$) by custom name.

Rule Body

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix base: <http://example.org/base#>
DELETE {
  $x$ $oldName$ $xName$ .
}
INSERT {
  $x$ $oldName$ $customName$ .
}
WHERE { }
```

(b) Inconsistency handling rule

Fig. 6. Inconsistency diagnosis and handling rules for resolving naming inconsistency

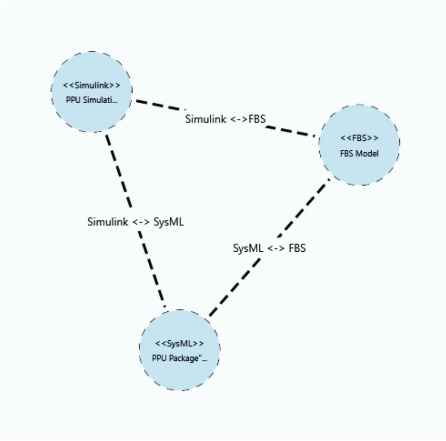
Additionally to the presented inconsistency diagnosis and handling rules, further rules can be introduced, e.g., to ensure that only the allowed entity types are linked to each other. Especially due to the expressiveness of RDF and SPARQL, we expect that a multitude of inconsistency types can be specified, diagnosed and handled through our approach.

5.4 Evaluation of the Visualization Approach

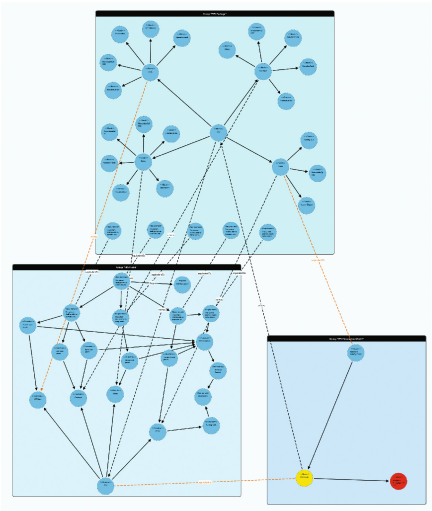
Although we argue that a multitude of different types of inconsistencies can be diagnosed and handled by our inconsistency management framework, there are certainly many cases in which a visualization is essential to supplement the process of inconsistency management. In these cases, stakeholders need to be supported for a better understanding of the relationships and dependencies. Consequently, using a pre-defined configuration of the visualization, both the hierarchy and the filtering concept are applied to the engineering models in the case study.

In particular, three distinct views are generated for the engineering models of the case study (see Fig. 7): one for a top-level overview on the model links (see Fig. 7a), one for a detail-level overview on the model links (see Fig. 7b) and one filtered view that only contains the linked model entities (see Fig. 7c). By means of switching between the different views, we argue that stakeholders get appropriate support in identifying the location and cause for specific inconsistency types.

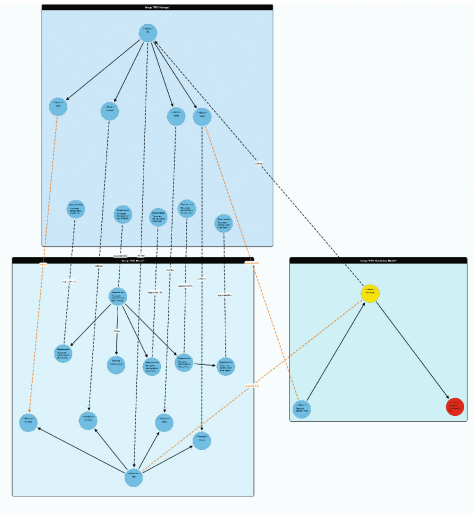
Within the top-level visualization (see Fig. 7a), all three engineering model instances are represented through a single node. Edges between the distinct model nodes represent the aggregated links between the models. Accordingly, the user can interact with the visualization by selecting one or more models or links and, hence, the detailed graphs show the respective clusters. These clusters are connected by inter-model links, according to the chosen hierarchy and filter.



(a) Top-level visualization



(b) Detail-level visualization



(c) Filtered detail-level visualization

Fig. 7. Visualization for the engineering models of the case study

Hierarchy Visualization. For each model, different hierarchy levels are chosen. For instance, for the MATLAB/Simulink model instance, it can be seen that all inner blocks of the simulated subsystem are not affecting the inconsistency management process. As a result, the rules are created in such a way that the whole simulated subsystem is contracted to a single node – only the input and output values of the simulation block are illustrated as their are interesting for inconsistency management. In contrast, the planning model instance is not abstracted and not contracted, because many entities on the most detailed abstraction level are directly affected by links to entities from other models.

For the third model, the SysML model, an abstraction level in between is chosen, as some of the entities within the model are interesting for the inconsistency management process. By means of color-coded inconsistencies (e.g., red for an error, orange for a warning and green for no error), stakeholders can be informed on the respective inconsistency status of their models.

Filtering. Additional to or instead of solely visualizing all elements of a certain hierarchy level, filtering may be applied. While the visualization result from above (cf. Fig. 7b) shows the structure of the model entities, the filtering provides the functionality to reduce the number of entities to a desired minimum. For instance, the entities of the same hierarchy level as above may be filtered for those, which could possibly be start or end point of a link (see Fig. 7c). Therein, only the requirements and modules for the planning model as well as the requirements and blocks of the same hierarchy level in SysML are visualized, while the MATLAB/Simulink model stays the same. Even though in Fig. 7c, many entities and their respective nodes are filtered out, the relationships among the remaining entities still gives an impression about the overall structure, e.g. the hierarchical requirement and module structure in the planning model instance as well as the structure of the physical parts in the SysML model instance are still contained. Despite the smaller amount of objects, which need to be grasped in the visualization, the amount of individually visualized inconsistencies is the same as without filtering.

6 Conclusion

This article introduces an interactive inconsistency management approach for engineering in the automated production systems (aPS) domain, which makes use of Semantic Web Technologies and combines the (semi-automatic) diagnosis and handling of inconsistencies with an interactive visualization approach. This interactive visualization approach allows stakeholders to pre-define views, which visualize the different engineering models on disparate, pre-defined levels of detail. The feasibility of the presented approach was illustrated at the hand of a lab-scale case study.

By means of the presented approach, we argue that a multitude of benefits can be achieved: For one, the visualization approach extends the already existing inconsistency management framework by comprehensive means to specify, diagnose and handle inconsistencies. Especially as views can be configured by means of pre-defined rules, any application of the framework is, basically, possible. Hence, stakeholders in the aPS domain are supported in identifying possibly erroneous parts in their engineering solution. The visualization of inter-model dependencies also supports the expansion of the stakeholders' individual mental models and leads to a common understanding of interfaces between and overlaps of the different models. This may also prevent inconsistencies in the future.

However, still a lot of research effort needs to be done. One essential step to be performed in future research works is an appropriate usability evaluation

together with experts from the aPS domain. Using the experts' feedback, appropriate visualization views (and, consequently, rules) can be identified and defined. Further on, in order to apply the presented concept, knowledge and expertise in the Semantic Web Technologies domain is required as, e.g., inconsistency diagnosis and handling rules as well as visualization rules must be configured. We argue that, through appropriate abstraction mechanisms (e.g., modelling techniques), such rules can be automatically generated from pre-defined models. This will especially increase transparency of the approach and, hence, make the concept more attractive for industrial engineering applications.

References

1. Apache Jena: Fuseki: serving RDF data over HTTP (2016). https://jena.apache.org/documentation/serving_data/
2. Basole, R.C., Qamar, A., Park, H., Paredis, C.J.J., McGinnis, L.F.: Visual analytics for early-phase complex engineered system design support. IEEE Comput. Graph. Appl. **35**(2), 41–51 (2015)
3. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless model-based development: From isolated tools to integrated model engineering environments. Proc. IEEE **98**(4), 526–545 (2010)
4. Cockburn, A., Karlson, A., Bederson, B.B.: A review of overview+detail, zooming, and focus+context interfaces. ACM Comput. Surv. **41**(1), 2 (2009)
5. Eclipse Foundation: Eclipse Modeling Framework (2016). http://www.eclipse.org/modeling/emf/
6. Feldmann, S., Herzig, S.J.I., Kernschmidt, K., Wolfenstetter, T., Kammerl, D., Qamar, A., Lindemann, U., Krcmar, H., Paredis, C.J.J., Vogel-Heuser, B.: A comparison of inconsistency management approaches using a mechatronic manufacturing system design case study. In: IEEE International Conference on Automation Science and Engineering, Gothenburg, Sweden, pp. 158–165 (2015)
7. Feldmann, S., Herzig, S.J.I., Kernschmidt, K., Wolfenstetter, T., Kammerl, D., Qamar, A., Lindemann, U., Krcmar, H., Paredis, C.J.J., Vogel-Heuser, B.: Towards effective management of inconsistencies in model-based engineering of automated production systems. In: 15th IFAC Symposium on Information Control Problems in Manufacturing, Ottawa, Canada, pp. 916–923 (2015)
8. Friedl, M., Weingartner, L., Hehenberger, P., Scheidl, R.: Model dependency maps for transparent concurrent engineering processes. In: Proceedings of the 14th Mechatronics Forum International Conference, Mechatronics 2014, pp. 614–621 (2014)
9. Gausemeier, J., Giese, H., Schäfer, W., Axenath, B., Frank, U., Henkler, S., Pook, S., Tichy, M.: Towards the design of self-optimizing mechatronic systems: Consistency between domain-spanning and domain-specific models. In: ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, pp. 1141–1148 (2007)
10. Herzig, S., Qamar, A., Reichwein, A., Paredis, C.: A conceptual framework for consistency management in model-based systems engineering. In: ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, Washington, DC, USA, pp. 1329–1339 (2011)
11. Hillairet, G.: EMF Triple (2016). https://github.com/ghillairet/emftriple

12. International Council on Systems Engineering: Systems engineering vision 2020. Technical Report INCOSE-TP-2004-004-02 (2007). http://oldsite.incose.org/ProductsPubs/pdf/SEVision2020_20071003_v2_%03.pdf
13. Jossic, A., Fabro, M.D.D., Lerat, J., Bézivin, J., Jouault, F.: Model integration with model weaving: A case study in system architecture. In: 1st IEEE International Conference on Systems Engineering and Modeling, pp. 79–84 (2007)
14. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: Merging models with the epsilon merging language (EML). In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MODELS 2006. LNCS, vol. 4199, pp. 215–229. Springer, Heidelberg (2006). doi:10.1007/11880240_16
15. Kovalenko, O., Serral, E., Sabou, M., Ekaputra, F.J., Winkler, D., Biffl, S.: Automating cross-disciplinary defect detection in multi-disciplinary engineering environments. In: Janowicz, K., Schlobach, S., Lambrix, P., Hyvönen, E. (eds.) EKAW 2014. LNCS (LNAI), vol. 8876, pp. 238–249. Springer, Cham (2014). doi:10.1007/978-3-319-13704-9_19
16. Leung, Y.K., Apperley, M.D.: A review and taxonomy of distortion-oriented presentation techniques. ACM Trans. Comput. Hum. Interact. **1**(2), 126–160 (1994)
17. Nuseibeh, B., Easterbrook, S., Russo, A.: Leveraging inconsistency in software development. IEEE Comput. **33**(4), 24–29 (2000)
18. Schmidt, D.C.: Guest editor's introduction: Model-driven engineering. IEEE Comput. **39**(2), 25–31 (2006)
19. Shneiderman, B.: The eyes have it: A task by data type taxonomy for information visualizations. In: IEEE Symposium on Visual Languages (1996)
20. Smirnov, A.: GraphX (2016). https://github.com/panthernet/GraphX
21. Spanoudakis, G., Zisman, A.: Inconsistency management in software engineering: Survey and open research issues. In: Handbook of Software Engineering & Knowledge Engineering: Fundamentals, vol. 1, pp. 329–380. World Scientific Publishing Co Pte. Ltd., Singapore (2001)
22. Vesse, R., Zettlemoyer, R., Ahmed, K., Moore, G., Pluskiewicz, T.: dotNetRDF (2016). http://dotnetrdf.org/
23. World Wide Web Consortium: SPARQL Protocol and RDF Query Language (SPARQL) 1.1 Query Language (2013). https://www.w3.org/TR/sparql11-query/
24. World Wide Web Consortium: SPARQL Protocol and RDF Query Language (SPARQL) 1.1 Update (2013). https://www.w3.org/TR/sparql11-update/
25. World Wide Web Consortium: Resource Description Framework (RDF) 1.1 Concepts and Abstract Syntax (2014). https://www.w3.org/TR/rdf11-concepts/