INTERNATIONAL DESIGN CONFERENCE - DESIGN 2018
https://doi.org/10.21278/idc.2018.0287

TOWARDS SYSTEMATIC INCONSISTENCY IDENTIFICATION FOR PRODUCT SERVICE SYSTEMS

M. R. Basirati, M. Zou, H. Bauer, N. Kattner, G. Reinhart, U. Lindemann, M. Böhm, H. Krcmar and B. Vogel-Heuser

Abstract

Value shift towards services led to emergence of product-service systems (PSS) as intertwined products and services. PSS development requires collaborating teams with higher domain diversity to tackle service side as well as product side. Since every domain employs a particular set of tools and models, it is challenging to manage consistency among them. However, the PSS literature lacks approaches for managing inconsistency among various type of models. This study proposes a framework that supports establishing a systematic solution for inconsistency identification during PSS development.

Keywords: product-service systems (PSS), model based engineering, modelling, systematic approach

1. Introduction

Competitive global economy necessitates manufacturers to increase sustainability and having a longer relationship with customers by providing new services in addition to the conventional products (Tukker, 2004). Besides, environmental considerations are putting restrictions on the products and the way they are developed leading manufacturers to rely more on value share of services (Mont, 2002; Meier et al., 2010). This shift towards services introduced concept of product-service-system (PSS) as a system consists of combined product(s) and service(s) (Baines et al., 2007).

To develop a PSS, diverse teams from managerial to technical level need to continually collaborate in order to address high level vision of the system as well as detailed functionalities and interactions between services and physical elements (Vasantha et al., 2012). Thus, not only the number of stakeholders raises in PSS development, but also the stakeholders are more likely to be heterogeneous from dissimilar domains. Accordingly, PSS development deals with multidisciplinary approaches, artefacts and models expanded over different levels of organization (Vasantha et al., 2012).

Thus, it is vital to manage consistency among the models from heterogeneous domains during a PSS development. While numerous studies addressed PSS modelling and proposed various modelling techniques such as Data Flow Diagrams (Durugbo et al., 2011) and Systems Modelling Language (Balmelli, 2007), there is lack of research on how to keep the complex network of models during PSS development consistent. Nevertheless, a consistent design process is crucial in both ensuring the system functionality and increasing interdisciplinary co-design. For example, in a digitalized machinery company, offered data services should meet requirements, allowed by the adopted sensor types and not violate country-specific privacy laws. Though various methods have been proposed to partially automate the inconsistency identification processes, a general and systematic underlying framework is missing (Zou and Vogel-Heuser, 2017).

Therefore, in this study, we investigate on inconsistencies in PSS development and develop a framework, which can be applied to identify the inconsistencies systematically. The reminder of this paper is structured as follows: Section 2 provides related work on inconsistency management as well as PSS modelling literature. Afterwards, we introduce different types of inconsistencies in Section 3. Subsequently the developed framework to identify inconsistencies is presented in Section 4. The framework is applied and evaluated on a case of e-bike-sharing system, presented in Section 5. We discuss the framework applicability and its limits in Section 6. Finally, Section 7 summarize the study and shed light on future works regarding managing inconsistencies during PSS development.

2. Related work

Inconsistency Management by definition is the process of handling dependencies in a way that the goals of stakeholders are concerned (Spanoudakis and Zisman, 2001). We reviewed the literature on inconsistency management from software engineering domain as well as mechanical engineering studies. Besides, we present few number of studies, which addressed inconsistency-related topics in PSS development.

Nuseibeh et al. (2000) and Finkelstein et al. (1996) have developed processes for the management of inconsistencies in the domain of software engineering. Both approaches share the common feature that inconsistencies must be determined with respect to defined consistency rules. The consistency rules, which are defined by experts, specify two essential necessities. First, they check whether a model is applicable as a valid model of a particular modelling language or not. In another words, this step analyses correctness of the models from syntactical point of view. Second, they ensure that the software models in a development process comply with valid standards of the software development project. In addition, both approaches involve activities to detect, diagnose and manage inconsistencies. There are other supporting activities, such as the specification and application of an inconsistency policy. These additional activities are not included in both approaches, which makes them different from each other. Besides, Finkelstein focuses on the term "inference", expressed as conflicts of interdependencies.

Spanoudakis and Zisman (2001) combine the both aforementioned approaches (Finkelstein et al., 1996; Nuseibeh et al., 2000) to create a process concentrated on special methods and techniques for inconsistency management. The approach is represented in six steps, which starts with the detection of overlaps, followed by the detection and the diagnosis of inconsistencies, handling of inconsistencies and finally, the specification and application of an inconsistency management policy.

Gausemeier et al. (2009) employ triple graph grammars and introduce a rule-based approach aimed at preserving consistency between domain-spanning and domain-specific models in the development of complex mechatronic systems. For this purpose, they developed an approach based on triple graph grammars to capture correspondences between models. However, this approach is limited to a special technique and its applicability on cross-domain models is weak.

Hehenberger et al. (2010) define consistency rules and propose a conceptual approach based on the automatic checking of the consistency rules. First, they use domain-spanning ontologies for identifying overlaps. Subsequently, they apply the consistency rules, which are simple conditions of a model that are either true or false, depending on whether the model satisfies them or not.

Qamar et al. (2012) address dependency modelling and present an approach to avoid inconsistencies in models, based on the explicit modelling of dependencies. They argue that impact of changing a model on the other models can be predicted by modelling dependencies, which enables easier inconsistency management.

Herzig and Paredis (2014) apply pattern matching, which is based on the transformation of models into graphs and a subsequent pattern-based identification of inconsistencies.

Feldmann et al. (2015) apply semantic web technology and present an approach where properties of semantic webs, which enable processing not only the linking of data but also their meaning, can be used to identify inconsistencies in models. A knowledge-base approach represents all Models in a Resource Description Framework (RDF) and reveals inconsistencies with SPARQL inconsistency query.

Dávid et al. (2016) introduce the inconsistency tolerance. In the context of this approach, semantic inconsistencies are quantified. Afterwards, it is decided how far they can be tolerated.

In PSS field, Shimomura and Hara (2010) introduce a conflict resolution method for PSS development. The method consists of two major strategies. First, an inconsistency is detected by lexical analysis of names and descriptions of functions and variables. If two objects' names include related words, the method investigates if there is an inconsistency or not. The second strategy applies a set-based approach to detect value overlaps that lead to inconsistencies. To this end, the related objects are identified manually. Nevertheless, the study does not address how heterogeneity of PSS models are handled.
Song and Sakao (2016) investigate on conflicts among services in product-service offerings, which also employs a linguistic techniques. However, the study does not cover the cross-models inconsistency problem.

3. Inconsistency types

In this study, we define inconsistency as any logical contradiction between two facts or two presentations of facts expressed in formal models as well as in informal artefacts such as requirements written in natural language. This definition excludes any high level conflict between goals, priorities and plans of people or organizational units. Therefore, we focus specifically on how heterogeneous models and artefacts can lead to inconsistencies during PSS development.
To this end, we developed a classification of inconsistencies that specifies what types of inconsistencies exist between models by extending the work of Feldman et al. (2015). The inconsistencies are classified based on two high level dimensions. First dimension determines how the two inconsistent elements are related to each other. We defined four high level relationship types, namely existence, equivalency, refinement and satisfaction. Furthermore, we determine if an inconsistency is occurred on syntactical level or semantical level. The syntactical level consists of notational and conventional types. The semantical level is divided to system/project specific as well as domain-specific types. The inconsistency types and examples are presented in Table 1. Besides, concrete examples of inconsistency types are demonstrated based on a case in the discussion, Section 6.

Table 1. Inconsistency types

	Existence	Equivalence	Refinement	Satisfaction
Notational	Missing Name; Typo	Different Names for Same Element	-	-
Conventional	Missing a Standard's Element	Incompliancy with a Standard's Element	-	-
System/Project-specific	Missing Element in a Model based on System's Logic	Different Value for Same Element in Different Models	Different Refined Value for an Element	Unsatisfied Logics of System
Domain-specific	Missing Element in a Model based on General Domain Rules	Different Value for an Element based on Domain Rules	Wrong Refinement based on Domain Rules	Unsatisfied Domain Rules

If an element is missing, which is supposed to exist in the model, it is considered as an existence inconsistency. For example, missing a name, a structural section in document or un-modelled part of a system can be assigned to this group. Similarly, if two elements are supposed to be equal, but they are not, there is an equivalence inconsistency. However, more complicated types of inconsistencies can be identified. If an element is refined version of another, but the refinement is done incorrectly, it can be recognized as a refinement inconsistency. For example, a Simulink model can be refined version of MATLAB codes. Finally, an inconsistency can arise, when an element is supposed to satisfy another element's conditions, domain-specific rules or project-specific rules.
Based on the other dimension, inconsistency can violate syntactical or semantical rationale. Notational instances are a typo or inconsistent use of terms for a same element. Besides, there might exist a set of conventional rules regarding the syntactic of a model or artefact. Various conventional rules can be set up including standards, guidelines, and quality attributes of specifications and so on. Accordingly, inconsistencies can violate conventional rules such as syntax of a modelling language or incompliancy

with a standard. System/project-specific inconsistencies are the ones that contradict logic and function of the system or project, which are under development. For example the relationship between components in the system, the specific defined values and so on. On the other hand, domain-specific inconsistencies are discipline-specific and root in general knowledge related to a domain, an example can be violating a physical law.

4. Framework for inconsistency identification in PSS

In this section, we describe a holistic framework that tackles the overall procedure for inconsistency identification in PSS development. Besides, there are many parameters that determines how an inconsistency identification procedure will be realised. For example it should be determined to what extent the process is performed automatically. Therefore, the framework includes such parameters and we explain them subsequently in the following.

4.1. Process of inconsistency identification in PSS development

To identify inconsistencies during PSS development, the framework considers five high level phases that need to be carried out. Since many departments and teams are involved in PSS development, it is necessary to first trace the information flow among them. Based on the information flow, we would be able to identify related models and artefacts as well as how they are related to each other. Subsequently, the dependencies among related models are detected in the next phase. At this stage, we have the related models and the dependencies among them. However, as the models are from heterogeneous domains and teams, there are expressed in different languages and forms. Therefore, a mutual form is needed to consolidate the information and enable comparing values. This is done in the fourth phase and finally, in the last phase, dependent information in the same form can be analysed with regard to some rationale and the inconsistencies will be exposed.

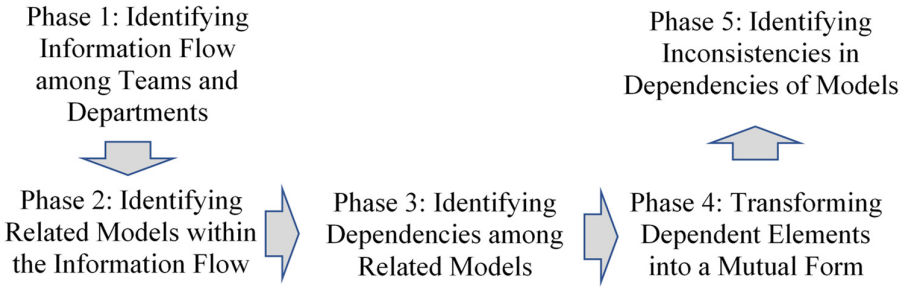

Figure 1. Inconsistency identification process

The described major phases for PSS inconsistency identification can be implemented by various methods and tools. For example, in the first and second phase, communication data of stakeholders can be investigated using information retrieval techniques or manually organizational structure of the project can be analysed. Reviewing such methods and tools for every phase is out of scope of this study and we address it in future work. However, the procedure for identifying inconsistencies among related models (phases 3 to 5) can be detailed more without lacking its general applicability.

To this end, we describe what steps are needed to be taken in every phase. These steps are elaborated based on three abstraction levels, namely meta-model, model and reality. As depicted in Figure 2, after identifying a dependency on model level, the exact elements which create the dependency have to be specified on model level. Afterwards, in order to systematically transform information from dissimilar models into a mutual form, it is required to trace dependency on the meta-model level. Specifying details on a meta-model level enables practitioners to generalize dependencies as well as inconsistencies and repeat the process systematically. In particular, such abstraction levels are essential for automating the procedure using defined algorithms for machine. After the relationship between two elements are identified on the meta-model level, the dependent meta-model level elements should be mapped to a corresponding element on the meta-model level of a mutual form. Consequently, we would be able to

capture the information in the same format. In the final phase, we need to reason inconsistencies based on some rationale. However, logics are expressed in a general manner such as domain-specific laws or system logics. Therefore, the rationale can be formulated on the meta-model level and ground the final identification as the final step.

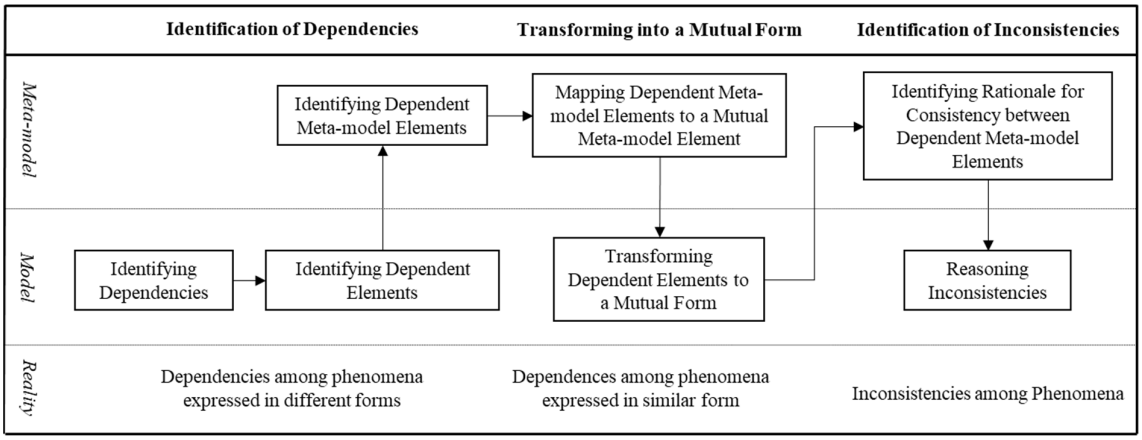

Figure 2. Inconsistency identification through abstraction levels

4.2. Parameters of inconsistency identification in PSS development

In addition to the process itself, there are parameters that determine how the process is realised. These parameters are related to the model and targeted inconsistency as well as the details of identification method. We distinguish between problem-side and solution-side parameters and explain them in detail. The problems-side parameters characterize the inconsistency and the situation, in which the consistency may emerge, while the solution-side parameters represent the techniques and technologies that are used to identify the inconsistency.

The framework addresses degree of formality, degree of criticalness and inconsistency type as problem-side parameters. Degree of formality determines to what extent the involved models in an inconsistency are formal. For example, mathematical equations have high formality, while artefacts written in natural language, such as user requirements can be considered as the most informal models. The degree of formality influences on how dependent elements can be identified and transformed into a mutual form. For example, graph-based formal models such as UML are easier to be mapped into a meta-model and accordingly transformed into a mutual form. However, informal models require different mechanisms. In particular, automation of the process in case of formal models is more straightforward than informal ones. Degree of Criticalness determines to what extent the targeted inconsistency is hazardous to the system and the project. A high critical inconsistency should not be missed by the implemented identification methods. Therefore, proper approaches and tools need to be employed in order to reach high recall for such inconsistences. Furthermore, dissimilar approaches can be employed based on the inconsistency type as it specifies the relationship between two elements and the rationale behind an inconsistency.

Similarly, there are three solution-side parameters that address general features of a solution for inconsistency identification in PSS development. First, considering the parameters of problem-side, degree of automation, based on which identification process is developed should be decided. How much automation is required directly impacts on the technologies and tools that are applied. In addition, there are in general two types of automatically re-employing acquired knowledge, formulating the knowledge into a set of rules or applying machine learning algorithms to seizure knowledge into a re-applicable model. These two types of automatic solutions necessitate different mechanisms and infrastructures. Thus, applicability of both methods should be analysed based on the PSS development settings and requirements. Finally, appropriateness of different technologies for the mutual form, to which the information from heterogeneous models are transformed, should be evaluated. Although this parameter determines how the overall process is realised, the other parameters such as degree of formality and

degree of automation influence on deciding it. For example, the technology of mutual form needs to be suited to the involved models as well as the tools and technologies used in automatizing the process.

Table 2. Influencing parameters for inconsistency identification

Problem-side	Solution-side
Degree of Formality	Degree of Automation
Degree of Criticalness	Rule-based vs. Machine Learning
Type of Inconsistency	Mutual Form Technology

5. Application: An inconsistency case

We describe application of the framework based on an e-bike-sharing system as an example of PSS. The e-bike-sharing system, called PSSycle, is developed by a group of students in context of an interdisciplinary research project that addresses cyclic innovation in PSS development. The large structure of the research project allows us to apply entire process of the framework and consider the complete relevant aspects. In the following, we describe the case and an instance of inconsistency, afterwards we apply the framework on identifying such an inconsistency.

5.1. PSSycle: An E-Bike-Sharing PSS

The research project consists of 7 research departments including Information Systems, Product Development, Mechatronic Design, Manufacturing Planning, Organizational Psychology, Automatic Control and Technology in Society. PSSycle development included all departments except Organizational Psychology, as they were focused on unrelated issues to this case. We briefly introduce the responsibility of every department in the development process. Department for Technology in Society investigates the needs for a PSS in society. Information System department tackled software parts of PSSycle as an e-bike-sharing system, including board computer and mobile app. Departments for Mechatronic Design and Product Development designed and developed the frame, the lock and the final assembly. The department for Manufacturing Planning modelled how the PSSycle will be manufactured in a real industrial settings. Besides, Automatic Control department was responsible for simulation of PSSylce.

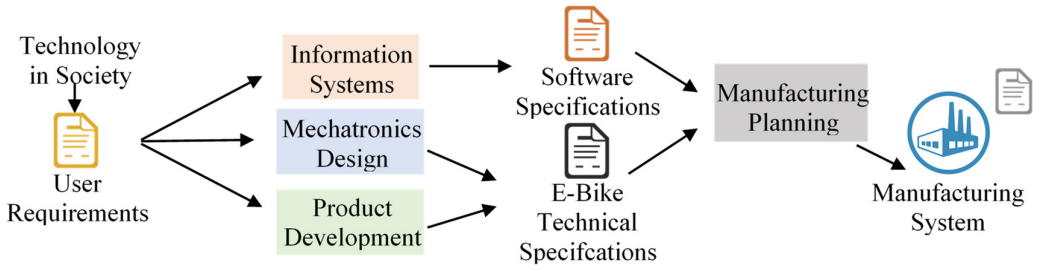

Figure 3. Information flow for PSSycle development

In an exemplary scenario, Technology in Society department finds out that there are two types of PSSycle users. A group of users ride e-bikes for very short trips, while another group, ride e-bikes for longer trips. However, the current e-bikes cannot ride such long trips completely, supported by the battery power. Therefore, from a business perspective it is decided that PSSycle should establish new services for customizing e-bikes for different purposes. As the first step, two types of e-bikes are offered based on their range: short-range (regular ones) and long-range. To this end, Departments for Mechatronics Design and Product Development decide to add a new e-bike with the same specifications as regular ones, but with a higher capacity battery, which has a bigger size than the previously used battery type for short-range e-bikes. For manufacturing such an e-bike, the gripper used in manufacturing line must support the size of the batteries. We apply the framework on the described situation and elaborate how an inconsistency can be identified.

As first step of the framework, we identify the information flow among the teams and departments in case of the described situation. Depicted in Figure 3, user requirements are formulated by Technology

in Society department. Afterwards, the user requirements are analysed, designed and translated into specifications. The e-bike specifications are detailed by departments for Mechatronics Design and Product Development. Besides, department for Information Systems delivers software specifications. Subsequently, department for Manufacturing Planning models how the e-bikes should be manufactured in an industrial settings.

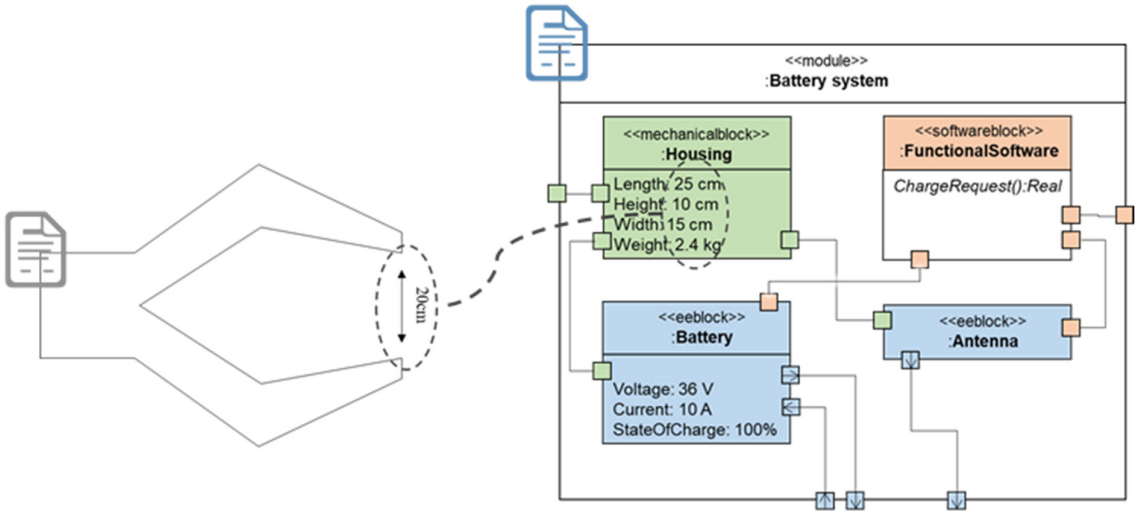

Figure 4. Example of model dependency for PSSycle

Based on the second step of the framework, we identify every department's models related to the battery manufacturing (we present several models as examples, however a complete model set is out of scope of this study). User requirements expressed in natural language can be considered as a model developed by Technology in Society department. UML models are from Information Systems Department. SysML4Mechatronics (Kernschmidt and Vogel-Heuser, 2013) and CAD models are respectively from departments for Mechatronics Design and Product Development. Finally, digital factory models are developed by department for Manufacturing Planning.

In the next step, dependencies among the models need to be identified. Based on the aforementioned scenario, we demonstrate a dependency between SysML4Mechatronics model of the e-bike battery and the gripper's model (depicted in Figure 4). Dependent elements are the gripper's supporting width and the battery's width and length, which it is planned to move.

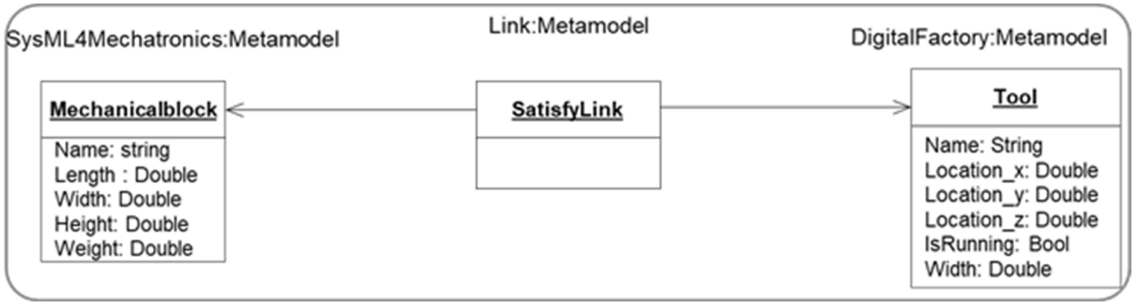

Figure 5. Meta-model level dependency

After identifying dependencies, the dependent elements should be transformed into a mutual form. This is an essential step for an automated approach. However, in a manual inspection approach, it depends on complexity of the dependent models to include or skip the phase because of high simplicity. In case of PSSycle, dimensions of the e-bike are specified in mechanicalblock of the SysML4Mechatronics model and the gripper's width is expressed in tool component of a digital factory model. Therefore, mechanicalblock and tool component are the dependent meta-model elements. We selected RDF for

mutual form that dependent elements are transformed to. In the following, we show how models are formulated in RDF.

```
<rdf:Description about="http://PSSycleModels/BatterySpec">
  <mechanicalblock:name>Battery</mechanicalblock:name>
  <mechanicalblock:length>25</mechanicalblock:length>
  <mechanicalblock:width>10</mechanicalblock:width>
  <mechanicalblock:height>15</mechanicalblock:height>
  <mechanicalblock:weight>2.4</mechanicalblock:weight>
</rdf:Description>
```

```
<rdf:Description about="http://PSSycleModels/GripperModel">
  <digitalfactorytool:name>Gripper</digitalfactorytool:name>
  <digitalfactorytool:location_x>1000</digitalfactorytool:location_x>
  <digitalfactorytool:location_y>1500</digitalfactorytool:location_y>
  <digitalfactorytool:location_z>1300</digitalfactorytool:location_z>
  <digitalfactorytool:width>20</digitalfactorytool:width>
</rdf:Description>
```

Figure 6. Transformed models into a mutual form (RDF)

In the final phase, consistency of dependent elements' values are investigated. To this end, we need to identify rationale that define whether a situation is in the state of consistency or inconsistency. The rationale capture general laws, logics and considerations, consequently, they can be assigned to meta-model level. In case of our example, the gripper must be able to pick up and move the battery from its either length or width dimension. Therefore, width of the gripper have to be equal or greater than the battery's length and width. This can be expressed on the meta-model level more formally as:

$$(\text{mechanicalblock:width} \leq \text{digitalfactorytool:width}) \text{AND} (\text{mechanicalblock:length} \leq \text{digitalfactorytool:width})$$

Based on such a rule on the meta-model level, an inconsistency can be identified between length of the battery and the gripper's width. The inconsistency on the model level is demonstrated in Figure 7.

Figure 7. PSSycle inconsistency example on model level

6. Discussion

In the previous section, we projected how overall process of the framework can support practitioners to systematically identify inconsistencies between heterogeneous models of PSS. Besides, we briefly showed an RDF example of transforming dissimilar models into a mutual form. In this section, based on the PSSycle case, we explain introduced parameters of the framework. Moreover, we discuss the limitations of the framework.

The battery was modelled in SysML4Mechatronics, which offers a high degree of formality. Hence, it allows us to adopt higher degree of automation. The inconsistency between the battery size and the gripper belongs to satisfaction and system/project-specific types (described in Section 3). In another words, the exposed inconsistency violated a satisfaction relation defined by the system development's logics. Regarding degree of criticalness, an inconsistency's criticalness depends on real settings of a project and the related practitioner's view. Therefore, it cannot be assessed objectively. However, future work can investigate on the factors that increase criticality of an inconsistency as well as measuring the criticality based on such factors.

The solution described in the exemplary case can be implemented highly automated. Because the models are mostly formal and it is elaborated in the previous section that how the dependent elements are related in the model as well as the meta-model level. Therefore, we consider high degree of automation for the presented example. Besides, we applied a rule-based method by defining a rule that determines whether the gripper is compatible with the battery size or not. Finally, as the mutual form technology we used RDF.

Nevertheless, in the PSSycle case scenario many other types of inconsistency could emerge. For instance, a conventional inconsistency could arise if width of the gripper was expressed in inches instead of centimetres. An example of refinement inconsistencies can emerge between RDF presentation of the battery and its SysML4Mechatronics model, as the RDF is refined version of the SysML4Methcatronics

model. Therefore, if the battery's width in RDF presentation is different to its value in SysML4Mechatronics, there is a system/project-specific refinement inconsistency. Besides, more complicated inconsistencies can happen between requirements collected from users and system specifications and functionalities. For example, if the battery's discharge rate is faster than the expectations of the users, this can be considered as an inconsistency.
Although, the inconsistency example of this study is rather simple, however in industrial settings with high number of interconnected elements, the need for a systematic approach to identify even trivial inconsistencies is crucial. This study establishes the fundamentals of such an approach that can be customised based on the needs and situations. For example, to increase dependability, formal methods can be employed to identify inconsistencies based on the framework. Hence, more in detail methods and analysis are required to address the introduced phases and parameters of inconsistency identification for PSS.
Furthermore, we cannot claim that the proposed framework includes all related aspects and parameters, however the high influential concepts, which are covered by the framework can be extended by future research through adding new parameters or customising the framework for a particular inconsistency problem. Moreover, the framework is developed from a PSS development perspective, nevertheless, future research can investigate and extend its applicability for general inconsistency identification purposes. Besides, the proposed framework only concentrates on the identification process and how the exposed inconsistencies should be handled is not tackled, which should be investigated in future research.

7. Conclusion

PSS development requires collaboration of diverse teams employing varied models and artefacts. Thus, there is a higher chance for occurrence of inconsistency among the models. Consequently, a systematic approach is required to support practitioners in order to identify the heterogeneous models' inconsistences during PSS development. To this end, first we introduced a classification of inconsistency types, based on which different methods and policies can be employed. Subsequently, we developed a holistic framework including the general procedure and influencing parameters of inconsistency identification. The framework can be customised based on different settings and needs. Based on an e-bike sharing system case, we applied the framework to identify an exemplary inconsistency. The case demonstrated a concrete problem of the profile fitness among different models, which is easy if it is designed by the same models and tools. However, it is complicated in the design phase of PSS, where diverse stakeholders work in parallel and distributed.
The proposed framework of this study addresses the basics of inconsistency identification. Thus, more studies required to investigate on every introduced aspect of inconsistency identification for PSS. To this end, we will apply the framework on a more complex PSS scenario to expose applicability of the framework more in detail. A future work can study what policies are more suited for different types of inconsistencies. Besides, most of the previous studies applied rule-based methods in order to detect inconsistencies among models, however, with extensive availability of data, the applicability of machine-learning techniques on models' inconsistency identification problem should be investigated. Hence, in future work, we plan to implement such a solution on higher number of models.

Acknowledgement

This work was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) collaborative research centre ‘Sonderforschungsbereich SFB 768 “Managing cycles in innovation processes – Integrated development of product-service-systems based on technical products”.

References

Baines, T.S., Lightfoot, H.W., Evans, S., Neely, A., Greenough, R. et al. (2007), “State-of-the-art in product-service systems.” *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture,* Vol. 221 No. 10, pp. 1543-1552. https://doi.org/10.1243/09544054JEM858

Balmelli, L. (2007), “An overview of the systems modeling language for products and systems development”, *Journal of Object Technology*, Vol. 6 No. 6, pp. 149-177. https://doi.org/10.5381/jot.2007.6.6.a5

Dávid, I., Syriani, E., Verbrugge, C., Buchs, D., Blouin, D. et al. (2016), "Towards inconsistency tolerance by quantification of semantic inconsistencies", *First International Workshop on Collaborative Modelling in MDE COMMitMDE 2016, Saint Malo, France, 2016*, CEUR-WS, pp. 35-44.

Durugbo, C., Tiwari, A. and Alcock, J.R. (2011), "A review of information flow diagrammatic models for product–service systems", *The International Journal of Advanced Manufacturing Technology,* Vol. 52 No. 9, pp. 1193-1208. https://doi.org/10.1007/s00170-010-2765-5

Feldmann, S., Herzig, S.J.I., Kernschmidt, K., Wolfenstetter, T., Kammerl, D. et al. (2015), "Towards effective management of inconsistencies in model-based engineering of automated production systems", *IFAC-PapersOnLine*, Vol. 48 No. 3, pp. 916-923. https://doi.org/10.1016/j.ifacol.2015.06.200

Finkelstein, A., Spanoudakis, G. and Till, D. (1996), "Managing interference", *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops, San Francisco, California, United States*, ACM, New York, NY, pp. 172-174. https://doi.org/10.1145/243327.243646

Gausemeier, J., Schäfer, W., Greenyer, J., Kahl, S., Pook, S. and Rieke, J. (2009), "Management of cross-domain model consistency during the development of advanced mechatronic systems", *Proceedings of the 17th International Conference on Engineering Design (ICED 09),* The Design Society, Glasgow.

Hehenberger, P., Egyed, A. and Zeman, K. (2010), "Consistency Checking of Mechatronic Design Models", *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference - 2010, Montreal, Quebec, Canada, August 15-18, 2010*, ASME, New York, NY, pp. 1141-1148. https://doi.org/10.1115/DETC2010-28615

Herzig, S.J.I. and Paredis, C.J.J. (2014), "A Conceptual Basis for Inconsistency Management in Model-based Systems Engineering", *Procedia CIRP*, Vol. 21, pp. 52-57. https://doi.org/10.1016/j.procir.2014.03.192

Kernschmidt, K. and Vogel-Heuser B. (2013), "An interdisciplinary SysML based modeling approach for analyzing change influences in production plants to support the engineering", *2013 IEEE International Conference on Automation Science and Engineering (CASE),* IEEE, pp. 1113-1118. https://doi.org/10.1109/CoASE.2013.6654030

Meier, H., Roy, R. and Seliger, G. (2010), "Industrial product-service systems—IPS 2." *CIRP Annals-Manufacturing Technology,* Vol. 59 No. 2, pp. 607-627. https://doi.org/10.1016/j.cirp.2010.05.004

Mont, O.K. (2002). "Clarifying the concept of product–service system." *Journal of Cleaner Production,* Vol. 10 No. 3, pp. 237-245. https://doi.org/10.1016/S0959-6526(01)00039-7

Nuseibeh, B., Easterbrook, S. and Russo, A. (2000), "Leveraging inconsistency in software development", *Computer*, Vol. 33 No. 4, pp. 24-29. https://doi.org/10.1109/2.839317

Qamar, A., Paredis, C.J.J., Wikander, J. and During, C. (2012), "Dependency Modeling and Model Management in Mechatronic Design", *Journal of Computing and Information Science in Engineering*, Vol. 12 No. 4, pp. 41009. https://doi.org/10.1115/1.4007986

Shimomura, Y. and Hara, T. (2010). "Method for supporting conflict resolution for efficient PSS development", *CIRP Annals-Manufacturing Technology,* Vol. 59 No. 1, pp. 191-194. https://doi.org/10.1016/j.cirp.2010.03.122

Song, W. and T. Sakao (2016), "Service conflict identification and resolution for design of product–service offerings", *Computers & Industrial Engineering,* Vol. 98, pp. 91-101. https://doi.org/10.1016/j.cie.2016.05.019

Spanoudakis, G. and Zisman, A. (2001), "Inconsistency management in software engineering: Survey and open research issues", In: Chang S.K. (Ed.), *Handbook of software engineering and knowledge engineering*, Vol. 1: Fundamentals, pp. 329-380. https://doi.org/10.1142/9789812389718_0015

Tukker, A. (2004), "Eight types of product–service system: eight ways to sustainability? Experiences from SusProNet", *Business Strategy and the Environment*, Vol. 13 No. 4, pp. 246-260. https://doi.org/10.1002/bse.414

Vasantha, G.V.A., Roy, R., Lelah, A. and Brissaud, D. (2012), "A review of product–service systems design methodologies", *Journal of Engineering Design*, Vol. 23 No. 9, pp. 635-659. https://doi.org/10.1080/09544828.2011.639712

Zou, M. and Vogel-Heuser, B. (2017), "Feature-based Systematic Approach Development for Inconsistency Resolution in Automated Production System Design", *13th Conference on Automation Science and Engineering (CASE), Xi'an, 2017*, pp. 687-694. https://doi.org/10.1109/COASE.2017.8256183

Mohammadreza Basirati, Msc.
Technical University of Munich, Information Systems
Boltzmannstraße 3, 85748 Garching, Germany
Email: basirati@in.tum.de