Collaboratively Enhanced Consistency Checking in a Cloud-based Engineering Environment

Michael Alexander Tröls
Johannes Kepler University
Linz, Austria
michael.troels@jku.at

Atif Mashkoor
Software Competence Center
Hagenberg GmbH & Johannes
Kepler University
Linz, Austria
atif.mashkoor@{scch\jku}.at

Alexander Egyed
Johannes Kepler University
Linz, Austria
alexander.egyed@jku.at

ABSTRACT

Software systems engineering involves many engineers, often from different engineering disciplines. Efficient collaboration among these engineers is a vital necessity. Tool support for such collaboration is often lacking, especially with regards to consistency between different engineering artifacts (e.g., between model and code or requirements and specifications). Current collaboration tools, such as version control systems, are not able to address these cross-artifact consistency concerns. The consequence is unnecessarily complex consistency maintenance during engineering. This paper explores consistent handling of engineering artifacts during collaborative engineering. This work presumes that all engineers collaborate using a joint, cloud-based engineering environment and engineering artifacts are continuously synchronized with this environment. The artifacts can be read and modified by both engineers and analysis mechanisms such as a consistency checker. The paper enumerates different consistency checking scenarios that arise during such collaboration.

CCS Concepts

•**Software and its engineering** → **Collaboration in software development;**

Author Keywords

Collaboration; Consistency Checking; Cloud Engineering; Linking

1 INTRODUCTION

Today's engineering landscape is filled with a multitude of widely used tools, meeting the diverse needs of engineers. In software engineering alone, we have tools for implementation, architecture & design, requirements engineering, testing and more. Collaboration among engineers is essential, yet the growing number and complexity of engineering artifacts captured through their tools makes this increasingly difficult. A single engineer is unlikely to be aware of, let alone understand or interpret, all artifacts [1]. This is in part caused by today's focus of collaborative tools, on single types of engineering artifacts. For example, version control systems such as Subversion[1] or Git[2], were originally developed for sharing code. It is technically possible to store different engineering artifacts in such systems as well, but the text-based nature of their mechanisms hinders a serious, fine-granular integration. This shortcoming results in a lack of information concerning artifact changes and their respective implications, meaning that the version control system may document changes, but not their intricate details (e.g., one could infer that a design document was altered but not which diagram or property within). This makes the analysis of changes rather cumbersome.

One major problem in this regard is, that inconsistencies - which are bound to arise, due to the heavily interdependent nature of engineering artifacts - become very hard to detect, before the integration of an engineer's work with the work of others. There have been various attempts at tackling consistency checking in general (e.g., [3, 4, 6, 7, 8, 11]), but the issue described beforehand makes it clear to us, that a mechanism providing more immediate feedback is required. There is a need for consistency checking enhanced by a collaborative environment and while some approaches have gone into that direction (e.g., [5, 9]) they mostly consider only homogenous engineering artifacts or require costly merging operations.

This paper presents a collaboratively enhanced consistency checking mechanism to better support the collaborative efforts of engineers. The approach is embedded in a cloud-based engineering environment, focusing on the integration of tools and synchronizing captured artifacts, to enable comprehensive reasoning, while engineers continue to use the tools they always have.

The rest of this paper is structured as follows: In Section 2, we present the DesignSpace engineering cloud [2] as the architectural foundation of our work. Realized on this platform, we present our approach of a collaboratively enhanced consistency checker in Section 3. We evaluate this work in Section 4 presenting the results of an experiment and two case studies.

EICS '19 June 18–21, 2019, Valencia, Spain

ACM ISBN 978-1-4503-6745-5/19/06…$15.00
DOI: https://doi.org/10.1145/3319499.3328232

[1] Apache Subversion: https://subversion.apache.org/
[2] Git: https://git-scm.com

The paper is concluded with an outlook towards the future work in Section 5.

2 THE DESIGNSPACE ENGINEERING CLOUD

The DesignSpace is a cloud-based collaboration platform. It is organized into artifact storage and collaboration services. Artifact storage provides a central and uniform place where engineering artifacts are stored. These artifacts typically originate from the various tools that engineers use (e.g., programming tools, modeling tools, requirements tools). Through tool adapters, the artifacts within tools are continuously synchronized with the DesignSpace. Engineers continue to use the tools they already know.

Collaboration services augment and analyze the artifacts in the cloud to assist engineers during their collaboration. By immediately reacting towards changes in the artifact storage, services provide live feedback and corresponding guidance to engineers. Collaboratively enhanced consistency checking is one such service. It provides the means to reason over artifacts stored in the cloud and is able to provide customized consistency feedback on arbitrary types of artifacts. This effectively means that consistency rules can be formulated beyond the boundaries of a single tool and compare artifacts from different tools with each other (e.g., model and code). The basic architecture of the DesignSpace is illustrated in Figure 1. In the following we discuss the basic aspects of this cloud environment that build the architectural foundation of our presented consistency checking approach.

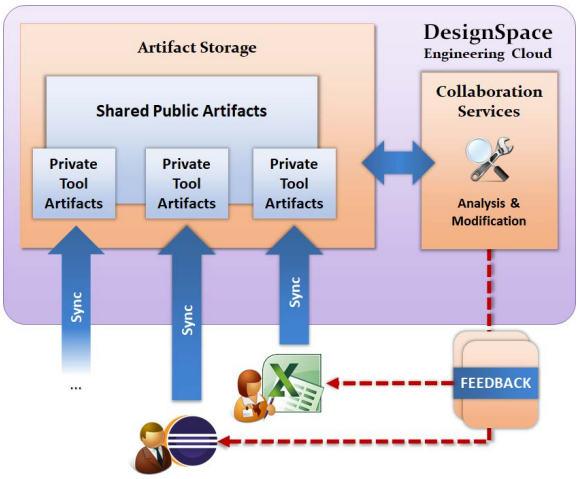

Figure 1. Illustration of multiple tools sharing artifacts on the DesignSpace and Services providing feedback

2.1 Uniform Data Representation

The artifact storage of the DesignSpace provides a uniform representation for arbitrary engineering artifacts. It ensures that engineers have access to all engineering artifacts and cloud services can parse them regardless of their origin. For this purpose, tool adapters translate the artifacts as they appear in the tools to a uniform data representation. This representation is a mapping between the properties of a data entity and their respective value. For example, a spreadsheet could be translated into the uniform data representation with properties for all relevant fields. The values mapped to these properties would represent the content of the respective fields. Likewise source code could be parsed such that classes, methods or fields become separate cloud artifacts where their properties reflect field or method names. The mapped values could be method bodies, return types, constant values or similar. The granularity of the artifact translation is defined in the tool adapters.

2.2 Live Feedback

As the engineering cloud is continously updated with the incremental data, every change triggers an event that can be forwarded through the cloud. Subsequently, these events can be caught by cloud services, e.g., to trigger mechanisms, or sent to tool adapters, which can translate them into user feedback, e.g., screen warnings.

2.3 Artifact Typing

Artifacts within the DesignSpace must adhere to a defined structure. We refer to these structures as types. Types can be defined by tool adapters, specifying certain properties as well as the primitive data type and cardinality of their values. The users can then easily instantiate the types and fill their values via a method provided by the DesignSpace.

2.4 Linking

Since artifacts are uniquely identifiable within the cloud environment, they can also easily reference each other, by simply storing each other's ID. The consistency checking service can utilize these links to reason beyond the boundaries of a single engineering artifact's domain.

2.5 Private Work Areas

Before being accessible to all engineers in the public repository (PR), artifact changes are synchronized with a private work area (PWA). This is an isolated view on the engineering artifacts the engineers are currently working on in their tools - respectively the delta of these artifacts in relation to their publicly available version. Assume engineers wish to make changes to an artifact stored within the PR. They would first checkout the artifact to their tool through the tool adapter. There they would make changes to the artifact's properties. These changes would then be stored in the PWA. Once the engineers decide to publish the changes, they are incrementally added to the PR.

2.6 Collaboration Services

As artifacts become available in the cloud, they can be analyzed by services. These services may modify artifacts on their own (e.g., to repair an error) or provide structured feedback to engineers (e.g., to warn about conflicts).

3 CONSISTENCY CHECKING

Consistency checking in general, is a technique to validate whether engineering artifacts adhere to a certain standard. This standard is normally regarded in isolation, i.e., they only matter for a certain type of artifacts. For example, Java code adheres to certain rules defined through the Java metamodel, certain IDE's evaluate written code against internal standards and some UML environments make sure that models stay

consistent with each other (e.g., by making sure lifelines in a sequence diagram correspond to a class in a class diagram). In broad terms, consistency checking differentiates between local and global consistency. The former describes the most common kind of consistency checking, as it only concerns itself with artifacts of a certain type adhering to internal standards (as described above). The latter, however, concerns itself with the consistency of several types of artifacts in relation to each other. Equivalently, one can also speak of intra- and inter-model consistency [10]. While systems for the former kind are rather frequent, the latter is rarely explored, mostly due to the lacking tool support. What is required for global consistency checking is a common ground on which engineering artifacts can be compared. Such a common ground is normally achieved through such activities as model merging. This is mostly connected with great efforts and thus rarely pursued. The DesignSpace, on the other hand, already provides a uniform data representation over which a consistency checker can reason. This is a major advantage since it allows us to countercheck changes on one typed artifact against values stored on a different type, e.g., one could synchronize both code and design documents with the DesignSpace. The consistency checker can then compare concrete related values and warn engineers if there are discrepancies.

3.1 Basic Architecture & Runtime Functionality

Since we covered both the technical pre-requisites, as well as potential benefits of a consistency checker within a collaborative cloud environment, we now discuss the technical details of its implementation and execution.

3.1.1 Data Structure

The consistency checker uses the uniform data representation not only for reasoning, but also as a data structure for internal information. Technically, this gives us the possibility to evaluate the consistency checking data for inconsistencies, but more importantly it grants a tight integration with the rest of the collaboration environment. Therefore, changes to its data structure are adapted incrementally and the resulting events can be forwarded through the cloud, for example, to trigger mechanisms of other services. Likewise they can be forwarded to users, e.g., when a consistency rule is broken. The consistency checker requires two major data structures:

- *Consistency Rule Definition Artifacts*: These artifacts define rules as a simple OCL[3]-like strings. They, furthermore, define a context for the rule's evaluation. This context is an artifact type for which the rule has to hold. Within the rule the context is refered to as "self".

- *Consistency Rule Instance Artifacts*: These artifacts are instanced for each artifact of a certain type referenced in a consistency rule definition. They store a context element (the concrete artifact instance of the context type) as well as a scope, which represents each artifact that is traversed by evaluating the consistency rule for the respective context element. Furthermore, the rule instances also hold the results of their individual rule evaluation.

[3]OCL: https://www.omg.org/spec/OCL/About-OCL/

As an illustrative example, consider the rule definition stated in Listing 1. In it, the declared context is a JavaClass, respectively the artifact type corresponding to the name JavaClass. The rule can be read like a simple object access in object oriented programming languages, i.e., from the context (self) the rule evaluation navigates through a reference "UML" to the name of the referenced UML artifact and compares it to the name field found in the context artifact. The consistency checker will instantiate a Consistency Rule Instance Artifact for each Java class artifact instance and evaluate the said instance according to the defined rule.

3.1.2 Rule Evaluation

In the following we will discuss a typical rule evaluation for the proposed consistency checking mechanism. As aforementioned, the consistency checker will instantiate Consistency Rule Instance Artifacts according to a respective definition. Each such instance references a certain context element for which the defined rule is evaluated. Such an evaluation happens in various steps:

- *Triggering the consistency checker*: The consistency checking mechanism can be triggered through various ways. It can either be started manually, by user input, or automatically, e.g., by catching change events on scope elements.

- *Retrieving the rule*: Once the consistency checker is active, it must first retrieve the rule corresponding with an artifact. For this the artifact triggering the mechanism must first be identified within a consistency rule instance artifact. It does not matter whether it is the context element or a scope element. Both kinds of changes can have an impact on the consistency state of a project. Once the artifact is found within a rule instance the referenced rule definition is retrieved.

- *Traversing the artifact structure*: Once the rule is retrieved, it can be executed. The rule itself describes a path from a context element to property values, which are then compared according to whatever operation is defined within the rule.

- *Write back results*: Once the rule is evaluated, the results are written back onto the Consistency Rule Instance Artifact.

- *Feedback*: Results can be forwarded to tool adapters and other services. The interpretation of the results (e.g., triggering screen messages or service mechanisms) is up to them.

During the evaluation process, the consistency checker also updates the respective consistency rule instances scope. Every artifact the consistency checker passes during the traversal of the artifact structure is stored in the scope for future evaluations.

```
Context:    Java::Class
Rule:       self.UML.name == self.name
```

Listing 1. A simple consistency rule comparing names between a JavaClass artifact and the corresponding UML diagram.

3.2 Collaborative Enhancement

With the deployment on the DesignSpace and the utilization of its architecture, the consistency checker can be enhanced through several collaborative features. In our approach, we covered the following:

- Full Activity Awareness
- Multi-Tool Consistency Checking
- Team-Driven Error Handling

In the following, we will discuss how individual features of the DesignSpace were exploited to achieve these features.

3.2.1 Full Activity Awareness

A way to collaboratively enhance consistency checking is to make the corresponding mechanism aware of each engineer's work. For our approach, three forms of awareness were established by deploying the consistency checker as a collaborative service:

1) Artifact Awareness: The most basic capability of an engineering cloud is to enable awareness of each others engineering artifacts. The lack of awareness of another engineers' work may lead to conflicts and unnecessary rework. This hinders collaboration as much as it hinders collaboratively enhanced consistency checking, which can in turn help us to reduce conflicts between engineering artifacts. Through the full integration of the consistency checker with the DesignSpace's uniform data representation the mechanism works natively on the cloud's data structure. Artifact awareness is given by providing the service with full access to the artifact storage. Contrary to the regular users the service does not only overlook a single PWA, but all work areas, including the PR. This also grants the consistency checker the possibility to retrieve a full representation of an engineering artifact within the context of each engineer's individual PWA. Each such context can be seen as an individual view of the consistency checker on engineering data. Such a view can be defined as follows:

$$V_{\mathrm{cc}}(A) = V(A(P_{\mathrm{m}} \cup P_{\mathrm{n}}))$$
$$P_{\mathrm{m}} = \{\forall p \mid p \in PWA\}$$
$$P_{\mathrm{n}} = \{\forall p \mid p \in PR \ \wedge \ p \notin PWA\}$$

Where V_{cc} is the view of the consistency checker on an artifact A. This is equivalent to the view on two property sets P_{m} and P_{n}, both in union making up the artifact A. P_{m} is a property set of all properties that have been changed within a PWA and are thus only available there in isolation, wheras P_{n} is a property set of all properties residing *only* within the PR. Since they have not been changed they do not reside within the PWA. In short: In the context of full artifact awareness, the consistency checker can view an artifact comprised of its public version complemented by the individual changes of the engineers.

2) Change Awareness: An engineering cloud can provide change awareness so that engineers and collaboration services see changes instantly. This is especially useful when the engineers' works have strong, immediate implications on each other. In such a scenario immediate consistency checking - triggered by change events - can be of high value. For this to happen the collaboration service in our approach directly listens to change events fired by both the PWAs and the PR. Catching these change events triggers the consistency checker in one of two possible modes:

- *Private Change Evaluation*: These are triggered by change events fired in a PWA. The evaluated consistency information is written into the private version of the respective Consistency Rule Instance. The newly computed results are only visible to the engineer using the PWA.
- *Public Change Evaluation*: These are triggered by change events fired in the PR. Since consistency information is also stored privately - and subsequently committed to the PR - change events triggered by users publishing their private changes are ignored. Instead, this form of rule evaluation can only happen if a service changes the PR directly. The newly computed results are visible to all users and services.

Both modes can perform local or global consistency checks, since the extent of a check is dependent on the nature of the written consistency rule. If a consistency rule contains global elements, like a link towards different engineering artifacts not available within the PWA, the consistency checker will automatically retrieve the publicly available version of the corresponding artifacts and integrate it into the rule evaluation.

3) Error Awareness: Concluding it's rule evaluation process, the consistency checker uses the direct connection between PWAs and tools to send the consistency feedback to all listening tool adapters as illustrated in Figure 1. The tool adapters can then visualize this feedback in a customized manner. In this context it is important to note that anybody can register for the error feedback of any PWA. This allows us to create a full error awareness for engineers, which can also be utilized collaboratively. For example, an electrical engineer and a software engineer could register to receive the consistency information of their respective changes. If, however, they choose to work as a group/team then they could choose to get feedback with regard to the union of the group's changes, while still working in isolation on their own engineering artifacts.

3.2.2 Multi-Tool Consistency Checking

A collaborative engineering cloud, such as the DesignSpace, gives its users the ability to augment tool knowledge with additional information, which tools are normally not able to capture themselves. To our understanding, one of the most crucial bits of additional information is the relationship between artifacts from different tools. Such relationships can be established in the DesignSpace through artifact links as aforementioned in Section 2.4. These links can be extensively utilized by a collaboratively enhanced consistency checker, especially with regards to multi-tool, respectively global consistency. Since links are defined through regular properties on an artifact, it is sufficient to refer to such a property in order to navigate from one artifact to another during the rule-evaluation process . Consider again the consistency rule given in Listing 1. In it, we refer to a property called "UML". In the DesignSpace this property can be defined as a reference towards a corresponding UML artifact. During the rule evaluation process the consistency checker will automatically navigate to

the referenced artifact and continue the evaluation from there. Subsequently, our approach can rely on such links to define conditions beyond the boundary of a single artifact or even the engineering domain. Additionally, it should be noted, that the consistency checker can also be used to validate such links, e.g., by making sure the properties are not empty. If, alternatively, links are established through separate artifacts featuring both a source and target property, the consistency checker can - given proper rules have been established beforehand - evaluate whether these artifacts have been created correctly.

3.2.3 Team-Driven Error Handling

Concurrent modification of engineering artifacts leads to errors - errors that are increasingly expensive to fix the longer they stay undetected. A collaboratively enhanced consistency checker can support cross workstation/tool/discipline error feedback on a scale that is otherwise not possible today. Especially with the help of change events being fired in both PWAs as well as the PR, the consistency checker can analyze arising inconsistency immediately and send the corresponding feedback to engineers. With this, fixing errors can become a collaborative effort, as the inconsistency feedback contains exact information about broken consistency rules, affected artifacts and the workspaces that issued changes leading to inconsistencies. The latter bit of information is especially important, since it gives the engineers indications on who to communicate with when fixing a problem. As an additional note: Enforcing consistency rules can also be used to prevent errors in the first place, e.g., by pre-checking changes and rejecting them if they violate any rule. However, since a temporarily inconsistent state of the isolated work is normally tolerated, this form of rule enforcement might be regarded as too restrictive by the engineers.

4 EVALUATION

The collaborative engineering cloud as well as the proposed collaboratively enhanced consistency checking mechanism have been evaluated via a prototype implementation in the context of an industrial experiment and two major case studies. In this section, we discuss them in more detail.

4.1 Prototype Implementations

The DesignSpace approach is different from much of its related work mentioned in Section 1, because it does not emphasize on certain collaboration styles but is meant to provide flexibility in collaboration. The goal is a freely definable and changeable collaboration approach where engineers can join groups, change artifact awareness, or engage arbitrary processes. The current implementation supports a range of tool adapters, such as Java Eclipse[4], IBM Rational Software Architect[5] (for UML), Microsoft Excel (for calculations), Creo[6] (for CAD Drawings), Eplan Electric P8[7] (for electrical layouts), and others. These tools demonstrate convincingly that it is possible to represent arbitrary tool artifacts in a cloud. The current implementation of the consistency checker makes extensive use of linked artifacts from different tools and as such augmented links with error rules to detect inconsistencies on the fly.

4.2 Experiments & Case Studies

Our collaboratively enhanced consistency checking approach has been validated on the basis of an experiment as well as two case studies, which are discussed in the following.

4.2.1 Industrial Experiment

Our consistency checking approach has been beneficially utilized in an extensive industrial experiment with Van Hoecke Automation[8]. There, consistency has been secured between electrical models as well as their software controllers. The corresponding artifacts were made available in the DesignSpace and altered in parallel, while continously being counterchecked against established consistency rules. Errors, respecitvely arising inconsistencies, were then fed back to the engineers, who worked with tool adapters for EPlan and Eclipse. With this the application of our approach could achieve full activity awareness. Multi-Tool consistency checking was established through various types of links, which were evaluated during the experimentas well. Artifacts representing spreadsheet cells, code and EPlan model elements have been linked manually through a separate tool. These links were typed artifacts and were not only used by the consistency checker, but also established traceability between engineering artifacts. This confirms not only the feasibility but adds to the usefulness of our approach, as traceability is an additional important issue covered through the DesignSpace. Both the detection of conflicting changes and also the prevention of errors has been done within the experiment. Feedback about inconsistencies was provided instantly after the consistency checker analyzed changes and evaluated them as erroneous. Alternatively, consistency feedback could be requested by the engineers manually.

4.2.2 FMTC Case Study

Full activity awareness was further evaluated during a case study with the Flander's Mechatronics Technology Center (FMTC[9]). In this case study, EPlan Electric P8 drawings were provided by a third-party company. These drawings had to be kept consistent with their respective code implementations according to user-defined, domain-specific consistency rules. Through customly implemented tool adapters, the respective artifacts were integrated in the DesignSpace, giving validation to our approach with regards to its multi-tool aspect. Links between artifacts were created with a custom DesignSpace tool. After issuing a series of changes, instant consistency feedback was provided to engineers. Detected discrepancies between the drawings and the code were reported, evaluating the notion of full activity awareness as well as team-driven error handling.

4.2.3 ACCM Case Study

To evaluate our approach within a different engineering domain, a case study was conducted with the Austrian Center of

[4]Eclipse IDE: https://www.eclipse.org/

[5]IBM RSA: https://www.ibm.com/developerworks/ downloads/r/architect/index.html

[6]Creo: https://www.ptc.com/de/products/cad/creo

[7]EPlan: https://www.eplanusa.com/us/2/

[8]Van Hoecke Automation: https://www.vha.be/

[9]FMTC: https://www.flandersmake.be/en

Competence in Mechatronics (ACCM/LCM[10]). There, the mechanical calculation for a robot arm was shared and analyzed on the DesignSpace. Full activity awareness was given with particular focus on error awareness, as engineers were notified about changes that have been counter-checked against established consistency rules. With the help of the notifications problem solutions concerning arising inconsistencies could be worked out in a team-driven way. The consistency rules required the established links between artifacts from different tools. In particular, these links involved relations between UML models, CAD drawings and spreadsheet artifacts, validating our approach with regards to its multi-tool capabilities.

5 CONCLUSION & FUTURE WORK

Software systems engineering is a highly complex activity that integrates knowledge from a variety of disciplines. Engineers attempt to manage and break down this complexity by focusing on what they can do individually in the context of tools they are using. However, this results in knowledge fragmented across many tools and an increased risk of inconsistencies. This paper has shown that a collaboratively enhancend consistency checker can rectify this issue. Our proposed mechanism both contributes to and benefits from a cloud-based engineering platform, that integrates engineering artifacts from different disciplines. We discussed various aspects of collaboration, that are affected positively by our approach, most importantly activity awareness, multi-tool consistency checking and team-driven error handling. With regards to the future work, we would like to extend the consistency checker's capabilities under particular consideration of the following scenarios:

Selected Group Awareness: Sometimes consistency information of a specific group context (e.g., a certain team within a project) can be of interest. While engineers might be reluctant to counter-check the unfinished work with publicly available engineering artifacts, they are more likely willing to check work-in-progress against selected sub-results of their own workgroup. This can be achieved by changing the context of consistency checks to a selected set of artifacts. The results can help engineers to integrate their own work with the rest of the team.

Change and Reuse The systematic linking of engineering knowledge also benefits change and reuse. Imagine a project about a robotic system which has similarities with past solutions - for example, parts of a robot arm. With all engineering knowledge available in the cloud and with all its artifacts linked, engineers could now extract the said robot arm's specifications, use cases or associated code. Consistency checking can simplify this process, since reused engineering artifacts must still adhere to the same or similar consistency rules.

6 ACKNOWLEDGEMENT

This work is supported by the Austrian Science Fund (FWF), grant no. P 31989-N31, the JKU Linz Institute of Technology (LIT), the state of Upper Austria, grant no. LIT-2016-2-SEE-019, Pro2Future, a COMET K1-Centre of the Austrian Research Promotion Agency (FFG), grant no. 854184 and the Software Competence Center Hagenberg GmbH.

[10]ACCM/LCM: https://www.lcm.at/unternehmen/kompetenzzentrum/

REFERENCES

1. Andreas Demuth, Roland Kretschmer, Alexander Egyed, and Davy Maes. 2016. Introducing Traceability and Consistency Checking for Change Impact Analysis across Engineering Tools in an Automation Solution Company: An Experience Report. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 529–538.
2. Andreas Demuth, Markus Riedl-Ehrenleitner, Alexander Nöhrer, Peter Hehenberger, Klaus Zeman, and Alexander Egyed. 2015. DesignSpace: an infrastructure for multi-user/multi-tool engineering. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 1486–1491.
3. Anthony CW Finkelstein, Dov Gabbay, Anthony Hunter, Jeff Kramer, and Bashar Nuseibeh. 1994. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering* 20, 8 (1994), 569–578.
4. Pascal Fradet, Daniel Le Métayer, and Michaël Périn. 1999. Consistency checking for multiple view software architectures. In *Software Engineering - ESEC/FSE'99*. Springer, 410–428.
5. Harald König and Zinovy Diskin. 2016. Advanced local checking of global consistency in heterogeneous multimodeling. In *European Conference on Modelling Foundations and Applications*. Springer, 19–35.
6. Christian Nentwich, Licia Capra, Wolfgang Emmerich, and Anthony Finkelsteiin. 2002. xlinkit: A consistency checking and smart link generation service. *ACM Transactions on Internet Technology (TOIT)* 2, 2 (2002), 151–185.
7. Steven P Reiss. 2006. Incremental maintenance of software artifacts. *IEEE Transactions on Software Engineering* 32, 9 (2006), 682–697.
8. Markus Riedl-Ehrenleitner, Andreas Demuth, and Alexander Egyed. 2014. Towards model-and-code consistency checking. In *2014 IEEE 38th Annual Computer Software and Applications Conference*. IEEE, 85–90.
9. Mehrdad Sabetzadeh, Shiva Nejati, Steve Easterbrook, and Marsha Chechik. 2008. Global consistency checking of distributed models with TReMer+. In *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 815–818.
10. Michael Alexander Tröls, Atif Mashkoor, and Alexander Egyed. 2019. Live and Global Consistency Checking in a Collaborative Engineering Environment. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)*. ACM, 1762 – 1771.
11. Michael Vierhauser, Paul Grünbacher, Alexander Egyed, Rick Rabiser, and Wolfgang Heider. 2010. Flexible and scalable consistency checking on product line variability models. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 63–72.