Multifaceted Consistency Checking of Collaborative Engineering Artifacts

1st Michael Alexander Tröls
Johannes Kepler University
Linz, Austria
michael.troels@jku.at

2nd Atif Mashkoor
Software Competence Center Hagenberg GmbH & Johannes Kepler University
Linz, Austria
atif.mashkoor@jku.at

3rd Alexander Egyed
Johannes Kepler University
Linz, Austria
alexander.egyed@jku.at

***Abstract*—In modern day engineering projects, different engineers collaborate on creating a vast multitude of different artifacts such as requirements, design specifications and code. While these artifacts are strongly interdependent, they are often treated in isolation and with little regard to their semantical overlappings. Automatic consistency checking approaches between these artifacts are rare and often not feasible. Therefore, artifacts become inconsistent and the consequences are costly errors.**

This work proposes a multifaceted consistency checking approach for different kinds of engineering artifacts, with the help of a collaborative engineering platform. The proposed approach enables engineers to automatically check the consistency of their individual artifacts against the work results of other engineers, without using different tools than the established ones of their fields and without merging their artifacts with those of others.

***Index Terms*—collaborative engineering, engineering artifacts, consistency checking**

I. Introduction

Engineering is a complex process involving different engineers from a multitude of varied disciplines. Each engineer produces artifacts such as code, requirements, models, hardware specifications and many more. The complexity and number of these artifacts is constantly growing as engineering projects become an increasingly important part of many different industries. A modern engineering project is no longer just concerned with the details of, e.g., a software solution or a mechatronical system – it acts as an inter-disciplinary process incorporating the knowledge from many different areas such as medicine, agriculture, logistics and more.

Likewise, the interdependendcy among engineering artifacts is strongly increasing as well. With the engineering process becoming more and more complex, the necessity to integrate and propagate changes from highly interdependent artifacts within a project becomes a more elaborate and time costly endeavour. For example, if hardware specifications change, all related engineering artifacts, first and foremost those using the specified hardware, must propagate such a change in one way or another. Frameworks may need to be updated, implementations might need to be adapted and designs may require restructuring in the light of new technological circumstances.

This combination of complexity, high interdependency and constant availability of the increasing amounts of engineering artifacts, has made the maintenance of the involved information a difficult task. As a result, keeping consistency among engineering artifacts is a critical aspect of each engineering project [1]. However, the available software solutions are lacking in this regard. While a regular engineering project may share its engineering artifacts among engineers in a multitude of different ways, this is not a sufficient solution to overcome the increasing complexity of the artifacts and their interdependencies. Despite its major industrial importance and constant reminders by the research community (e.g., [2]–[9]), maintaining consistency among artifacts is treated as an afterthought.

Therefore, this work proposes a novel, multifaceted way of checking consistency between engineering artifacts within a collaborative environment. The consistency checks are automatically triggered by changes engineers perform on their aritfacts and happen under the consideration of the semantical overlappings of the different types of engineering artifacts involved in a project. This enables engineers to approach the problem of the strongly interdependent nature of engineering artifacts and simplifies the engineers' task of maintaining their work results' consistency. We tackle the artifacts' physical, virtual and syntactical separation by synchronizing them into a collaborative cloud environment, where they share a common representation. There they can be arbitrarily linked to each other, which allows us to establish semantical equivalencies between the different artifacts. These links can subsequently be used for consistency checking on the basis of a cloud service and a set of user-determined consistency rules, which are automatically checked on the synchronization of a change with the cloud.

The rest of this paper is organized as follows: In Section II we discuss the current state of collaborative engineering and address problems that may commonly occur during modern engineering projects. In Section III we introduce our approach for tackling these problems. In Section IV we discuss details on the realization of our approach. We conclude this paper with a discussion of the approaches benefits in Section V.

II. Problem Statement

A single engineering project may involve a multitude of different collaboration tools, many of which are used in parallel. Such tools include repositories, e.g., Git[1] or SVN[2], time planning tools, communication tools or various team management solutions. Each of these tools incorporates engineering artifacts in one way or the other. Artifacts may be stored in a repository or shared through communication tools - yet, while engineering artifacts are passed around through and stored within these collaboration solutions, they are often not truly integrated within the tool. A traditional repository, for example, may store different engineering artifacts, but does not offer ways to natively incorporate the relationships between them. As a result, different engineering artifacts remain logically isolated from each other, even when they are virtually stored in the same space. This naturally makes the maintenance of consistency among engineering artifacts a highly complex task, as it has to be treated in an equally separate way for each type of artifact. No true incorporation of different engineering artifacts in a single consistency check is feasibly possible. Yet lacking such consistency checking may lead to a multitude of different problems. In the following we discuss some of the most critical problems, as well as other circumstances within current collaboration solutions that hinder feasible consistency checking.

- **Syntactic differences of engineering artifacts**: Most modern collaboration tools allow the storage of a single type of engineering artifacts. For example most repository solutions are text- respectively line-based and therefore, mostly used for code. Different engineering artifacts, such as design models, can technically be stored on these repositories as well, however, their formats - especially if they are stored in binary files - are often not natively supported. As a result, meaningful analysis of different engineering artifacts in one space is very difficult.
- **Artifact interdependency**: Despite the high interdependency between different engineering artifacts, collaboration solutions often provide no way of properly representing this circumstance efficiently. This complicates the effort of maintaining consistency among artifacts, as a consistency checker – even if it can interpret all engineering artifacts – has no possibility of making use of the semantic overlappings between different artifacts. Therefore, similar concepts, represented in different kinds of engineering artifacts, become isolated from each other and the task of keeping them consistent might fall short.
- **Limited collaboration techniques**: Due to the large variety of engineering artifacts, there is a limited support for comprehensive, native integration within most collaboration solutions. While collaboration techniques like merging or branching are common for text-based artifacts – e.g., code in Git or SVN – such operations are not possible for many other kinds of engineering artifacts. For example, design models or circuit diagrams are rarely meaningfully supported in this regard.
- **Difficult merging process**: Current collaboration solutions provide only limited conflict identification for engineering artifacts. Once the work of one engineer has to be integrated with the work of others, this may lead to a difficult merging process. This may further lead to unnecessary delays, which can become a crucial problem, especially when the integration process of engineering artifacts is tackled close to deadlines. As a result, hasty resolutions may often be incorporated into the final product, which potentially lead to severe errors.
- **Refactoring efforts**: When the merging process leads to the identification of problems, the consequence is often an extensive refactoring phase. Engineers may have to carefully refactor their work into a state where it is consistent with the work of others again. For example, both design models and code of a software engineering project may become out of sync, which makes it necessary to update one or both sides, in order to keep the project's documentation up-to-date.
- **Tolerated inconsistencies**: It often occurs that even though inconsistencies are identified during the merging process, they are being tolerated to a certain extent. This is, for example, done to simplify development processes during engineering projects or out of organizational reasons. However, at some point the identified inconsistencies must be corrected. To guarantee this, not only the identification, but the documentation of inconsistencies is a critical task. Engineers must know which issues arose in which context at what time to meaningfully merge the final engineering artifacts and subsequently remove the documented inconsistencies. Often, since cross-documentation of different types of engineering artifacts and their relationships rarely exist, the removal of older inconsistencies is entirely based on the memory of individual engineers. This, of course, holds massive potential for errors in the final product, especially if an engineering project has long development cycles or engineers are replaced during the project. Consequently, many undocumented inconsistencies remain hidden for the rest of the development. As a result, costly errors can end up in the final product, which are – if they are detected at all – hard to fix.
- **Error propagation:** Artifacts which underly constant changes by several engineers also produce constant consequences for related artifacts. So, while a single change may not cause an inconsistency in itself, the change may lead to errors once it is propagated through into other engineering artifacts. If that is the case, it is important to know the exact change history of engineering artifacts and the associated consistency state. Lacking the possibility to fall back on such documentation diminishes the possibility of identifying the original source of a problem. Even if an inconsistency can be found, the unawareness of its further consequences may nullify the attempt to repair it and turn the repairs into sources of new errors.
- **Symptomatic inconsistency repairs:** Given that a consistency checker finds an inconsistency, repairing the

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

inconsistency does not necessarily eradicate the problem. If the inconsistency is correlating with previous changes, the actual root of the problem can not be gotten rid of by fixing the latest effect. As such, engineering projects enter a cycle of fighting the symptoms of old, largely invisible problems.

- **Follow-up errors:** Naturally, the practice of fixing symptomatic problems can lead to follow-up errors. Especially if later project specifications start to clash with the original problem. These decisions may propagate through one or several engineering projects for a very long time, before the real problems are finally recognized. A typical example of this are long lasting errors in software solutions based on architectural mistakes. If the same base architecture is re-used during the production of new software, respectively new versions of same software, old errors may lead to new work arounds and new errors.
- **Late or no recognition of inconsistencies:** If, at last, the original problem causing an inconsistency is - perhaps by chance - identified, the damage is most likely already done. The effects are potentially immense costs, radical changes of established systems or product lines and possibly even legal consequences. This makes the early recognition of inconsistencies - and thus the recognition of correlating changes - not just a side issue, but an existential necessity for the smooth execution of engineering projects.

As can be seen, the support of different engineering artifacts, as well as the documentation of their relationships with each other, is an important aspect of engineering. Yet, the current landscape of collaboration solutions is severly lacking in these regards. Therefore, the identification of inconsistencies becomes very difficulty.

III. Multifaceted Consistency Checking

The issues discussed in Section II show a wide array of different aspects that are affected or could be averted by a multifaceted consistency checking approach fit for the growing complexity, number and required availability of engineering artifacts. As such, we suggest a consistency checking mechanism featuring five core characteristics in this work:

- Cloud Deployment
- Global Constraints
- Live Capabilities
- Context Views
- Group Orientation

Each of these characteristics offers individual advantages for the engineer, which are discussed in the following.

A. Cloud Deployment

To meet the requirement of constant availability, storing engineering artifacts in a single, network-connected storage space comes as a natural practice in most engineering projects. Considering consistency checking, this means, that the respective mechanism could be deployed on such a storage, instead of being run individually for each tool, on each machine contributing to a project. The core issue with this is not the technical limitation of data storage, e.g., by synchronizing artifacts into a cloud environment. The problem is to natively store them in an integrated way that allows us to handle their complexity, especially with regards to their syntactic differences. A consistency checker must either operate on a single uniform engineering artifact representation or be able to interpret all different engineering artifact formats that are synchronized with the storage. Naturally the former is a more feasible solution.

In this work we settle for a uniform artifact representation, which can be further exploited for advantageous concepts that aid us in tackling the complexity of engineering artifacts. In fact, a core issue that encompasses many of the problems discussed in Section II is the syntactic difference of engineering artifacts. While a multitude of artifacts is stored in various textual representation (coding languages, plain text, HTML, XML, etc) many others are, for example, stored in a binary format (images, CAD drawings, etc). Regardless of their format these artifacts share interdependencies and semantical overlappings that need to stay consistent. A classic example would be the names of a class diagram and its concrete implementation in code. To keep these artifacts consistent requires either a consistency checker that is able to read all the involved formats and interpret them, or a transformation of the artifacts into a single uniform representation in which they can be read and analyzed on a syntactic common ground. With such a uniform representation present, the deployment of a consistency checker within a cloud environment becomes both feasible and advantageous.

B. Global Constraints

One way to advantageously exploit the uniform representation of artifacts synchronized with the cloud environment, is the possibility to represent their interdependencies in the same abstracted manner. While engineering artifacts feature many different semantic overlappings, the resulting interdependencies between them are only rarely comprehensively documented. Even with traceability being a common challenge for each engineering project, most tools concerned with it, are limited to parallel representations of the matter in further separate tools. The approach presented in this paper suggests a more integrated way of representing the interedependencies of artifacts. In our approach we represent semantic overlappings, respectively the interdependency of engineering artifacts, through the manipulation of the uniform artifact representation directly in the artifact storage space. This allows us to arbitrarily extend the engineering artifacts with additional meta-information, e.g., the relationship to other artifacts. If, for example, a certain piece of code realizes a certain UML diagram, this relationship can be expressed through the addition of a simple link on the respective code artifact, pointing towards the uniform artifact representation of the UML diagram artifact. Alternatively, a new artifact can be created which then points towards both the code and the UML artifact as source and target respectively.

These links can then be used by the consistency checker to analyze the impact of an engineering artifact's change beyond the boundaries of a single discipline. This can be done by formulating consistency rules in a way that builds on the uniform artifact representation and therefore utilizes the links that can be set therein. This gives engineers the possibility to implement global constraints for entire engineering projects, which can be applied in multiple scenarios, from model-code consistency checking, to checking the integrity between implementation and corresponding circuit diagrams ([9], [10]).

C. Live Capabilities

To prevent the late recognition of inconsistencies and to tackle the aspect of ever changing artifacts having an impact on artifacts related to them, a consistency checker capable of reacting live towards changes becomes a necessity. Exploiting the cloud deployment and the uniform artifact representation therein can help tackle this problem. As a single, cloud-based consistency checker, familiar with the uniform artifact representation can be responsible for all engineering artifacts within an engineering project, the computation of consistency states can be entirely server sided. This way consistency checks can be done in parallel to engineers working. Their work is not interrupted. At the same time, the server-sided computation can apply load-balancing strategies that help minimizing the computation time for a consistency check. As a result consistency feedback can be given through the network directly to the engineers, live as they synchronize their artifact changes with the cloud. Immediate feedback of such nature can be utilized to foster a more team-driven approach towards error handling, as well as awareness of collaborative activities within the team of engineers [8].

Handling the amount of computations and feedback is no trivial task in this context. The organization of the cloud's artifact storage into several substorages can be advantageous in this context. In this work we apply both a private and a public work area - the earlier holding only work of a single engineer, which is then pushed to the latter and made publicly available for every engineer using the cloud environment. Every change within one of the substorages fires an event. Feedack that is not related to a certain work area can easily be filtered out for the connected engineer. Further our data model allows us to minimize the number of (necessary) executed consistency checks. This adds to the scalability of our consistency checking approach.

D. Context Views

An engineer's work can always be regarded in two different consistency contexts: By itself - e.g., code corresponding to certain syntax - and integrated with the work of others - e.g., code corresponding to UML designs. To prevent merging conflicts as well as extensive refactoring processes, our approach can regard the consistency state of an engineer's work from the context of a private work area. This way, the engineer's work is projected on top of publicly available engineering artifacts. The consistency checker can then deliver information on what would be the consistency state of both the private and the public artifacts, would the engineer decide to push his work to the public area. Since feedback can be given live, the engineer can always be aware of this information.

E. Group Orientation

Adding on top of context views, multifaceted consistency checking can also regard the consistency of engineering artifacts within a group context. Then the engineers' work, held in their private work area, is not primarily projected on top of publicly available artifacts, but on top of the work present within a group of other private work areas. The consistency state computed from this context allows engineers to be aware of their work's consistency integrated with the ongoing work of other engineers. This further prevents a complex merging process as conflicts can be detected before engineers decide to make their engineering artifacts publicly available.

IV. Realization

The approach discussed in this work is a live consistency checker of global consistency rules, representing the interdependent aspects of engineering artifacts. It reacts towards the synchronization of artifact changes onto a collaborative engineering cloud environment, where all engineering artifacts are stored in full or partially. The snychronization is handled by tool adapters, which are used with the regular engineering tools the engineers use in their respective fields. Our approach has no need of switching tools. Engineers may work with what they already know. In this Section we outline how the approach tackles the problems discussed in Section II, respectively the concepts discussed in Section III with regards to their realization.

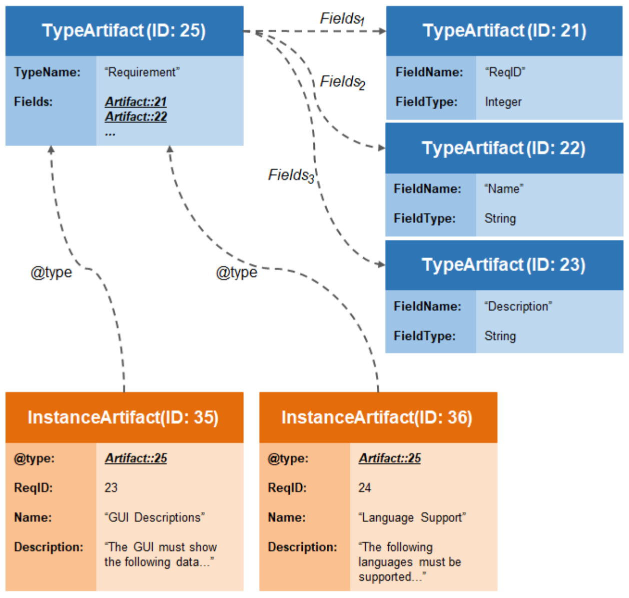
A. Uniform Data Representation

A problem that has to be adressed before realizing multifaceted consistency checking of varied engineering artifacts, is the equally varied landscape of different artifact formats. A large portion of engineering tools store their outputs in unique data structures, respectively file formats. This is a major hurdle, when it comes to the meaningful integration of engineering artifacts in a single storage space and subsequently, the analysis of their semantically overlapping aspects. Therefore, this approach applies a uniform data representation, into which engineering artifacts - or parts thereof - are transformed, before they are stored in a cloud environment. This allows a cloud service to analyse the stored data, without knowing the specifics of the original artifact format.

The uniform data representation engineering artifacts are transformed into, is a typed, uniquely identifieable mapping of named values with timestamps. These named values are refered to as properties. A single property always contains a list of values, which represent the property's change history.

Every artifact in the cloud environment is created according to a corresponding type. Such types can either be created manually by users, or automatically by tool adapters (e.g. during the adapter's initialization process). Further, the type

Fig. 1. Exemplary types and instantiations for requirement artifacts.



contains a set of definition fields that describe the name, cardinality and primitive data type of an artifact's properties. Once artifacts are synchronized with the cloud environment, the type is automatically instantiated and the respective properties are filled with values. The mapping process is done on the side of the synchronizing engineering tool, with the help of a tool adapter - which can be written as a plugin, using the cloud environment's API.

The data representation is willfully kept abstract to allow the biggest possible number of different engineering artifacts to be stored in the cloud environment. A figurative transformation of engineering artifacts can be seen in Figure 1, where a requirement type is present in the cloud environment and two instances have been created with corresponding values.

B. Linking

To represent the interdependencies between different engineering artifacts, the applied uniform data representation allows us to link different engineering artifacts together, by referencing their unique identifier within a property. This way, semantical overlappings between engineering artifacts from different fields, can be easily expressed. In the current approach, the linking process is handled manually - however, automated approaches can easily be deployed on the cloud environment in the form of a further service. The linking tool used in this approach can be seen in Figure 4, showing an overview of the artifact structure, as well as a detailed view on a selected artifact. The links themselves can be set as simple references on artifacts, or be instantiated as artifacts themself - with source and target properties pointing towards the respective linked engineering artifacts. This allows our approach to apply different link types and enhance them with further meta-information. Such structures can again be used by a consistency checker for the analysis of relationships between artifacts. Furthermore, the timestamped history of property values gives information on which artifacts were linked together at what point in time, which automatically documents re-arrangements within the relationships of artifacts.

C. Artifact Storage

Once artifacts are transformed into a uniform representation, they can persistently be stored in the cloud environment. More precisely, the artifact storage stores a set of property changes, the sum of which describes the full representation of an artifact. The artifact storage itself is separated into two types of work areas.

- **Public Work Area:** There is only a single public work area in the cloud environment. Engineers can push their changes into the public work area, once they are done editing them in their respective private work area. The public work area can be accessed by any engineer at any time. It acts as the root within a hierarchy of work areas.
- **Private Work Area:** There are several private work areas within the artifact storage. Each tool adapter synchronizing engineering artifacts with the cloud environment is bound to a single, unique work area. When synchronizing an engineering artifact, the private work area only stores a change, respectively a delta of the artifact with respect to the publicly already available information on the said artifact. This restricts the contents of the private work area to a relatively small set of changes. The private work areas can be parented to each other, with the public work area always acting as the highest parent. The parenting concept allows us to provide specific, change-oriented views on engineering artifacts. Tool adapters, retrieving engineering artifacts from the cloud, always receive the changes they synchronized with their bound private work area projected on top of whatever artifact information is available in the private work area's parents. An example for such information retrieval can be seen in Figure 2, where the public area of the artifact storage contains an instance of a "Robot Arm" engineering artifact, describing length and variant of the hardware. From the perspective of private work area "WA1", the robot arm has the length of 0.75 meters, since its contents are projected on top of the information within the public area. Work area "WA2" on the other hand still retrieves the length as 1.00 meters. If requested properties cannot be found within a private work area, they are retrieved from the parents - in this case the public work area. Vice versa "WA1" can not retrieve the latest name and variant description of the robot arm, as changes on these properties are only available within work area "WA2".

Since many engineering projects are organized in development teams, the artifact storage also allows for the grouping of work areas. These groups can be established by the users and alter the data retrieval process. When a property can not be found within a grouped private work area, the cloud environment will first check the other work areas within the group instead of the private work area's direct parents. If the property is found in multiple members of the work area group, the cloud environment retrieves the latest version of it, based on its value's timestamp. This way - when a full artifact is retrieved from the artifact storage - the changes of a private work area are first projected onto whatever relevant properties can be found within the group, which are then projected onto what can be found in the public work area. Coming back to Figure 2, would "WA1" and "WA2" be grouped together, the latest name and variant changes - present in "WA2" - could be retrieved from the perspective of work area "WA1". Vice versa, the latest length - present in "WA1" - could be retrieved from the perspective of work area "WA2". Property duplicates would have no effect, since there are only two work areas and the work area from whose perspective data is retrieved overrules the group.

D. Live Services

With engineering artifacts being persisted within the artifact storage, our approach can now access the said storage to analyze and potentially alter its content. This is done through live services, which are running in parallel to the cloud's activities. Every cloud activity, for example the synchronization of a change, triggers a corresponding event, which can be listened to by the services. As a result, every service can immediately react towards changes synchronized by the engineers. This holds the advantage that it allows us live consistency checking, which supports the effort of keeping inconsistent states of an engineering project as short as possible, and which prevents the late recognition of engineering artifact inconsistencies.

E. Consistency Checking

With the discussion of the problems listed in Section II the importance of comprehensive consistency checking becomes very clear. Maintaining consistency and keeping the complexity of engineering artifacts in check is a critical aspect of every large engineering project - not just for the sake of documentation, but to ensure a level of quality in the final product by preventing unnecessary flaws. Our approach builds on the cloud environment's concepts introduced beforehand. As such our consistency checking approach is implemented in the form of a live service. It listens to changes synchronized with the artifact storage, analyses these changes for their relevancy and re-evaluates the altered artifacts in parallel to other cloud activities. The results are fed back to the users through their tool adapters. This process is illustrated in Figure 3. This section goes into detail on the specifics of the consistency checker's data model within the cloud environment, as well as its concrete functionality whenever a change happens within the artifact storage.

Fig. 2. An overview of an artifact as represented in the context of a work area hierarchy.

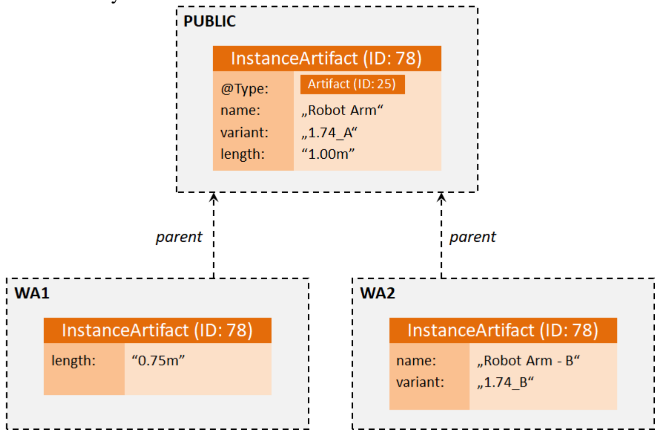
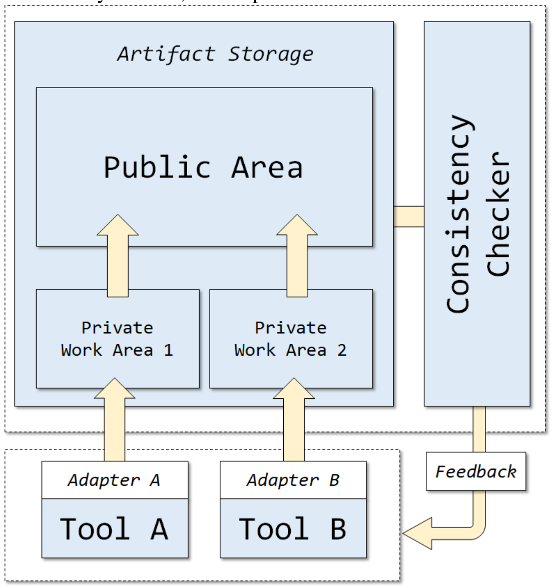


Fig. 3. Tool adapters synchronize artifacts with private work areas, from where they can be pushed to the public area. The artifact storage is observed by the consistency checker, which provides feedback to the tools.



1) Data Model: To realize the consistency checking service, our approach fully utilizes the uniform data representation by creating its own artifact types as well as the corresponding instantiations for them. This first and foremost comes with the advantage that the service handles its own documentation as it stores its results - which in themselves are engineering artifacts again - alongside other engineering artifacts. It references the concrete values that lead to the stored results and offers information on the involved artifacts.

For the storage of its own internally used data, the consistency checker creates two different engineering artifact types:

- **Consistency Rule Definiton Artifact:** A Consistency Rule Definition Artifact defines a certain rule for a single, specific artifact type. It contains a name, a description, a reference to the specific type artifact and the concrete rule in a simple string, each of which are provided by an engineer. Every time an instance of the type is created, the consistency checker automatically realizes the definition in the form of a Consistency Rule Evaluation Artifact.
- **Consistency Rule Evaluation Artifact:** A Consistency Rule Evaluation Artifact realizes a consistency rule definition for the instantiation of a certain artifact type. For example, if an engineer creates a Consistency Rule Definition Artifact for a requirements artifact type, as seen in Figure 1, every instantiation of the requirement type (ID:35 and ID:36) will receive its very own Con-

sistency Rule Evaluation Artifact. This artifact contains a reference to its definiton, a reference to the artifact type instantiation (the so called context element), the latest consistency checking result of this Evaluation Artifact and a so called scope, which keeps track of all properties involved in the computation of the result.

Both of these artifact types, respectively their instantiations are utilized during the consistency rule evaluation process. Within this process the Consistency Rule Definition Artifacts have an organizational role, while the Consistency Rule Evaluation Artifacts are concerned with documenting the results. The consistency rules themselves are stored as a string that is parsed and executed during the evaluation. These strings have the following structure:

$$Conjunction(Operator(ExpressionA, ExpressionB)^{1...*})$$

Expression pairs are compared via operations. Many operations can be connected into a logical conjunction, formalizing the full rule. The expressions themselves are held in an OCL-like language[3], which describes the navigation path from a context element towards a certain property value, based on the uniform artifact representation. Consider again the structure of the requirements artifact in Figure 1 and the following expression:

$$self.@type.TypeName$$

This expression, for example, would retrieve the name of the context elment's type (the context element is always the first point of reference within a rule and refered to as "self"). The result sets of two of such expressions would be compared via the enclosing operation. In our approach we implemented regular operations (equals, greater than, etc) for both sets and single values. A set of values may be retrieved when the cardinality of a property is defined accordingly in the artifact type's field definitions.

2) Rule Evaluation: Our consistency checking approach reacts immediately towards changes synchronized within the cloud environment's artifact storage. In the following we discuss every step from the fired change event to the concrete consistency state results.

- **Change Analysis:** When engineers perform a change within their tools, the corresponding tool adapter transforms the change into a form that can be incrementally added onto the uniform data representation. This change is then uploaded to the cloud environment and stored within the artifact storage on the respective artifact's property. Each change fires a change notification event, which can be listened to by the cloud's live services. The change notification event contains information on the identifier of the corresponding artifact, the name of the changed property, the nature of the change (Creation, Update, Deletion), as well as the new value of the property. This way, the consistency checker receives all required changes performed on the synchronized artifacts and can immediately react by computing a new consistency state. Naturally, not every single change is relevant, nor would it be performant to re-evaluate with every change notification event. Therefore, the consistency checker only reacts towards three types of change events:
  a) *Creation of an artifact:* When an engineer creates a new artifact in the cloud environment, the consistency checker must first analyze whether the type of the created artifact is referenced in an existing Consistency Rule Definition Artifact. If that is the case, the service automatically creates the corresponding Consistency Rule Evaluation Artifact and sets the context element.
  b) *Update of a scoped property:* When an engineer changes an existing artifact, the cloud environment stores this change on the respective property and fires a corresponding change notification event. To minimize the number of consistency rule re-evaluations, the consistency checker only reacts towards changes on properties that are part of a scope. This is possible, since a scope (which is built during the first re-evaluation of a Consistency Rule Evaluation Artifact) references all properties that are relevant for a consistency rule. Naturally, when one of these scoped properties changes, the Consistency Rule Evaluation Artifact containing the scope must be retrieved.
  c) *Deletion of an artifact:* When an engineer deletes an artifact, the cloud fires a corresponding change notification. The consistency checker then analyzes whether the deleted artifact was a context element or referenced through a scoped property. If the earlier is the case, the corresponding Consistency Rule Evaluation Artifacts are removed as well. If the latter is the case, the corresponding Consistency Rule Evaluation Artifacts' results are set to "invalid" until a change on a related scope element (referencing the deleted artifact) triggers the re-evaluation process again.
  The created, respectively retrieved Consistency Rule Evaluation Artifacts are marked down for "Data Gathering". The following steps must be executed for each Consistency Rule Evaluation Artifact individually.
- **Data Gathering:** For the data gathering process, the consistency checker first retrieves the stored consistency rule from the Consistency Rule Definition Artifact referenced in the Consistency Rule Evaluation Artifact. The said rule must then be parsed. Each step within the rule represents a navigation step through the artifact structure, starting at the context element. Each part of an expression refers to a certain property on a certain artifact. The navigated artifacts, respectively their properties are stored within the scope. Once a full expression has been navigated through, the final results (the property values of the last step) are marked down for the "Rule Evaluation" process. This is done for every expression within the rule.
  It should be noted, that when work area groups exist within the artifact storage, this step is slightly different. If a property can not be found within a work area, it is first

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

looked for within the group. If more than one result can be found (i.e., if more than one work area has changed the property) these are compared via their timestamp. The latest version of the respective property is retrieved. If the property can not be found in the group, the consistency checker falls back on the work area's parent hierarchy and eventually the Public Work Area.

- **Rule Evaluation:** For the concrete rule evaluation process, the gathered property values are substituted for the consistency rules expressions and the encapsulating operations are executed. The resulting boolean values are connected in the overall logical conjunction. The boolean result of this conjunction denotes whether the consistency rule holds or not.
- **Storing Results:** Once a rule evaluation result has been computed, the said result must be stored within the artifact storage for the documentation of consistency states. In our approach, the result is stored as a simple change on the according property of the corresponding Consistency Rule Evaluation Artifact. This change is of course only stored within the context work area that triggered the original rule re-evaluation. Once the change is pushed to the public work area, the latest result is simply added on the corresponding artifact's result property with a new timestamp. With the help of timestamps the history of consistency states is stored within this property.
- **Providing Feedback:** Once the result is added to the Consistency Rule Evaluation Artifact a new change notification event is fired within the cloud environment. This change event can be used by further live services that provide detailed feedback. Such feedback can then be provided to the engineers by contacting the tool adapter plugin bound to the work area, for which the latest re-evaluation was conducted. Alternatively tool adapters can simply listen to change notification events based on the Consistency Rule Evaluation Artifact type. This way, engineers will immediately be notified if their work area, respectively the changes therein, are in a new consistency state. This information can then of course be used for further visualizations by the tools. For example, potential inconsistencies can be marked on the according engineering artifacts (e.g. classes within a UML diagram that do not correspond to a naming convention, formulated in a rule definition, could be colored red - the concrete realization of feedback is up to the tools themself).

V. Discussion

The application of multifaceted consistency checking on different engineering artifacts comes with several benefits. However, it also comes with the drawback that engineering artifacts must first be transformed into a syntactically equivalent form. This means additional effort for programmers - implementing tool adapters - as well as domain experts - consulting the programmers during the implementation process. We believe, that the benefits are well worth the additional effort, as the uniform data representation also allows engineers to perform arbitraty enhancements on their data, e.g., through the addition of properties (or referenced artifacts) holding meta-information. In the case of the consistency checker, such added information would be the consistency states of an engineering artifact, which add further to the documentation of an engineering project.

This flexible way of adding information helps us tackle the complexity of the growing number of engineering artifacts, as it also allows us to document the various interdependencies between artifacts in a single location. In our approach this is done via a specific cloud tool, which can be seen in Figure 4. Further the uniform artifact representation allows the sharing and merging of engineering artifacts in a fine-grained way that is not possible in most modern collaboration solutions. This simplifies the integration process of one engineers work with the work of other engineers.

The consistency checker, with its ability to check the consistency of private changes against public knowledge (as the earlier are projected on top of the latter), allows engineers to stay consistent with the work that has already been done and simplifies the eventual merging process. With the work area grouping mechanism this advantage can not only be drawn from public knowledge, but from knowledge about the private changes of other engineers as well. This way, engineers are notified about inconsistencies in their work, before a merging process even starts. This minimizes potential refactoring efforts and prevents late recognition of major conflicts in ongoing work.

Since the consistency checker, respectively its Consistency Rule Evaluation Artifacts, keeps track of the consistency state history of a certain engineering artifact, the consistency checker provides documentation for temporarily tolerated inconsistencies within a project, which may otherwise be forgotten and go unnoticed until the final product.

With an overview over the fully connected, navigatable artifact structure of an engineering project, the consistency checker can further help tracking down the origin of various errors, as it keeps track of all engineering artifacts that are involved in the computation of a consistency state. This helps engineers to avoid symptomatic inconsistency repairs and potential follow-up errors.

All in all, consistency checking in a cloud environment provides several highly advantageous aspects. We believe, these aspects outweigh the additional effort of writing custom tool adapters, as our approach can help us tackling some of the major issues arising through the constantly increasing complexity and number of engineering artifacts in a modern engineering project. Further, a range of tool adapters has already been implemented, 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]

[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/

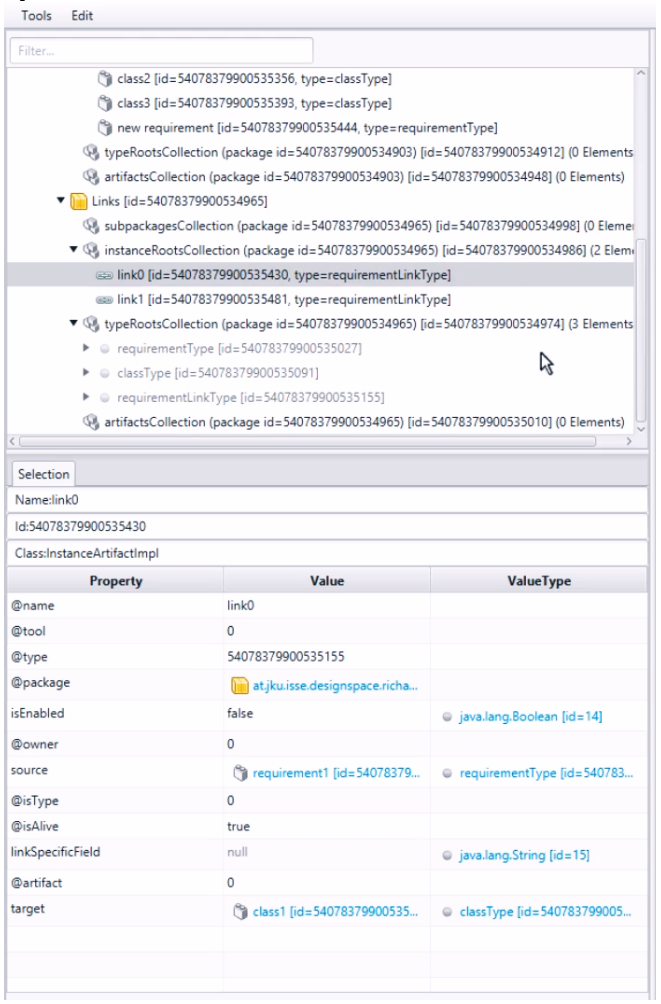


Fig. 4. An overview of the linking tool used in this approach. Both a part of the full artifact structure as well as a detailed view of a link artifact are depicted.

```
        class2 [id=54078379900535356, type=classType]
        class3 [id=54078379900535393, type=classType]
        new requirement [id=54078379900535444, type=requirementType]
      typeRootsCollection (package id=54078379900534903) [id=54078379900534912] (0 Elements
      artifactsCollection (package id=54078379900534903) [id=54078379900534948] (0 Elements)
  ▼ Links [id=54078379900534965]
      subpackagesCollection (package id=54078379900534965) [id=54078379900534998] (0 Eleme
    ▼ instanceRootsCollection (package id=54078379900534965) [id=54078379900534986] (2 Elem
        link0 [id=54078379900535430, type=requirementLinkType]
        link1 [id=54078379900535481, type=requirementLinkType]
    ▼ typeRootsCollection (package id=54078379900534965) [id=54078379900534974] (3 Elements
      ▸ requirementType [id=54078379900535027]
      ▸ classType [id=54078379900535091]
      ▸ requirementLinkType [id=54078379900535155]
      artifactsCollection (package id=54078379900534965) [id=54078379900535010] (0 Elements)
```

Selection

Name:link0

Id:54078379900535430

Class:InstanceArtifactImpl

Property	Value	ValueType
@name	link0	
@tool	0	
@type	54078379900535155	
@package	at.jku.isse.designspace.richa...	
isEnabled	false	java.lang.Boolean [id=14]
@owner	0	
source	requirement1 [id=54078379...	requirementType [id=540783...
@isType	0	
@isAlive	true	
linkSpecificField	null	java.lang.String [id=15]
@artifact	0	
target	class1 [id=54078379900535...	classType [id=540783799005...

(for electrical layouts), and others. Our approach has been evaluated in multiple ways. There has been an empirical study with regards to its computational feasibility [9]. Further, two case studies have been conducted to evaluate its applicability within different engineering projects ([10], [11]), as well as within an industrial environment [10].

VI. Related Work

While a multitude of consistency checking approaches exist in the current landscape of engineering solutions (e.g., [2]–[7]) most of them do not consider the core aspect of this work, which is the multifaceted application of the mechanism on a varied set of different engineering artifacts. Nevertheless, individual concepts of this work have been the focus of various research efforts in the past. Consistency checking itself, as well as the collaborative aspect of engineering, with the strong interdependency between various engineering artifacts in mind, have gotten attention in various works.

Some consistency checking approaches rely on the partial merging of metamodels to incorporate different model elements with each other. This way they can check consistency on the overlapping, respectively interdependent parts of engineering artifacts without performing a full integration. Koenig et al. [12] considered various engineering artifacts by incorporating individual model elements of heterogeneous multimodels. These elements were then compared on the basis of global constraints. Minimizing the set of compared elements significantly lowers the efforts required for the matching and merging process between different types of artifacts. The rest of the artifacts - which did not overlap with other artifacts - were checked as locally as possible. Likewise, Sabetzadeh et al. [13] proposed global consistency checking by the partial merging and the comparison of metamodels. For their approach they provided merging operations for requirements, behaviour models, design and implementation. Contrary to these two approaches, our approach transforms existing engineering artifacts into a different format, which can be arbitrarily linked. This is a very lightweight alternative to the sometimes extensive effort of model merging.

Collaborative approaches towards model-driven engineering have also been the focus of various works [14]–[20]). A mapping study conducted by Franzago et al. [21] documented available solutions for collaborative modeling, which showed that only very few approaches allow for the editing of model artifacts in an asynchronous way. A live capability for various applications, such as a consistency checker, is partly hindered by this fact. This issue is overcome by our application of a collaborative engineering cloud which directly integrates the consistency checker and forwards activity events towards it.

Another major way in which our approach deviates from related work is the fact that its live change-adaptive capabilities are based on live synchronization through a tool plugin. This - depending on the concrete plugin - means that our consistency checker is reacting live towards changes directly handled in the engineering tools, which are immediately propagated to the cloud environment. This way, the cloud always receives engineering artifacts directly from the internal data structure that is present within the tool itself, which is in contrast to most other approaches. In most consistency checking approaches the engineering artifacts are synchronized from a file- respectively document-base. One such approach was developed by Aldrich et al. [22]. In their approach architecture description language was coupled with the concrete implementation in a Java project. It utilizes a type system guaranteeing the integrity between architecture and code. Similarily Ubayashi et a. [23] secures the integrity between architectural design and code, by providing both a programming interface, as well as a description language for architectures. In their approach, architectural constraints within the implementation guarantee the traceability and integrity between model and code. An approach that also considers the aspect of artifact availability was suggested by Nentwich et al. [24]. In it, XML-based engineering artifacts are checked for consistency after links have been generated between them. The artifacts are web distributed, which is in contrast to our own approach which centralizes artifacts in a single storage place.

VII. Conclusion & Future Work

In this paper we discussed the various aspects of modern day collaborative engineering and the major problems that arise through the constantly growing complexity in engineering artifacts as well as with their number and required availability. To tackle these problems, this work introduced the novel way of multifaceted consistency checking within an engineering cloud environment. In contrast to most related work, our approach takes the diverse nature of strongly interdependent engineering artifacts into account, by synchronizing them into a single space and offering a way to represent their relationships. These relationships can then be exploited for consistency checking. Our approach further takes full advantage of the cloud environment's structural concepts. Thanks to this, the results of a consistency state computation are directly integrated with the rest of the engineering artifacts, which further adds to the documentation of an engineering project. The consistency checking mechanism itself strongly builds on the availability of various work areas within the cloud environment. It takes advantage of various context views on engineering artifacts. These views can be constructed by combining different work areas during the data gathering process of the consistency checker, e.g., in the form of a parent hierarchy, where the changes of one work area are projected on top of other changes – or in the form of work area groups, where the latest version of a certain property within a group is retrieved for the consistency check. We concluded this paper with a discussion of various benefits of our approach, as well as with a discussion of the related work.

For future work we would like to expand on the mechanism for distributing consistency feedback. Currently a lot of consistency feedback is produced, not all of which may be interesting to the engineers. Therefore, we would like to introduce a way of propagating only information that is relevant to the engineers' current focus.

VIII. Acknowledgements

This work is partially supported by the Austrian Science Fund (FWF): FWF P 31989-N31, the JKU Linz Institute of Technology (LIT), the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy and the Province of Upper Austria in the frame of the COMET center SCCH.

References

[1] A. Finkelstein, "A foolish consistency: Technical challenges in consistency management," in *Database and Expert Systems Applications* (M. Ibrahim, J. Küng, and N. Revell, eds.), pp. 1–5, Springer Berlin Heidelberg, 2000.

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

[3] P. Fradet, D. Le Métayer, and M. Périn, "Consistency checking for multiple view software architectures," in *Software Engineering - ESEC/FSE'99*, pp. 410–428, Springer, 1999.

[4] S. P. Reiss, "Incremental maintenance of software artifacts," *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 682–697, 2006.

[5] A. Reder and A. Egyed, "Model/analyzer: a tool for detecting, visualizing and fixing design errors in UML," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pp. 347–348, ACM, 2010.

[6] M. Vierhauser, P. Grünbacher, A. Egyed, R. Rabiser, and W. Heider, "Flexible and scalable consistency checking on product line variability models," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pp. 63–72, ACM, 2010.

[7] M. Riedl-Ehrenleitner, A. Demuth, and A. Egyed, "Towards model-and-code consistency checking," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, pp. 85–90, IEEE, 2014.

[8] M. A. Tröls, A. Mashkoor, and A. Egyed, "Collaboratively enhanced consistency checking in a cloud-based engineering environment," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2019, Valencia, Spain, June 18-21, 2019*, pp. 15:1–15:6, 2019.

[9] M. A. Tröls, A. Mashkoor, and A. Egyed, "Live and global consistency checking in a collaborative engineering environment," in *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)*, pp. 1762 – 1771, ACM, 2019.

[10] A. Demuth, R. Kretschmer, A. Egyed, and D. Maes, "Introducing traceability and consistency checking for change impact analysis across engineering tools in an automation solution company: an experience report," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 529–538, IEEE, 2016.

[11] A. Egyed, K. Zeman, P. Hehenberger, and A. Demuth, "Maintaining consistency across engineering artifacts," *Computer*, vol. 51, no. 2, pp. 28–35, 2018.

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

[13] M. Sabetzadeh, S. Nejati, S. Easterbrook, and M. Chechik, "Global consistency checking of distributed models with TReMer+," in *2008 ACM/IEEE 30th International Conference on Software Engineering*, pp. 815–818, IEEE, 2008.

[14] A. García Frey, J.-S. Sottet, and A. Vagner, "Ame: an adaptive modelling environment as a collaborative modelling tool," in *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*, pp. 189–192, ACM, 2014.

[15] S. Krusche and B. Bruegge, "Model-based real-time synchronization," in *International Workshop on Comparison and Versioning of Software Models (CVSM14)*, 2014.

[16] H.-m. Cai, X.-f. Ji, and F.-l. Bu, "Research of consistency maintenance mechanism in real-time collaborative multi-view business modeling," *Journal of Shanghai Jiaotong University (Science)*, vol. 20, no. 1, pp. 86–92, 2015.

[17] M. Franzago, H. Muccini, and I. Malavolta, "Towards a collaborative framework for the design and development of data-intensive mobile applications," in *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, pp. 58–61, ACM, 2014.

[18] D. Wüest, N. Seyff, and M. Glinz, "Flexisketchteam: collaborative sketching and notation creation on the fly," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pp. 685–688, IEEE Press, 2015.

[19] C. D. Francescomarino, C. Ghidini, M. Rospocher, L. Serafini, and P. Tonella, "A framework for the collaborative specification of semantically annotated business processes," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 4, pp. 261–295, 2011.

[20] J. Gallardo, C. Bravo, and M. A. Redondo, "A model-driven development method for collaborative modeling tools," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 1086–1105, 2012.

[21] M. Franzago, D. Di Ruscio, I. Malavolta, and H. Muccini, "Collaborative model-driven software engineering: a classification framework and a research map," *IEEE Transactions on Software Engineering*, vol. 44, no. 12, pp. 1146–1175, 2018.

[22] J. Aldrich, C. Chambers, and D. Notkin, "Architectural reasoning archjava," *ECOOP*, pp. 334–367, 2002.

[23] N. Ubayashi, J. Nomura, and T. Tamai, "Archface: A contract place where architectural design and code meet together," *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, pp. 75–84, 01 2010.

[24] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelsteiin, "xlinkit: A consistency checking and smart link generation service," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 151–185, 2002.