Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

SPECIAL ISSUES PAPER

Ensuring Safe and Consistent Co-Engineering of Cyber-Physical Production Systems: A Case Study

Michael Alexander Tröls[1] | Atif Mashkoor[1,2] | Andreas Demuth[3] | Alexander Egyed[1]

[1]Johannes Kepler University, Linz, Austria
[2]Software Competence Center Hagenberg GmbH, Hagenberg, Austria
[3]Dynatrace Austria GmbH, Linz, Austria

Summary

In today's engineering projects, companies continuously have to adapt their systems to changing customers or dynamic market requirements. This requires a flexible, iterative development process in which different parts of the system under construction are built and updated concurrently. However, concurrent engineering becomes quite challenging in domains where different engineering artifacts from different disciplines come into play, such as safety-critical cyber-physical systems, where the involved engineering artifacts are quite heterogeneous in nature. In such systems, it is of utmost importance that different artifacts remain consistent in order to guarantee a correctly functioning end product. In this article, we discuss our experiences (with a leading company working in the areas of production automation and product processing) in maintaining the consistency between electrical models and the corresponding software controller, when both are subject to continuous changes. The article discusses how we let engineers describe the relationships between electrical models and the corresponding software controller code in the form of links and consistency rules. Additionally, we demonstrate that how our approach, through a process of continuous consistency checking, notifies engineers about the erroneous impact of their changes in various engineering artifacts.

KEYWORDS:
Traceability, Incremental Consistency Checking, Model-driven Engineering, Software Evolution, Safety Analysis

1 | INTRODUCTION

Modern engineering projects often span across multiple domains and disciplines. It has become common that engineers with diverse backgrounds collaborate in order to develop complex systems. For example, developing a cyber-physical system typically requires software engineers, hardware engineers, electrical engineers, requirements engineers, mathematicians, physicists, and others to work together closely. Typically experts from these different domains are responsible for different engineering artifacts. For instance, the requirements the system under development must fulfill may be negotiated with stakeholders by the requirements engineer. The requirements may then be used by a mathematician to calculate data that must be known to realize the project. This data may then, in turn, be used by an electrical engineer to create a model of the hardware setup. Based on this hardware setup and other calculated data, a software engineer may, finally, develop the source code that controls the hardware components.

During the initial development of the aforementioned artifacts, there is typically communication between experts from different domains. For example, once the hardware design is finished, the hardware engineer informs the software engineer about which hardware components will be used and thus have to be controlled by the control software. Each engineer will also quite likely inform the corresponding supervisor about a

finished artifact. Therefore, during the initial development, knowledge is typically passed on quite well between involved engineers and different artifacts are likely to be consistent with respect to each other. However, such communication often happens informally, and the knowledge passed during this communication might not be documented. This becomes a problem especially when an artifact is evolved past its initial stage, for example, in an iterative development process individual artifacts are evolved multiple times until they reach their final state when the project is finished. In this case, the rationale for certain decisions is often no longer available for engineers. For instance, it might no longer be clear why certain source code fragments are needed for implementing the controller of a piece of hardware. If an engineer removes a component from the hardware model, software engineers might have a hard time locating the corresponding source code fragments. As a consequence, changes because of evolution or maintenance may easily introduce inconsistencies between different artifacts and the impact of a change is hard to estimate. This is especially challenging in cyber-physical systems due to the highly interwoven nature of their engineering artifacts. The development of such systems commonly involves engineers from highly diverse backgrounds. The more diverse the engineering disciplines in a development project are, the more difficult it becomes to maintain consistency between artifacts throughout multiple development phases.

The maintenance of consistency is critical for the safety aspect of cyber-physical systems. Since safety plays a major role in modern cyber-physical systems [1,2,3], safety analysis must accurately capture the potential harm a system may cause to its operators. With inconsistencies between engineering artifacts and the propagation of arising errors, the results of such analysis may be distorted. From a safety perspective it is therefore highly important to guarantee both a reliable and comprehensive analysis of artifact consistency.

These are exactly the issues that the Belgian company Van Hoecke Automation[1] (VHA) – a leading provider of automation solutions – is facing. In this article, we present the results of a collaboration between VHA, Flanders Make[2], and Johannes Kepler University[3] (JKU) that addresses these issues. In particular, we describe a solution that enables traceability, consistency checking, and change impact analysis between different artifacts in the VHA's development process. The solution relies on existing principles of traceability and incremental consistency checking (e.g., the *DesignSpace Information Integration Platform* [4] and the *Model/Analyzer Consistency Checking Framework* [5,6]). Moreover, new technologies for establishing traceability and the application of consistency checking to different artifacts have been developed.

This article is an extension of our previous work [7]. The extension includes a broad update of definitions and concepts according to the latest implementation standards of the presented solution. We also report improvements on the principles of engineering artifact integration, the adaption of types in the integration platform, the organization of data within the artifact storage as well as the inner workings of the consistency checking mechanism.

The remainder of the paper is organized as follows. In Section 2, we give background information about Van Hoecke Automation and the Flanders' Mechatronics Technology Centre. In Section 3, we present a challenge problem that Van Hoecke Automation actually has been facing. In Section 4, we outline the specific goals and requirements of Van Hoecke Automation regarding a better support of artifact evolution and maintenance. Section 5 then describes the solution that has been developed to address the challenge problem and to meet the stated goals and requirements. Section 6 then shows how the developed solution is applied to the challenge problem. In Section 7, we discuss how the developed solution actually meets the goals and requirements. Moreover, in this section we discuss aspects such as expected applicability and benefits of employing the solution, and the challenges that we encountered during the development. Section 8 discusses the related work. The paper is concluded in Section 9.

2 | BACKGROUND

Van Hoecke Automation is a leading Belgian Family company, founded in 1990, providing advanced solutions for production automation, product processing, and product inspection and tracking. Their main sectors of operation include the make industry with an important international activity in the automotive and OEM sector, and the food industry. Key competences of VHA regarding the discipline of software engineering include the development of production supervisory systems by means of standard tools and the development of customer specific machine applications. As it is typical for companies dealing with cyber-physical systems and production automation, VHA employs experts from diverse domains such as business management, software engineering, mechanical engineering, electrical engineering, mathematics, or physics.

Flanders Make is the strategic research centre for the manufacturing industry with establishments in Lommel and Leuven, Belgium. The centre collaborates with research labs at various Flemish universities as well as with major companies and SMEs. Together, they focus on product and process innovation based on the challenges and needs of the industry. The research focus is on 4 technological domains: power electronics and energy storage, mechatronics and design methods, production processes, and people-driven system development. The primary goal of the collaboration is to yield product and process innovations in 3 fields of application: vehicles, machines, and factories.

[1] https://www.vha.be/ (last visited: 25.06.2020)
[2] https://www.flandersmake.be/ (last visited: 25.06.2020)
[3] https://www.jku.at/ (last visited: 25.06.2020)

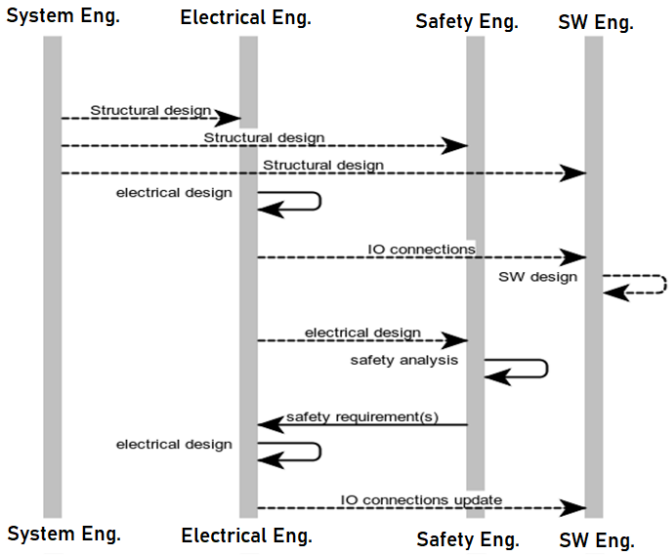


FIGURE 1 Incremental development of the conveyor belt system

In this project, VHA and Flanders Make collaborated with JKU to combine their experience in the development of mechatronical systems with leading research on software traceability and consistency checking. The results are meant to not only be beneficial for the industrial partners with respect to their development process, but also to serve as a case study for the application of research technologies from academia in an industrial context.

3 | PROBLEM STATEMENT

VHA produces conveyor belts which are powered by asynchronous motors. The motors are controlled by a programmable logic controller (PLC). The design of the conveyor belt involves four different engineers: i) a systems engineer, ii) an electrical engineer, iii) a safety engineer, and iv) a software engineer. The process is depicted in Fig. 1. Each engineer is using specific tooling during the following individual steps of the scenario:

1) [System concept] The systems engineer creates the specification of the initial system concept (as shown in Fig. 2(a)).

2) [Electrical design] Eplan Electric P8[4] is used by the electrical engineer to specify the electrical connections between actuators (e.g., a motor) and the PLC (a high-level view of the electrical design is shown in Fig. 2(b), a snippet from the actual EPlan P8 model is shown in Fig. 2(c)). The EPlan P8 model snippet depicts the control signal of the conveyor motor as the component `21K3` and the normal output of the Siemens PLC as the component `Q70.0`.

3) [Software design] In the example, the Eclipse IDE with Java is used as a proxy for the regular PLC code in which the control logic of the actuators are implemented (as shown in Fig. 2(d)). Notice that for the two IO ports of the Siemens PLC, there are two variables defined: i) `hw_IO_out0_1` representing the normal output (`Q70.0` in Fig. 2(c)), and ii) `hwIO_out1_1` representing the safety output of the Siemens PLC (not depicted in Fig. 2(c)). Moreover, the variable `fb_motor_control` represents the control signal of the motor conveyor (`21K3` in Fig. 2(c)). In the last line of Fig. 2(d), the linking of the components `Q70.0` and `21K3` from Fig. 2(c) is reflected.

4) [Safety analysis] Microsoft Excel is used by the safety engineer for the analysis of potential harms to (human) operators caused while operating the system, using a template specifically designed at VHA (as shown in Fig. 2(e)).

5) [Electrical design update] The electrical engineer performs updates of the electrical connections to avoid the harms identified in the previous step (as shown in the high-level view in Fig. 2(f) and the actual EPlan P8 model snippet in Fig. 2(g)). Notice that in Fig. 2(g) the control signal (component `21K3`) is now connected to a different component (`Q30.1`), which represents the safety output of the Siemens PLC.

6) [Software design update] The software engineer performs updates of the software so that it is in sync with the updated electrical connections (as shown in Fig. 2(h)). Notice that in the mapping of IO ports, the control signal (`fb_motor_control`) is now linked to the safety output of the Siemens PLC (`hwIO_out1_1`).

[4]Eplan Electric P8: https://www.eplanusa.com/us/solutions/product-overview/eplan-electric-p8/ (last visited: 25.03.2020)

Note that after Step 5, the update of the electrical design, there is an inconsistency between the electrical design and the control software, and therefore an update of the software design is required. Specifically, in the EPlan P8 model snippet in Fig. 2(g) it can be seen that the component `21K3` is no longer connected to the component `Q70.0` (as in Fig. 2(c)), but instead it is now connected to the safety output of the Siemens PLC, which is modeled as the component `Q30.1`. The source code from Fig. 2(d) still reflects that `21K3` and `Q70.0` are connected. However, VHA has experienced that in such a scenario the inconsistency may be overlooked by engineers and may thus remain undetected, resulting in Step 6 being not performed in practice. This might be caused by a lack of support for traceability, change impact analysis, and inconsistency detection. The consequences of not detecting the inconsistency and not performing Step 6 in the scenario include, for example, delays in the development progress or failure to deliver a functioning system. Therefore, it is crucial for VHA to introduce support for traceability, change impact analysis, and inconsistency detection in their development process.

4 | GOALS AND REQUIREMENTS

Let us now summarize the goals and specific solution requirements stated by VHA. The main goals of the project presented in this article are:

- to increase awareness about consistency between spreadsheet data, source code, and electrical models,
- to increase traceability between these engineering artifacts, and
- to get feedback about consistency and change impact.

A key requirement stated by VHA is the support for the following technologies: Microsoft Excel Spreadsheets and Eplan Electric P8. Additionally, in the example, the Eclipse IDE with Java is used as a proxy for the regular PLC code development. Engineers using either tool must be notified about arising inconsistencies. Moreover, it must be possible to analyse the impact of a change and to track dependencies between different artifacts. Additionally, the current workflows of the individual disciplines should not be disrupted significantly (i.e., the specialized engineering tools used currently must not be replaced). As we have already illustrated VHA's challenge problem and discussed the goals and requirements they stated, now we present the solution that was developed in the project.

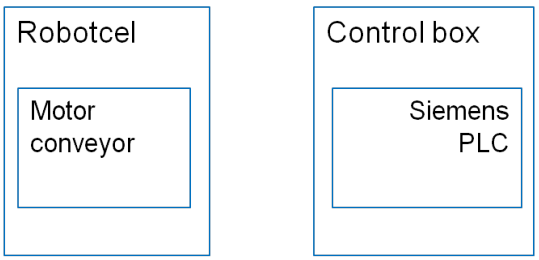
5 | APPROACH

One concern about the realization of the outlined goals was the potential interference with the established development process of VHA. Therefore, we decided to apply a solution based on existing approaches. In the following, we first discuss a general overview of our solution's architecture and then describe its individual parts.

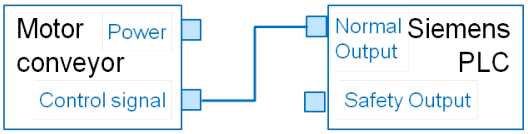
As discussed above, incremental consistency checking and traceability between different artifacts are key features our solution must provide. We abstract these concepts in a way that they are adaptable on a heterogeneous engineering artifact basis. The original engineering artifacts (e.g., Excel files) should remain unchanged. Therefore, they are duplicated in the employed data integration platform. In this work, we exploit the DesignSpace cloud environment[4]. A service running on top of the DesignSpace platform is used for consistency checking, which has been discussed in previous works (e.g., [8,9,10]). Further, we apply another service enabling traceability between different existing artifacts. We first discuss how artifact integration works in our solution, then we present how traceability can be established using the integrated data. We finally show how consistency checking is realized on the integrated data and existing traceability information.

5.1 | Artifact Integration

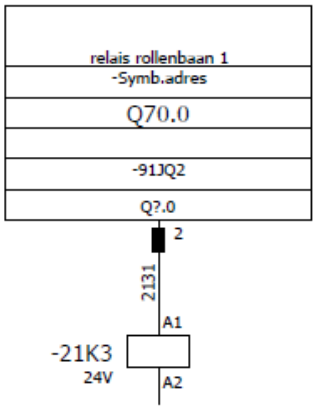
As discussed above, the DesignSpace integration platform is used in this project. This platform provides support for the transparent integration of arbitrary engineering artifacts. In this work, all engineering artifacts remain unchanged at their original location (e.g., on an engineer's local hard drive) and the DesignSpace only holds a replication of the original information that is synchronized live. Thus, artifact integration with the DesignSpace does not affect existing processes and practices. Engineers are still able to use their favorite engineering tools for developing their artifacts. This is important for two reasons. First, engineers are typically reluctant to change the used tools, which is understandable because engineering tools are typically highly specialized for performing certain tasks as efficient as possible. Thus, trying to replace such highly specialized tools is typically a non-optimal solution with respect to the quality and the efficiency of results. Second, developing a new engineering tool for a specific domain requires years of development, with only small chances of coming up with a tool that leads to better — or even similar — results than the leading existing tools for the domain.



(a) Initial system concept from Step 1.



(b) Initial electrical design from Step 2.



(c) Initial EPlan P8 model from Step 2.

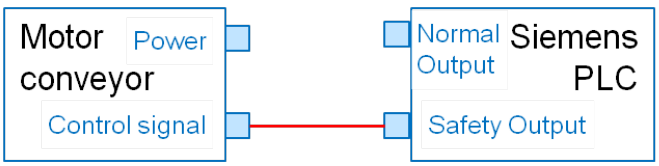
```
//Q70.0
int hwIO_out0_1 = 0xc001;
//Q30.1
int hwIO_out1_1 = 0xc001;
//21K3
int fb_motor_control = 0xc002;

//linking
IO_mapping.put(hwIO_out0_1, fb_motor_control);
```

(d) Initial software design from Step 3.

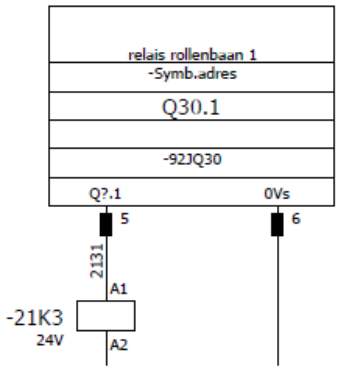
Danger	Likelihood	frequency	…	Corrections
Pinched in between conveyor belt	Probably	Daily	…	fail safe on conveyor motor

(e) Initial safety analysis from Step 4.

	Required SRP/CS level			
Safety Functions	S	F	P	SILr
Fail safe conveyor motor	S2	F1	P2	2



(f) Updated electrical design from Step 5.



(g) Updated EPlan P8 model from Step 5.

```
//Q70.0
int hwIO_out0_1 = 0xc001;
//Q30.1
int hwIO_out1_1 = 0xc001;
//21K3
int fb_motor_control = 0xc002;

//linking
IO_mapping.put(hwIO_out1_1, fb_motor_control);
```

(h) Updated software design from Step 6.

FIGURE 2 Engineering artifacts of a conveyor belt system. Initial versions (a)–(e) and updated versions (f)–(h).

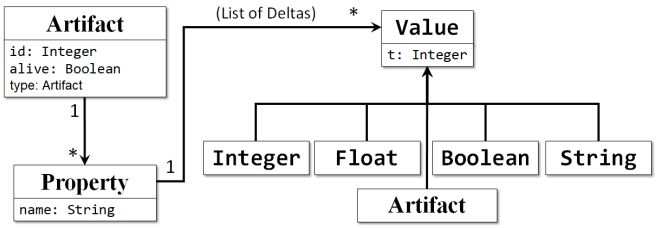


FIGURE 3 The uniform artifact representation employed in the DesignSpace

In the DesignSpace, engineering artifacts are represented in the form of property-value mappings. An artifact consists of several (named) properties, which may point toward basic value types (integer, float, boolean, string) or yet more artifacts (references). This data structure is illustrated in Fig. 3. It is of importance to note at this point, that a value, basic or referential, is not merely a single value, but a list of deltas, that describe the change history of the property. Nothing in the DesignSpace is ever truly removed, but merely overwritten or flagged as "not alive". It is saved for every artifact, on which an engineer performed the last modification along with a timestamp when this modification happened. Further, every artifact is uniquely identifiable with an ID and adheres to a specific user-defined type.

The type of an artifact defines the structural components of its concrete instantiation. This means, an artifact type describes properties, i.e., property names, cardinalities (whether a value is a list or a single value), and the datatype. When an engineering artifact is synchronized with the DesignSpace, it automatically instantiates the corresponding type and fills the properties accordingly.

To synchronize an engineering artifact with the DesignSpace, we may employ one of two principles:

- **File-Adapters**: One option is to use file-adapters that translate file contents into the DesignSpace's internal data structure. With file-based integration, the synchronization of the original engineering artifact and its duplicate in the DesignSpace can only be done when the file is saved, requiring the use of diffing tools or inefficient overrides of existing data.
- **Tool-Adapters**: The second option is to use tool-adapters that directly translate and synchronize information from engineering tools with the DesignSpace. Note that this can be done live and incrementally while engineers work on their respective engineering artifacts. Moreover, by using tool-adapters it is possible to present information about consistency or trace information to the engineer directly in the engineering tool. Therefore, the use of tool-adapters is generally preferable.

For each of the three tools that are used by VHA in their challenge problem, we developed three different tool adapters, which we present next.

5.1.1 | Microsoft Excel

For Microsoft Excel, data integration is performed at cell level through an Excel plugin written in C#. We decided not to duplicate all cells in a spreadsheet with the DesignSpace, but only those cells that are of relevance for other engineering artifacts. This decision is based on the fact that, due to the nature of spreadsheets, a vast majority of existing cells may be either empty or contain information that is required only for some internal calculations but does not have any effect on other artifacts. Instead, those cells in the spreadsheet that are of relevance and should therefore be synchronized with the DesignSpace for later use by the traceability and consistency checking services are annotated by an engineer. This avoids unnecessary communication overhead and allows for easier establishing of traceability, as we will see below. The annotations are added to each cell individually using Excel's comment function. To distinguish comments for data integration from others, the data integration comments must conform to the following syntax: `$name=<name> $unit=<unit> $meta=<meta information>`. The `$name` attribute allows engineers to specify a name by which the cell can later be referenced for traceability and consistency checking. The `$unit` attribute defines the unit of the value saved in the annotated cell. The attribute `$meta` allows engineers to specify additional meta information, for example, if a cell is representing some domain concepts. As an example, consider in Fig. 2(e) the cell with the value `Probably` (2nd row, 2nd column). Using the `$meta` attribute, an engineer can define that this cell's value represents a likelihood. Note that another option would have been to use predefined templates that define which cells should be synchronized. However, using annotations provides more flexibility to engineers in case templates are changed.

5.1.2 | Eclipse IDE with Java

For the source code development, in this project we relied on the Eclipse IDE. Therefore, we developed a tool-adapter plugin for the Eclipse IDE and the Java programming language. Notice again that in practice PLCs are not programmed using Java. However, in this project we agreed with

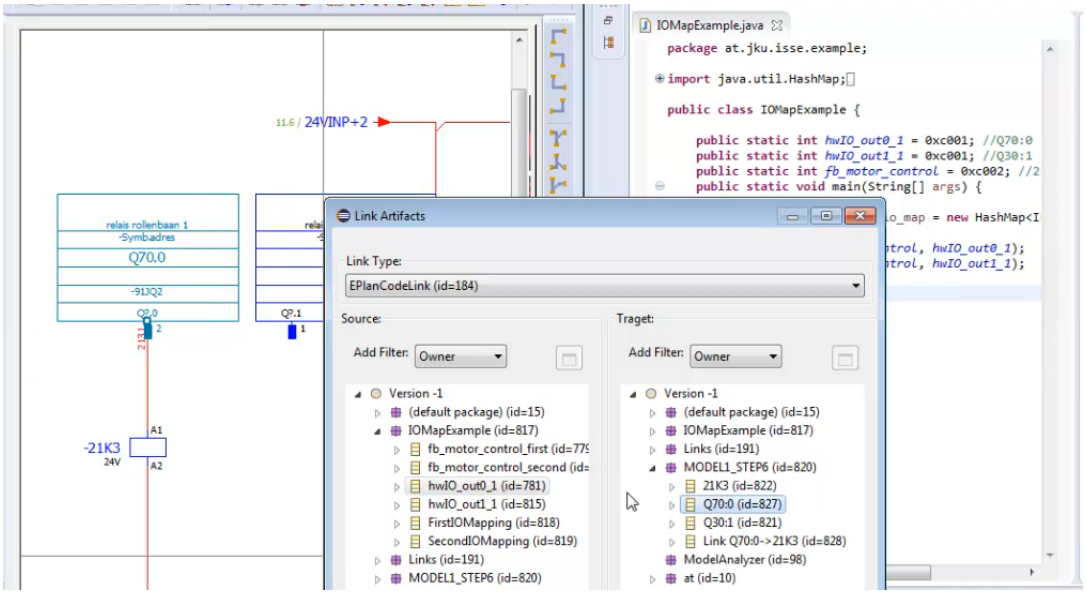


FIGURE 4 Wizard for creating a link between EPlan model element and its corresponding source code fragment.

VHA to use Java for demonstrating the feasibility of checking consistency between EPlan P8 models and the source code. VHA is confident that if consistency checking is feasible with Java, it is also feasible with an actual PLC programming language. Similarly to the tool-adapter for Microsoft Excel, this adapter does not synchronize the entire source code with the DesignSpace platform, but only those parts of it that are relevant for other artifacts. As for excel, this reduces the amount of information that is available in the DesignSpace for establishing traceability. Engineers can mark relevant parts by adding annotations as comments. Note that such annotations can be made for any element of a program (e.g., classes, methods and functions, and individual statements). The synchronization with the DesignSpace is not performed immediately when an annotation is added, but it is performed each time the Eclipse IDE's compiler is executed, which is typically after each save triggered by the user.

5.1.3 | EPlan Electric P8

For electrical models created with EPlan Electric P8, we followed a similar approach as for Microsoft Excel and the Eclipse IDE with Java. A tool-adapter is used that observes the electrical model. Any model elements that should be synchronized with the DesignSpace have to be marked by adding a custom property to the element (e.g., to a modeled component). However, in EPlan it is not possible to select connections between elements and set this property. Therefore, the tool-adapter also synchronizes all connection elements that connect two (or more) already marked (and therefore synchronized) elements. For instance, if a motor element is connected to a controller element, then the connection between the motor and the controller is also synchronized with the DesignSpace. An example of this behavior is shown in Fig. 4. Specifically, the left-hand side of the figure shows three model elements that are synchronized with the DesignSpace: `Q70.0`, `Q30.1` (only visible partially to the right of `Q70.0`), and `21K3`. The corresponding DesignSpace representation of the EPlan model is shown on the right-hand side of the linking wizard in the center of Fig. 4. Notice that there are four entries: one for each of the marked model elements, and one additional representation of the link between `Q70.0` and `21K3`. This link has been created in the DesignSpace automatically.

5.1.4 | Integrated Type Information of Engineering Artifacts

As discussed above, the DesignSpace platform internally uses a simple, graph-like data structure that represents any information as artifacts and properties. Alongside this graph of artifacts, the DesignSpace also stores corresponding type information. This type information describes the structure of engineering artifacts as they are represented on the DesignSpace.

The type information can be drawn from corresponding metamodels (e.g., the EPlan metamodel, which describes the structure of EPlan models). Since not all engineering artifacts have defined metamodels, type information can also be created from scratch by a domain expert. While the engineering artifacts are modeled as logical instantiations of their types, the representations of both instantiations and their types, are runtime elements of the DesignSpace's architecture. Therefore, it must be distinguished between linguistic and ontological levels. Linguistic levels define technical instantiation at runtime, whereas ontological levels describe logical instantiation.

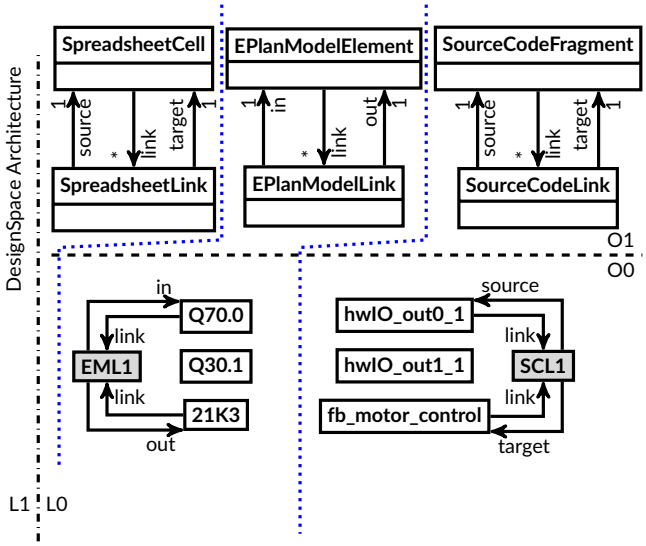


FIGURE 5 Engineering artifacts and their types integrated in the DesignSpace.

In Fig. 5, the linguistic and ontological levels are depicted. Notice that we use the same notation for representing the individual engineering artifacts at the ontological level `O0` and their respective types at the ontological level `O1`. This indicates that both levels are runtime instances of the linguistic level `L1`. The distinction between the two ontological levels is purely virtual and based on interpretation of the runtime data structures. This means, that both instantiations and types are based on the same abstract entity within the data structure (i.e., everything is treated as an artifact - even type information). For each integrated engineering artifact its corresponding type remains unchanged in principle – the type may only be extended with additional information but the original concepts are retained. This means that any such issue would be avoided that typically arise when trying to merge different metamodels into a single one [11,12]. In this project, we used metamodels to create the corresponding typing information. However, the typing information of synchronized engineering artifacts has been simplified. This was not only done to simplify the development of tool adapters but also to reduce the complexity of linking and consistency checking from the perspective of the involved engineers. Specifically, the typing information of EPlan models contains only two entities: `EPlanModelElement` and `EPlanModelLink`. The former is used to represent any first-class entities of EPLan models (e.g., components), whereas the latter is used for representing any links that are drawn between first-class entities (e.g., when an input of a component is linked to another component's output). For the source code, the typing information contains also two classes: `SourceCodeFragment` and `SourceCodeLink`. A `SourceCodeFragment` can be any part of source code, e.g., a class, a method, an individual statement, or even a simple comment. A `SourceCodeLink` is used to model linking and mapping of the source code fragments. For instance, a `SourceCodeLink` between two variables is used to model that the two variables are used as a key-value pair in a map. For Microsoft Excel spreadsheets, the tool adapter only synchronizes individual cells. Therefore, a single typing element, `SpreadsheetCell` is sufficient for this project.

At the ontological level `O0` in Fig. 5, the state of the DesignSpace after step 4 from the challenge problem in Section 3 is shown. Note that the relevant elements from each of the three involved engineering artifacts have been synchronized with the DesignSpace automatically by tool-adapters after they have been annotated by engineers in their respective engineering tools. For the source code, the three variable and the IO-port mapping from Fig. 2(d) are represented. For the EPlan P8 model, the three components that model the control signal of the motor conveyor and the two outputs of the Siemens PLC are represented. The links between the elements, which have been generated automatically by the respective tool adapters, are drawn as boxes with grey background. For the Excel sheet from Fig. 2(e), there are no annotated cells, meaning that nothing has to be synchronized with the DesignSpace.

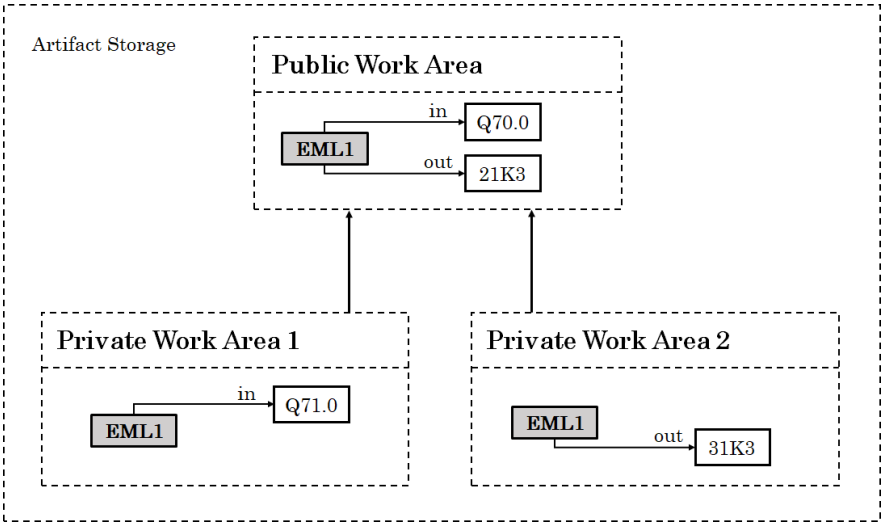


FIGURE 6 A layered hierarchy of work areas.

5.2 | Work Areas

Once synchronized, the engineering artifacts are stored in the DesignSpace's artifact storage. The artifact storage consists of both a public area (PA) and a set of private work areas (PWAs). The PA contains all information on artifacts that have been synchronized and published by the engineers. The PWAs contain only deltas (changes) of artifacts with regards to their publicly available version. The work areas are arranged in a tree structure, where the PA is the root. PWAs are attached as leaves below the root. Retrieving engineering artifacts from a PWA layers the work area's deltas on top of the respective property values in its parent. This way an artifact can always be retrieved from a certain engineer's perspective, incorporating their changes. This holds the major advantage that changes can be seen in relation to publicly available (and potentially changing) properties without integrating these changes into the public version of the artifact. This concept can be exploited in consistency checking, as it allows us to check the consistency of changes that have yet to be published. The full concept of layered work areas can be seen in Fig. 6. In this illustration, all changes in the PWA overwrite their respective equivalent in the PA. The link represented as the artifact `EML1` holds a property `in` towards `Q70.0` and a property `out` towards `21K3`. Both PWA1 and PWA2 hold different changes on these two properties respectively. When the artifact `EML1` is retrieved in full, the DesignSpace will return the public version, complemented by the changes of the PWA used for retrieval. For example, PWA1 will retrieve `EML1` with an `in` property towards `Q71.0` and an `out` property towards `21K3`. The PWA2 will retrieve the same properties with `Q70.0` and `31K3`.

This concept of keeping engineering artifacts in a layered hierarchy of work areas comes with various advantages. There is a separation of personal changes and public knowledge. This gives engineers an individualized view on their engineering artifacts. Changes are always kept separate from public knowledge until they are published. Yet a full engineering artifact is always retrieved on the basis of its potentially changing public version. This means the individual private perspective of the engineer is always kept up to date (unlike in most contemporary version control systems, where being up to date with a public repository requires checking out its latest state). With regards to consistency checking this provides the advantage, that a private change can always be checked with regards to publicly available knowledge. This allows consistency checking from the perspective of a specific engineer's work, without publishing the said work. Supporting such private perspectives puts this work into contrast with traditional consistency checking approaches (e.g., [13,14,15,16]), which are either entirely focused on publicly available knowledge or on an incomplete subset thereof.

5.3 | Traceability

A major goal of VHA for this project was to allow for traceability between engineering artifacts. After a change has been performed it should be possible to easily identify other artifacts that might be affected by the change. Further it should be possible to identify the associated engineers.

A major challenge with regards to traceability is the heterogeneous nature of engineering artifacts. Different kinds of engineering artifacts underlie different syntactic and semantic rules. Connecting these artifacts for the purpose of traceability has to overcome these semantic and syntactic differences. This problem is often tackled by managing traceability information in separate traceability tools. In these tools, traceability links as well as engingeering artifacts are captured indepdendently from the actual engineering artifacts. In other words, information is duplicated and analysis is done on a minor subset of the original engineering artifacts. This complicates a comprehensive analysis of the resulting traceability

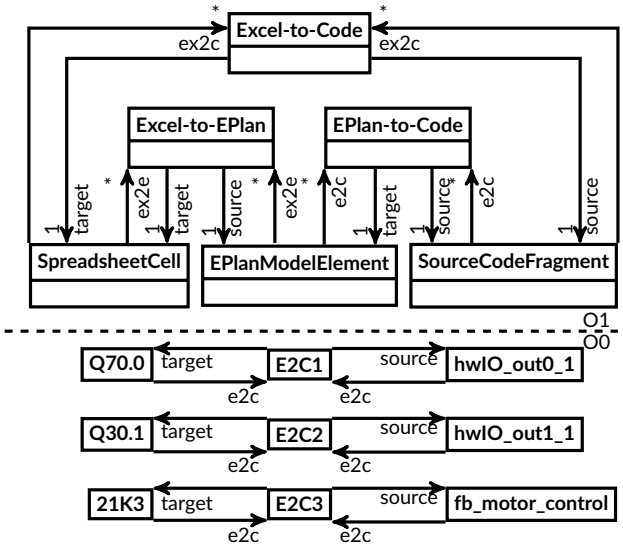


FIGURE 7 Extension of type information for traceability and linked engineering artifacts.

graph. It further comes with the disadvantage that traceability links have to be maintained independently. This means, whenever an engineering artifact changes in a way that affects traceability, this change has to be propagated through various tools. Not only does this cause extra costs for maintenance, but without highly detailed documentation this process may quickly become a source of error.

We avoid these problems in our approach by live synchronizing changes of engineering artifacts with the DesignSpace. Every potential change to traceability information is immediately realized in the platform holding this information. Further, we can analyse any change through a consistency checker, making sure that traceability information is correct and complete. Further, the constructed traceability graph can be used to analyze the propagation of inconsistencies identified in an engineering artifact.

To realize traceability maintenance on the DesignSpace, we first need to integrate all involved engineering artifacts in the uniform artifact representation as described in Section 5.1. After integrating all involved artifacts with the DesignSpace data integration framework, its linking tool can be used to establish traceability between (parts of) the different artifacts. Two artifacts can either be linked together by referencing each other through a property, or by referencing an interposed link artifact. This typed link artifact stores both a source and a target, as well as additional information (e.g., name and description). Using an additional artifact for linking has the advantage that the links themselves become concrete engineering artifacts. For traceability, this means that traceability links can easily be retrieved for further analysis and manipulation.

For creating, updating, and deleting links between the different artifacts, we developed a standalone desktop application, the DesignSpace *Linking Tool*. While the creation of links may be done (semi-)automatically using state-of-the-art heuristics to identify the parts that are logically connected (e.g., a cell in a spreadsheet that is used for defining a constant in the source code), the solution we developed relies on manual linking performed by engineers.

The application visualizes any information present in the DesignSpace uniformly and traces can be established between arbitrary pieces of information. For example, one cell in an Excel sheet may be linked to another cell in the same spreadsheet, to another cell in a different spreadsheet, to a constant or also to a method in the source code, or to an element of the electrical model. To create a link, an engineer simply has to specify the two pieces of information that are logically connected. Different kinds of links can be used to express additional meaning or to restrict the information that is eligible as a source or a target of the link. For example, for this project, in addition to the linking artifacts already presented above - which are responsible for linking information between engineering artifacts of the same type - we provide by default three kinds of links that may be used to link different types of engineering artifacts: `Excel-to-Code`, `Excel-to-EPlan`, and `EPlan-to-Code`. These kinds of links simply restrict the source and target information to specific artifact types. For instance, an `Excel-to-Code` link allows only cells from an Excel spreadsheet as a source and only elements of the source code as a target. These three kinds of links are considered bidirectional; they just express that there exists a connection between the linked elements. Fig. 7 depicts the type information for linking artifacts used in this project. Note that it contains one artifact type for each of the pre-defined types of traces at the ontological level `O1`. In Fig. 4, the creation of an `EPlan-to-Code` link is shown. Specifically, the left-hand side of the figure shows the EPlan model, whereas on the right-hand side the Eclipse IDE with the Java source code is shown. The component named `Q70.0` in the EPlan model (selected and drawn in blue), which models the `Normal Output` IO-port of the `Siemens PLC` in Fig. 2(b), should be linked to the variable in the source code that reflects this IO-port `hwIO_out0_1`. In the center of Fig. 4, the linking tool

for creating such a link is shown. In this tool, the engineer has opened the information available in the DesignSpace of the EPlan model (right-hand side of the tool) and the source code (left-hand side of the tool). The engineer simply selects the DesignSpace representation of the IO-port (`Q70:0`) and the corresponding source code fragment representation (`hwIO_out0_1`) to establish the desired link. After linking each EPlan P8 model element to its corresponding source code fragment, the resulting DesignSpace representation is depicted in the bottom part (i.e., the ontological level `00`) of Fig. 7.

Note that additional kinds of links can be defined by engineers freely and on-demand. For example, an engineer may want to connect certain elements in the electrical model to a requirement, while another engineer may want to use a link to express that a certain constant in the source code exactly reflects a value that is defined in a spreadsheet. Note that it is possible for engineers to express link semantics through additional link constraints. This allows, for instance, for different units in different artifacts. For example, a link may be enriched with a constraint that ensures correct translation between a metric value in a spreadsheet (e.g., 2,400 mm) to an imperial value in an electrical model (e.g., 94.48819 inches), or 2.4 kW to 2,400 W. The *Object Constraint Language (OCL)*[5] is used for defining these constraints. Using OCL, more complex and sophisticated constraints can also be stated and validated. Next, we present in detail how constraints are written and validated in this work.

5.4 | Consistency Checking

As discussed in Section 4, the main objective of this project was to support consistency checking among VHA's major engineering artifacts. Above, we have seen how these artifacts are integrated with the DesignSpace platform and how they can be connected in the DesignSpace to establish traceability. Besides its obvious benefits, traceability is also a requirement for enabling the employed consistency checking approach. Specifically, our solution relies on a variation of the Model/Analyzer consistency checking framework[5,17] adapted for the DesignSpace platform. The Model/Analyzer framework allows for highly efficient, incremental consistency checking of any information that is available in the DesignSpace.

In our variation of the framework, the Model/Analyzer is directly integrated into the DesignSpace as a platform-sided service. This allows for the propagation of consistency information, directly through the internal artifact structure of the DesignSpace. This gives us the possibility to treat consistency information as an additional type of engineering artifact that is maintained alongside the other engineering artifacts. Naturally this also requires us to introduce Model/Analyzer concepts in the form of typed artifacts. For this work, we introduced two types of artifacts:

- **Consistency Rule Definition Artifacts (CRDs)**: A Consistency Rule Definition holds both a consistency rule (in the form of an OCL expression) and a reference to the artifact type for which the rule should hold (the context of the rule). These artifacts are provided by the users of the DesignSpace.
- **Consistency Rule Evaluation Artifacts (CREs)**: A Consistency Rule Evaluation Artifact realizes a CRD for a specific engineering artifact and stores a result and a scope. These artifacts are automatically instantiated by the DesignSpace consistency checking service.

When a CRD is created, the consistency checking service automatically creates the corresponding CREs for each artifact related to the defined context.

It is important to note that both CRDs and CREs underlie the same conditions of layered representation as all other engineering artifacts in the DesignSpace. This means if somebody changes a CRD, the change will be stored within that person's PWA. Naturally, this has an effect on the distribution of CREs as well. When a change happens in a work area, the CRE must be re-validated for the said change. The result of this re-validation is stored as a change on the CRE artifact in the PWA from which the re-validation was triggered. This enables comprehensive consistency information for every PWA containing consistency-relevant changes in the DesignSpace. This allows to provide a work-area-specific perspective on the consistency state of engineering artifacts stored on the DesignSpace. To enable incremental re-validation after changes of information, our consistency checking service reacts to changes in the data structure of the DesignSpace. A change in the DesignSpace causes an event that can be listened to by a service. Therefore, a re-validation via the consistency checker is triggered whenever the information available in the DesignSpace changes (i.e., whenever an engineer performs a change in his or her local tool and the change is synchronized with the DesignSpace). However, the re-validation works incrementally, meaning that only those consistency rules are re-validated that are potentially affected by the change. This is done with the help of a `scope` that is attached to each CRE. The scope documents every property – respectively artifact – that is involved in the re-validation of a consistency rule. If one of these properties is changed, the consistency checker can identify associated scopes with the help of the change event. The CREs carrying these scopes are then re-validated.

It has been shown that this incremental re-validation of consistency rules after changes to the checked data is typically performed within milliseconds. Thus, the information about changes regarding consistency between engineering artifacts can be provided instantly to engineers. All tasks regarding consistency checking (e.g., definition of consistency rules or the notification about the visualization of inconsistencies) are done in a custom tool developed for this project.

[5]OCL: https://www.omg.org/spec/OCL/About-OCL/ (last visited: 26.03.2020)

```
1 [Context: EPlan-to-CodeLink]
2 self.source.e2c->contains(self) and
3 self.target.e2c->contains(self)
```

Listing 1: Consistency Rule for `EPlan-to-CodeLink`

```
1 [Context: SpreadsheetLink]
2 self.source<>self.target and
3 self.source.link->contains(self) and
4 self.target.link->contains(self)
```

Listing 2: Consistency Rule for `SpreadsheetLink`

Indeed, in order to check consistency among information from different artifacts, besides the links between these artifacts, there must exist consistency rules that express their desired relation. These consistency rules are written in OCL, as already briefly discussed above. For each consistency rule, a *context* must be chosen. This context defines which artifact types are affected by the rule. Specifically, the consistency rule is validated individually for each logical instantiation of the defined context (i.e., a rule context at the ontological level `L1` means that each instantiation of the rule context at `L0` is validated). When using our variation of the Model/Analyzer and the DesignSpace platform, any piece of information available in the DesignSpace may be used as a consistency rule's context. For instance, a consistency rule with the context `Class::Java` means that the rule is validated for every occurrence of a Java class in the DesignSpace. In the following, we discuss four relevant consistency rules used in this project.

5.4.1 | EPlan-to-CodeLink

Let us first show how consistency rules can be used to ensure that the traceability features discussed above are used correctly. We must make sure that both the `EPlanModelElements` and the `SourceCodeFragments` can refer to the respective links between them. As an example, consider the link created between the electrical component `Q70.0` and the source code fragment `hwIO_out0_1` in Fig. 4. This particular link should be navigable from both `Q70.0` and `hwIO_out0_1` by using the reference named `ECLink` of the respective DesignSpace representations of the two components (i.e., `(Q70.0).ECLink` and `(hwIO_out0_1).ECLink` should point to collections of `EPlan-to-Code` links that the respective elements are part of, and these collections should both include the specific link). This can be ensured with the consistency rule shown in Listing 1. The context of this consistency rule is `EPlan-to-CodeLink`. Therefore, the consistency rule is validated individually for each occurrence of such a link, including the `EPlan-to-CodeLink` between `Q70.0` and `hwIO_out0_1`.

5.4.2 | SpreadsheetLink and SourceCodeLink

Similarly to the `EPlan-to-CodeLink`, we want to ensure that links connecting different spreadsheet cells or source code fragments are used correctly. As shown in the type information for linking in Fig. 7, a spreadsheet cell's representation in the DesignSpace may be linked to another cell by a `SpreadsheetLink`. However, it is not desired that a cell can be linked to itself. Therefore, the consistency rule in Listing 2 does not only check that the link is also accessible from the cells it connects, but also that it really connects distinct cells. The same checking can be done for links between source code fragments. The corresponding consistency rule is equivalent to the one shown in Listing 2 but with `SourceCodeLink` as a context.

5.4.3 | EPlanModelLink

Finally, for the sake of completeness, links between elements in EPlan P8 models should also only exist between distinct elements, which is ensured with the consistency rule shown in Listing 3.

Now that we have described consistency rules that ensure correct traceability, let us move on to consistency rules for ensuring correct relations between different engineering artifacts such as EPlan P8 models and Java source code. Specifically, we want to make sure that whenever two components in an electrical model are linked, their respective representations in source code are also linked. The corresponding consistency rule, which is instantiated for every `EPlanModelLink`, is shown in Listing 4. Indeed, this can only be checked if both of the linked model elements (i.e., the link's `in` and `out` `EPlanModelElements`) are represented by a source code fragment (i.e., the target of a `EPlan-to-Code` link). This can be ensured by checking whether both elements are part of at least one `EPlan-to-Code` link (see line 2 of Listing 4). If that is the case, then both

```
1 [Context: EPlanModelLink]
2 self.in<>self.out and
3 self.in.link->contains(self) and
4 self.out.link->contains(self)
```

Listing 3: Consistency Rule for `EPlanModelLink`

```
1 [Context: EPlanModelLink]
2 (self.in.e2c->size()>0 and self.out.e2c->size()>0) implies
3  self.in.e2c->exists(
4   x:EPlan-to-Code|self.out.e2c->exists(
5    y:EPlan-to-Code|x.source.link->exists(
6     z:SourceCodeLink|
7      ((z.source=x.source and z.target=y.source) or
8      (z.source=y.source and z.target=x.source)))))
```

Listing 4: Advanced Semantics Consistency Rule for `EPlanModelLink`

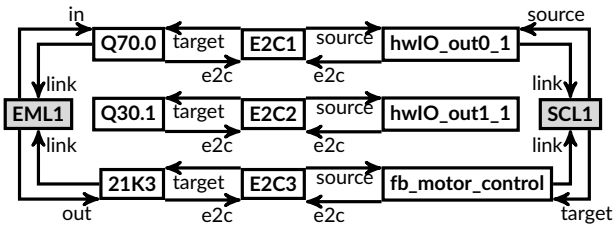


FIGURE 8 Engineering artifact representation in the DesignSpace after step 3.

linked `EPlanModelElements` must be part of `EPlan-to-Code` links to `SourceCodeFragments` that are connected by a `SourceCodeLink`. Those `EPlan-to-Source` link for the `in` and `out` `EPlanModelElements` are named `x` and `y`, respectively, in the consistency rule (see lines 4–5 of Listing 4). The existence of an appropriate `SourceModelLink` between the source code fragments is checked in lines 6–8 of Listing 4. Since `SourceCodeLink` should only express a connection between source code fragments without a prescribed direction but `SourceCodeLinks` have a source and a target `SourceCodeFragment`, it is necessary to check for the two possible directions (see lines 7–8 in Listing 4). Now that we have presented how engineering artifacts are integrated, which kinds of links can be used to establish traceability between different engineering artifacts, and how this traceability can be used to define consistency rules, we next have a look at how this can be applied to the challenge problem from Section 3.

6 | APPLICATION TO CHALLENGE PROBLEM

We will now revisit the challenge problem from Section 3 and discuss how our solution is applied during the most important steps.

6.1 | After Step 3

We start with Step 3. After this step, engineers have defined the initial versions of the EPlan P8 model and the corresponding source code, as depicted in Fig. 2(c) and Fig. 2(d), respectively. These two engineering artifacts have already been annotated by the engineers and the corresponding DesignSpace representations have been created automatically by the tool-adapters. Moreover, engineers have established traceability between the engineering artifacts, as shown by the presence of `EPlan-to-Code` links. Figure 8 shows the resulting DesignSpace representation. At this point, there are no inconsistencies. Notice that for the `in` and `out` elements of the `EPlanModelLink`, the corresponding source code fragments are also connected via `SourceCodeLink`. Therefore, at this stage of development all involved artifacts are consistent.

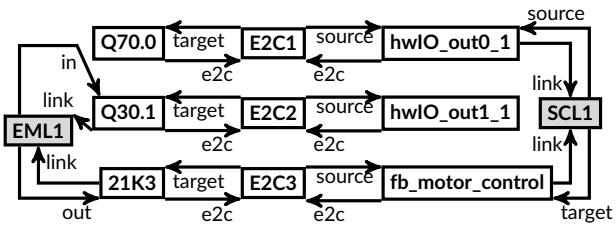


FIGURE 9 Engineering artifact representation in DesignSpace after step 5.

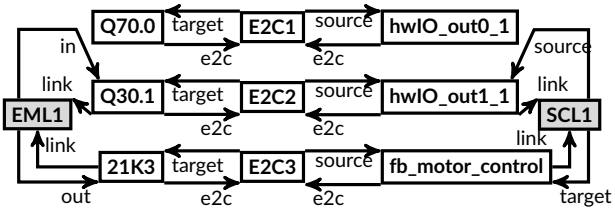


FIGURE 10 Engineering artifact representation in DesignSpace after step 6.

6.2 | After Step 5

After Step 5, the EPlan P8 model has been changed by the electrical engineer to reflect the requests issued by the safety engineer in Step 4. The DesignSpace representation of the electrical model has been updated automatically by the EPlan P8 tool-adapter, the result of this update is depicted in Fig. 9. At this point, the consistency rule from Listing 4 is re-validated for the link that now connects the components `21K3` and `Q30.1` (instead of `Q70.0`). However, since the source code has not been changed, there is no link in the source code representation that connects the corresponding source code fragments. Therefore, an inconsistency is detected and visualized right after the change of the electrical model. The information presented to the engineers shows exactly that the reason for the inconsistency is that the mapping in the source code is not conforming to the mapping in the EPlan P8 model. Moreover, the Workbench also lets the electrical engineer, who performed the change of the electrical model, identify the software engineer(s) responsible for the involved source code fragments. Of course, the software engineers will also be informed in the Workbench about the inconsistency. Therefore, not only the inconsistency is detected, but more specific information about the impact of the changed electrical model is available.

6.3 | After Step 6

After reacting to the inconsistency that was detected after Step 5 by performing Step 6, the inconsistency is removed by the software engineer by changing the mapping between source code fragments. In Fig. 10, the resulting representation of engineering artifacts in the DesignSpace is depicted. Notice that the `SourceCodeLink` `SCL1` has changed compared to Fig. 9. Specifically, it now uses as source the safety output of the Siemens PLC, `hwIO_out1_1`. When re-validating the consistency rule from Listing 4 for the `EPlanModelLink` `EML1`, the consistency checker detects that now a `SourceCodeLink` exists that connects right source code fragments (i.e., those that represent the `EPlanModelElements`, which are linked by `EML1`). Again, the change is processed immediately by the consistency checker and the engineer is informed immediately that the change actually removed the inconsistency.

7 | DISCUSSION

Now that we have presented the solution developed in this project and its application to the challenge problem, let us revisit the goals and requirements stated in Section 4 and discuss whether these are met by the solution. Moreover, in this section, we will discuss the major challenges that were encountered during the project.

7.1 | Goals

7.1.1 | Traceability

As far as traceability is concerned, our solution provides the means to establish traceability between the integrated artifacts (i.e., spreadsheets, electrical models, and source code). The DesignSpace workbench can be used not only to establish traces of the pre-defined types, but also new types of traces with custom semantics can be defined by engineers. Using our solution, engineers at VHA can easily create and maintain traceability between different engineering artifacts as standard tasks that are added to their established development process. Engineers are expected to incrementally build traceability information by creating traces each time they make use of existing information they received from other engineers. While managers at VHA and Flanders Make are confident that this practice can be adopted by engineers, they believe that more support for engineers is still necessary.

7.1.2 | Consistency Checking

For consistency checking, we have developed a set of consistency rules that are validated automatically and incrementally by the employed Model/Analyzer consistency checking framework. VHA believes that these consistency rules are sufficient to check electrical models and source code for consistency, which was the main goal of this project. Moreover, feedback about inconsistencies is provided instantly (or at any desired point in time) to engineers. However, more consistency rules may be added to extend the current set of consistency rules to also cover, for instance, spreadsheet data.

7.1.3 | Change Impact Analysis

Based on traceability and consistency checking, our solution provides the necessary information for comprehensive change impact analysis. This allows engineers to quickly identify which engineering artifacts may be (or are) affected after a performed change and which engineers are responsible for those engineering artifacts. This provides important information for further analysis, especially with regards to the safety critical components of a cyber-physical system. Therefore, our solution provides the information desired by VHA. Overall, the developed solution helps VHA to realize their goals of establishing traceability and consistency checking in order to improve the efficiency of their development process.

7.2 | Requirements

The requirements stated in Section 4 are met by the developed solution. In particular, the stated tools have been integrated with tool-adapters. Moreover, change impact analysis is now possible and dependencies between engineering artifacts are managed and visualized by the DesignSpace Workbench application.

7.3 | Challenges and Lessons Learned

One factor that has been underestimated in the early phases of the project was the development of tool-adapters. Developing a tool-adapter for a commercial, closed-source tool with a defined API required us not only to familiarize ourselves with that API, but it also required all involved stakeholders to analyse and understand exactly which information is actually available to plug-ins or extensions, and in which form. Only after an in-depth analysis of the available information, it is possible to successfully develop a tool-adapter that is capable of synchronizing the relevant tool information with the DesignSpace. For example, the EPlan P8 tool-adapter developed in this project has to automatically discover the links between two or more components that are synchronized with the DesignSpace, as it is not possible for a user of the tool to mark these links. Moreover, it can be an additional challenge to observe a tool's internal data structure for incremental changes in order to synchronize its data live with the DesignSpace (or any other cloud solution, for that matter). However, typically tools allow at least for a synchronization at some events during their typical workflow, for example, when the user saves changes. Additionally there are significant differences between tools with respect to their extensibility and the expressiveness of information available for third-party plug-ins or extensions. However, for this project we were able to access the relevant information in all three integrated tools. Moreover, it has been shown that in general commercial engineering applications can be extended appropriately [18].

Overall, the results of the project show that even the basic forms of traceability and consistency checking that our solution provides to VHA may help to improve the efficiency of a development process significantly. Engineers and managers at VHA were generally perceiving the use of the provided tools as straightforward and easy to learn. While only time can tell whether the practice of establishing and managing traces is actually performed continuously by engineers, the involved stakeholders agree that this additional task does not impose a significant amount of extended work that would severely impact their existing workflow.

8 | RELATED WORK

Indeed, the topics of traceability and consistency checking have been discussed extensively in literature and they have also been addressed by commercial tool vendors. In this section, we will discuss those approaches and tools that are closest to the solution developed in this project and highlight the differences between existing approaches and tools and the results of this project.

For consistency checking, various approaches have been proposed by academics and several commercial tools exist that perform some sort of consistency checking (e.g., [15,17,19,20,21,22,23,24,25,26,27]). Unfortunately, to the best of our knowledge, there is no commercial or academic tool available that supports consistency checking of EPlan P8 models, let alone support for checking consistency between EPlan P8 models and other engineering artifacts. For academic approaches, as with commercial tools, they do not provide out-of-the-box support for EPlan P8 models, meaning that an adaptation of any existing approach was necessary. We opted for employing the Model/Analyzer consistency checker [5,17] as it has been shown that it scales well and provides instant feedback about inconsistencies, even for large-scale industrial models [28]. Moreover, the Model/Analyzer not only provides engineers with information about inconsistencies, but it also provides features that automatically, based on individual inconsistencies, provide engineers with suggestions on how to repair them [29]. Of course, there also exist other technologies that allow for automatic model repairs (e.g., [20]). However, such approaches typically select and execute repairs automatically. Discussions with VHA and Flanders Make showed that such automatically performed changes are not desired as engineers wish to have ultimate control about the engineering artifacts and do not want the artifacts to be changed by fully automated technologies. Thus, the use of the Model/Analyzer, which only informs about inconsistencies and, if desired, possible repair options, is a valid choice.

Another crucial aspect of the consistency checking approach applied in this work is the heterogeneity of engineering artifacts. The possibility to check consistency across different engineering domains has been researched by various other works (e.g. [30,31,32]) One major problem is the integration of type information in the platform used for consistency checking. Most approaches do not provide automated synchronization of artifacts in a uniform representation, but rely on costly meta-model merges. In line with this problem, an extension of type information with further meta-information (e.g., properties for linking) poses a problem as well. Our approach easily allows for such extensions. Artifact type information can be extended at will and may contain properties that are not available in the engineering tool the artifacts originate from. These properties can then be filled by services running on the DesignSpace (e.g., traceability links that are filled by the traceability service).

The heterogeneity of engineering artifacts plays a similarly important role in the field of systems engineering, where modeling languages such as SysML[6] are used for the specification, analysis, design, verification and validation of large systems. These languages are supported in various tools, such as Papyrus[7] or Eclipse PolarSys[8]. However, most system engineering tools do not support a holistic, fine-grained integration of engineering artifacts for team-oriented manipulation. This is largely due to the fact, that SysML - being a variation of UML - is mostly focused on design diagrams. Engineering artifacts representing concrete implementation (e.g., code) find little to no consideration in many SysML-related tools (with the exception of code skeleton generation). Our approach not only considers the consistency between heterogeneous engineering artifacts, but also produces immediate consistency feedback, in relation to recently adapted changes. This collaborative aspect of passing information about inconsistencies to engineers in a timely manner is absent from currently available SysML tools.

In model-driven engineering on the other hand, multiple tools and approaches partly incorporate collaborative concepts, some of which were covered in the literature review of Bruneliere et al. [33] . The main focus of these approaches is the unified consideration of different models within a singular environment (e.g., hardware design and software implementation). This is related to our own consideration of heterogeneous engineering artifacts. Similarly, models corresponding to multiple meta-models are manipulated together. Managing consistency in such an environment is a major factor in these works. EMF Views [34] Kitalpha, [35], Model Join [36] and Viatra [37] are similar to our own work in this regard. However, only Viatra recomputes consistency information in an incremental way. This is done on a change basis. The difference to our work is the application of these principles in a cloud environment with immediate feedback. Similar differences can be identified in approaches such as Vitruvius [38] and ModelJoin. While conceptually these works are similar, the resulting collaboration possibilities differ, mostly due to the lack of immediate consistency feedback.

For traceability, different approaches have been proposed in the past, mainly focusing on automatic creation and management of traces [39], and tools often support some level of traceability, for example, through change tracking (e.g., Excel). However, discussions with VHA and Flanders Make showed that automatic discovery of traces was not desired as there is always a chance of false positives that must be investigated. Therefore, we decided to rely on manually managed traces. Relying solely on different tools' traceability features was indeed not an option as none of the involved tools allowed to specify any kind of connection to engineering artifacts of other tools.

[6] SysML: http://www.omgsysml.org/ (last accessed: 22.06.2020)
[7] Papyrus: https://www.eclipse.org/papyrus/ (last accessed: 22.06.2020)
[8] Eclipse PolarSys: https://projects.eclipse.org/projects/polarsys (last accessed: 22.06.2020)

9 | CONCLUSIONS AND FUTURE WORK

In this paper, we presented the results of a collaboration between Van Hoecke Automation, Flanders Make, and JKU to establish traceability, consistency checking, and impact analysis in a company that focuses on the development of automation solutions. During the project, a software solution was developed that effectively helps the company to improve its development process by providing traceability capabilities for different engineering artifacts and instant consistency checking between them. Both traceability and consistency checking allowed for a comprehensive change impact analysis. This directly influences the safety-critical aspect of the development process by supporting safety analysis. We found that one of the major challenges was the adaptation of existing technologies to support the domain-specific commercial tools used by the company. For future work, we plan to investigate possibilities of making the integration of such tools easier so that the effort for establishing traceability and consistency checking technologies can be reduced and the acceptance and application of these technologies increase.

ACKNOWLEDGEMENTS

The research reported in this article has been partly funded by the Austrian Science Fund (FWF) grant # P 31989-N31, the LIT Secure and Correct Systems Lab, LCM, Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK), the Federal Ministry for Digital and Economic Affairs (BMDW), and the Province of Upper Austria in the frame of the COMET - Competence Centers for Excellent Technologies - program managed by Austrian Research Promotion Agency FFG. We would also like to extend our gratitude to Van Hoecke Automation NV and Flanders Make for their cooperation. In particular, we would like to thank Davy Maes, from Flanders Make and Roland Kretschmer, from JKU, for their contribution to the previous work[7].

References

1. Biró M, Mashkoor A, Sametinger J, Seker R. Software Safety and Security Risk Mitigation in Cyber-physical Systems. *IEEE Software* 2018; 35(1): 24–29. doi: 10.1109/MS.2017.4541050

2. Mashkoor A, Biró M, Messnarz R, Palacios RC. Selected functional safety and cybersecurity concerns in system, software, and service process improvement and innovation. *J. Softw. Evol. Process.* 2018; 30(5). doi: 10.1002/smr.1955

3. Mashkoor A, Sametinger J, Biro M, Egyed A. Security- and safety-critical cyber-physical systems. *Journal of Software: Evolution and Process* 2020; 32(2): e2239. doi: 10.1002/smr.2239

4. Demuth A, Riedl-Ehrenleitner M, Nöhrer A, Hehenberger P, Zeman K, Egyed A. DesignSpace: an infrastructure for multi-user/multi-tool engineering. *Proceedings of the 30th Annual ACM Symposium on Applied Computing* 2015: 1486–1491.

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

6. Egyed A. Automatically Detecting and Tracking Inconsistencies in Software Design Models. *IEEE Trans. Softw. Eng.* 2011; 37: 188–204.

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

8. Tröls MA, Mashkoor A, Egyed A. Live and Global Consistency Checking in a Collaborative Engineering Environment. *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)* 2019: 1762 - 1771.

9. Tröls MA, Mashkoor A, Egyed A. Collaboratively enhanced consistency checking in a cloud-based engineering environment. *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* 2019: 1–6.

10. Tröls MA, Mashkoor A, Egyed A. Multifaceted Consistency Checking of Collaborative Engineering Artifacts. *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* 2019: 278–287.

11. Fahad M, Moalla N, Bouras A. Towards ensuring satisfiability of merged ontology. *Procedia Computer Science* 2011; 4: 2216–2225.

12. Kotis K, Vouros GA, Stergiou K. Towards automatic merging of domain ontologies: The HCONE-merge approach. *J. Web Sem.* 2006; 4(1): 60–79. doi: 10.1016/j.websem.2005.09.004

13. Fradet P, Le Métayer D, Périn M. Consistency checking for multiple view software architectures. *Software Engineering - ESEC/FSE99* 1999: 410–428.

14. Finkelstein AC, Gabbay D, Hunter A, Kramer J, Nuseibeh B. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering* 1994; 20(8): 569–578.

15. Sabetzadeh M, Nejati S, Easterbrook S, Chechik M. Global consistency checking of distributed models with TReMer+. *2008 ACM/IEEE 30th International Conference on Software Engineering* 2008: 815–818.

16. Nentwich C, Emmerich W, Finkelstein A, Ellmer E. Flexible consistency checking. *ACM Trans. Softw. Eng. Methodol.* 2003; 12(1): 28-63.

17. Egyed A. Instant consistency checking for the UML. *Proceedings of the 28th international conference on Software engineering* 2006: 381–390.

18. Egyed A, Balzer R. Integrating COTS Software into Systems through Instrumentation and Reasoning. *Autom. Softw. Eng.* 2006; 13(1): 41-64.

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

20. Nentwich C, Emmerich W, Finkelstein A. Consistency management with repair actions. *25th International Conference on Software Engineering, 2003. Proceedings.* 2003: 455–464.

21. Silva dMAA, Mougenot A, Blanc X, Bendraou R. Towards automated inconsistency handling in design models. *International Conference on Advanced Information Systems Engineering* 2010: 348–362.

22. Easterbrook S, Nuseibeh B. Using ViewPoints for inconsistency management. *Software Engineering Journal* 1996; 11(1): 31 -43.

23. Blanc X, Mounier I, Mougenot A, Mens T. Detecting model inconsistency through operation-based model construction. *2008 ACM/IEEE 30th International Conference on Software Engineering* 2008: 511–520.

24. Reiss SP. Incremental maintenance of software artifacts. *IEEE Transactions on Software Engineering* 2006; 32(9): 682–697.

25. Riedl-Ehrenleitner M, Demuth A, Egyed A. Towards model-and-code consistency checking. *2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC)* 2014: 85–90.

26. Xu C, Cheung SC, Chan WK. Incremental consistency checking for pervasive context. *Proceedings of the 28th international conference on Software engineering* 2006: 292–301.

27. Blanc X, Mougenot A, Mounier I, Mens T. Incremental Detection of Model Inconsistencies Based on Model Operations. *Advanced Information Systems Engineering* 2009: 32-46.

28. Reder A, Egyed A. Incremental consistency checking for complex design rules and larger model changes. *International Conference on Model Driven Engineering Languages and Systems* 2012: 202–218.

29. Demuth A, Riedl-Ehrenleitner M, Lopez-Herrejon RE, Egyed A. Co-evolution of metamodels and models through consistent change propagation. *Journal of Systems and Software* 2016; 111: 281–297. doi: 10.1016/j.jss.2015.03.003

30. Vangheluwe HL. DEVS as a common denominator for multi-formalism hybrid systems modelling. *Cacsd. conference proceedings. IEEE international symposium on computer-aided control system design* 2000: 129–134.

31. Herzig SJ, Qamar A, Paredis CJ. An approach to identifying inconsistencies in model-based systems engineering. *Procedia Computer Science* 2014; 28: 354–362.

32. König H, Diskin Z. Advanced local checking of global consistency in heterogeneous multimodeling. *European Conference on Modelling Foundations and Applications* 2016: 19–35.

33. Bruneliere H, Burger E, Cabot J, Wimmer M. A feature-based survey of model view approaches. *Software & Systems Modeling* 2019; 18(3): 1931–1952.

34. Bruneliere H, Perez JG, Wimmer M, Cabot J. EMF views: A view mechanism for integrating heterogeneous models. *International Conference on Conceptual Modeling* 2015: 317–325.

35. Langlois B, Exertier D, Zendagui B. Development of modelling frameworks and viewpoints with Kitalpha. *Proceedings of the 14th Workshop on Domain-Specific Modeling* 2014: 19–22.

36. Burger E, Henss J, Küster M, Kruse S, Happe L. View-based model-driven software development with ModelJoin. *Software & Systems Modeling* 2016; 15(2): 473–496.

37. Debreceni C, Horváth Á, Hegedüs Á, Ujhelyi Z, Ráth I, Varró D. Query-driven incremental synchronization of view models. *Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling* 2014: 31–38.

38. Kramer ME, Burger E, Langhammer M. View-centric engineering with synchronized heterogeneous models. *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling* 2013: 1–6.

39. Cleland-Huang J, Gotel OC, Huffman Hayes J, Mäder P, Zisman A. Software traceability: trends and future directions. *Proceedings of Future of Software Engineering* 2014: 55–69.