Model and Data Differences in an Enterprise Low-Code Platform

Arvid Butting, Timo Greifenberg, Katrin Hölldobler
mgm technology partners gmbh
Aachen, Germany
{firstname}.{lastname}@mgm-tp.com

Timo Kehrer
University of Bern
Bern, Switzerland
timo.kehrer@unibe.ch

***Abstract*—The comparison of versions and variants of models is a well-known challenge in model-driven software engineering. In the context of low-code platforms, models are rarely text-only and the modelers are not necessarily familiar with programming. Therefore, presenting the comparison results in an understandable way and on a suitable level of abstraction is a challenging problem vital, e.g., for enabling asynchronous collaborative low-code development.**

This paper describes an approach for calculating and displaying differences between versions or variants of models in A12, a low-code platform for enterprise applications that employs different types of models to represent data and its presentation. The differencing infrastructure is built of reusable modules to avoid redundancies of its commonalities across different A12 model types. Despite that, it can be applied for calculating and displaying differences of data conforming to models in low-code applications. The approach is integrated into A12, but the concepts behind it are transferable to other low-code platforms.

***Index Terms*—Low-Code Platform, A12, Model Differencing**

I. Introduction

In software engineering, comparing development artifacts with each other is a common use case. Different *versions* of an artifact are compared in the context of software evolution or version control systems like Git [1]. *Variants* of an artifact are compared, e.g., in the context of software product lines [2], particularly when extracting a product line from a set of variants in order to analyze their commonalities and differences [3]. In other scenarios, such as clone detection [4], even conceptually unrelated artifacts are compared.

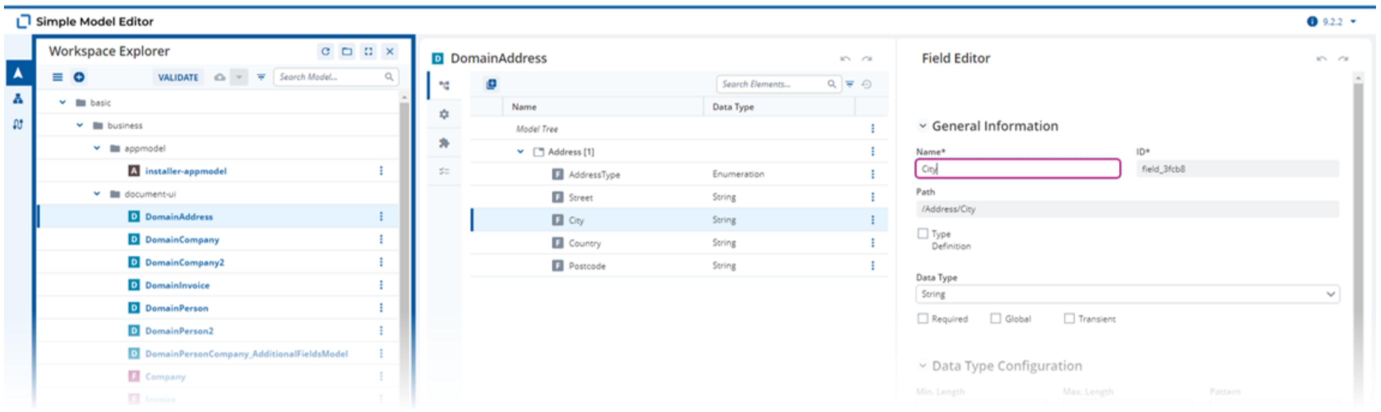
Model-driven development (MDD) [5] and low-code platforms [6], [7] (aka. low-code development platforms [8]) use models as central development artifacts throughout the software lifecycle. In the context of this paper, we consider models as special kinds of artifacts that conform to a meta-model [9] and have a well-defined meaning in terms of a semantics [10]. Low-code platforms in general, and those for enterprise applications in particular, often have models whose syntax is diagrammatic or borrows elements of user interfaces via projectional editors [11]. While such models can be compared based on a textual representation using line-based diff algorithms as adopted, e.g., by version control systems such as Git, it has long been recognized that this is not an adequate solution [12]. The major disadvantage of line-based differencing is that it is not aware of the language's syntax and semantics. Therefore, line-based approaches may report changes that are not semantically relevant, such as white-space changes to mention only one of the most simple examples. Moreover, it is difficult for users to interpret the calculated differences for models that have no textual syntax.

The MDD research community has tackled the challenge of model differencing from various angles. Most notably, a number of approaches for comparing models take into account the abstract syntax [13], i.e., the internal structure of models. However, virtually all approaches presented in the literature have studied model differencing from an algorithmic perspective, while the presentation of differences has been largely neglected. We argue that properly displaying model differences is of utmost importance for modelers who do not necessarily have a programming background, such as *citizen developers* [8] in the context of low-code platforms.

Modelers use low-code development platforms to create models that describe aspects of (low-code) applications. A12 [14] is an enterprise low-code platform relying on projectional editors that comprise editor elements, such as forms, overviews (also referred to as tables), and trees that are also commonly used in low-code applications. To that end, we argue that displaying model differences in dedicated *diff editors* of the platform is an option for integrating the calculated differences with the low-code platform. The diff editors aim to reduce the accidental complexity [15] of citizen developers needing to comprehend differencing results in notations they are unfamiliar with. Another use case for diff editors in the context of low-code platforms is the differencing of data conforming to models in applications engineered with the low-code platform. Hence, approaches for model (and data) differencing in the context of low-code platforms have particular requirements and deserve special treatment.

This paper presents an approach for calculating differences between A12 models and displaying these in a notation close to the one used for originally creating the models. The approach relies on a systematic method and, hence, can be adapted to new model types with little effort. Moreover, as A12 model editors are meta-modeled using A12 models, the display of differences can be applied both for model differences in the development platform and for data differences in low-code applications.

The remainder of this paper is structured as follows: Section II explains foundations of the A12 low-code platform,



```
basic
  business
    appmodel
      installer-appmodel
    document-ui
      DomainAddress
      DomainCompany
      DomainCompany2
      DomainInvoice
      DomainPerson
      DomainPerson2
      DomainPersonCompany_AdditionalFieldsModel
      Company
      Invoice
```

Name	Data Type
Model Tree	
Address [1]	
AddressType	Enumeration
Street	String
City	String
Country	String
Postcode	String

Name*: City
ID*: field_3fcb8
Path: /Address/City
☐ Type Definition
Data Type: String
☐ Required ☐ Global ☐ Transient

Fig. 1. User interface of the Simple Model Editor (SME) of the A12 enterprise low-code platform.

Section III introduces the terminology for model differences used in this paper, and Section IV summarizes the requirements for model differencing in A12. Section V presents the concept for the display of differences between A12 models and their integration into A12's model editor tool. Section VI describes how difference editors are composed from reusable building blocks, and Section VII discusses our approach along with an outlook on potential future extensions. Finally Section VIII compares our approach to related work, before we conclude in Section IX.

II. The A12 Enterprise Low-Code Platform

A12 [14] is an enterprise low-code platform for the development, integration, maintenance, and operation of complex business applications in large-scale IT landscapes. It combines a low-code approach enabling business experts to create application components without any programming knowledge with enterprise-grade custom software development and system integration. The overall A12 platform comprises two main parts: The *modeling platform* and the *runtime platform*. The modeling platform enables *developing* low-code applications and consists of the *simple model editor (SME)*, different model types, and textual domain-specific languages (DSLs). The runtime platform enables *operating* low-code applications and consists of several modular client- and server-side components.

A. A12 Runtime Platform

Among other components, the A12 runtime platform comprises A12 engines, widgets, and data services. A12 engines are client-side model interpreters of the A12 runtime platform. Currently, A12 provides three different engines for forms, tables, and tree structures. Each of these takes a data model, a user interface (UI) model, and data as input. Engines compose UI widgets from the A12 widget library to render different interactive UIs depending on the corresponding model type. Widgets are UI components, mainly for the input and display of data. *Data services* is the main server-side component of A12. Besides storing the data conforming to A12 data models, it provides the models for client-side usage and offers data filter and query capabilities.

A large number of extension points allow combining MDD with professional custom software development and system integration. However, even without any custom development, the resulting business applications provide a rich feature set including localization, responsive design, themeability, and accessibility.

B. Model Types

Following the "data first" modeling paradigm [14], A12 modeling begins with defining the enterprise entities and their interrelationships in terms of data models. The entities are described by *document models* (DMs) that contain a tree structure of groups that each can contain fields of different types. Moreover, DMs contain validation and calculation rules for fields. The interrelationships between entities are described by *relationship models* that connect pairs of document models.

Modelers realize the application logic by defining validation rules and computations via the integrated domain-specific language and workflows via BPMN models. UI models usually refer to A12 data models and establish connections between the fields of data models and UI elements. Instead of a what-you-see-is-what-you-get principle, the UI models describe the underlying structure of the user interface. For each engine, there is a specific UI model type, namely *form*, *tree*, and *overview* model. The *application* model orchestrates the overall client application.

C. The Simple Model Editor (SME)

The SME is the main modeling tool for A12. Except for BPMN models, which are edited using a third party modeling tool, it supports all other A12 model types. Models are stored and loaded from the local file system of the modeler. The SME lists models in the *workspace explorer*, which gives an overview of the available models. When a user selects a single model, a model type-specific editor is opened. All SME parts that are specific to a certain model type are contained in the model type *module* (e.g., the DM module).

Figure 1 depicts a screenshot of the SME. The workspace explorer is located on the left, while the *document model editor* is in the center and the right of it and shows the content of the DM *DomainAddress*. If instead, e.g., a form model (FM) is

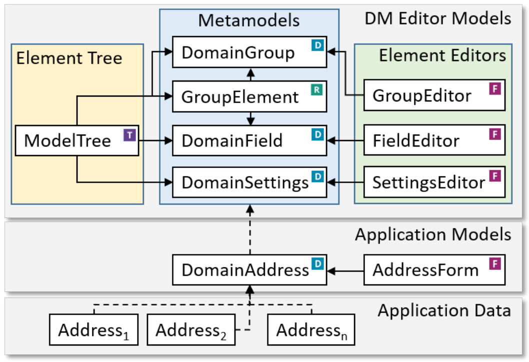


Fig. 2. Example of SME-level and application-level models and data

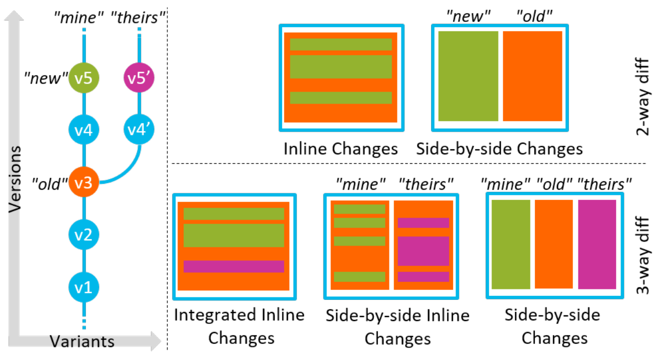


Fig. 3. Displaying differences between model versions and variants

selected in the workspace explorer, the SME displays the FM editor in the center of the SME. The DM editor itself is divided into two parts. The model's *element tree* in the center of the SME gives an overview of the model elements, and the *field editor* on the right shows properties of the field selected in the element tree. Field properties are modified through the field editor. If another DM element (e.g., a group) is selected in the model tree, the DM editor displays the corresponding editor (in this case, the *group editor*) on the right side. DM elements can be added, moved, and deleted via the model tree. Furthermore, the DM editor has a side menu on the left side, where users can switch between the model tree and, e.g., a form-based editor for model settings and an overview-based table displaying type definitions. Type definitions in DMs enable reuse of recurring configurations of field properties (e.g, a number with a specific lower and upper bound) for different fields. The model settings include information about the model, such as its name and the supported locales.

D. Meta-modeling

While the SME is the model editor for A12, it is also built using A12. Especially, the specific model editors make extensive use of model-driven components. Technically, a specific model editor is a composition of different metamodels, editor models, and (customized) engines. The engines have data models as input, which together are regarded as the *metamodel* for the corresponding model type. Moreover, the different UI models are used to model the UI of the specific editors. We refer to these as *editor models*.

The DM editor, for example, uses tree, overview, and form models and, hence, all three engine types as part of its implementation. Each A12 model is stored as a file. At runtime of the SME, application level models are converted into data that can be interpreted by the engines together with the metamodels and editor models. Beside the composition of engines, the editors are implemented using different programming extension points of the runtime platform components.

Figure 2 shows different models used for building the SME as well as A12-based low-code applications. The top part shows a small excerpt of the models that are used for the development of the DM editor that is part of the SME. The DM metamodel used to build the DM editor is composed of different DMs and a single relationship model. The editor models in the figure include the UI for the element tree (see Figure 1) defined by the tree model `ModelTree`. Moreover, the sub editors for specific model elements such as `Group` or `Fields` are modeled using FMs. All editor models reference their underlying data model.

When using the SME to create application-level models, these models conform to the metamodels used to build the SME. Figure 2 shows two exemplary application-level models. Again, the UI model `AddressForm` references the data model `DomainAddress`. Both models conform to their metamodels, which is shown by the dashed arrow. Note that the metamodel for FMs is omitted in the figure, but is present in the SME project.

The bottom part of the figure displays exemplary entries for application data provided to an A12 application via the data services. In this example, the data comprises data entries conforming to the model `DomainAddress`.

III. Difference, Deltas, Versions, and Variants

Regarding the conceptual contents of a model difference, approaches to model differencing can be classified into semantic and syntactic differencing. The semantic difference between two models is a set of counterexamples (aka. witnesses) against the proposition that the compared models are equivalent [16], [17]. For example, the semantic difference between two UML class diagrams comprises the set of instances (i.e., UML object diagrams) being valid for one of the class diagrams but not for the other one [17]. In general, however, this set of witnesses is typically infinite and thus does not represent a complete description of the difference between two models. In this paper, we consider syntactic model differences, as it is customary in model versioning in practice. The syntactic comparison of two models A and B yields a sequence of transformations [18], where each transformation in the sequence describes the addition, removal, or modification of a model element of A and is referred to as a *delta*. If all deltas are applied to the model A, the result is the model B.

The evolution of models leads to different *versions* of a model (cf. (linear) timeline in the version graph depicted on the left side of Figure 3), for pairs of which the difference can be described by an (ordered) list of deltas. Different circumstances can lead to the presence of *variants* of a model. These include the concurrent modification of the same model in distributed modeling environments, or so-called *clone-and-own* [19]reuse of models by copying an existing model and adapting it to individual needs. In the version graph shown in Figure 3, variants are depicted on the horizontal axis. Variants always have a common ancestor version – unrelated variants have the empty model as their common ancestor.

For the display of differences, we distinguish the "two-way diff" that compares two models with each other from the "three-way diff" that compares two models that have a common ancestor. In the context of version control systems, two-way diffs are often applied for comparing two versions of the same model and three-way diffs are utilized for comparing (co-evolved) variants of a model.

In general, we distinguish two forms of displaying the difference in a two-way diff. As depicted in the top of Figure 3, changes can be displayed in an *inline* representation that interleaves the display of deltas with the original model. In the alternative *side-by-side* representation, one model is displayed on the left and the other one on the right side. Model elements affected by deltas are highlighted. The bottom of Figure 3 depicts alternatives for displaying a three-way diff, which are combinations of inline and side-by-side representations.

IV. Requirements for Model Differences in A12

For the realization of calculating and displaying the differences between models in an enterprise low-code platform such as A12 beyond pure textual differencing, we first did a careful requirements analysis before diving into design and implementation. The requirements were elicited in collaboration with different A12 users. One class of users was from the user group of business analysts who use the platform as modelers. Another class of users was software developers who mainly contribute code to A12 low-code applications but who are sometimes also also involved into modeling activities.

RQ1 Modelers should be able to get an overview of all differences in a model

RQ2 Modelers should be able to inspect model differences in a notation close to the original modeling environment

RQ3 Only semantically-relevant differences should be displayed to modelers

RQ4 The differencing infrastructure should be applicable both for models in low-code development and for data on the application level

RQ5 Developers should be able to build new diff editors from reusable parts

We restrict the models that are comparable with the differencing infrastructure to those that are of the same model type in the same model type version. Any two models for which this holds can be compared, regardless of, e.g., whether they share a common ancestor version or not. Models that have the same model *type* but different model type *versions* can be compared if the model with the older model type version is upgraded to the other model's type version via a dedicated model migration tool.

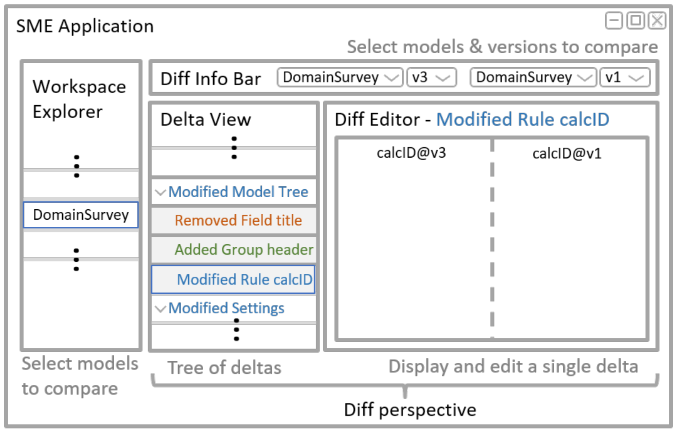


Fig. 4. UI components involved in the comparison of models in the SME

V. Calculating & Displaying Model Differences

The user-visible parts of comparing models in a low-code platform are centered around the display of model differences and its integration into the user interface of the platform.

A. Integration into the Platform UI

Platform users have to be able to trigger the comparison of two models or two versions of a single model via the platform. In the SME, the workspace explorer provides an overview of all files in a workspace and, hence, is an appropriate user interface location for integrating this trigger. A button in the workspace explorer opens a dialog, in which models and model versions can be selected for comparison. In the context of integrating a version control system (VCS) into a low-code platform, the model comparison should also be triggered in certain VCS-specific use cases, e.g., to annotate locally changed models in the overview with a specific icon or to inform about local changes that will be committed to a branch as part of a commit operation dialog.

When a user action triggers the model comparison, the comparison result must be displayed in the platform's UI. In the SME, a user selection of models to compare opens up the *diff perspective* as depicted in Figure 4. This perspective comprises three UI components: the *diff info bar* at the top, the *delta view* on the left side and the *diff editor*. The diff info bar displays the names of the two models that are compared, the model versions which are to be compared may be selected via drop-down menus. Via drop-down selections, users can modify these values. The figure depicts the comparison between the versions `v1` and `v3` of the model `DomainSurvey`.

The delta view is a tree of deltas that result from the model comparison. Because model editors in A12 are usually composed of different overviews, trees, detailed form editors, and editor tabs, the delta view enables identifying the changed model parts at a glance (cf. **RQ1**). The delta view contains two kinds of entries: *editor-level deltas* and *detailed deltas*, which are described in Section V-B. Furthermore, the elements in

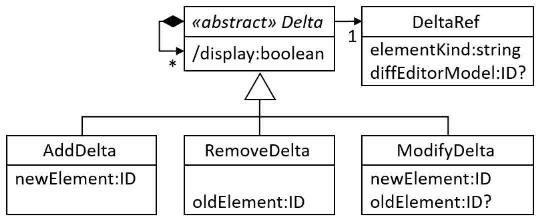


Fig. 5. Data structure of deltas

the delta view are clickable and open the diff editor for the selected delta. This way, users can also navigate between the deltas.

The main purpose of the diff editor is to gain a detailed explanation of the difference described by a certain delta. To do so, the diff editor uses the same form of representation that is used when a model element is created or modified (cf. **RQ2**). For instance, the DM editor comprises a model tree editor to display and edit DM elements, such as fields, groups, and rules. Moreover, it uses individual form-based editors to edit properties of the DM elements. Hence, the diff editor for an "added field" delta is based on the model tree editor, because fields are added there. For a "modified field" delta, the diff editor is based on the form-based field editor, because field properties are modified there.

B. Data Structure for Deltas

The data structure for deltas has a tree shape, i.e., a delta can be composed of other deltas on a lower level and may belong to at most one enclosing delta at an upper level. Currently, our approach uses three levels of deltas: Deltas on the top-most level describe modified (model or data) artifacts. For example, a delta describes that the model `DomainSurvey` has been modified. On the level below, deltas describe changes that affect entire (sub) editors, such as the field editor or the model tree editor. For instance, these deltas describe that the model tree has been modified or that the DM settings have been modified. Deltas on this level are displayed in the delta view as editor-level deltas. One level below, deltas describe changes of elements inside an editor, such as a field that has been added to the model tree. In the delta view, these deltas are represented as detailed deltas.

The data structure, however, is not limited to these three levels but is more general to prepare for further use cases in the future. Hence, each delta has a Boolean flag indicating whether the delta should be displayed in the delta view or not.

Figure 5 depicts the data structure for deltas in A12. Besides the composition of deltas and the flag for displaying deltas in the delta view, each delta indicates an affected model element kind via the `DeltaRef`. For instance, when comparing DMs, a delta can affect the DM element kind `Field`. An affected model element kind comprises a string representation, e.g., used for the delta view, and an (optional) identifier pointing to the editor model that should be displayed if the delta is opened in a diff editor. For instance, the diff editor model for modifying a `Field` is `DiffFieldEditor` (cf. Section VI). The value for the Boolean flag `display` is derived; it is *true* if the diff editor model ID is present.

Currently, we distinguish three delta kinds: `AddDelta`, `RemoveDelta`, and `ModifyDelta`. Add deltas refer to the ID of the new element that has been added, whereas remove deltas indicate the ID of the old element that has been removed. Modify deltas can indicate both new and old element IDs where the old element ID is optional and is only present in case the ID changed due to the modification the delta describes. The latter is useful, e.g., for clone detection in which matching model elements are determined by other means than the ID.

C. Calculating Deltas

While the data structure for deltas is agnostic to concrete model types, the calculation of actual deltas is specific to the type of the models that are compared. Among other reasons, this is due to the fact that only semantically-relevant changes should be displayed (cf. **RQ3**). Hence, every model type module that supports model differencing in A12 must provide an implementation of the delta calculation. The input for the delta calculation are two models and its result is an instance of the delta data structure. In each delta calculation, the first step is to check whether the models are considered comparable, by comparing the model type and the model type version (cf. Section IV).

A12 models have an intrinsic tree structure that can be traversed, e.g., by applying the visitor pattern [20]. The actual realization and the order of traversal, however, is set for each model type individually. The strategy for calculating deltas relies on the tree structure of the models. This tree is traversed in a top-down manner and corresponding elements in the models are matched. If child elements are present then the matching continues with matching the child elements. By default, the elements are matched via their identifiers but the matching criteria can be customized. A limitation of this strategy, however, is the detection of move operations. In this case, an add and a remove delta are calculated, which can be combined to a move delta later [21], [22].

For document models, for example, the first step is comparing the model settings. The comparison of models settings is composed of comparing the model names, the supported characters and locales, labels, roles that should have access to the model, and annotations. Comparing these, again, can be composed of more detailed comparisons. If any of these comparisons results in a difference, the comparison of the model settings results in a "modified" delta that is displayed in the delta view. Since all DM settings are editable via a joint form-based editor, the ID of the diff editor model is the one from the diff editor of the DM settings editor (see below). Finer-grained deltas that are part of the delta for the modification of the settings would all be displayed in the same form. Therefore, the delta calculation could stop at this point, but could also continue to calculate finer-grained deltas that are not displayed. Afterwards, the comparison continues with the model tree elements.

D. Displaying Deltas

All displayable deltas between two models are shown in the delta view. If the difference between two models does not contain any displayable delta, the two models are equal or there is no semantically relevant difference between the models. In both cases, the delta view remains empty. With the delta data structure, the labels for the delta view entries can be calculated by concatenating the delta kind, the affected model element kind, and a human readable form of the element ID. For instance, a remove delta for a field with the path[1] `title` has the label "Removed Field title". Additionally, the delta kind is represented by an individual font color. The delta view is implemented in an extensible way to support more functionality, e.g., filtering and sorting of deltas in the future.

All entries of the delta view are clickable. When a user clicks an entry in the delta view, the ID of the diff editor model indicated in the delta data structure is used to load the diff editor data. The model type of the diff editor model determines which engine is used for the diff editor to open and display the diff editor data.

While, in general, a model type module may use arbitrary diff editor models, the implementation in A12 relies on a common, systematic workflow to derive diff editor models from editor models. Section VI describes the workflow, the editor models, and the metamodels for differencing forms, tables, and trees in more detail. Currently, the diff editors produced via the workflow are two-way diff editors in a side-by-side layout. However, this is determined only via the diff editor models and it is also possible to produce other diff editor layouts.

VI. Systematic Engineering of Diff Editors

Due to the large number of different model types in A12 (which is common for enterprise low-code platforms) and because diff editors should be usable for comparing data in low-code applications created with A12, a structured workflow for creating new diff editors from reusable editor parts is beneficial. This is covered by the requirement **RQ5**.

A. Architectural Overview

To increase the reuse, the differencing infrastructure is separated into model-type-specific and model-type-agnostic infrastructure parts. The latter parts can be reused across all differencing infrastructures with optional customization and result in the fact that the delta view can be reused in a model-type-agnostic way. Section VI-B explains how it integrates with the model-type-specific delta calculation and delta display. Apart from the model-type-specific delta calculation, the differencing infrastructure for each model type must provide diff editor models for each model element that may be referenced in the delta view of that model type. As

[1]Field paths (cf. Figure 1) must be unique within a document model and, hence, can be used as a human readable form of the element ID for fields.

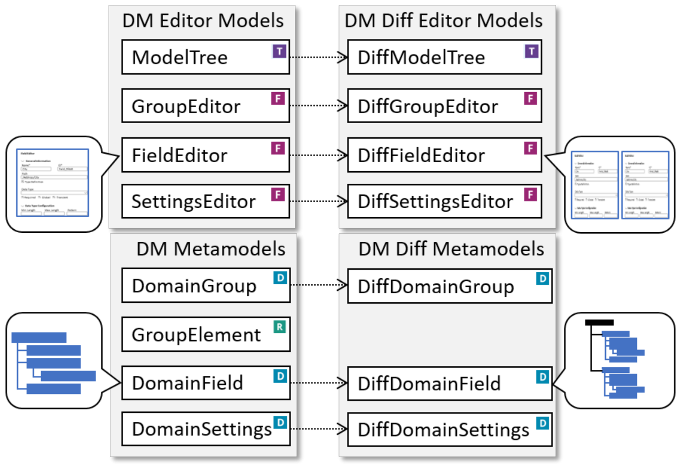


Fig. 6. Deriving diff editor models from editor models and diff metamodels from metamodels

the editor models rely on data from the metamodels, it is also necessary to derive diff metamodels from the metamodels. The diff editor models rely on data from the diff metamodels.

For example, Figure 6 illustrates DM diff editor models for the DM editor models as depicted in Figure 2. For each model used to build the original editor, there must be a corresponding diff editor model, such as `DiffModelTree` and `DiffFieldEditor`. These new models are systematically derived from the original models. In the same way, diff metamodels (e.g., DiffDomainField) are derived systematically from the metamodels (e.g., DomainField). The diff metamodel `DiffDomainField` describes and its instances provide the data required for displaying the deltas between two DM fields. The diff editor model `DiffFieldEditor` describes the form-based UI of the diff editor for comparing two fields. `DiffModelTree` is a tree model that describes the tree-based diff editor for comparing two model trees.

Currently, the derivation process of diff editor models is a manual activity that is planned to be automated in the future. With a code generator [23], recommended diff editor models can be synthesized and, if desired, adjusted manually afterwards. The systematic derivation processes for displaying differences in forms, tables, and trees are explained in Sections VI-C and VI-D.

Note that the derivation process for models of the original SME editors can also be applied to models on the application level. For instance, the models `DiffDomainAddress` and `DiffAddressEditor` can be derived from the corresponding application-level models depicted in Figure 2. Together with a custom delta calculation algorithm for addresses and the reused delta view, an application-level diff perspective that displays differences between application data conforming to the DM `DomainAddress` can be created with little effort (cf. **RQ4**).

B. Model-Type-Agnostic Delta View

The differencing infrastructure includes the delta data structure and the delta view (cf. Section V) as model-type-agnostic

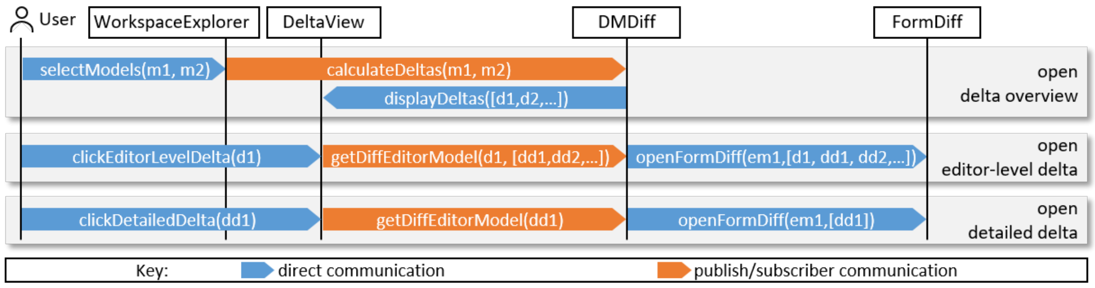


Fig. 7. Communication between components when comparing and displaying the differences between A12 document models

components. Furthermore, the workspace explorer of the SME is not directly aware of model types. Model type modules are registered to the workspace explorer as plug-in modules. The calculation of deltas is model-type-specific and, therefore, typically part of a model type module. Similarly, the models for the model type's diff editors are derived with a generic, systematic process, but still specific to the model type.

Figure 7 depicts an exemplary overview of the communication between the model-type-specific and the model-type-agnostic parts of the infrastructure involved in displaying the diff perspective. The central tasks that users can trigger are to *open the delta view*, to *open an editor-level delta* and to *open a detailed delta*.

Typically, users trigger the display of the delta view (and, therefore, to open up the entire diff perspective) via the workspace explorer, by selecting two models $m1$ and $m2$ that should be compared. The comparison of versions and the one of variants is not conceptually different, hence $m1$ and $m2$ can also be two versions of a single model. The selection starts a publish/subscriber communication to all registered model type module plug-ins of the workspace explorer. The published message is to calculate the deltas for $m1$ and $m2$. If a module is capable of handling the model type of $m1$ and $m2$ (which must be the same, cf. Section IV), it carries out the calculation of the deltas. In the example of Figure 7, the models are DMs and the module that can handle these contains the `DMDiff` delta calculator. As a result, the module feeds the delta view with the calculated deltas.

By clicking an element in the delta view, a user triggers the display of the delta in the corresponding diff editor. More precisely, after an element has been clicked, the delta view starts a publish/subscriber communication to open the diff editor model referenced in the instance of the delta data structure. Through this, the delta view does not need to communicate directly with any model type module. In the example, the clicked delta $d1$ has an affected model element kind from the DM module described by the diff editor model $em1$, which is a form model. Therefore, the DM module responds with opening the form-based diff editor described by $em1$ with the corresponding (customized) form engine.

C. Displaying Differences in Forms

To display the differences between two FMs, our approach employs a diff editor model that is, again, an FM (cf. Figure 6). This requires that the diff metamodel has been derived from the metamodel of the original editor to describe the data displayed in the diff editor. The diff metamodel is derived from an original metamodel by introducing a new root group that has two subgroups *left* and *right*. Each subgroup includes the root group of the original document model, which builds the foundation for a side-by-side diff editor (cf. Section II). For example, Figure 6 contains a visualized abstraction of the model tree contained in the metamodel `DomainField`. The derived diff metamodel `DiffDomainField` contains this tree twice, and an additional new root node. In a similar way, each screen described by an original form model is divided into two sub screens *left* and *right* with all elements of the original model. The screen elements on the *left* part of the form model refer to diff DM elements of the *left* group. Vice versa, the same holds for the *right* screen part and group. For example, Figure 6 contains a visualized abstraction of the screen described by the FM `FieldEditor`. The derived diff editor model contains this screen twice as sub screen.

If a user selects an editor-level delta in the delta view that references a form diff editor model (i.e., a diff editor model of the model type FM), the corresponding form-based diff editor is displayed and highlights all detailed deltas contained in this editor-level delta. If a user selects a detailed delta in the delta view, the same diff editor is displayed but highlights only the selected delta.

The form engine interprets the form diff editor models which basically consist of two identical editor models *left* and *right*. The elements of the two editor models form tuples where for each element of the left editor model (e.g., an input field or a checkbox) there is a corresponding element in the right editor model and vice versa. The form engine was customized to highlight these elements if the values differ, e.g., if a checkbox is checked in the left but not in the right editor. Through this, the form can be edited and the differences are updated instantly.

D. Displaying Differences in Tables and Trees

Displaying the difference in tables and trees is generally similar to displaying the difference between forms. The difference is displayed in the same model kind, i.e., if the original editor model is a tree model, the diff editor is also based on a tree model. Selecting an editor-level delta with a tree diff editor model in the delta view opens the tree-based diff editor and highlights all changes described by detailed deltas contained in this editor-level delta. Selecting a contained detailed delta in the delta view opens the same diff editor, but highlights only the selected delta. Therefore, different delta view entries may display the same table (or tree), but highlight different parts of it. The same holds for tables and overview-based (diff) editors. Moreover, the tree and overview elements of tree- and overview-based diff editors are typically clickable for the purpose of navigation, if they are in the original tree- and overview-based editor models. For instance, field entries in the model tree editor of DMs are clickable to inspect the field in the form-based field editor. Therefore, field entries in the model tree diff editor are also clickable and open form-based diff editors. We display the difference between two tables or trees with an inline diff editor (cf. Section II).

An overview model uses a DM to describe the data that is displayed. The data source for a diff editor model that is an overview model is a diff metamodel derived in the same way as for the differencing of forms described in the previous section. If the value of a left-side diff metamodel element can be matched to a value of the corresponding right-side diff metamodel element, the values are compared. If these values differ, the overview diff editor model cell is highlighted as being modified. If there is no corresponding right-side diff metamodel element, the overview diff editor model cell is highlighted as being added. Vice versa, the overview diff editor model cell is highlighted as being removed if there is no corresponding left-side diff metamodel element for a given right-side diff metamodel element. The same is applied to model elements that represent entire rows of the overview diff editor model.

For the user-visible part of the approach, the display of differences between tables and the one for differences between trees are almost equal. However, instead of a single DM that describes the data for an overview model, the data for a tree model is described by multiple DMs and relationship models among these. The diff metamodels for this are derived as described above. For relationship metamodels, there is no dedicated diff metamodel. Instead, on the data level, a relationship is either present or not.

VII. Discussion

A. Major Design Decisions

The main technical contribution of this work is a method for integrating model differencing capabilities into a low-code platform, addressing a number of specific requirements arising on top of the general challenge of model differencing in MDD. Through the reusable delta view and the systematic approach for deriving diff editors, highly customized diff perspectives for new model types are created as follows:

1) For each editor model, derive a diff editor model via the systematic process.
2) For each metamodel used by an editor model, derive a diff metamodel via the systematic process.
3) Implement a model-type-specific diff algorithm or reuse a generic diff algorithm
4) Integrate diff editors and diff algorithm with diff view and establish communication among these

We have prototypically implemented this approach for selected model types of the low-code platform A12 and based on the following design decisions wrt. difference calculation and display.

Our approach currently uses diff editor models for the display of diff editors and corresponding (diff) metamodels. Alternatively, it is possible to display deltas by opening the original editor twice, where each instance displays the data of one of the compared models/model versions. Additionally, the widgets these editors use could be customized to highlight differences. This would allow using the original editor models and metamodels for displaying deltas. However, this requires more effort in synchronizing the editor states, e.g., regarding foldable or conditionally displayed sections or scroll bars.

Moreover, the approach for deriving diff editors produces 2-way diff editors (cf. Figure 3). 3-way diff editors can be derived by slightly adjusting the produced diff models, such that these contain a third "center" model part.

The diff overview of our approach currently contains a list of delta operations whereas the tree data structure for deltas can also be leveraged to calculate and display the deltas in a more detailed level. We made this decision, because it is in line with the diff editor that is opened when a delta is selected. If, in the future, we obtain feedback from modelers that a higher level of detail is desired in the diff overview, we will exchange the list view with a tree view, where sub items can be collapsed and expanded.

Semantically irrelevant changes, such as, the actual order of persisted array elements where the array order has no meaning in the abstract syntax, should not be presented to the users. Sometimes, the model type alone does not pose a meaning to a difference, but the implementation part in a low-code platform does. For instance, FMs have a list of annotations. Depending on the customization of the form engine in a concrete application, an annotation can be given a meaning and then, the order of FM annotations could be meaningful for the model interpretation. Hence, we consider all changes to the model that users can make via the projectional editors as semantically relevant. Any changes that do not modify the projectional representation of the model is not considered semantically relevant. This includes changes, which are not modifiable through the projectional editors that users make by directly editing the models.

B. Future Research Perspectives

Currently, our editors only support to inspect the differences between models, but they do not support model merging. However, making the diff editors (partially) editable is the foundation for realizing merge editors, which we will investigate in the future.

More generally, we are convinced that a holistic approach to model versioning and evolution in low-code platforms not only requires to model diff editors that enable a highly customized presentation of model differences, but we also want to customize our differencing facilities wrt. to the underlying edit operations that are meaningful from a user perspective. Again, we will draw from foundational work in the context of research on in MDD [21], [24]–[26]. This line of research relies on an executable yet declarative specification of model edit operations based on graph transformation concepts, enabling a bunch of further use cases for effectively managing model evolution. From an analytical perspective, low-level model differences comprising elementary changes (i.e., adding, deleting and modifying single model elements) may be semantically lifted to higher-level edit operations including model-type specific refactorings, thus providing a meaningful yet compact description of model differences serving as basis for further reasoning [21], [24]. From a constructive perspective, executable edit operations may be wrapped by editing commands which are being integrated into existing model editors, thus enabling a more sophisticated editing support as known from modern development environments in classical source code-centric development. Recent research results even combine the analytical and constructive perspective, e.g., for the sake of repairing models by complementing incomplete editing processes that leave a model (or pairs of interrelated models) in an inconsistent state [26].

The challenge in adopting all of these techniques in the context of low-code platforms is the same as for adopting techniques for model differencing: Existing research has mostly focused on the algorithmic perspective of the developed solutions and its empirical evaluation in terms of controlled experiments, while aspects such as integration into existing ecosystems, presentation and user interaction have been largely neglected. Our aim for future work is to bridge this gap by wrapping the underlying foundations and mostly formal notations by a modeling approach being accessible to the various stakeholders of a low-code development platform.

VIII. Related Work

Out of the variety of existing low-code platforms [6], [8], [27], we consider Salesforce [28], Appian [29], Pega [30], Mendix [31], OutSystems [32], and Microsoft Power Apps [33] to be the ones most related to A12. Similar to A12, Pega, Mendix, and Salesforce are low-code platforms to develop complex enterprise application that use visual data modeling, validation, and relations. In contrast to these low-code platforms, A12 relies on the data-first approach that allows to focus on modeling the data and flexibly use it for different applications. While low-code platforms can easily be used by software developers, A12 as well as Appian, Outsystems, Mendix, Salesforce, and Microsoft Power Apps also target citizen developers as their users. A model differencing approach focused on presenting differences in a way similar to editing the models like the one presented in this paper would also be applicable for these low-code platforms.

As already mentioned in the introduction, the scientific literature on model differencing mainly focuses on the algorithmic perspective, but largely neglects the aspect of presenting model differences in a user-friendly way. Nonetheless, there are a few tools that originate from an academic context and that enable basic forms of a difference presentation. The most prominent one, EMF Compare [34], provides a viewer for side-by-side changes as illustrated in Figure 3, however, by reusing a generic tree-based editor provided by the Eclipse Modeling Framework (EMF) that presents EMF models in terms if their abstract syntax. LemonTree [35] is a standalone model versioning tool integrated with the modeling tool Enterprise Architect [36]. Similar to the delta view in our approach, LemonTree contains a tree browser providing an overview of model differences. Moreover, detailed deltas can be inspected in a diagram visualization and in a property viewer. LemonTree has a rich feature set and is integrated with various technologies, e.g., version control systems. To the best of our knowledge, it does not focus on representing non-diagrammatic models in a notation close to the one for editing the models. The same limitation applies to the academic tool known as SiLift [37], whose difference display component cannot be generalized to non-diagrammatic models as used in low-code platforms.

A few academic papers have presented sketches of further diagrammatic representations of model differences, notably so-called unified diagrams [38] in the context of model merging [18], [39]. A unified document is a brute force merge of two models in the sense that all information contained in both models is united. In case of competing name changes, both names are shown side by side at this model element. Deleted model elements are not deleted, but only marked as deleted. However, to the best of our knowledge, none of these sketches has been ever implemented in a dedicated tool that has even come close to being mature enough to be used in practice. The reason for this is the complexity of an implementation of unified diagrams, which can hardly reuse an existing diagram editor of the underlying model type. Even if a unified document looks very similar to an original model it is actually a new type of model with a different meta-model which represents information about deltas between the models. On the contrary, our approach is centered around the requirement to build diff editors from reusable parts.

More generally, supporting software evolution, collaborative development, variability management, and version control have been identified as critical challenges for low-code platforms [40]–[42]. We argue that the foundation for integrating solutions to these challenges into low-code platforms is a suitable form of representation of model differences, such as, via the diff editors presented in this paper.

IX. Conclusion

For the purpose of being used within low-code platforms, classical model differencing approaches need to be rethought and adapted. In this paper, we presented an approach for calculating and displaying differences between models in the context of low-code platforms. The approach comprises UI elements that integrate both into the low-code development platform and its applications. The UI elements enable modelers to control the model difference calculation, to obtain an overview of the difference between two models, and to inspect the comprising deltas in detail. The latter uses a form of representation close to the one employed for creating the models. The UI elements as well as the data structure for model differences and the differencing processes are composed of reusable building blocks. This enables the integration of differencing facilities for new model types into the low-code platforms and fosters the applicability of differencing on the application level. The approach is planned to be integrated into the enterprise low-code platform A12 and it can be applied to other low-code approaches as well.

References

[1] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "An Introduction to Model Versioning," *Formal Methods for Model-Driven Engineering: 12th Intl. School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM 2012)*, pp. 336–398, 2012.

[2] K. Pohl, G. Böckle, and F. Van Der Linden, *Software Product Line Engineering*. Springer, 2005, vol. 10.

[3] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski, "A survey of variability modeling in industrial practice," in *7th Intl. Workshop on Variability Modelling of Software-intensive Systems*, 2013, pp. 1–8.

[4] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and Evaluation of Code Clone Detection Ttechniques and Tools: A Qualitative Approach," *Science of computer programming*, vol. 74, no. 7, pp. 470–495, 2009.

[5] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2013.

[6] A. Bucaioni, A. Cicchetti, and F. Ciccozzi, "Modelling in Low-Code Development: A Multi-Vocal Systematic Review," *Software and Systems Modeling*, vol. 21, no. 5, pp. 1959–1981, 2022.

[7] A. C. Bock and U. Frank, "Low-Code Platform," *Business & Information Systems Engineering*, vol. 63, pp. 733–740, 2021.

[8] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, "Supporting the Understanding and Comparison of Low-Code Development Platforms," in *46th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2020, pp. 171–178.

[9] C. Atkinson and T. Kuhne, "Model-Driven Development: A Metamodeling Foundation," *IEEE Software*, vol. 20, no. 5, pp. 36–41, 2003.

[10] D. Harel and B. Rumpe, "Meaningful Modeling: What's the Semantics of "Semantics"?" *IEEE Computer Journal*, vol. 37, no. 10, pp. 64–72, October 2004.

[11] M. Völter, J. Siegmund, T. Berger, and B. Kolb, "Towards User-Friendly Projectional Editors," in *7th Intl. Conf. on Software Language Engineering*. Springer, 2014, pp. 41–61.

[12] K. Altmanninger, P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "Why model versioning research is needed!? an experience report," in *MoDSE-MCCM Workshop @ MoDELS*, vol. 9, 2009, pp. 1–12.

[13] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, "Different Models for Model Matching: An Analysis of Approaches to Support Model Differencing," in *ICSE Workshop on Comparison and Versioning of Software Models*. IEEE, 2009, pp. 1–6.

[14] H. Mehmanesh, S. Gandenberger, A. Weiss, T. Kneist, and S. Lorenz, "Whitepaper A12 Enterprise Low Code," mgm technology partners GmbH, Tech. Rep., 2022.

[15] F. Brooks and H. Kugler, *No Silver Bullet*. April, 1987.

[16] A. Butting, O. Kautz, B. Rumpe, and A. Wortmann, "Semantic Differencing for Message-Driven Component & Connector Architectures," in *Intl. Conf. on Software Architecture*. IEEE, 2017, pp. 145–154.

[17] S. Maoz, J. O. Ringert, and B. Rumpe, "Cddiff: Semantic differencing for class diagrams," in *25th European Conference Object-Oriented Programming*. Springer, 2011, pp. 230–254.

[18] M. Alanen and I. Porres, "Difference and Union of Models," in *UML 2003-The Unified Modeling Language. Modeling Languages and Applications: 6th Intl. Conf.* Springer, 2003, pp. 2–17.

[19] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An Exploratory Study of Cloning in Industrial Software Product Lines," in *17th European Conference on Software Maintenance and Reengineering*. IEEE, 2013, pp. 25–34.

[20] E. Gamma, R. Helm, R. Johnson, R. E. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson, 1995.

[21] T. Kehrer, U. Kelter, and G. Taentzer, "A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning," in *26th Intl. Conf. on Automated Software Engineering*, 2011, pp. 163–172.

[22] P. Langer, M. Wimmer, P. Brosch, M. Herrmannsdörfer, M. Seidl, K. Wieland, and G. Kappel, "A posteriori operation detection in evolving software models," *Journal of Systems and Software*, vol. 86, no. 2, pp. 551–566, 2013.

[23] K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches," in *2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, vol. 45, no. 3, 2003, pp. 1–17.

[24] T. Kehrer, U. Kelter, and G. Taentzer, "Consistency-preserving edit scripts in model versioning," in *28th Intl. Conf. on Automated Software Engineering*. IEEE, 2013, pp. 191–201.

[25] D. Strüber, K. Born, K. D. Gill, R. Groner, T. Kehrer, M. Ohrndorf, and M. Tichy, "Henshin: A usability-focused framework for emf model transformation development," in *10th Intl. Conf. on Graph Transformation*. Springer, 2017, pp. 196–208.

[26] M. Ohrndorf, C. Pietsch, U. Kelter, L. Grunske, and T. Kehrer, "History-based model repair recommendations," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 2, pp. 1–46, 2021.

[27] Best Low-Code Development Platforms. [Online]. Available: https://www.g2.com/categories/low-code-development-platforms

[28] Salesforce App Cloud Platform Overview. [Online]. Available: https://developer.salesforce.com/platform

[29] Appian platform overview. [Online]. Available: https://www.appian.com/

[30] Pega Low Code App Development Overview. [Online]. Available: https://www.pega.com/de/products/platform/low-code-app-development

[31] Mendix Platform Features. [Online]. Available: https://www.mendix.com/platform/

[32] Outsystem Platform Features. [Online]. Available: https://www.outsystems.com/platform/

[33] Microsoft Power Apps Platform Overview. [Online]. Available: https://docs.microsoft.com/en-us/powerapps/maker/

[34] C. Brun and A. Pierantonio, "Model differences in the eclipse modeling framework," *UPGRADE, The European Journal for the Informatics Professional*, vol. 9, no. 2, pp. 29–34, 2008.

[35] LieberLieber Lemon Tree Website. [Online]. Available: https://www.lieberlieber.com/lemontree

[36] Enterprise-Architect Visual Modeling Platform. [Online]. Available: https://www.sparxsystems.eu/enterprise-architect

[37] T. Kehrer, U. Kelter, M. Ohrndorf, and T. Sollbach, "Understanding model evolution through semantically lifting model differences with silift," in *28th Intl. Conf. on Software Maintenance*. IEEE, 2012, pp. 638–641.

[38] D. Ohst, M. Welle, and U. Kelter, "Differences between versions of uml diagrams," in *9th European Software Engineering Conf. held jointly with 11th ACM SIGSOFT Intl. Symp. on Foundations of Software Engineering*, 2003, pp. 227–236.

[39] A. Mehra, J. Grundy, and J. Hosking, "A generic approach to supporting diagram differencing and merging for collaborative design," in *20th Intl. Conf. on Automated Software Engineering*, 2005, pp. 204–213.

[40] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer, "Low-Code Development and Model-Driven Engineering: Two Sides of the Same Coin?" *Software and Systems Modeling*, vol. 21, no. 2, pp. 437–446, 2022.

[41] K. Rokis and M. Kirikova, "Challenges of Low-Code/No-Code Software Development: A Literature Review," in *Perspectives in Business Informatics Research: 21st Intl. Conf. on Business Informatics Research*. Springer, 2022, pp. 3–17.

[42] A. Bragança, I. Azevedo, N. Bettencourt, C. Morais, D. Teixeira, and D. Caetano, "Towards Supporting SPL Engineering in Low-Code Platforms Using a DSL Approach," in *20th Intl. Conf. on Generative Programming: Concepts and Experiences*, 2021, pp. 16–28.