The Journal of Systems and Software 222 (2025) 112285

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

SPL-DB-Sync: Seamless database transformation during feature-driven changes☆

Delfina Ramos-Vidal [a,*], Wesley K.G. Assunção [b], Alejandro Cortiñas [a], Miguel R. Luaces [a], Oscar Pedreira [a], Ángeles Saavedra Places [a]

[a] Universidade da Coruña, Centro de Investigación CITIC, Database Lab., Elviña, 15071, A Coruña, Spain
[b] Department of Computer Science, North Carolina State University (NCSU), Raleigh, USA

ARTICLE INFO

Keywords:
Software evolution
Variability management
Database management
Data synchronization

ABSTRACT

Software Product Line (SPL) Engineering is a reuse-oriented approach to developing a suite of software products that share common components but vary in specific features. The advantages of SPLs (e.g., reducing development costs and time while improving quality) have already been proven in practice. However, despite the success in deriving new products from an SPL, challenges arise in evolving existing products. Altering the feature selection (e.g., adding or removing a feature) for an already existing product poses a challenge regarding the application data stored and managed by derived products, particularly when the features impact an already populated database. In many cases, these modifications imply loss of data or constraint violations. However, in both the state of the art and practice, there are no approaches to support feature and data evolution simultaneously for SPL products.

This paper reports a novel evolution approach, SPL-DB-Sync, with actions required for database adjustments when adding or removing features for existing SPL products. Actions delineate modifications necessary within the database. These modifications are associated with the SPL features and linked to the components of the data model they influence. SPL-DB-Sync facilitates the automatic readjustment of the database while preserving clear traceability between features and elements of the data model. The applicability of our evolution model is detailed in four practical scenarios of in-production products of an SPL for Digital Libraries. The contributions of this work are: present a novel evolution approach for SPLs with databases; define an SPL Evolution Model considering data transformation/migration; advance the state of practice between software reuse and data management; and provide insights for practitioners that face the same challenges of evolving both business logic and its data in software products.

1. Introduction

Software Product Line Engineering (SPLE) refers to the application of methods, tools, and techniques for creating a collection of software products sharing a common foundation but differing in specific features (Clements and Northrop, 2002). A Software Product Line (SPL) allows engineers, or even end users, to create tailored products by specifying the desired features (Pohl et al., 2005). After selecting features, different variability mechanisms can be used to orchestrate the assembly and customize software components accordingly. Such variability mechanisms can be either *annotative*, which indicate pieces of code that should be compiled/included (e.g., `ifdef` preprocessor directives), or *compositional*, in which separated reusable assets are composed during derivation (e.g., aspect-oriented programming) (Kästner et al., 2008; Apel et al., 2013). In addition to leveraging systematic reuse of software components, SPLs are also used to derive database schemas that meet the needs of practitioners of specific domains or market segments (Khedri and Khosravi, 2013; Bouarar et al., 2015). However, while SPLs offer a systematic strategy for creating software products with shared components, the dynamic nature of modern applications poses challenges in effectively managing the evolution of product configurations (Laguna and Crespo, 2013; Marques et al., 2019). This evolution is especially challenging when changes impact not only the product logic but also the existing database schemas and populated databases (Herrmann et al., 2015; Rhizadi et al., 2019).

During the evolution of an SPL, when a new configuration of features is required for a product that was already derived, the data already existing in the database (i.e., the already populated database)

☆ Editor: Laurence Duchien.

* Corresponding author.
E-mail address: delfina.ramos@udc.es (D. Ramos-Vidal).

https://doi.org/10.1016/j.jss.2024.112285
Received 5 February 2024; Received in revised form 4 October 2024; Accepted 15 November 2024
Available online 2 December 2024
0164-1212/© 2024 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

must be transformed accordingly. Database schema evolution as a reaction to software system evolution has been acknowledged as a practical problem for decades (Roddick, 1995) because of the need to retain current data. Thus, a challenge for SPL engineering lies in automatically transforming or migrating data as a reaction to new feature selection. As the domain or market segment requirements evolve, changes in feature selection become inevitable, with consequences for the entire SPL (Marques et al., 2019; Ataei et al., 2021). Modifying the feature selection of a product generated using an SPL already in production entails changes to various system artifacts. Software components can be relatively easily regenerated and redeployed to support new features (Pohl et al., 2005). However, adapting the database is more intricate, especially when considering that the product is actively in use and contains crucial data. Our central inquiry focuses on the complexities that arise when changes to feature selections ripple through and alter the database structure.

While evolution is an important aspect of SPL research, the database layer is typically not considered, and vice versa. The existing literature addresses the evolution of source code artifacts but neglects the consideration of how these changes affect the database schema (Laguna and Crespo, 2013; Marques et al., 2019). Furthermore, in scenarios where the product is actively used and the database is populated, the change not only requires a modification in the data model but also involves data transformation/migration.

To fill the existing limitations in the state of the art and practice, we delve into the intricacies of how the evolution of product variants within an SPL influences the database schema and examine the propagation of these modifications across different versions of a software product. More specifically, in this work, we present a novel semi-automated approach for seamless database adjustments when adding or removing features for existing SPL products. Our approach is composed of a metamodel that enables engineers to outline the actions that need to be performed in existing databases during product evolution. These actions are intricately linked to the SPL features and associated with the components of the data model they influence. The scripts generated for selecting or deselecting features are designed to apply to all products within the SPL, ensuring consistency and efficiency across the entire product line. Our approach also supports the automatic readjustment of the database while maintaining clear traceability between the features in the feature model and the possible changes in their selection. The applicability of this evolution model is evaluated through real-world use cases. We describe how our approach can be applicable in four scenarios involving in-production products in the Digital Libraries domain.[1]

SPL-DB-Sync builds upon our previous work (Cortiñas et al., 2023), which introduced a preliminary model to capture changes in feature selection for already-derived SPL products and the subsequent actions needed to adjust the product database. This initial solution relied on the domain engineer to explicitly define the evolution of the database based on the changes in product configuration. Recognizing the limitations of this approach, we saw the need for a more sophisticated and automated method. The SPL-DB-Sync approach offers several contributions: it addresses the intricate challenge of evolving already-derived products with populated databases; systematically classifies modifications to feature selections and connects them to corresponding changes in the database schema; ensures clear traceability, transparency, and adaptability of evolution changes; and facilitates the reuse of data transformation actions. The structured model aids decision-making throughout the evolution process, ensuring data consistency and alignment with evolving requirements. SPL-DB-Sync provides a structured and practical solution for practitioners managing the evolution of derived products, laying the groundwork for future research to refine and extend methodologies in this domain. Despite its promising results, the approach has some limitations, including a primary focus on database schema evolution, potential user-dependent challenges, and the need for future work to optimize the evolution model.

The remainder of this paper is organized as follows: Section 2 motivates our work and describes the practical problem we address. In Section 3, we review previous research, laying the foundation for upcoming discussions. Section 4 introduces SPL-DB-Sync, delving into its intricacies and real-world applications. Section 5 provides a practical illustration within a digital library SPL, evaluating its effectiveness through specific use cases. Finally, Section 6 discusses the implications and limitations, while Section 7 summarizes key findings and contributions, outlining potential future research directions in SPL evolution and database management.

2. Motivating example and problem statement

To illustrate the complexities of evolving SPL variants and their associated databases, we present a simplified example within the context of a Human Resources (HR) Management SPL based on the Oracle HR Sample Schema (Oracle, 2019). This SPL caters to diverse organizations, each requiring tailored solutions for managing their departments and employees. Fig. 1 provides an overview of the Feature Model (FM) for this SPL, in which each organization can configure its HR system based on specific needs. Departments may or may not have managers (`Manager`), and they may possess zero, one, or multiple locations (`Location`, `OneLocationOnly`, `ManyLocations`). Employees, on the other hand, must be linked to departments, with the flexibility to work in one or more departments simultaneously (`CanWorkInManyDepartments`). To ensure secure access to the system, all employees require passwords, which can be stored in plain text or encrypted using MD5 (`PasswordType`, `Plain`, `MD5`). The model introduces variability in the retribution structure for employees (`Retribution`). They can receive a fixed salary (`FixedSalary`), a commission on sales (`Comision`), or a combination of both, allowing for diverse remuneration schemes. Additionally, certain employees may opt for a commission-only compensation structure, resembling collaborators who are not regular employees. Others, like partners of the organization, may forego both commission and salary.

Nevertheless, stakeholder needs are subject to change, leading to potential modifications in the product configuration. Take, for example, an organization that has been using a software product created with the HR-SPL to manage its departments and employees. Initially, an organization configures the HR system to allow employees to work in only one department. Now, consider a situation where the organization undergoes substantial growth, triggering the emergence of a new requirement: employees need the flexibility to work in multiple departments simultaneously. This organizational change is challenging when it comes to modifying the already-derived products with populated databases to accommodate this new relationship between employees and departments while supporting the principles of SPLE.

Fig. 1 also illustrates the evolution of the HR-SPL. Initially, the database schema reflected a one-to-many relationship between employees and departments because the feature model did not include `CanWorkInManyDepartments`. With the addition of this feature, the schema must now undergo a significant transformation to model the new many-to-many relationship. This requires creating an intermediate table to store and manage these complex associations, involving substantial database schema evolution and data migration.

Based on the discussion above, we realize that the evolution of SPL products with populated databases faces two prevalent challenges:

Challenge 1: Current methodologies primarily address the evolution of source code artifacts (Montalvillo and Díaz, 2023; Mitschke and Eichberg, 2008; Hoff et al., 2020), neglecting the implications for the

[1] The products referenced in this study are intellectual property of a third party, and we are not authorized to share them. Additionally, the execution framework is not yet open-source, but we plan to release it in the future according to the terms of our funding.

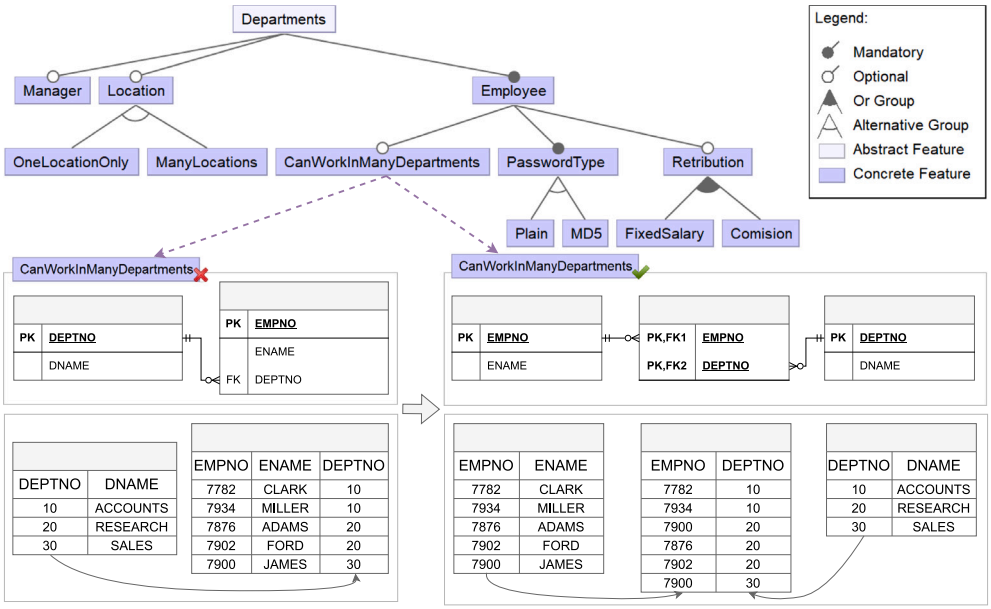

Fig. 1. Motivating example.

database schema. This complexity goes beyond merely integrating the feature into the existing product; it requires a modification of the database schema, influencing the storage and retrieval of information concerning employee-department relationships.

Challenge 2: When the product is already in production and actively used, changes in its configuration demand not only modifications in the database schema but also entail data migration, which involves transferring data between different storage systems. Additionally, in some cases, such migration requires data transformation, which entails converting data from one format or structure to another.

This example illustrates the main challenges that we addressed in our work. Despite being a prevalent problem in practice, the existing literature does not present solutions to these challenges, as discussed in the next section.

To address these challenges, we present a solution that facilitates the seamless evolution of existing SPL variants and their associated databases when new organizational requirements impact fundamental aspects of the system architecture. Our evolution model aids engineers by minimizing the risk of human errors, enhancing data consistency, and simplifying the process of keeping the system aligned with the organization's evolving requirements. Consequently, it reduces the overall effort needed to update and maintain the product.

3. Related work

We present the existing related work grouped in two categories, namely studies on SPL evolution and database management as part of the evolution (Section 3.1) and database schema management (Section 3.2). Then, we summarize the main challenges that still need to be addressed (Section 3.3).

3.1. SPL evolution and database management as part of the evolution

SPLE emphasizes the creation of reusable and configurable software artifacts (Clements and Northrop, 2002). It has proven crucial to efficiently develop software products with a common foundation and adaptable features (Pohl et al., 2005). SPL evolution has gained prominence in software engineering research, covering various essential topics for successful product line development and maintenance (Laguna and Crespo, 2013; Marques et al., 2019; Güthing et al., 2024; Krieter et al., 2023). SPLs efficiently produce a variety of software products through systematic reuse of common components and features (Apel et al., 2013). However, the evolution of SPLs, especially the evolution of products derived from them, introduces intricate challenges. This complexity becomes apparent when feature selections change, which subsequently influences the underlying database schema (Herrmann et al., 2015; Rhizadi et al., 2019). In what follows, we first present pieces of work on SPL evolution, and then the studies that introduce database management as part of this evolution.

The predominant focus in the existing literature revolves around the evolution of SPL in response to changing domain requirements. Montalvillo and Díaz (2023) provide an overview of the challenges encountered during SPL evolution. The extended life cycles of SPLs inherently involve changing customer needs over time. However, the intricate nature of SPLs may impede SPL products from adapting swiftly, potentially jeopardizing one of the key advantages of SPLs: decreasing time to market. Notably, their approach concentrates on the evolution of SPLs and the propagation of changes from the feature model (FM) to the product, yet it does not directly address the specific challenge we are tackling: evolving the feature selection of an already-derived product.

Thüm et al. (2009) have categorized alterations to FMs into four distinct groups based on their impact on valid configurations. The first category, labeled "Refactorings", encompasses changes that do not add or delete any valid settings, but instead optimize or restructure the FM itself. The second category, "Generalizations", pertains to modifications introducing additional valid configurations, thereby expanding the list of permissible setups. In contrast, the third category, "Specializations", involves changes that eliminate previously viable configurations, reducing the pool of potentially valid items. The fourth category, "Arbitrary Edits", includes both the addition of new valid configurations and the removal of previously acceptable ones. Their proposal provides an algorithm to determine the relationship between two FMs, relying on a retrospective analysis of inconsistencies between two FM snapshots but does not provide a strategy to perform the evolution. They propose that by clearly modeling the evolution process, computational tasks related to FM evolution management could be executed more efficiently. Nonetheless, their solution specifically addresses alterations to the FM and does not extend to examining their impact on the generated products.

Mitschke and Eichberg (2008) present a versioning model that establishes traceability among features, products, and artifacts through

feature-driven versioning. Their solution involves creating versioned containers within the project repository, and linking features to artifacts. The approach offers versatility in its application, accommodating both feature versions and implementation artifact versions. This model serves as a foundational framework for handling the evolution of SPLs, offering traceability information to ensure maintainability and data consistency. However, it is important to note that their solution primarily addresses the versioning of artifacts and does not extend to scenarios where the database schema is impacted by feature evolution.

Schwägerl and Westfechtel (2023) also present a conceptual framework and tool support to manage evolving SPLs, SuperMod. Their framework supports logical, historical, and collaborative versioning, where every change is connected to the set of variants it affects. The paradigm underlying SuperMod is denoted as product-based product line engineering, which blurs the distinction between domain and application engineering. However, it is essential to note that this solution heavily relies on Model-Driven Software Engineering (MDSE), introducing challenges in terms of complexity and learning curve for development teams not accustomed to MDSE methodologies. Furthermore, the approach primarily focuses on managing artifacts and versions without considering the intricate impact of evolving SPLs on the data model. Their framework prototype demonstrated promising results, but their work does not encompass the broader spectrum of data management challenges that arise when evolving products within an SPL. Thus, their conceptual framework is limited to explicitly addressing the challenges arising from database schema modifications.

3.2. Database schema management

In the realm of database schema variability management, researchers explored techniques to address the challenge of how feature changes affect the database (Siegmund et al., 2009). Khedri and Khosravi (2013) propose an innovative method that uses delta-oriented programming (DOP) and SQL Data Definition Language (DDL). Their approach involves creating core and delta modules to handle mandatory and optional features, respectively, allowing for the generation of consistent database schemas tailored to specific feature selections. This method emphasizes scalability, modularity, and minimal reliance on specialized tools, which makes it highly practical for industrial applications. Their methodology is validated through a case study on university software systems, demonstrating its effectiveness in real-world scenarios.

Humblet et al. (2016) introduce the SVL Tool, designed to manage variability in database applications using the Simple Variability Language (SVL). Integrated with the DB-Main CASE tool, SVL enables explicit modeling of feature variability and its mapping to database schema elements. The tool facilitates the generation of specific database schemas based on selected features, as evidenced by its application to the OSCAR Electronic Medical Record system. This approach underscores the importance of variability modeling in legacy database systems, providing a systematic solution for generating customized schemas to meet diverse deployment needs.

The work by Herrmann et al. (2015) proposes a database evolution toolkit for SPLs, DAVE, that enables the definition of the diff between the database schema of two products. They use a DSL to specify the data model of each component of the SPL, and the Database Development Tool generates SQL templates for the evolution that the developer must manually fill with the actual data to be migrated. Additionally, they do not differentiate the types of actions, so users must check the script manually to ensure no data loss occurs. In contrast, we propose an automated schema evolution system that does not require manually defining the differences between database schemas. Instead, changes to the database are associated with each feature in the feature model. Even if a manual definition of changes is required, it is done once in conjunction with the feature model, allowing automatic generation of migration scripts each time a product evolves. This eliminates the need to manually fill templates for every migration, reducing the risk of human error and improving efficiency. Moreover, our system provides an informed solution by clearly differentiating the types of actions, helping users avoid unintentional data loss.

3.3. Summary

Various approaches in the SPL literature have provided valuable insights into the evolution of SPLs and the management of their artifacts. A few studies have considered databases as part of the evolution, however, there remains a significant gap in addressing the intricate challenges posed by evolving already-derived products with populated databases. Existing methodologies, though comprehensive in their coverage of FM evolution, fall short when it comes to explicit modifications to the database schema, especially in scenarios where the products are already in use and contain data. The challenges introduced by changes in feature selections and their impact on the data model (as discussed in Section 2) require dedicated solutions to ensure data consistency and seamless evolution. This work aims to fill this gap by introducing a novel evolution model explicitly tailored to address the challenges of evolving already-derived products with populated databases, emphasizing the need for comprehensive solutions that extend beyond the traditional focus of the existing literature.

4. The SPL-DB-Sync approach

To address the practical problem of evolving SPL products with populated databases and fill the gap with existing work in the literature, in this section, we present our approach, SPL-DB-Sync. Illustrated in Fig. 2, our solution is designed to effectively manage changes in SPL feature selections that impact the database schema, ensuring accurate and efficient synchronization.

SPL-DB-Sync is a semi-automated approach that consists of two steps, *Step 1. Evolution Model Definition* and *Step 2. Product Evolution*. This latter step (Step 2) is composed of two sub-steps. In Step 1, the SPL engineer manually defines the evolution model for all SPL features. For that, the engineer uses the FM of the product line and its corresponding database schema as primary sources of information, considering how the variability of the FM may impact the evolution model. The evolution model conforms to the evolution metamodel (presented later in Section 4.2), which describes how adding or removing a feature from the feature selection of an existing product affects its database schema. The engineer also outlines the actions required to implement these changes, with each action linked to a corresponding script, which contains the necessary database migration operations. These scripts are executed during *2.1. Data Transformation and Migration*, which forms part of the semi-automated Step 2. Step 2 requires specific inputs: (i) an existing product with a populated database and implementation artifacts (e.g., source code and configuration files) to be evolved; (ii) a new feature selection that is the focus of the evolution; and (iii) the evolution model produced in the first step. Based on the evolution model, SPL-DB-Sync knows which changes in the product database and the data must be performed, which is part of the automated sub-step *2.1. Data Transformation and Migration*. The sub-step *2.2. Artifact Generation* is the automatic re-assembling (i.e., re-derivation or re-generation) of the product artifacts for the new feature selection. The output of SPL-DB-Sync is an evolved product, implementing new features and keeping the data consistent.

SPL-DB-Sync addresses the two described challenges (Section 2). Firstly, it orchestrates the adaptation of the database schema to accommodate new feature selections, thereby addressing Challenge 1. This involves understanding and executing the necessary database schema modifications based on the selected features. Secondly, SPL-DB-Sync manages data migration and conditional behaviors inherent in the feature changes, ensuring data integrity throughout the evolution process, which aligns with Challenge 2.

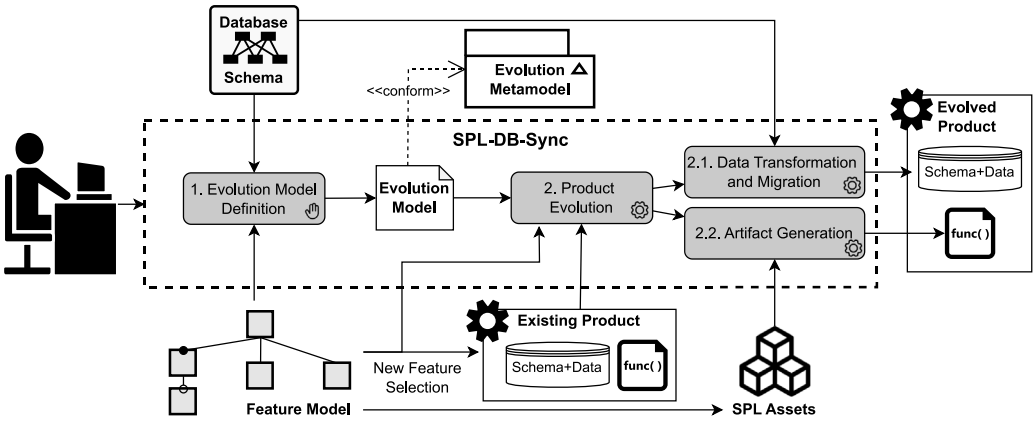

Fig. 2. Overview of the SPL-DB-Sync to evolve SPL products with populated databases.

4.1. Preliminary approach

SPL-DB-Sync is an extension of our previous work (Cortiñas et al., 2023), which presented a preliminary model to capture changes in the feature selection of already-derived SPL products and also the subsequent actions required to adjust the product database. This initial solution operated under the assumption that the domain engineer must explicitly define the evolution of the database resulting from changes in the product configuration. While serving as an initial exploration, we recognized the need for a more sophisticated, automated, and intelligent approach. In the following, we describe the evolution of our work, leading to the development of the SPL-DB-Sync approach, and we showcase its effectiveness through practical use cases.

An SPL provides the mechanisms and assets to assemble the desired products. However, after a product is assembled (i.e., derived or generated), it may become necessary to modify the selected features (Krieter et al., 2023). To tackle this, in our preliminary approach, we model the changes in feature selection and how they affect the underlying database schema. This involves actions that impact entities, properties, and relationships within the database schema. Our approach considers three main requirements: (i) it should enable the definition of potential modifications to feature selection, even if they result in data loss; (ii) it should determine which modifications are feasible and how they connect to particular features while preserving traceability between the modifications to feature selection and the data model elements affected by those modifications; and (iii) it should leverage the reuse of data transformation actions across various features. Based on these requirements, we define a preliminary model as part of our previous work, which is presented in Fig. 3. The metamodel represents both the FM (at the top of Fig. 3) and the database schema (at the bottom). In the middle of Fig. 3, we have two main elements used by our approach, which are "Change" and "Action", described in detail below.

According to the first requirement above, modifications may or may not have an impact on the database of the existing products that will be evolved. Considering this, our approach categorizes potential modifications in feature selection based on their impact on database schema and data, delineating them into four different "Change Types", as follows:

- *NOOP* (*No Operation*): This change type indicates that the change does not affect the data or the database schema; hence, no database schema adaptation actions are necessary to carry out this change.
- *ALLOWED*: This change type indicates that the change is permissible (i.e., the database schema can be changed) and does not result in data loss, implying that database modifications can be carried out while maintaining all existing data.
- *MISSING DATA*: This change type indicates that the modification to the feature selection of an existing product is permitted, but will result in the loss of some data. The system informs the user of the specific data that will be lost through the chosen interface, ideally the SPL feature selection interface. Given the traceability between the changes and the database schema elements that are affected, engineers have information about specific data that may be lost.
- *FORBIDDEN*: This change type indicates that the change is not permitted for a given SPL product. While our approach allows for various changes, it also provides the flexibility to prohibit certain changes if necessary, keeping the consistency of SPL products and their data. An example of a change of this type would be deselecting simultaneously the features `Plain` and `MD5` in the illustrative example in Fig. 1 because that would imply leaving the database and the product without an attribute to store the password. *FORBIDDEN* changes may be redundant because they can probably be represented by cross-tree constraints in the feature model. However, we believe including this information in the evolution model is informative for the engineer. Additionally, SPL engineers might use the FORBIDDEN change type to restrict the possibility of changing the selection of certain features as part of the SPL design decisions.

Each "Change" comprises a set of fundamental "Actions", as represented by the composition relationship between `Action` and `Change` in Fig. 3. Actions are responsible for performing modifications to elements within the database schema when the change type is *ALLOWED* or *MISSING DATA*. For example, considering our illustrative example in Fig. 1, the change of including the `CanWorkInManyDepartments` is *ALLOWED*, and requires several actions, such as the creation of the `EMP_DEPT` table. A change can consist of one or multiple actions. In the latter case, the association between a change and its constituent actions defines the sequence in which these actions should be executed, as defined by `Order` in Fig. 3. Decomposing a change into multiple actions allows us to reuse modifications that may be part of different changes, thereby enhancing efficiency.

Our approach maintains traceability in two key aspects. Firstly, it ensures traceability between features and the changes they trigger when selected or deselected. This linkage is explicitly managed within our approach, where each feature is associated with specific changes that declare the feature name they are related to. Secondly, we place equal importance on traceability between changes and the elements within the database schema that are affected by those changes. This traceability is facilitated by our metamodel, which defines a relationship called "affects" (see Fig. 3) that connects `Action`, in the evolution model, to `Entity`, `Property`, or `Relationship` in the database schema. The evolution model offers a clear and structured understanding of each modification's consequences and ensures

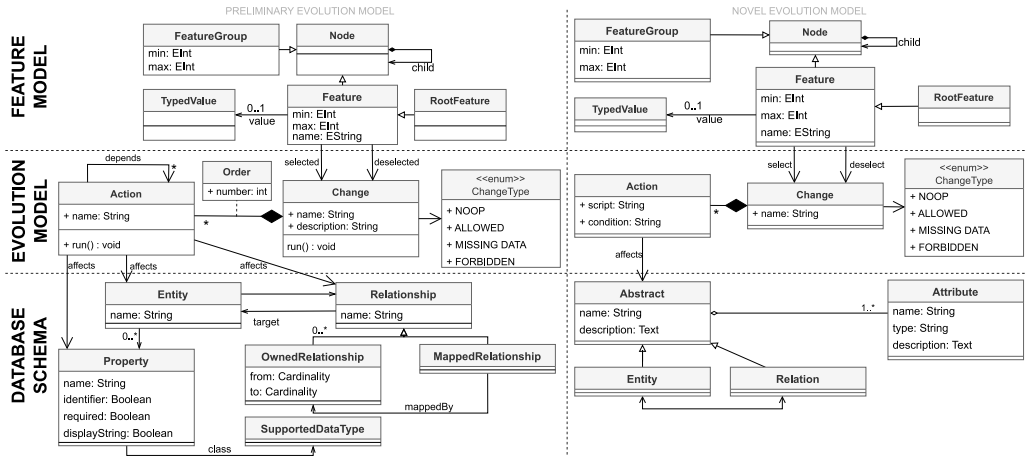

Fig. 3. Metamodel of preliminary and novel evolution models.

users are aware of the nature and implications of changes, maintaining traceability between features, changes, and affected database elements. The approach's step-by-step process enhances transparency by outlining a clear roadmap for evolution activities. In our current representation, we assume that the database schema is structured using Entity-Relationship (ER) notation. Nevertheless, our approach could be adapted to work with different modeling paradigms as long as the relation between features and database elements can be modeled. It can also be easily modified to represent the database schema with a different notation or even to directly consider the logical model of the database (i.e., establishing relationships between changes and tables, columns, and keys). In addition to the flexible schema representation, adaptability is achieved through the reusable evolution model, automated data transformation, and modular actions that support easy updates and modifications.

One of the key issues when it comes to identifying and implementing changes in the database schema according to the feature selections, which affect the database schema, is the order of execution of these changes. This issue was addressed in our preliminary approach by identifying all possible existing changes. In other words, for each set of features that modify the same part of the database schema, all possible combinations of changes were identified. Managing the execution order in this way leads to a combinatorial explosion, considering the possible combinations of features in an FM. This is specifically harder for FMs with several features affecting the same part of the database schema. Thus, managing all possible changes becomes quite challenging.

We use the example based on the management of departments and employees (see Section 2) to examine the modifications that must be made to the database schema in order to accommodate the newly chosen features using our previous approach. For example, when dealing with a single feature, we model two possible changes: (i) we model going from deselected to selected, which typically results in an *ALLOWED* change, or (ii) from selected to deselected, which typically results in a *MISSING DATA* change. Since our previous approach identified all possible combinations and interdependencies of changes, we registered each change in the state of a feature, from selected to deselected. Table 1 describes the two possible combinations for feature `Manager`. This feature does not present any problems because it is independent of any other feature. Furthermore, `Manager` does not have dependent children.

Issues arise when the behavior of features becomes interdependent due to constraints or hierarchical requirements, dictating the existence of one feature before another. Table 2 illustrates all conceivable combinations resulting from alterations in the features `Plain` and `MD5`, which belong to the feature `PasswordType` (see Fig. 1). Notice that we intentionally violate FM constraints to demonstrate all potential cases. For instance, the absence of passwords (where neither `Plain` nor `MD5` is selected) is invalid, as the `PasswordType` feature is mandatory. Conversely, in the presence of a password system, only one of these features can be selected at a time. Consequently, any combination where both `Plain` and `MD5` are chosen is labeled as `FORBIDDEN`. This example computes the potential combinations of two interrelated features. Since each feature can be in one of two possible states (selected or deselected), the total number of combinations is $2^{2\times2} = 2^4 = 16$. Although some of these combinations result in NOOP (no operation) due to the lack of changes in the status of a feature, they are not represented in Table 2.

For every feature in our FM, we must consider two potential states: selected or deselected. Given the scenario of our LPS-BIDI (described in detail in Section 5), we have a group of six alternative features (as shown in Fig. 4), which originate from the feature `LO_E_DR_FileTypeSupported`. Since we have six features, each of which can either be selected or deselected (two states), we need to compute all possible combinations of these states. This results in $2^{6\times2} = 2^{12} = 4096$ possible combinations. Recognizing the exponential growth in configurations, we identified a significant combinatorial explosion problem, dependent on the FM design. To address this, our approach does not rely on feature dependencies to structure the evolution model. Instead, it interprets each feature as an independent element whose behavior may be conditioned by the presence or absence of other features in the feature selection. By treating each feature independently,

Table 1
Changes and combinations of Manager feature.

Manager	*Change*	*Change type*
$\neg Manager \rightarrow Manager$	SelectManager	ALLOWED
$Manager \rightarrow \neg Manager$	DeselectManager	MISSING DATA

Table 2
Changes and combinations of Password features.

Plain	MD5	*Change*	*Change type*
$\neg Plain \rightarrow Plain$	$\neg MD5 \rightarrow \neg MD5$	SelectPlain	ALLOWED
$\neg Plain \rightarrow \neg Plain$	$\neg MD5 \rightarrow MD5$	SelectMD5	ALLOWED
$\neg Plain \rightarrow Plain$	$\neg MD5 \rightarrow MD5$	Impossible	FORBIDDEN
$\neg Plain \rightarrow \neg Plain$	$MD5 \rightarrow \neg MD5$	Impossible	FORBIDDEN
$\neg Plain \rightarrow Plain$	$MD5 \rightarrow \neg MD5$	MigrateToPlain	MISSING DATA
$\neg Plain \rightarrow Plain$	$MD5 \rightarrow MD5$	Impossible	FORBIDDEN
$Plain \rightarrow \neg Plain$	$\neg MD5 \rightarrow \neg MD5$	Impossible	FORBIDDEN
$Plain \rightarrow \neg Plain$	$\neg MD5 \rightarrow MD5$	MigrateToMD5	ALLOWED
$Plain \rightarrow Plain$	$\neg MD5 \rightarrow MD5$	Impossible	FORBIDDEN
$Plain \rightarrow \neg Plain$	$MD5 \rightarrow \neg MD5$	Impossible	FORBIDDEN
$Plain \rightarrow Plain$	$MD5 \rightarrow \neg MD5$	Impossible	FORBIDDEN
$Plain \rightarrow \neg Plain$	$MD5 \rightarrow MD5$	Impossible	FORBIDDEN
$Plain \rightarrow Plain$	$MD5 \rightarrow MD5$	Impossible	FORBIDDEN

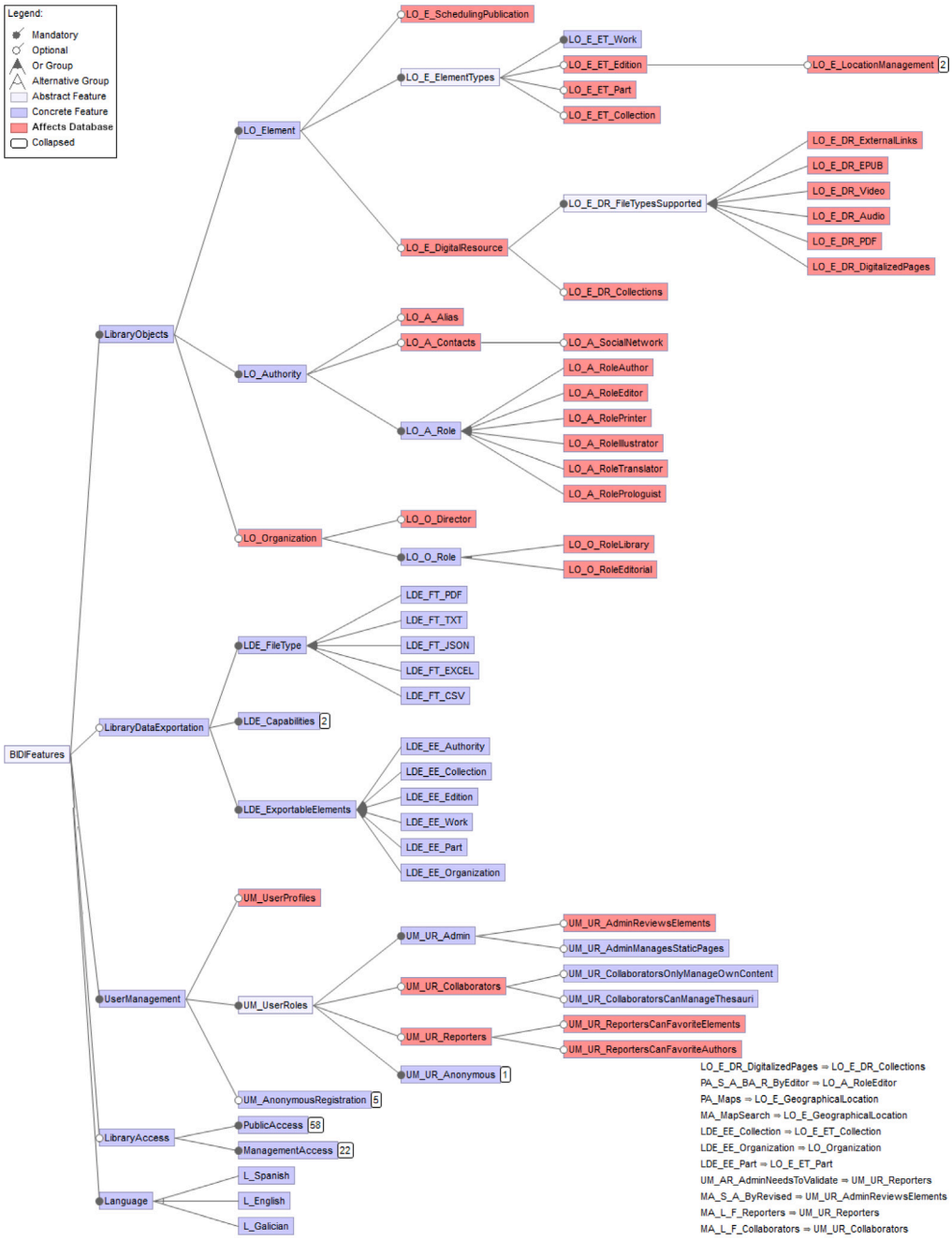

Fig. 4. Feature model for LPS-BIDI, with features related to the database schema highlighted in red.

we ensure that we only have to consider two states per feature (selected or deselected) without having to account for the combinations of states with correlated features. By interpreting each feature as independent and applying a condition that checks whether any of the other features have been selected previously or not, we reduce the complexity. This strategy allows us to manage the feature states more efficiently and avoids the need to evaluate an exponentially larger number of combinations, thus mitigating the combinatorial explosion.

4.2. Novel approach

In our preliminary work, the suggested multi-model facilitated traceability of changes, offering flexibility with the ability to leverage various modeling paradigms. Nonetheless, limitations persisted, notably in the hierarchical relationship between features and changes being overlooked. Furthermore, the employed database schema deviated from standard data models. To address these drawbacks, we present a novel approach in this contribution. This new approach incorporates clear semantics through an algorithm and adopts a standard data schema, such as the ER model schema, aiming to overcome the identified issues.

To address the identified issues, we modified our approach by modeling changes at the individual feature level, regardless of their interdependencies with other features. The right side of Fig. 3 presents the metamodel for database evolution of our novel approach, highlighting several key differences from the previous evolution model. The differences between the Preliminary Metamodel and our Novel Metamodel are the following:

- **Changes have only a name:** In the new metamodel, changes are simplified and identified solely by their name.
- **No predefined order:** Changes and actions are not required to follow a specific order, as their execution is governed by newly defined conditions.
- **Independent actions:** Each action now includes a script and a condition, and they operate independently without relying on one another.
- **ER database schema:** The new metamodel explicitly requires that the database schema be represented using the ER model.

We still have the same working assumption: (i) each feature entails two potential changes: a selection change and deselection; (ii) the same type of changes are maintained, namely *NOOP*, *ALLOWED*, *MISSING DATA*, and *FORBIDDEN*; and (iii) each change involves a series of actions necessary for database transformations, each requiring a specific script to be executed against the database. We account for potential constraints that may prevent an action from being executed through `conditions` associated with each `Action`, whether derived from a parent–child hierarchy or explicitly specified by the domain engineer. As the hierarchy is already inherent in the FM structure, we now define potential changes in an atomic manner, eliminating the need to specify a particular execution `order` for the `Actions` inside a `Change`.

For implementing the database transformations, we assume that domain engineers are responsible for outlining the potential changes and their corresponding actions for each feature. In our current implementation, the definitions of possible changes are articulated in a JSON file,[2] where each feature is depicted as an object encompassing both changes: selection and deselection. An example can be seen in Listing 1. Each change is delineated as a subordinate object of the feature, complete with a name, a set of actions, and its type. Additionally, the JSON file specifies the actions required to alter the database schema and, when necessary for the changes, migrate associated data. Each action includes scripts, conditions, and a list of database elements that it specifically affects.

```
{"FeatureName": {
    "change": {
        "name": "changeName",
        "actions": [
            {
                "condition": "condition",
                "script": "scriptName",
                "affects_selection": [ "Table"],
                "affects_attribute": [ {
                    "relation": "Table",
                      "attribute": "attribute"} ],

                "affects_fk": " [ {
                       "relation": "Table",
                       "fk": "attribute"} ]
                }
        ],
        "type": "ChangeType"
    }
} }
```

Listing 1: JSON schema for change definition

Given the example of departments and employees (see Section 2), Listing 2 provides an example of how the JSON would be defined for the feature `Manager`. In this brief example, we define the changes `selectManager` (Line 2) and `deselectManager` (Line 20) with their corresponding actions. The `selectManager` change involves two actions: creating the `Manager` attribute (Line 6) and establishing the relationship from Employee to itself (Line 12). The first change, `createAttributeManager` (Line 6), defines how the table `Employee`

```
1  {"Manager": {
2      "select": {
3          "name": "selectManager",
4          "actions": [
5              {
6                  "script": "createAttributeManager",
7                  "affects_selection": [ "Employee"],
8                  "affects_attribute": [ {
9                      "relation": "Employee",
10                       "attribute": "manager"} ] },
11             {
12                 "script":
       "createRelationEmployeeToEmployee,
13                 "affects_selection": [ "Employee"],
14                 "affects_fk": " [ {
15                        "relation": "Employee",
16                        "fk": "manager"} ] }
17         ],
18         "type": "ALLOWED"
19     },
20     "deselect": {
21         "name": "deselectManager",
22         "actions": [
23             {
24                 "script":
       "dropRelationEmployeeToEmployee",
25                 "affects_selection": [ "Employee"],
26                 "affects_fk": " [ {
27                        "relation": "Employee",
28                        "fk": "manager"} ] },
29             {
30                 "script": "dropAttributeManager",
31                 "affects_selection": [ "Employee"],
32                 "affects_attribute": [ {
33                     "relation": "Employee",
34                       "attribute": "manager"} ] },
35         ],
36         "type": "MISSING DATA"
37     }
38 } }
```

Listing 2: Example of change definition for Manager feature

(Line 7) gains a new attribute called `manager` (Lines 8–10). The second change, `createRelationEmployeeToEmployee` (Line 12), establishes that `Employee` (Line 13) must have a new foreign key relation to the attribute `manager` of `Employee` (Lines 14–16).

Each change explicitly details the entity of the database affected by this change, `Employee` (Lines 7 and 13). For creating an attribute, we specify the affected attribute, `manager` (Line 10). Simultaneously, the definition of the relationship requires explicitly stating the attribute serving as the foreign key, `manager` (Line 16). The change description concludes by specifying the type of change, `ALLOWED` in this instance (Line 18). Conversely, the reverse process is outlined for the `deselectManager` change, involving the opposite two actions: initially, dropping the relationship from Employee to itself (Line 24) to prevent constraint violations, and subsequently, dropping the `Manager` attribute (Line 30).

5. Evaluation of SPL-DB-Sync

To further understand the practical implications of our proposal for evolving SPL products and their impact on the database, we provide real-world use cases in which the problem at hand highlights the benefits of our work. The phenomenon in this case is the SPL-DB-Sync approach, evaluated in real-world circumstances. The study provides a detailed investigation of practical use cases of SPL-DB-Sync, allowing us to assess its tangible benefits and downsides in managing feature selection changes.

[2] https://www.json.org/json-en.html.

Our research follows the design science methodology (Wieringa, 2010), which is appropriate for evaluating technological artifacts such as SPL-DB-Sync. Design science involves the creation and evaluation of artifacts to solve identified problems, thereby extending the boundaries of human and organizational capabilities (Wieringa, 2014). We chose this methodology because it aligns well with our goal of developing an approach to manage the evolution of SPL products and their impacts on databases in SPLs. This approach includes both conceptual and empirical evaluations to ensure a comprehensive understanding of the artifact's utility.

5.1. Subject system

The SPL used as the subject system is in the domain of digital libraries. LPS-BIDI is an SPL developed by the Database Laboratory of the University of A Coruña (Pedreira et al., 2021). A digital library denotes a collection of literature and its corresponding metadata (data about data) stored electronically (Hull et al., 2008; Lesk, 2005). These digital artifacts can include literature, multimedia, research data, and the like. Digital libraries find application in various contexts, ranging from extensive public libraries to specialized research endeavors. While digital libraries are widely recognized, their characteristics and objectives exhibit diversity according to the particular context in which they are employed (Arms, 2000).

In several countries (e.g., Spain), a significant number of digital libraries were developed in collaboration with academic organizations specializing in the humanities and literature. These research initiatives required the rapid delivery of highly customized software solutions with little financial investment. This particular domain proved to be a nurturing ground for the application of an SPL. This is the case of LPS-BIDI, described in more detail in our previous contributions (Pedreira et al., 2021).

The domain analysis to develop LPS-BIDI was performed taking into consideration previous digital libraries developed by the research group, such as the Documentary Center for the Association of Writers in Galician (AELG),[3] the Galician Socio-Pedagogical Association,[4] the Edition During the Franco Era,[5] and the Poliantea Library.[6] Fig. 4 presents the FM for LPS-BIDI, in which some sub-features have been collapsed due to space limitations. The thorough study of previous libraries resulted in a broad range of 162 features, which naturally fit into four major groups: (i) library objects such as publications and authors; (ii) data exportation; (ii) user management; and (iv) library access that specifies which functions can be accessed by certain user roles. In addition, the FM for LPS-BIDI has 13 cross-tree constraints that impact 22 features. Upon computing all possible feature combinations while respecting cross-tree constraints, the SPL can create 7806 distinct products.

A subset of features directly influences the data schema of the SPL products, highlighted in red in Fig. 4. Table 3 presents the number of features in each group as well as the total number of features that impact the database design. As we can see, the `LibraryObjects` feature group comprises 35 features, of which 28 are directly related to the database schema. The significance of the `LiteraryObjects` group lies in its role in determining how information related to libraries is stored, encompassing details about authors (`LO_Authority`), literary elements (`LO_Element`), and publishing houses (`LO_Organization`). The conceptual database schema for the LPS-BIDI, centered on the entities impacted by the `LibraryObjects`-related features, is presented in Fig. 5.

`LiteraryObjects` are the core element of a digital library, and so they will always be present in any possible feature selection.

Table 3
Distribution of features into groups.

Feature group	# Features	# Affecting DB
LibraryObjects	35	28
UserManagement	16	6
DataExportation	20	0
LibraryAccess	69	0

Our FM defines four different types of literary elements. Literary works (`LO_E_ET_Work`) such as books, are the only kind of literary publication that is always present in the database. The database elements belonging to the other types, which correspond to editions of literary works (`LO_E_ET_Edition`), smaller pieces that compose a literary work (`LO_E_ET_Part`), and collections of editions (`LO_E_ET_Collection`), are dependent on whether their features are selected as part of the product configuration.

The selection of the feature `LO_Organization`, implies the presence of an entity dedicated to organizations in the database with detailed information (i.e., the name, the location, or the opening date). On the contrary, the engineer holds the responsibility of deciding whether to incorporate information about publishing entities (organizations). Upon selecting the feature `LO_Organization`, a new database entity is established for this specific purpose. In the absence of the feature `LO_Organization`, the only information concerning organizations retained in the database is confined to an attribute within the Edition entity labeled "Editorial", providing a space for users to input the name of the publishing organization. Additionally, the engineer has the option to activate the `LO_O_Director` feature if they wish to include details about the directors of these organizations.

The feature group `User Management` also includes certain features with implications for the database. These features relate to the capabilities of the application regarding registration, authentication, user profiles, and the type of user role identification within the product, to cite some. The functionalities within the `DataExportation` and `LibraryAccess` categories have no impact on the structure of the database or current status. In terms of `DataExportation`, these features allow the engineer to decide whether the digital library will provide options to export data in different formats (e.g., PDF, plain text, or JSON), as well as to select which items can be exported. Although these functionalities require data retrieval from the database, their integration into the product solely impacts the software and does not impact the database schema. Likewise, `LibraryAccess` features allow the engineer to designate which user profiles will be granted access to search, navigate, and visualize information through maps. Once again, incorporating these features into a product impacts the software but does not alter the database schema or its existing data. Consequently, any modifications made to the feature selection in these two categories can be made without considering any implications for the database.

Fig. 6 presents the entity-relationship model of the database designed for LPS-BIDI, including all feature annotations. Annotations structured as `[if` $FeatureName$`]` are used to visually represent FM variability. These annotations indicate when certain elements are present in the database based on the selection of the corresponding feature in the FM. On the other hand, `[if not` $FeatureName$`]` annotations indicate configurations in which the data model is changed because a specific feature is not included in the selection. Dashed lines denote foreign keys whose presence depends on the selection of their corresponding features. The model in Fig. 6 delineates the correlation between the database schema and the FM, elucidating the impact of each feature on particular tables or attributes within the database. While we use this model as a guide to elucidate how features influence our database schema and existing data, it is important to note that the relational model is not reflected in the metamodel presented in Fig. 3.

[3] Association of Writers in Galician (AELG): https://www.aelg.gal/.
[4] Galician Socio-Pedagogical Association: https://www.as-pg.gal/.
[5] Edition during the Franco era: https://ediciongalizafranquista.udc.gal/.
[6] Poliantea: https://www.bidiso.es/Poliantea/.

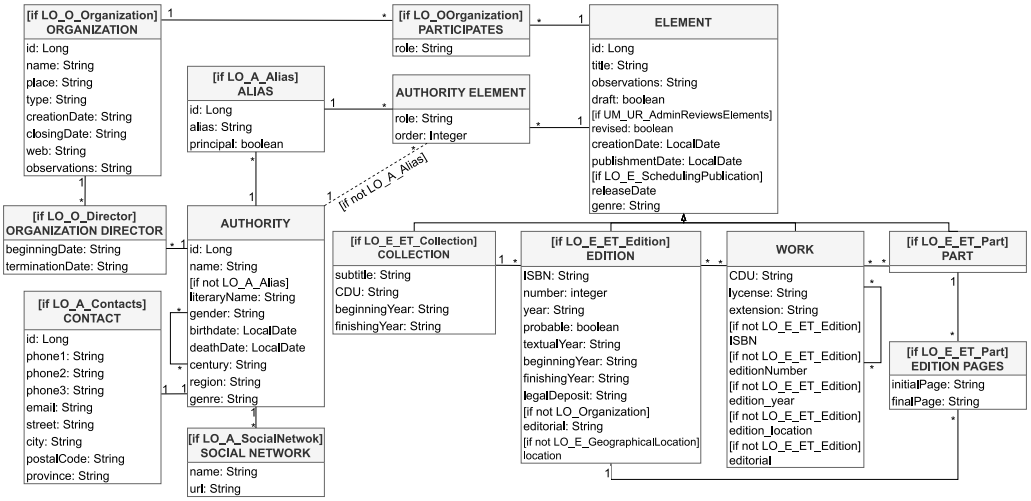

Fig. 5. Entities in the conceptual database schema impacted by `LiteraryObjects`.

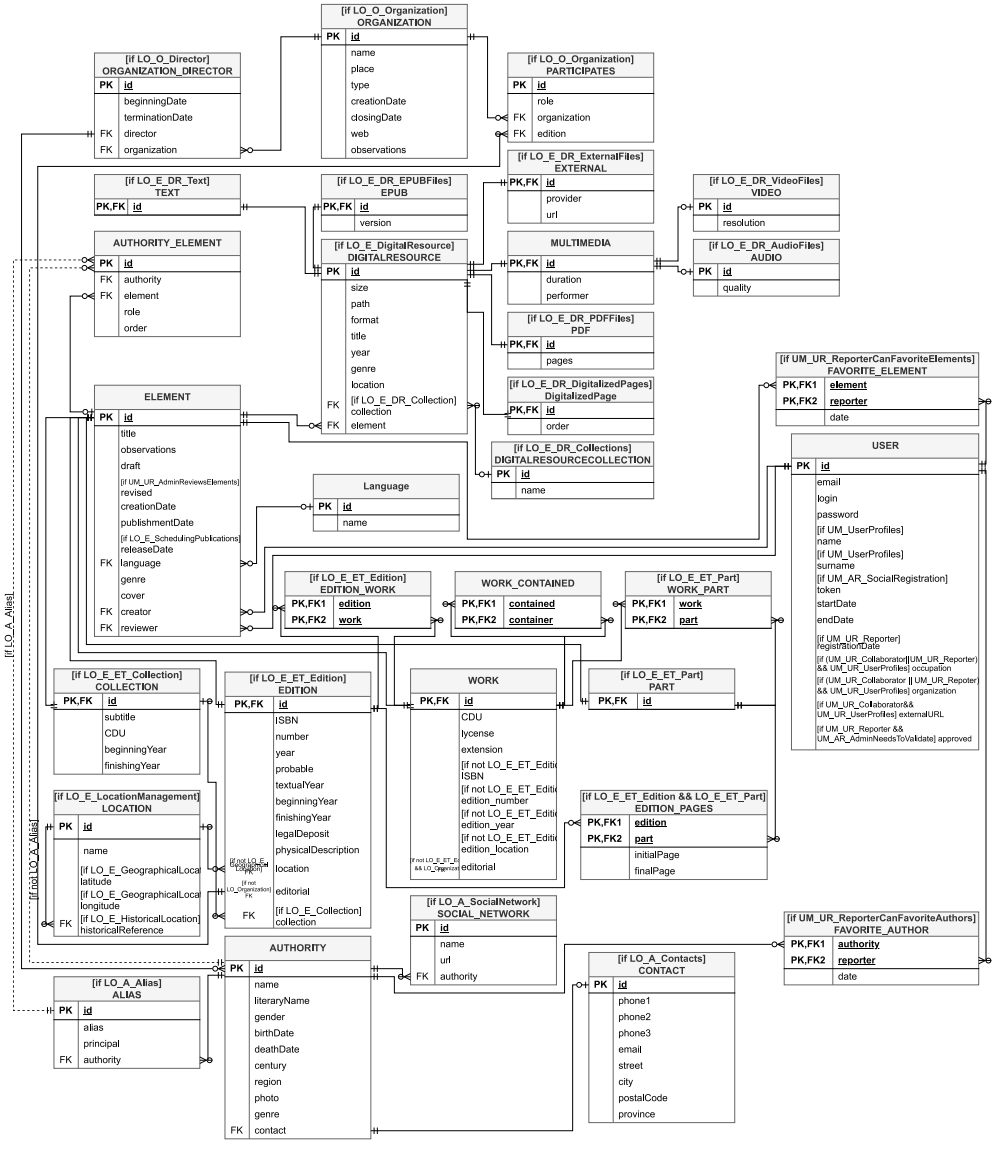

Fig. 6. LPS-BIDI entity-relationship model annotated with corresponding features present in the FM.

5.2. Subject product

Our use cases (i.e., scenarios) center on a specific product from LPS-BIDI, focusing on the real-world evolution of the Galician Virtual Library (BVG).[7] The BVG project was started in 2002, led by the University of A Coruña, and funded by the Deputation of A Coruña and the Galician Royal Academy. It was developed by experts in Galician-Portuguese philology and the Database Laboratory (LBD). Over the years, the BVG has become a valuable repository for Galician literature, offering linguistic materials, bibliographic information about writers, and additional content such as images, audio, and video files. Currently, it encompasses records for 353 writers and 4770 literary resources, featuring around 1000 items with digital content in the form of photos, text, audio, and video. After more than two decades of operation, the BVG has experienced changes in its requirements and the functionalities it is expected to provide.

5.3. Product evolution scenarios

The need for evolving BVG required changes in the feature selection of the product, as well as evolution in the database schema and transformation/migration in the existing data. These changes reflect the dynamic nature of SPLs, where the evolving needs of stakeholders may prompt the inclusion of novel features not selected in the previous product instance and the removal of initially selected features deemed unnecessary over time. This evolution consequently impacts the underlying database schema. In the subsequent sections, we illustrate four real-life scenarios of changes that BVG underwent and how our approach effectively addressed them.

5.3.1. Including new feature with data migration

The first scenario deals with the implications of introducing a change to the database schema that involves selecting a new feature within the SPL. Specifically, the scenario involves the addition of a feature that requires the creation of a new table in the database (Challenge 1) and the migration of data to the new table (Challenge 2), resulting in consequential impacts on the overall data structure.

During the initial BVG design phase, the focus was on preserving the first or most significant edition of each book. Nevertheless, as the dataset grew, it became apparent that the richness of data surpassed the constraints of a singular edition. Acknowledging the significance of documenting various relevant editions, the upcoming product iteration will introduce a feature to incorporate editions as literary elements, denoted as `LO_E_ET_Edition`, which allows to store many editions of each literary work.

Fig. 7 illustrates the conceptual model before and after the inclusion of the `LO_E_ET_Edition` feature, while Listing 3 outlines the concrete definition of the change `selectLO_E_ET_Edition` (Line 3). The entity dedicated to editions is structured as an inheritance of the class `Element`, mirroring the structure of the `Work` entity. Attributes within the class `Work`, responsible for storing information about the first edition, have been transferred to a distinct entity in the database. To facilitate this transformation, the table `Edition` is created as a preliminary step (Line 6–7), followed by the migration of relevant attributes from `Work` to `Edition` (Lines 9–25). With both tables in place, the intermediate `EditionWork` table is established (Lines 27–35) to represent the *Many-To-Many* relationship between them. Notably, the inclusion of the `LO_E_ET_Part` feature would trigger a conditional action (Lines 37–46). If selected, an intermediate table between `Edition` and `Part` is created to model their *Many-To-Many* relationship. However, as `Part` is not included in the feature selection, this condition is not triggered.

[7] Galician Virtual Library website: https://bvg.udc.es/.

```
1  { "LO_E_ET_Edition": {
2      "select": {
3          "name": "selectLO_E_ET_Edition",
4          "actions": [
5              {
6                  "script": "createTableEdition",
7                  "affects_selection": [ "Edition" ] },
8              {
9                  "script":
       "migrateAttributesFromWorkToEdition",
10                 "affects_selection": [ "Edition",
       "Work" ],
11                 "affects_attribute": [ {
12                     "relation": "Work",
13                     "attribute": "ISBN" },
14                     {
15                     "relation": "Work",
16                     "attribute": "edition_number" },
17                     {
18                     "relation": "Work",
19                     "attribute": "edition_year" },
20                     {
21                     "relation": "Work",
22                     "attribute": "edition_location" },
23                     {
24                     "relation": "Work",
25                     "attribute": "editorial" } ] },
26             {
27                 "script": "createTableEditionWork",
28                 "affects_selection": [ "Edition_Work" ]
       ,
29                 "affects_fk": " [
30                     {
31                     "relation": "Edition_Work",
32                     "fk": "Edition" },
33                     {
34                     "relation": "Edition_Work",
35                     "fk": "Work" } ] },
36             {
37                 "condition": "LO_E_ET_Part",
38                 "script": "createEditionPagesTable",
39                 "affects_selection": [ "Edition_Pages"
       ],
40                 "affects_fk": " [
41                     {
42                         "relation": "Edition_Pages",
43                         "fk": "Edition" },
44                     {
45                         "relation": "Edition_Pages",
46                         "fk": "Pages" } ] }
47         ],
48         "type": "ALLOWED"
49     }
50 } }
```

Listing 3: Change to include `LO_E_ET_Edition`

If this change had to be performed manually, a developer would need to analyze both the existing and the new schema to understand the effects of selecting `LO_E_ET_Edition`, write SQL scripts to create tables `Edition` and `Edition_Work`, and script the operations to migrate the relevant attributes from `Work` to `Edition` with the necessary transformations (e.g., INSERT INTO SELECT Statement, CAST functions). Additionally, the developer would need to ensure the SQL script correctly performs the data migration without errors, and then execute these scripts on the production database.

5.3.2. Including new features with data migration and condition

In this scenario, the selection of a new feature within the SPL requires a more intricate adaptation of the database schema. This particular change entails the creation of two new tables in the database

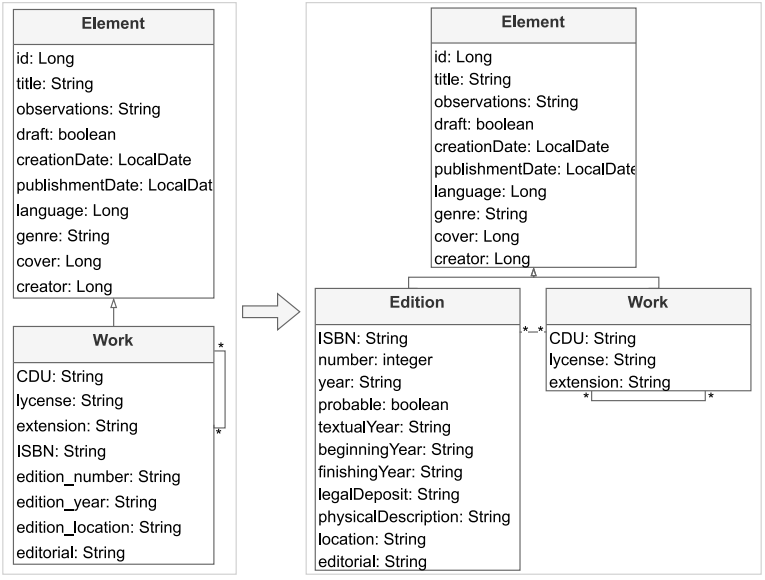

Fig. 7. Conceptual data model representation of the change `SelectEdition`.

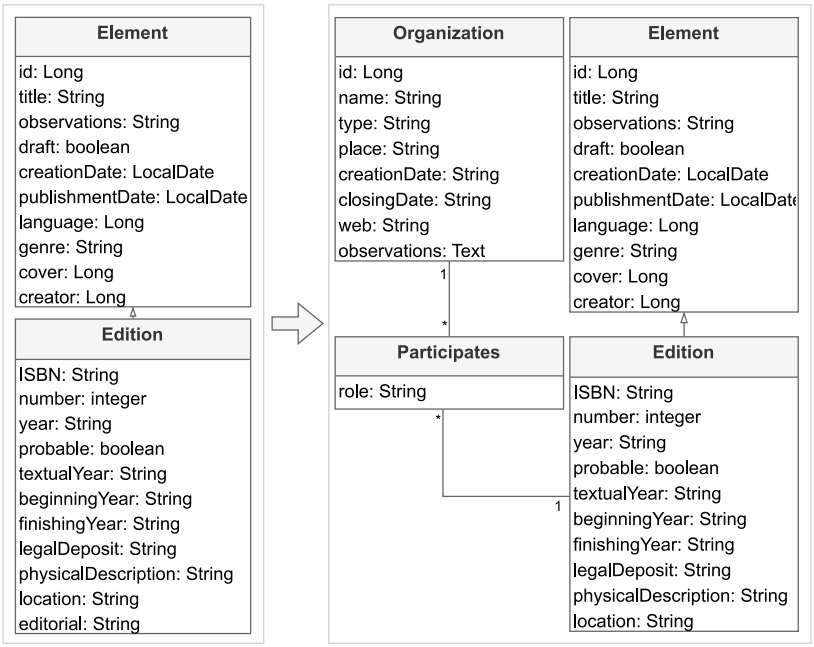

Fig. 8. Conceptual data model representation of the change `SelectOrganization`.

(Challenge 1). Moreover, the inclusion of this feature brings about conditional behavior, where the subsequent actions and table creations depend on the specific conditions being met. Additionally, this scenario requires migrating existing data into the newly created tables, ensuring data consistency and accuracy (Challenge 2).

Philologists, experts in language and literature, have cultivated partnerships with publishing houses, resulting in a collaborative synergy. Publishing houses actively share newsletters and important data about their most recent books with philologists; this not only raises the visibility of new works but also provides philologists with valuable insights into emerging trends in literature. Including this curated data is crucial for capturing the evolution of language studies and guarantees that BVG becomes a valuable resource for researchers.

The initial library design, focused on authors and bibliographies, omitted the `LO_Organization` feature. Initially, we deemed a single attribute for edition publishers adequate, without foreseeing the necessity to store extensive information about the involved publishing organizations. As BVG evolved, we opted to incorporate the feature `LO_Organization` into the new feature selection. Fig. 8 illustrates the transformation to the conceptual model, and Listing 4 includes the definition of the change to select the feature `LO_Organization`.

This change involves creating the `Organization` class in the database (`createTableOrganization`, Lines 6–7). Following this, we establish a *Many-To-Many* relationship, referred to as `Participates`, between `Organization` and `Edition` (`createTableParticipates`, Lines 9–11; `createRelationOrganizationParticipates`, Lines 13–18 and `createRelationEditionParticipates`, Lines 20–25). To complete this transition, data from the `Edition` class needs to be migrated to the new `Organization` class. During this process, the editorial attribute transforms into the names of the new entities. As a final step, the editorial attribute is removed from the `Edition` class to ensure data

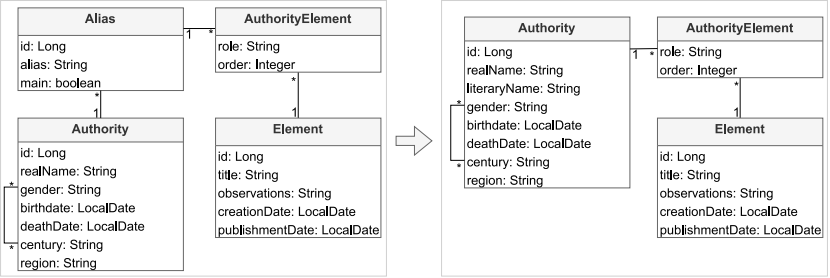

Fig. 9. Conceptual data model representation of the change DeselectAlias.

consistency (migrateEditorialAttributeToParticipates, Lines 27–32).

Performing this change manually would require creating two new tables, namely Organization and Participates, along with establishing the necessary foreign key constraints between Organization and Participates, and between Participates and Edition, so that we can establish the *Many-To-Many* relationship between Organization and Edition. Additionally, the developer would need to manually implement data migration operations. This process involves transferring data from the existing editorial attribute to Participates. Specifically, the developer would first migrate the existing editorial data into new organization entries in the Organization table (e.g., SELECT DISTINCT and INSERT statements). Then, they would cross-reference these new entries with the legacy editorial data to populate the Participates table effectively (e.g., INSERT INTO SELECT Statement, JOIN clauses).

```
{ "LO_Organization": {
    "select": {
        "name": "selectLO_Organization",
        "actions": [
            {
                "script": "createTableOrganization",
                "affects_selection": [ "Organization"]
     },
            {
                "condition": "LO_E_ET_Edition",
                "script": "createTableParticipates",
                "affects_selection": [ "Participates"]
     },
            {
                "condition": "LO_E_ET_Edition",
                "script":
     "createRelationOrganizationParticipates"
                "affects_selection": [ "Participates"]
     ,
                "affects_fk": " [ {
                    "relation": "Participates",
                    "fk": "organization" } ] },
            {
                "condition": "LO_E_ET_Edition",
                "script":
     "createRelationEditionParticipates"
                "affects_selection": [ "Participates"]
     ,
                "affects_fk": " [ {
                    "relation": "Participates",
                    "fk": "edition" } ] },
            {
                "condition": "LO_E_ET_Edition",
                "script":
     "migrateEditorialAttributeToParticipates",
                "affects_selection": [ "Edition"],
                "affects_attribute": [ {
                    "relation": "Edition",
                    "attribute": "editorial" } ] }
        ],
        "type": "ALLOWED"
    }
} }
```

Listing 4: Change to select LO_Organization

5.3.3. *Removing features with data loss*

In scenarios where we deselect a feature, it may initiate a process of adapting the database schema (Challenge 1). This change entails the removal of specific tables from the database structure, introducing the challenge of managing potential data loss. However, to mitigate the impact, we implement measures to handle the preservation of desired information (Challenge 2).

In certain digital libraries, authors denoted as Authority, may use an Alias to sign their works. Initially, the library aimed to store multiple aliases for each author with the inclusion of the feature LO_A_Alias. However, over a decade of usage revealed that out of 1800 authorities, only three authors had more than two aliases. Recognizing the substantial effort and resources required for maintaining alias data, a decision was made to reconfigure the product by deselecting the feature LO_A_Alias. In this situation, executing this change impacts both the configuration of the database and its existing data. Removing the Alias feature leads to the deletion of data associated with aliases used by the authors to sign their works. Fig. 9 illustrates the transformation to the conceptual data model.

In Listing 5 the actions needed to deselect LO_A_Alias are defined. The change DeselectAlias (Line 3) involves a set of five corresponding actions. To minimize data loss, the engineer must encode data preservation mechanisms within the script. For instance, the script MigratePrincipalAliasToAuthority (Lines 6–14) is specifically designed to preserve the 'main alias' by migrating it to the Authority table, filling the LiteraryName attribute. Subsequently, we dissolve the relationship between AuthorityElement and Alias (DropRelationAliasAuthorityElement, Lines 16–20) and the link between Alias and Authority (DropRelationAliasAuthority, Lines 22–26). The relationship must now be established between Authority and AuthorityElement (createRelationAuthorityToAuthorityElement, Lines 28–32). Finally, Alias is removed entirely (DropTableAlias, Lines 34–35).

If the change were to be performed manually, the developer would need to specify a preservation technique, such as using INSERT INTO SELECT statements with appropriate WHERE clauses, to migrate the principal alias data to the Authority table. They would also need to add foreign key constraints to establish relationships between Authority and AuthorityElement. Finally, the developer would be responsible for manually dropping the Alias table and ensuring that all dependent relations are properly managed to maintain database integrity.

```
{ "LO_A_Alias": {
    "deselect": {
        "name": "deselectLO_A_Alias",
        "actions": [
            {
                "script":
    "migratePrincipalAliasToAuthority",
                "affects_selection": [ "Alias",
    "Authority"],
                "affects_attribute": [
                    {
                        "relation": "Alias",
                        "attribute": "alias"},
                    {
                        "relation": "Authority",
                        "attribute": "literaryName"} ]
    },
            {
                "script":
    "dropRelationAliasAuthorityElement",
                "affects_selection": [ "Alias",
    "AuthorityElement"]
                "affects_fk": " [ {
                    "relation": "AuthorityElement",
                    "fk": "alias"} ] },
            {
                "script":
    "dropRelationAliasAuthority",
                "affects_selection": [ "Alias",
    "Authority"]
                "affects_fk": " [ {
                    "relation": "Alias",
                    "fk": "authority"} ] },
            {
                "script":
    "createRelationAuthorityToAuthorityElement",
                "affects_selection": [ "Authority",
    "AuthorityElement"]
                "affects_fk": " [ {
                    "relation": "AuthorityElement",
                    "fk": "authority"} ] },
            {
                "script": "dropTableAlias",
                "affects_selection": [ "Alias"] } ],
        "type": "MISSING DATA"
    }
} }
```

Listing 5: Change to select `LO_A_Alias`

```
{
"LO_E_DR_Audio": {
       "select": {
           "name": "selectLO_E_DR_Audio",
           "actions": [
               {
                   "condition": "!LO_E_DR_Video",
                   "script": "createTableMultimedia",
                   "affects_selection": [
    'Multimedia"]
               },
               {
                   "condition": "!LO_E_DR_Video",
                   "script":
    "createRelationDigitalResourceToMultimedia",
                   "affects_selection": [
    'Multimedia"]
                   "affects_fk": " [
                       {
                           "relation": 'Multimedia",
                           "fk": "id"
                       }
                   ]
               },
               {
                   "script": "createTableAudio",
                   "affects_selection": [ "Audio"]
               },
               {
                   "script":
    "createRelationMultimediaToAudio",
                   "affects_selection": [ "Audio"]
                   "affects_fk": " [
                       {
                           "relation": "Audio",
                           "fk": "id"
                       }
                   ]
               }
           ],
           "type": "ALLOWED"
       }
   }
}
```

Listing 6: Change to select `LO_E_DR_Audio`

5.3.4. Including new features with conditions

The last scenario illustrates the introduction of new features, prompting the creation of additional tables in the database schema (Challenge 1). However, conditions are applied to align the database structure with the entity-relationship model, as depicted in Fig. 6, even though the direct correlation to specific features is not explicitly shown in the model.

Originally focused on video content, specifically author interviews, our digital resource selection has expanded to include audiobooks with the introduction of the `audio` feature. To accommodate this, we introduced the feature `audio`. This addition requires the creation of a dedicated database table, as shown in Fig. 10. However, our analysis of the entity-relationship model revealed the presence of an intermediary `multimedia` table connecting digital resources and audio files, which is not explicitly linked to features such as `Audio`, `Video`, or `DigitalResource`, even though the `multimedia` table creation is only necessary when either `Audio` or `Video` exists. In this scenario, we selectively introduce the `Audio` feature, ensuring the existence of the `multimedia` table. The shared inheritance of `Video` and `Audio` entities necessitates careful checks before executing scripts to maintain data consistency, ensuring the structural soundness of our evolving digital resource ecosystem.

Listing 6 defines the changes and actions to modify the feature selection by selecting `LO_E_DR_Audio`. The first action involves creating the `Multimedia` table (Lines 7–9). To maintain database integrity and prevent duplicate tables, a conditional check ensures that the intermediate table has not been created before (Line 7). This condition verifies the absence of the `LO_E_DR_Video` feature in the current feature selection, as `Video` also depends on the intermediate table. If `LO_E_DR_Video` is already selected, indicating the prior creation of `Multimedia`, the table creation is skipped. The same principle applies to establishing the relationship between `DigitalResource` and `Multimedia`, which is not executed in this scenario due to the presence of `LO_E_DR_Video` in the previous product configuration. As an exemplification of the reuse of transformational actions, the actions dependent on the condition would be the same if selecting `LO_E_DR_Video` and checking the presence of `LO_E_DR_Audio`. As the last step, the `Audio` table (Lines 23–24) and the relationship between `Audio` and `DigitalResource` (Lines 27–32) are created.

Performing this change manually would involve initially verifying the existence of the `Multimedia` table. If it does not exist, the developer would need to create it. Subsequently, they would add foreign key constraints to link `DigitalResource` with `Multimedia`, thereby

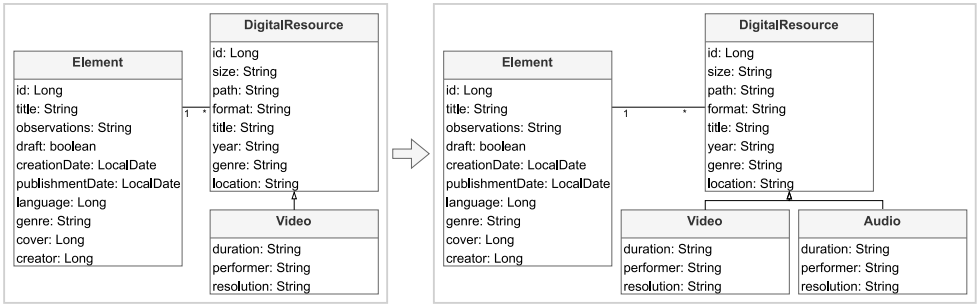

Fig. 10. Conceptual data model representation of the change `SelectAudio`.

implementing the table per entity inheritance structure in the database. Next, the developer would create the `Audio` table and establish the necessary foreign key constraint to reflect the inheritance from `Multimedia` to `Audio`. Throughout this process, meticulous attention would be required to ensure that all implementations align with the designed database structure.

Evaluation summary: The real-world scenarios presented above highlight the high effort required to perform manual data migration in the evolution of SPL products. This manual process introduces significant risks, including the potential for human errors when writing and testing the SQL scripts, and it overburdens the developer with repetitive and time-consuming tasks. As in any automated software engineering technique, an error in the database evolution actions could introduce errors in the application of SPL-DB-Sync in a real scenario. However, provided that those data evolution actions are correct, our proposal eliminates the possibility of introducing human errors in the product's evolution and database adaptations.

SPL-DB-Sync leverages pre-implemented scripts integrated with the evolution model, which can be reused across any instance of the products generated by our SPL. This means that a one-time effort in developing the scripts allows for automated changes, eliminating the need for manual migration and reducing the risk of errors.

5.4. Limitations and threats to validity

The primary limitations and threats to validity of our work, along with the mitigation strategies we used, are outlined below.

5.4.1. Limitations of the approach

While SPL-DB-Sync introduces a novel perspective to address practical challenges and contributes to the state of the art and the practice, we acknowledge certain limitations of the approach and potential threats to the validity of the evaluation.

A limitation in scope may arise as the primary focus of the approach centers on the evolution of the database schema, specifically designed for situations where alterations in feature selection influence the database schema. Although various aspects of SPL evolution, including artifact evolution, are considered, the detailed exploration of the interaction between these challenges is not a focal point of our investigation.

The successful implementation of SPL-DB-Sync hinges on user-dependent aspects, such as the manual definition of the evolution model for all SPL features. This includes specifying the actions needed to implement these changes, with each action linked to a corresponding script executed during data transformation and migration. Therefore, the expertise of the domain engineer in defining these changes and actions is crucial. Misinterpretation or oversight during the process may lead to unsuccessful data transformation.

5.4.2. Threats to validity

In the following, we explain how the threats to validity were addressed in the validation presented in this section, according to the categories presented by Wohlin et al. (2012):

Internal Validity: The subject system we used in the validation of SPL-DB-Sync is a real data-intensive SPL, in which many of the features affect the underlying data model. Although we consider this subject system to present all the possible cases regarding database adaptation upon updates in the feature selection, other systems may present new challenges. Regarding the application of the model to the subject system, we do not perceive internal validity threats, since the relationships between features and database elements are objective, as well as the evaluation of how our proposal addresses those relationships on updates to the product's selection of features.

External Validity: Selecting or deselecting features post-production is a typical scenario in software engineering with SPLs. While this use case has demonstrated the suitability of SPL-DB-Sync for managing database evolution based on updates to the selection of features, it is possible that new needs or challenges appear when applying the model to other SPLs in different contexts, which can affect the generalizability of the conclusions or our validation. Regarding the replicability of the validation, the artifacts we used are not public due to third-party property constraints.

Construct Validity: The success criteria of the use case was based on the suitability of SPL-DB-Sync to address the database evolution upon changes to the selection of features of the subject system. More specifically, the proposal was considered to be suitable if it was able to support the database evolution actions automatically, without needing any additional actions. The potential use case.

Reliability: The relationship between features and the elements of the data model they impact is not subjective—it is a process that any user could replicate. In terms of the application of SPL-DB-Sync, we did not identify any factors that would suggest significant variability in outcomes based on the individual applying the model.

6. Implications

The discussion and implications of our work revolve around the key insights and advantages of our approach. SPL-DB-Sync addresses the challenge of evolving variants within an SPL when changes to the feature selection impact the database schema. Our contribution lies in the definition of an SPL Evolution Model, based on the classification of modifications to feature selection based on their effects on the database schema and data, categorized into four unique "Change Types". We provide a systematic framework for understanding and managing changes to feature selection, ensuring clear traceability, and facilitating the reuse of data transformation actions across different products and features.

For practitioners, one implication of our approach is its practical utility in SPL development. By offering a structured model for defining

potential modifications to feature selection and their impact on the database, our approach empowers engineers to navigate and implement changes seamlessly. The approach categorizes changes into four types — NOOP, ALLOWED, MISSING DATA, and FORBIDDEN — helping engineers make informed decisions about modifications. To use SPL-DB-Sync, engineers would start by defining the evolution model for their specific SPL, which includes identifying how each feature affects the database schema. This model helps engineers anticipate the impact of adding or removing features, including potential data loss. Engineers can then leverage the semi-automated steps of SPL-DB-Sync, which guide them through data migration and schema updates, ensuring that changes are applied correctly and consistently.

For tool builders, our approach can be a starting point to extend existing tools to develop SPLs. For instance, currently, our approach is semi-automated, with some steps being done manually by engineers. However, all our descriptions and logic of such manual steps can be used as inspiration for further automation. Also, the traceability mentioned above can be used for additional impact analysis or as a source of information for the optimization of the SPL, for example, improving non-functional requirements.

For researchers, SPL-DB-Sync contributes to advancing the state-of-the-art in software reuse and data management. It fills a significant gap in the literature by explicitly addressing the challenges of evolving already-derived products with populated databases. Our approach consists of carefully designed steps to systematically apply modifications to the database schema. This not only enhances the flexibility of SPLs but also ensures data consistency and alignment with evolving organizational needs. Additionally, SPL-DB-Sync lays a foundation for future research on refining and extending methodologies for handling complex feature selections and their impact on the database schema. This work is the provision of a practical, structured, and systematic approach, SPL-DB-Sync, that addresses the intricate challenge of evolving already-derived products with populated databases. The approach emphasizes clarity through classification, traceability, and transparency, serving as a foundational contribution to navigating the complexities of SPL evolution and data management.

7. Conclusions and future work

This paper presents SPL-DB-Sync, an approach specifically designed to address the intricate challenge of evolving products generated using SPLs, with populated databases. Categorizing modifications to feature selection based on their impact on the database schema, SPL-DB-Sync offers a structured methodology that ensures clear traceability, classification, and transparency. Our work contributes to SPL engineering by addressing the nuanced challenges of evolving the products generated by SPLs once they are already in the production phase. SPL-DB-Sync addresses a gap in the literature by presenting a practical and systematic approach to address the challenges posed by evolving SPLs with populated databases. Although SPL-DB-Sync provides a practical solution, it acknowledges limitations.

Future research can explore the interaction between various challenges in SPL evolution, optimize the evolution model implemented in SPL-DB-Sync for more efficiency, and aim to reduce user-dependent aspects by automating the definition of changes and actions. Additionally, new studies should evaluate the scalability of our proposed SPL-DB-Sync approach, particularly given the prevalence of modern software applications that are data-intensive with numerous database tables and large volumes of data. In addition to scalability considerations, future research should focus on evaluating the usefulness and practical applicability of SPL-DB-Sync across diverse domains within real-world software engineering contexts.

Expanding our approach to encompass the evolution of the SPL itself represents an important direction for future work. Currently, SPL-DB-Sync focuses on evolving products generated by an SPL. Extending this capability to the evolution of the SPL would involve several key considerations. Future work should explore how to automatically propagate changes such as adding, altering, or deleting features from the FM to the evolution model. This includes studying how these changes can be represented in the evolution metamodel and ensuring that they are correctly reflected in the evolution process. Yet, investigating scenarios where constraints within the SPL change, potentially rendering existing products invalid, will be essential.

Researchers should also explore mechanisms to update affected products and maintain consistency within the SPL. Studying how alterations in the definition of a feature, particularly those affecting the backend in ways that impact the database schema, can be represented within our evolution metamodel. This involves understanding how such changes can be effectively managed and integrated into the database schema evolution process. Exploring the feasibility of implementing a version control system that allows for the restoration of data lost due to changes like the "MISSING DATA" type. This would enable the undoing of changes that lead to data loss, mitigating potential risks.

CRediT authorship contribution statement

Delfina Ramos-Vidal: Writing – review & editing, Writing – original draft, Supervision, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Wesley K.G. Assunção:** Writing – review & editing, Supervision, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Alejandro Cortiñas:** Writing – original draft, Supervision, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Miguel R. Luaces:** Writing – review & editing, Writing – original draft, Supervision, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Oscar Pedreira:** Writing – review & editing, Writing – original draft, Supervision, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Ángeles Saavedra Places:** Writing – review & editing, Supervision, Resources, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no financial or non-financial conflict of interest that are directly or indirectly related to this work submitted for publication.

Acknowledgments

CITIC is funded by the Xunta de Galicia, Spain through the collaboration agreement between the Department of Culture, Education, Vocational Training and Universities and the Galician universities for the reinforcement of the research centers of the Galician University System (CIGUS); GRC: ED431C 2021/53, partially funded by GAIN/Xunta de Galicia; TED2021-129245B-C21 (PLAGEMIS): partially funded by MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR; PID2021-122554OB-C33 (OASSIS): partially funded by MCIN/AEI/10.13039/501100011033 and EU/ERDF A way of making Europe; PID2022-141027NB-C21 (EarthDL): partially funded by MCIN/AEI/10.13039/501100011033 and EU/ERDF A way of making Europe; PID2020-114635RB-I00 (EXTRACompact): partially funded by MCIN/ AEI/10.13039/501100011033; PRE2021-099351, partially funded by MCIN/AEI/10.13039/501100011033 and "FSE+"Fondo Social Europeo Plus". Funding for open access charge: Universidade da Coruña/CISUG.

Data availability

The authors do not have permission to share data.

References

Apel, S., Batory, D., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines: Concepts and Implementation. Springer Publishing Company, Incorporated, http://dx.doi.org/10.1007/978-3-642-37521-7.

Arms, W.Y., 2000. Digital Libraries. MIT Press.

Ataei, P., Li, Q., Walkingshaw, E., 2021. Should variation be encoded explicitly in databases? In: 15th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS). ACM, pp. 1–9. http://dx.doi.org/10.1145/3442391.3442395.

Bouarar, S., Jean, S., Siegmund, N., 2015. SPL driven approach for variability in database design. In: Model and Data Engineering. Springer International Publishing, Cham, pp. 332–342. http://dx.doi.org/10.1007/978-3-319-23781-7_27.

Clements, P., Northrop, L., 2002. Software product lines. SEI series in software engineering, Addison-Wesley Boston.

Cortiñas, A., Luaces, M.R., Pedreira, O., Places, Á.S., 2023. Adapting the database to feature changes in software product lines. In: 27th ACM International Systems and Software Product Line Conference (SPLC) - Volume a. ACM, pp. 194–200. http://dx.doi.org/10.1145/3579027.3608990.

Güthing, L., Bittner, P.M., Schaefer, I., Thüm, T., 2024. Explaining edits to variability annotations in evolving software product lines. In: 18th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS). ACM, New York, NY, USA, pp. 93–102. http://dx.doi.org/10.1145/3634713.3634725.

Herrmann, K., Reimann, J., Voigt, H., Demuth, B., Fromm, S., Stelzmann, R., Lehner, W., 2015. Database evolution for software product lines. In: 4th International Conference on Data Management Technologies and Applications. DATA, pp. 125–133. http://dx.doi.org/10.5220/0005484101250133.

Hoff, A., Nieke, M., Seidl, C., Sæther, E.H., Motzfeldt, I.S., Din, C.C., Yu, I.C., Schaefer, I., 2020. Consistency-preserving evolution planning on feature models. In: 24th ACM Conference on Systems and Software Product Line: Volume a. SPLC, ACM, New York, NY, USA, pp. 65–76. http://dx.doi.org/10.1145/3382025.3414964.

Hull, D., Pettifer, S.R., Kell, D.B., 2008. Defrosting the digital library: Bibliographic tools for the next generation web. PLoS Comput. Biol. 4 (10), 1–14. http://dx.doi.org/10.1371/journal.pcbi.1000204.

Humblet, M., Tran, D.V., Weber, J.H., Cleve, A., 2016. Variability management in database applications. In: 1st International Workshop on Variability and Complexity in Software Design. VACE, pp. 21–27. http://dx.doi.org/10.1145/2897045.2897050.

Kästner, C., Apel, S., Kuhlemann, M., 2008. Granularity in software product lines. In: 30th International Conference on Software Engineering. ICSE, ACM, New York, NY, USA, pp. 311–320.

Khedri, N., Khosravi, R., 2013. Handling database schema variability in software product lines. In: 20th Asia-Pacific Software Engineering Conference (APSEC)). Vol. 1, pp. 331–338. http://dx.doi.org/10.1109/APSEC.2013.52.

Krieter, S., Krüger, J., Leich, T., Saake, G., 2023. VariantInc: Automatically pruning and integrating versioned software variants. In: 27th ACM International Systems and Software Product Line Conference - Volume a. SPLC, ACM, New York, NY, USA, pp. 129–140. http://dx.doi.org/10.1145/3579027.3608984.

Laguna, M.A., Crespo, Y., 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. Sci. Comput. Program. 78 (8), 1010–1034. http://dx.doi.org/10.1016/j.scico.2012.05.003.

Lesk, M., 2005. Understanding Digital Libraries, second ed. Morgan-Kauffman Publishers, San Francisco, http://dx.doi.org/10.1016/B978-155860924-2/50004-9.

Marques, M., Simmonds, J., Rossel, P.O., Bastarrica, M.C., 2019. Software product line evolution: A systematic literature review. Inf. Softw. Technol. 105, 190–208. http://dx.doi.org/10.1016/j.infsof.2018.08.014.

Mitschke, R., Eichberg, M., 2008. Supporting the evolution of software product lines. In: ECMDA Traceability Workshop (ECMDA-TW). pp. 87–96.

Montalvillo, L., Díaz, O., 2023. Evolution in software product lines: An overview. In: Handbook of Re-Engineering Software Intensive Systems Into Software Product Lines. Springer International Publishing, Cham, pp. 495–512. http://dx.doi.org/10.1007/978-3-031-11686-5_20.

Oracle, 2019. Database sample schemas. URL https://docs.oracle.com/en/database/oracle/oracle-database/19/comsc/index.html.

Pedreira, O., Ramos-Vidal, D., Cortiñas, A., Luaces, M., Saavedra-Places, A., 2021. Development of digital libraries with software product line engineering. J. Web Eng. 20, 2017–2058. http://dx.doi.org/10.13052/jwe1540-9589.2072.

Pohl, K., Böckle, G., Van Der Linden, F., 2005. Software Product Line Engineering. Springer, http://dx.doi.org/10.1007/3-540-28901-1.

Rhizadi, F., Fadhlillah, H.S., Azurat, A., Afriyanti, I., Apriani, N.F., 2019. Database generator to support product derivation in SPL. In: International Conference on Advanced Computer Science and Information Systems. ICACSIS, IEEE, pp. 1–8. http://dx.doi.org/10.1109/ICACSIS47736.2019.8979688.

Roddick, J.F., 1995. A survey of schema versioning issues for database systems. Inf. Softw. Technol. 37 (7), 383–393. http://dx.doi.org/10.1016/0950-5849(95)91494-K.

Schwägerl, F., Westfechtel, B., 2023. Managing software product line evolution by filtered editing: The SuperMod approach. In: Handbook of Re-Engineering Software Intensive Systems Into Software Product Lines. Springer International Publishing, Cham, pp. 429–451. http://dx.doi.org/10.1007/978-3-031-11686-5_17.

Siegmund, N., Kästner, C., Rosenmüller, M., Heidenreich, F., Apel, S., Saake, G., 2009. Bridging the gap between variability in client application and database schema. In: Datenbanksysteme in Business, Technologie Und Web (BTW) – 13. Fachtagung Des GI-Fachbereichs Datenbanken Und Informationssysteme. DBIS, Gesellschaft für Informatik e.V., Bonn, pp. 297–306.

Thüm, T., Batory, D., Kästner, C., 2009. Reasoning about edits to feature models. In: 31st International Conference on Software Engineering. ICSE, IEEE Computer Society, USA, pp. 254–264. http://dx.doi.org/10.1109/ICSE.2009.5070526.

Wieringa, R., 2010. Design science methodology: principles and practice. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2. ICSE, ACM, New York, NY, USA, pp. 493–494. http://dx.doi.org/10.1145/1810295.1810446.

Wieringa, R.J., 2014. Design Science Methodology for Information Systems and Software Engineering. Springer, http://dx.doi.org/10.1007/978-3-662-43839-8.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B., Wessln, A., 2012. Experimentation in Software Engineering. Springer Publishing Company, Incorporated.

Delfina Ramos-Vidal is a predoctoral researcher at the University of A Coruña (UDC) in Spain, working under an FPI-MEC contract. She obtained her BSc. in Computer Science from University of A Coruña (UDC) and her MSc. in Data Science from Polithecnic University of Madrid (UPM). She is affiliated with the CITIC (Centro de Investigación de Tecnologías de la Información y de la Comunicación) and specializes in areas such as Software Engineering and Data Engineering. Her research interests include on Software Product Line Engineering, Software Modernization, Variability Management, and Digital Libraries.

Wesley K.G. Assunção is an Associate Professor with the Department of Computer Science at North Carolina State University, USA. Previously, Dr. Assunção worked as a University Assistant in the Institute of Software Systems Engineering (ISSE) at Johannes Kepler University Linz (JKU), Austria (2021–2023); a Postdoctoral Researcher at Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil (2019–2023); and an Associate Professor at Federal University of Technology - Paraná, Brazil (2013–2020). He obtained his M.Sc. and Ph.D. in Computer Science from Federal University of Paraná (UFPR) also in Brazil. His research interests are: Software Modernization, Variability Management, Software Quality, Model-Driven Engineering, Collaboration in Systems Engineering, Software Testing, and Search-Based Software Engineering. Dr. Assunção' work has been published in several software engineering venues. He has also been serving as reviewers for many conferences and journal, and as organizer of conference, symposiums, workshops, competitions, and meetings. Further information: https://wesleyklewerton.github.io.

Alejandro Cortiñas is an Assistant Professor at the Database Lab of the Universidade da Coruña (Spain). He received a Ph.D. in Computer Science from the same university in 2017 for his thesis, entitled "Software Product Line for web-based Geographic Information Systems". His research topics of interest include software product lines, generative programming, geographic information systems, and spatial big data.

Miguel R. Luaces received his M.S. degree in Computer Science from the University of A Coruña (Spain) in 1998 and a Ph.D. in Computer Science from the same university in 2004. He undertook research in the area of spatial, temporal and Spatio-temporal databases at the FernUniversität Hagen (Germany) under the ChoroChronos project funded by the European Union. Today, he is an Associate Professor at the University of A Coruña, and he is currently the coordinator of the Databases Laboratory of the University of A Coruña. His research interests include Geographic Information Systems, Spatio-temporal Databases, Software Engineering, and Web-based Information Systems.

Oscar Pedreira has an M.Sc. and Ph.D. degree in Computer Science from the University of A Coruña, Spain. He is an Associate Professor since 2008 at the same institution. He is a researcher of the Database Laboratory. His research interests include topics in databases (algorithms for similarity search, data structures and algorithms for graph databases, geographic information systems), and in software engineering (process improvement, testing, MDE, and SPL). He has co-authored many articles published in journals and conferences relevant to the research areas mentioned. He has continuously participated in research projects and technology and knowledge transfer projects with different companies.

Ángeles Saavedra Places is currently an Associate Professor at the Computer Science Department of the University of A Coruña. She received her Ph.D. in Computer Science in 2003 from the same university. Her research interests are in the areas of Digital Humanities, Web Information Systems, Geographic Information Systems and Software Engineering. For further details about her CV see: http://lbd.udc.es/ShowResearcherInformation.do?id=5.