Adapting the Database to Feature Changes in Software Product Lines

Alejandro Cortiñas*
Universidade da Coruña, CITIC, Database Lab.
A Coruña, Spain
alejandro.cortinas@udc.es

Miguel R. Luaces
Universidade da Coruña, CITIC, Database Lab.
A Coruña, Spain
miguel.luaces@udc.es

Oscar Pedreira
Universidade da Coruña, CITIC, Database Lab.
A Coruña, Spain
opedreira@udc.es

Ángeles S. Places
Universidade da Coruña, CITIC, Database Lab.
A Coruña, Spain
angeles.saavedra.places@udc.es

Abstract

Software Product Lines (SPL) support the development of families of software products that share a set of core assets but differ in certain features. To generate a new product, the engineer selects the desired features and the SPL assembles and adapts the implementation of the core assets. In real scenarios, we may need to update a product by adding a feature not initially selected. Similarly, we may need to remove a feature that is no longer necessary. Modifying the selection of features of a product in use poses a challenge from the point of view of the product's database. If the added/removed features affect the database schema, we may need to adapt the schema and the data stored in the database. This paper addresses this scenario and proposes an evolution model to define actions to be executed in the database when features are added or removed. Our proposal allows us to model those adaptations and to automate them when modifying the selection of features of a product. The evolution model describes changes to be made in the database, each composed of different actions that adapt certain elements of the database. Changes are associated with the features that may trigger their execution, and the change's actions are associated with the data model elements they affect. In this way, the evolution model supports automatic adaptation of the database, and we keep traceability between features and the elements of the data model they affect.

Keywords

software product lines, database evolution

1 Introduction

Software Product Lines (SPL) support the semiautomatic development of a family of software products. The products in this family are built from a common set of core assets and share many elements, but differ in certain features [5, 15]. One of the key elements of a software product line is the feature model that describes the variability of products in the family [2, 3]. The SPL allows the user to generate any product of the family by specifying the features that must be present in that product. The SPL automatically generates the product by assembling and adapting the core assets according to the selected features [14].

One of the challenges in SPL practice is software evolution, which is a necessity for both the SPL itself and the products generated with that SPL. The SPL platform requires maintenance, just like any other software system. This problem has attracted significant attention from the research community in recent years [11, 13]. In this paper, we do not address the evolution of the SPL platform but the evolution of the software products generated with an SPL.

A product generated with an SPL is defined by the selection of features it must include. Software requirements change over time. Therefore, after the product has been generated and it has been used for some time, the product's selection of features may need to be modified to meet changing requirements: we may need to add a feature not originally selected, we may need to remove a feature that is no longer needed, or we may need to replace a feature by another one. The SPL should support these changes in the selection of features of its products.

Adding or removing a feature to/from one of the generated products generated with an SPL has different implications in different parts of the product's architecture. From the point of view of software, such a change does not necessarily have to create a technical problem. We can use the SPL again, modify the original selection of features, rebuild the product, and deploy it to replace the old one. If the modified features do not affect other elements, such as integrations with other systems, the product can be redeployed without significant problems.

However, regenerating the product with a new selection of features can be a problem from the point of view of the product database, since it may already contain data that must be adapted to that new selection of features[1]. Therefore, the change in the product's selection of features creates the need for database transformation, both in its structure and the data it contains. In this case, a possible solution would be to migrate the data manually, that is, developing scripts or using some application that adapts the data to the new schema. However, this option does not align with the automation in the development achieved with SPLs. This is the problem we address in this paper: *defining a solution that allows to automatically*

*All authors have contributed equally and the names are listed in alphabetical order

[1] In the rest of the paper, we use the term database to refer to the database's schema and/or state. Furthermore, our examples assume that a relational database is used, although our proposal is applicable to any other type of database system.

This is the author's preprint of the following published work:
Cortiñas, A.; Luaces, M. R.; Pedreira, O.; Places, A. S.: "Adapting the Database to Feature Changes in Software Product Lines", in *Proceedings of the 27th ACM International Systems and Software Product Line Conference (SPLC 2023)*, ACM Digital Library, Tokyo (Japan), 2023, pp. 194–200. DOI: 10.1145/3579027.3608990

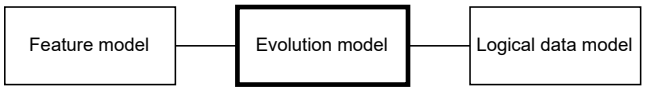

Figure 1: Evolution model for database evolution

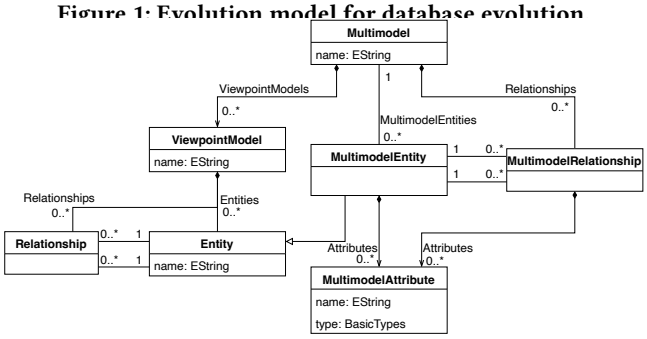

Figure 2: A metamodel of multimodels [4].

adapt the database schema and/or state of products generated with an SPL when the selection of features of those products is modified.

Our proposal follows a multimodel approach that establishes traceability between three system models (see Figure 1): (i) the feature model, (ii) the data model, and (iii) the *evolution model* that defines the changes that can be made in the product's selection of features and the actions they imply in the product's database. In our proposal, the *Features* of the feature model are associated with possible *Changes* in the selection of those features (that is, a selected feature is deselected in the next version of a product, or vice versa). These changes are defined in the evolution model. Each of these changes is composed of database adaptation *Actions* to be executed on the database if the change is applied. Each action involves a modification of the schema and/or the state of the database. Each of these actions is associated with the different elements of the logical data model that it can affect (relations, attributes, etc.), allowing us to maintain traceability from each feature in the feature model to each of the database elements it may affect if added to or removed from a specific product. In this way, we can have information on which elements of the database would be affected by a change in the selection of features.

By allowing us to model the changes that we can make in the product's selection of features and the actions they would trigger in the database, our proposal allows to automate the database adaptation of the generated products when they evolve and their selection of features changes. At the same time, identifying the changes that may be executed automatically for the products of an SPL, and defining them, facilitates their evolution since we know in advance how the feature selection can change effortlessly.

The rest of the article is structured as follows: Section 2 briefly presents background and related work. In Section 3 we present the evolution model, a proposal based on multimodels for modelling the changes in the database triggered by the selection or deselection of the features included in the product. Finally, Section 4 presents the conclusions of the article and lines for future work.

2 Background and Related Work

SPL Evolution and its Implications on the Database. The term *SPL evolution* generally refers to the evolution of the SPL platform, and sometimes also to the evolution of the products generated with that platform [9, 10, 12, 13]. Like any other software, an SPL evolves and undergoes changes that correct errors, modify functionalities, or add new elements. On many occasions, these changes must be propagated to the products generated using that SPL.

There is almost no literature that has considered the database evolution together with the evolution of products generated with an SPL. Herrmann et al. [8] address the problem of database evolution in the context of the evolution of the core components of a software product line. In their article [8], they present an industrial experience in which they use a framework called DAVE, which allows the integration of database scripts that derive from the evolution of each component and combines them into a single script that contains all changes to the database.

Multimodels in Software Product Lines. Traceability management has been a recurring concern in research in software product lines [1, 16]. The goal is to maintain the relationships between the different models, features, components, and other elements of an SPL. Multimodels are a tool that can be used to model the relationships between elements belonging to different systems' models.

A multimodel comprises different models of a system and different relationships between them. Each model represents the system from a specific point of view, with the relevant concepts and the relationships between them. In many cases, it is necessary to establish relationships between elements of different system models, either for traceability reasons or because one model has some implication in the elements of another model. Figure 2 shows the proposal of multimodels by [4]. The diagram shows that the multimodel defines specific associations between elements of different models, although these are kept outside each of them. In this way, each model is defined by itself, maintaining relationships between models in another specific model. In this article, we use the concept of multimodel. Our proposal contemplates that in an SPL we have its feature model and its data model, to which we add a product evolution model that defines the changes that can be made in selecting characteristics and the adaptations they imply in the database.

Multimodels have been used, for example, in [6, 7] to model the relationships between features of an SPL, components of the architecture, and quality attributes. In this case, the multimodel allowed for automatic analysis of which quality attributes were satisfied given a specific selection of features.

3 Modeling Database Adaptation Based on Feature Selection Modifications

In this section, we present our proposal for modeling and automating changes in the selection of features of a specific product and their associated actions to adapt the product's database schema and/or state. Without losing generality, in the examples we provide we assume the products use a relational database, although the proposal can be equally applied to any other database technology.

3.1 Description of the Problem

Changing the selection of features of a product generated with the SPL which is already in production implies changes in many system artifacts. For example, different modules or software components will change to implement the new behavior determined by the new

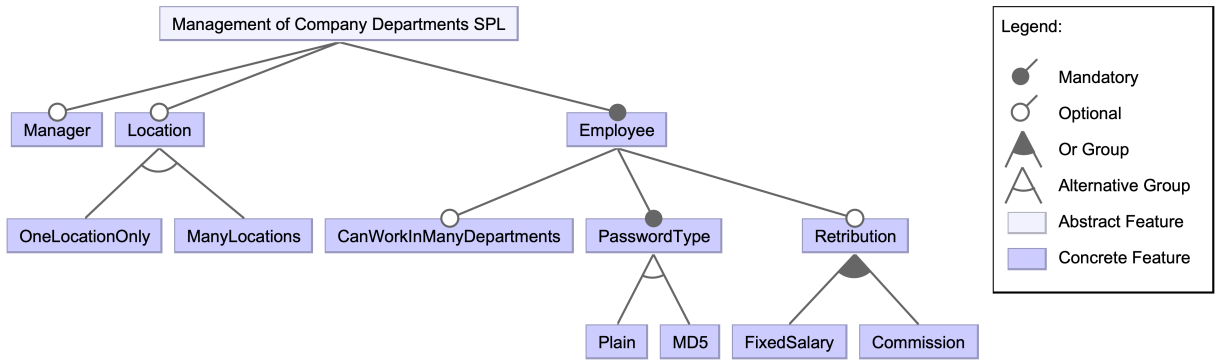

Figure 3: Sample feature model.

selection of features. The database would also have to change to support those features. The changes in the software components pose no significant challenges, since they have to be regenerated to support the new features and deployed again. However, the changes are more challenging in the case of the database since the product has already been in use and stores data that must be taken into account and, sometimes, adapted to the new features.

During the rest of this section, we will use a simple feature model as a running example, shown in Figure 3.

This small example considers a hypothetical SPL that supports the development of products for the management of the departments and the employees of a company. Let us think of this family of products as web applications that allow the administrator of a company to manage the different departments existing in the company, choosing which employees work in which department, identifying the manager of a department, setting the location of the departments, and also setting the salary of the employees. These employees can also log into the products of this SPL to check data about their departments and salary. For the rest of the paper, we will focus on the data supporting these features, but keep in mind that the features represent capabilities or functionalities of the products, not the data itself.

This SPL generates products for any company depending on the specifics of its organization and includes the features described below. 1) *Manager*: the departments of the company can optionally have a manager. 2) *Location*: departments can have zero, one, or many locations. 3) *Employee*. Departments have employees. 4) *CanWorkInManyDepartments*: whether employees can work in one single department or in many departments. 5) *PasswordType*: each employee will have access to the application, so they must have a password, for which we can decide whether it will be stored in plain text or encrypted using MD5. 6) *Retribution*: there is also variability present in the employee retribution scheme. An employee can have a fixed salary, but also the employee can have a commission on sales if we select the corresponding feature. The example feature model also considers that some employees can only have a commission but not a fixed salary (for example, for collaborators not hired as personnel in our company) or neither a salary nor a commission (as could happen in the case of partners of the organization).

In this example we can see many cases in which the database would be affected by an update in the selection of features of a product that has been already in use.

An interesting case is the possible changes in the features within the sub-tree of *PasswordType*. For a particular product, we can generate it storing the employee's passwords in plain text (*Plain*) or ciphered using MD5[2] (*MD5*). After the product is generated and used, the password of all employees will be stored in the database. What happens with the passwords in the database if the requirements of the product change and we need to switch from one alternative to another? Changing to *Plain* from *MD5* is a feasible change, since we can cipher existing data during the upgrade. However, changing to *MD5* from *Plain* implies losing all the passwords.

Another possible change in our simplified SPL is the selection or deselection of *Manager*. In this case, the change in the schema is to add or remove a *foreign key* (the Entity-Relationship supporting these products is shown in the bottom of Figure 5, where all the orange elements are variant depending on the selected features). If we select *Manager*, there is no loss of data. However, if we are deselecting *Manager*, there is a loss of data, the information of who is the manager of each department, and the platform should alert the engineer generating the new version. In fact, if there are no managers set, the change would not provoke loss of data, but in order to check that the SPL would require access to the products database, which is also an interesting idea.

As we can see with this example, changes in the selection of features of a product already in use may imply adaptations of the data managed by that product. We consider that these adaptations should be supported and automated by the SPL platform. However, as we have seen in this subsection, different modifications in the selection of features may imply data adaptations that must be solved in different ways.

3.2 Feature Selection and Database Evolution Model

In the application engineering phase, when creating a new product, an engineer decides which features have to be selected based on the specific requirements for it, and the software product line would assemble the components to generate the new product itself. Once the product is generated, it can be used as expected, and therefore all the data managed by the product shall be created normally.

[2]In practice, there is no point on storing a password in plain text in the database of an application, but, of course, this is a simplified example to illustrate the problem.

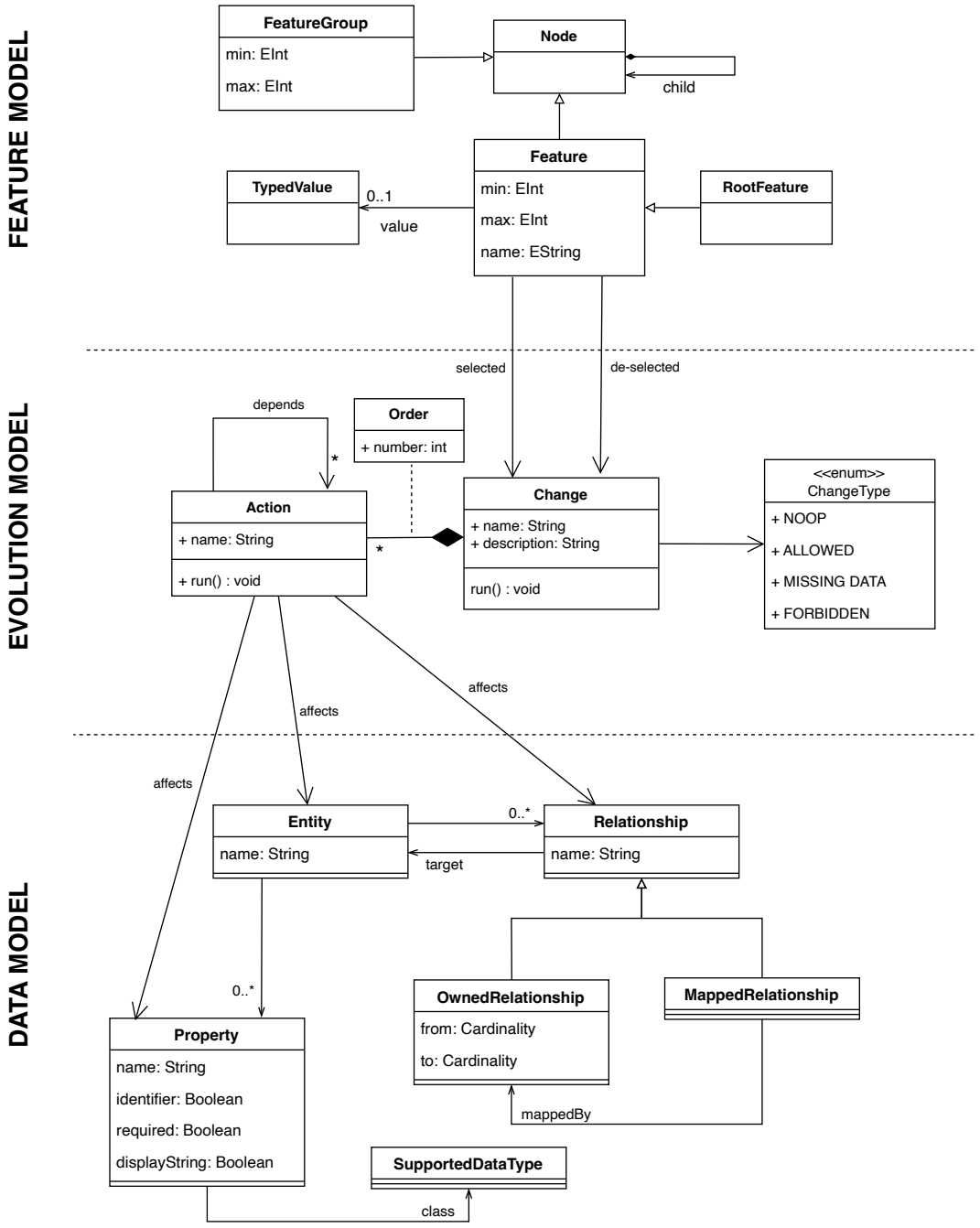

Figure 4: Metamodel of the Database Evolution Model.

At some point during the life cycle of the product, the requirements can change over time, and the engineer would generate a new version of the product with a new selection of features. Our solution aims to model the possible changes in the selection of features for iterative versions of products within an SPL. In the development of this model, we considered three requirements:

- *Definition of possible changes:* From the selection of features for the current version of a product, there are some features that could be selected or deselected for the next version of the product. That is, there is a set of possible changes. For example, if we generate a first version of a product for our running example SPL with the feature *Plain* selected (sub-feature of *PasswordType*), we could easily switch to the *MD5* in the next version of the product. This change is very simple; in order to preserve the data already being used in the application, we just need to cipher the passwords of the

employees. However, the opposite change, that is, changing from *MD5* to *Plain* for the next version of a product, cannot be done without losing data, as MD5 is not reversible. Therefore, we should define the first change as possible, and the second change as feasible but involving data loss.

- *Traceability:* Our model must know which features are related to each change. The model should also maintain traceability between changes in the features and elements of the data model that would be affected by those changes. In this way, the engineer would know the implications in the database of a change in feature selection.
- *Reuse of transformation actions:* Our model must consider that some data transformation actions will consist of smaller steps that may need to be reused among different changes. This avoids repeating the implementation of those change actions and reusing them in as many features as needed.

Figure 4 shows the metamodel of our solution to model the changes in the feature selection of a specific product. As we can see in the diagram, our product evolution model considers the feature model and the data model of the system. We add a new *evolution model* that considers the potential *Changes* in the selection of features. Each change is described by its name and description. In addition, each change is associated with the features that may trigger its execution. One change can imply deselecting one feature but selecting other features. Therefore, a change can be associated with many features, indicating for each of them if the change implies selecting or deselecting the feature. In this way, we maintain full traceability between the features in the feature model and the possible changes in their selection. This allows the SPL to show the consequences that a change in the selection of a feature would have on other features. In addition, each change has a *Change Type*:

- *NOOP* (*no operation*): this type of change indicates that the change does not affect the data or the data model, so it can be done without running any data model adaptation action.
- *ALLOWED*: This type indicates that the change is allowed and does not imply losing data; that is, the adaptations in the database can be done while preserving all the data we currently have.
- *MISSING DATA*: This type indicates that the change in the product's feature model is allowed but would imply losing some data. As we will see later, our solution allows identifying which information can be lost since we also keep traceability between the changes and the elements of the data model it affects.
- *FORBIDDEN*: This type indicates that the change is not allowed in the SPL. This would allow us to implement different changes but forbid some of them if necessary.

Each *Change* is composed of basic *Actions* that modify elements of the data model. A change can have one or many actions. In this last case, the association between a change and the actions it comprises defines the order in which those actions must be executed. Decomposing a change into many actions allows us to reuse modifications that may be part of different changes.

In addition, to keep the traceability between the features and the changes they imply when selected or deselected, we also consider it important to keep the traceability between the changes and the elements of the data model they affect. Our multimodel defines a relationship *affects* that connects *Actions* in the evolution model with the entities, properties, or relationships of the data model. In this case, we are assuming that our data model is modeled using Entity-Relationship, although our proposal could be easily modified to represent the data model with a different notation, or even to consider directly the model of the database (that is, establishing relationships between changes and tables, columns, and keys).

3.3 Application Example

Figure 5 shows an example of the application of our proposal. It addresses the changes in *PasswordType* for our running example. First, let us explain the figure itself and then proceed with the example. In the upper part of the figure, we can see the *feature model*. In the middle part of the figure, there is the *evolution model* with the changes and actions. Lastly, at the bottom of the figure, there is the *data model* represented as Entity-Relationship. The links between the features and the changes represent the selection (green arrow) or the deselection (red arrow) of a feature and the associated change. The numbers within the links between changes and actions represent the order in which the action is executed (see Figure 4). The elements of the data model which are painted in orange are variant, depending on the selected features and actions related. Since we focus on the changes in *PasswordType*, which affect two attributes, we colored in yellow the relationships with the attribute *plainPassword*, and in blue the relationships with the attribute *md5Password*.

There are four possible *changes* related to the feature *PasswordType*. We can see this just by looking at the feature model. The first time we generate a product, we shall select *Plain* or select *MD5*. From that moment on, every future version of the product can switch from one to another, but it can never be deselected since both *Employee* and *PasswordType* are mandatory. Since we also model the first selection as a change, we have just four changes: *SelectPlain*, *SelectMD5*, *MigrateToMD5*, and *MigrateToPlain*.

Each of these changes is linked to the features that trigger the change. For example, if the feature *Plain* is the only selected one, the change *SelectPlain* is triggered. However, if the feature *Plain* is selected and, at the same time (for the same upgraded version of a product), the feature *MD5* is deselected, then we are switching from one to another and the change *MigrateToPlain* is triggered. The same happens in both cases for *MD5* and *SelectMD5*. Each change has a *type*, depending on whether the change implies losing data (`MISSING DATA`) or not (`ALLOWED`).

Then, each of these changes executes one or many actions. For example, the change *SelectPlain* execute the action *CreatePlainColumn*, which is implemented as an SQL string. The change *MigrateToMD5* involves three actions that are linked in order: *CreateMD5Column*, *TransformPasswordToMD5*, and *DropPlainColumn*[3].

Finally, just to keep traceability between features, changes, and data model, we can see how each action, in this example, is related to the attribute *plainPassword* or with the attribute *md5Password*. In other cases, such as the ones described in Section 3.1, the actions could be linked to other elements of the data model, such as the

[3]In a real scenario probably the column for the password should not change, but for the purpose of the example, it is more clear if we have separated columns

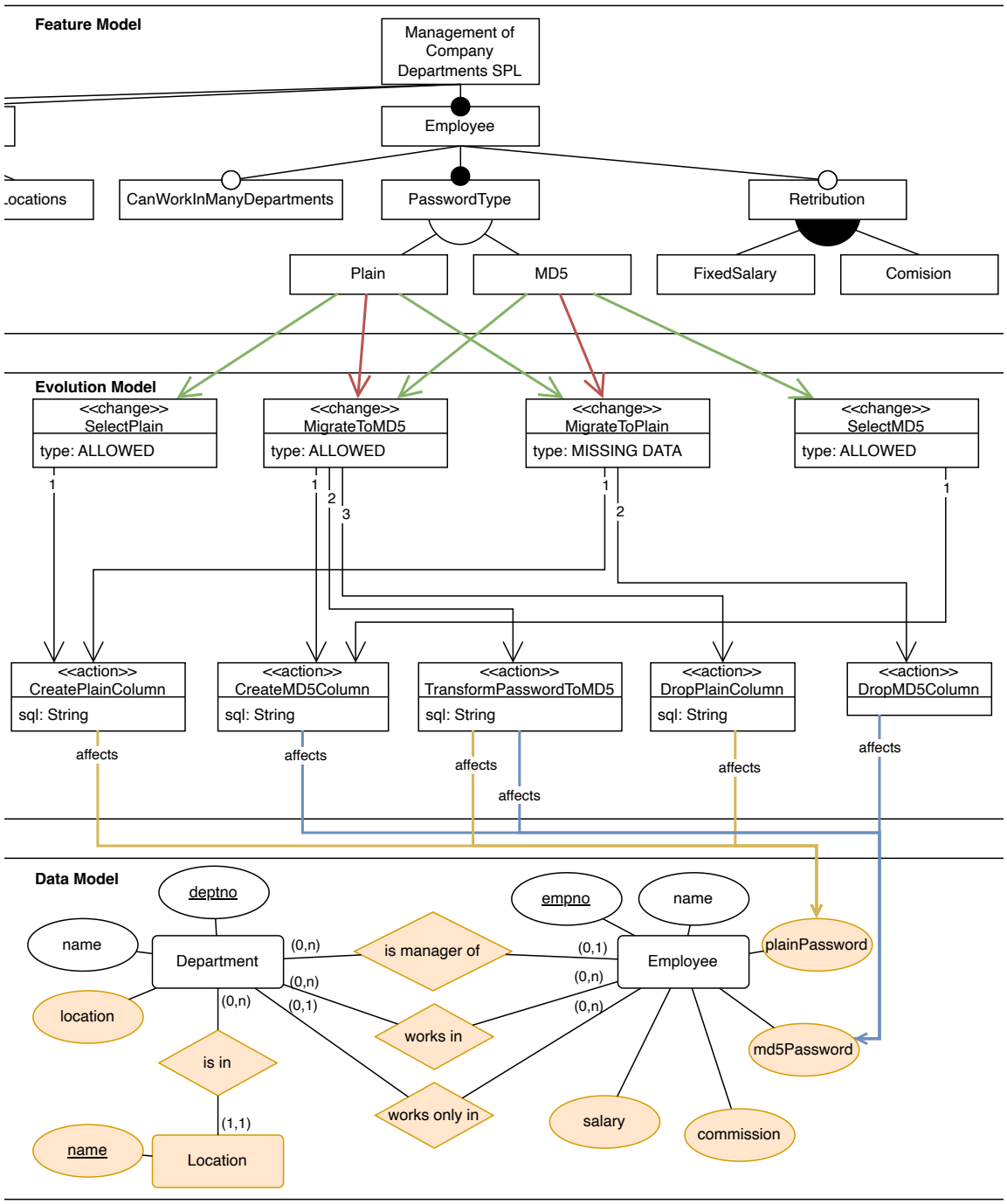

Figure 5: Example of changes and actions on the data model in a sample feature model.

relation *Location* or the relationships *works in* and *works only in* (which have different cardinality to support the feature *CanWorkInManyDepartments*).

As we can see in this application example, modeling the changes in the selection of features and the actions they may trigger to adapt the database allows the SPL to support the automatic update of the selection of features of products generated with the SPL that have already been in use and, therefore, contain data.

4 Conclusions

In this paper, we addressed the problem of modifying the feature selection of a product generated using a software product line that has been already in use and, therefore, contains data that may need

to be adapted to the new features. Adding or removing a feature from a generated product does not necessarily pose a challenge from the point of view of the software. Still, it does so from the point of view of the database since adding or removing a feature can affect the database's schema and state. The database could be adapted manually, but in this paper, we presented an evolution model that allows us to define the possible changes and their associated actions to be executed on the database if adding or removing a feature triggers those changes. We formalize the evolution model using the concept of multimodels. In this way, the definition of the SPL should include the feature model, the data model, and the evolution model. Our evolution model defines *Changes* triggered by changes in the feature selection of the product. Each change comprises many adaptation *Actions* on the database, which affect the elements of the data model. In this way, we keep the traceability between features, changes, actions, and elements of the data model, which allows us to have information on which elements of the database would be affected by a modification of the features of the product.

Our model defines many types of changes. Some changes do not affect the database. Others affect the database without losing data (`ALLOWED`) while others can be done but involve a loss of data (`MISSING DATA`). Our model also defines a `FORBIDDEN` type of change that allows us to restrict certain changes.

Some aspects of this project remain as lines for future work. For example, in this paper, we propose modeling predefined actions that reconfigure the database on a change in the selection of features. Still, each of those actions is then implemented as a script (either in SQL or in a general-purpose programming language). We are exploring the definition of these database adaptation actions in terms of transformations in the data model, since this would elevate the abstraction level of the solution.

Acknowledgments

Partially funded by: PID2021-122554OB-C33 (OASSIS) MCIN/AEI-/10.13039/501100011033 and EU/ERDF A way of making Europe; TED2021-129245B-C21 (PLAGEMIS) MCIN/AEI/10.13039/50110001-1033 and NextGenerationEU/PRTR; PDC2021-121239-C31 (FLAT-CITY-POC) MCIN/AEI/10.13039/501100011033 and NextGenera-tionEU/PRTR; PDC2021-120917-C21 (SIGTRANS) MCIN/AEI/10.-13039/501100011033 and NextGenerationEU/PRTR; PID2020-11463-5RB-I00 (EXTRACompact) MCIN/AEI/10.13039/501100011033; MA-GIST: PID2019-105221RB-C41 MCIN/AEI/10.13039/501100011033; GRC: ED431C 2021/53, partially funded by GAIN/Xunta de Galicia; CITIC is funded by the Xunta de Galicia through the collaboration agreement between the Department of Culture, Education, Voca-tional Training and Universities and the Galician Universities for the reinforcement of the research centers of the Galician University System (CIGUS).

References

[1] Nicolas Anquetil, Uirá Kulesza, Ralf Mitschke, Ana Moreira, Jean Claude Royer, Andreas Rummler, and André Sousa. 2010. A model-driven traceability framework for software product lines. *Software and Systems Modeling* 9, 4 (2010), 427–451. doi:10.1007/S10270-009-0120-9

[2] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2016. *Feature-oriented software product lines.* Springer.

[3] Sven Apel and Christian Kästner. 2009. An overview of feature-oriented software development. *Journal of Object Technology* 8, 5 (2009), 49–84.

[4] Edward Barkmeyer, Edward Barkmeyer, Peter Denno, Allison Barnard Feeney, David Flater, Donald E Libes, Michelle Potts Steves, and Evan K Wallace. 2003. *Concepts for automating systems integration.* Technical Report NISTIR 6928.

[5] Paul Clements and Linda Northrop. 2002. *Software product lines.* Addison-Wesley Boston.

[6] Javier Gonzalez-Huerta, Emilio Insfran, and Silvia Abrahao. 2012. A multimodel for integrating quality assessment in model-driven engineering. In *Procs. of the 8^{th} Int. Conf. on the Quality of Information and Communications Technology (QUATIC 12).* IEEE Press, 251–254. doi:10.1109/QUATIC.2012.14

[7] Javier Gonzalez-Huerta, Emilio Insfran, and Silvia Abrahão. 2013. Defining and Validating a Multimodel Approach for Product Architecture Derivation and Improvement. In *Procs. of 16th Int. Conf. on Model-Driven Engineering Languages and Systems (MODELS 13)*, Vol. LNCS 8107. Springer, 388–404. doi:10.1007/978-3-642-41533-3

[8] Kai Herrmann, Jan Reimann, Hannes Voigt, Birgit Demuth, Stefan Fromm, Robert Stelzmann, and Wolfgang Lehner. 2015. Database Evolution for Software Product Lines. In *Procs. of 4th Int. Conference on Data Management Technologies and Applications (DATA 2015).* IARIA, 125–133.

[9] Jacob Krüger, Wardah Mahmood, and Thorsten Berger. 2020. Promote-Pl: A Round-Trip Engineering Process Model for Adopting and Evolving Product Lines. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A* (Montreal, Quebec, Canada) *(SPLC '20).* Association for Computing Machinery, New York, NY, USA, Article 2, 12 pages. doi:10.1145/3382025.3414970

[10] Miguel A. Laguna and Yania Crespo. 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming* 78, 8 (2013), 1010–1034. doi:10.1016/j.scico.2012.05.003

[11] Maíra Marques, Jocelyn Simmonds, Pedro O Rossel, and María Cecilia Bastarrica. 2019. Software product line evolution: A systematic literature review. *Information and Software Technology* 105 (2019), 190–208. doi:10.1016/j.infsof.2018.08.014

[12] Leticia Montalvillo and Oscar Díaz. 2015. Tuning GitHub for SPL Development: Branching Models & Repository Operations for Product Engineers. In *Proceedings of the 19th International Conference on Software Product Line* (Nashville, Tennessee) *(SPLC '15).* Association for Computing Machinery, New York, NY, USA, 111–120. doi:10.1145/2791060.2791083

[13] Leticia Montalvillo and Oscar Díaz. 2023. *Evolution in Software Product Lines: An Overview.* Springer International Publishing, Cham, 495–512. doi:10.1007/978-3-031-11686-5_20

[14] Linda M. Northrop and Paul C. Clements. 2012. *A framework for software product line practice, Version 5.0.* Technical Report. Software Engineering Institute (SEI), Carnegie Mellon University.

[15] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. 2005. *Software product line engineering.* Springer.

[16] Tassio Vale, Eduardo Santana de Almeida, Vander Alves, Uirá Kulesza, Nan Niu, and Ricardo de Lima. 2017. Software product lines traceability: A systematic mapping study. *Information and Software Technology* 84 (2017), 1–18. doi:10.1016/J.INFSOF.2016.12.004