Towards Managing Data Variability in Multi Product Lines

Niloofar Khedri and Ramtin Khosravi
School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran

Keywords: Multi Product lines, Software Product Line Engineering, Data Model Variability, Delta-oriented Programming.

Abstract: Multi product lines (MPLs) are systems consisting of collections of interdependent software product lines (SPLs). The dependencies and interactions among the SPLs cause new challenges in variability management. In the case of a large-scale information system MPL, important issues are raised regarding integration of the databases of the individual SPLs comprising the main system. The aim of this paper is to introduce a method to manage the variability in the data model of such systems. To this end, we first address the problem of developing a universal feature model of the MPL, obtained from integrating the feature models of the individual SPLs, incorporating the data interdependencies among the features. Further, we develop the data model of the MPL using a delta-oriented technique, based on the universal feature model. Our method addresses the problem of possible conflicts among the data model elements of different SPLs and proposes techniques to resolve the conflicts based on data model refinements.

1 INTRODUCTION

Software product line (SPL) approach is based on defining the commonalities and variabilities of the products and building reusable platforms. A feature model contains the compact representation of all products of software family in terms of features (Pohl et al., 2005).

"Multi product lines (MPLs) are sets of several self-contained but still interdependent product lines, together representing a large-scale or ultra-large-scale system" (Holl et al., 2012a). As MPL is a system including dependent systems, the main issues to manage variability include the dependency management among multiple systems and consistency checking across them (Holl et al., 2012a). The main role of the data and the importance of managing data variability in information systems as well as the lack of variability management method in MPLs lead us to the study of the variability in data model of MPLs in the area of information systems. Having an MPL of information systems implies some kind of data integration between the system. Designing a single integrated database schema is a solution to this problem which benefits from the capabilities such as defining referential integrity across the subsystems, global optimizations, partitioning and clustering.

The main goal of the study is to present a method to develop a single data model for an MPL of several information systems, each having a separate data model. Our method to manage the variability in the data model of the MPLs is based on a delta-oriented technique which uses the feature model as the basis to model variability in terms of delta modules. Hence, we need a way to represent the variabilities in the MPL in a universal feature model. As there is no common well-known method to derive such a feature model for MPLs, we first solve this problem by integrating the feature models of the subsystems focusing on the data interdependencies among them. Based on the universal feature model, we present a delta-oriented method to generate the data model for the MPL.

We have used our method in a case study which is a family of university software systems and library information systems. This study is part of a project, currently being done in the Software Architecture Laboratory in University of Tehran on developing a middle-sized MPL for university-related information systems. Figures 1 and 2 illustrate the simplified feature models of our running example of the university and library product lines. Our main contributions in this paper are summarized below:

- Providing a method to create a universal feature model for MPLs focusing on the data interdependencies among systems.
- Presenting a method to generate the MPLs data model with regards to the interdependencies among systems.

Khedri N. and Khosravi R..
Towards Managing Data Variability in Multi Product Lines.
DOI: 10.5220/0005227005230530
In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2015), pages 523-530
ISBN: 978-989-758-083-3

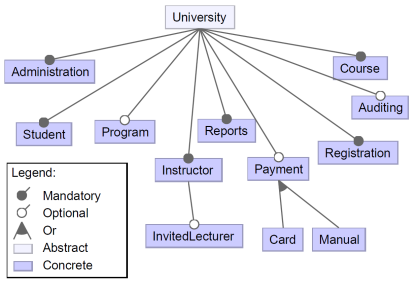

Figure 1: Simplified university feature model.

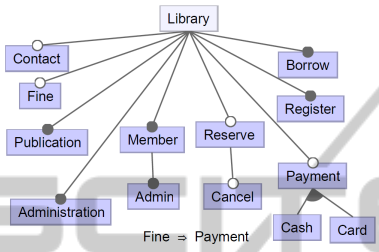

Figure 2: Simplified library feature model.

- Reducing the complexity of generating MPLs data model through systematic reuse across data model development of systems by presenting the MPLs core data model and delta models.

2 DATA MODEL VARIABILITY IN SPLS

In our previous work(Khedri and Khsoravi, 2013), we presented an approach to manage variability in database design of information system product lines based on delta-oriented programming (DOP) technique (Schaefer et al., 2010). In DOP, a core module contains the mandatory features of the product family and delta modules are linked to the alternative and optional features. A product is created by applying delta modules to the core module incrementally.

In our method (Khedri and Khsoravi, 2013), the core includes the DDL scripts of the mandatory (and some selected alternative and optional) features and a delta consists of the DDL scripts of the changes to the core to implement the related feature. The DDL scripts related to the deltas includes the related tables, columns, primary keys, foreign keys and constraints to implement the feature in data model. The deltas are applied to the core in a sequence, derived by applying topological sort on the dependency graph of the selected configuration.

The consistency of the resulting database is checked during the process and the conflicts are handled by creating new DDL scripts or changing the existing ones. For instance, the selected configuration of the sample SPL is depicted in Figure 3. The core consists of the mandatory features A and B. The dependency graph related to selected configuration is shown in Figure 3. One sequence of applying deltas to the core according to the result of applying topological sort on the dependency graph is *C*, *G* and *H*. The whole method of applying DDL delta scripts to the core is illustrated in Figure 3.

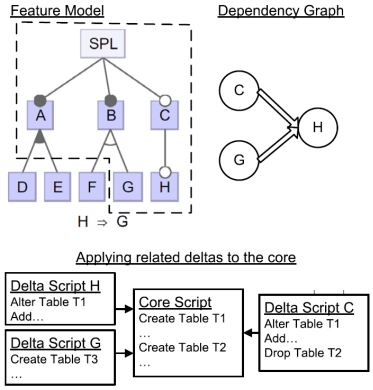

Figure 3: A sample SPL configuration and applying the related deltas to the core.

3 GENERATING FEATURE MODEL FOR MPL

In this paper, we generate the feature model of the MPLs from that of the product families and their data model, in contrast to the SPL approach, in which the feature model is generated at the end of the domain requirement engineering through analysis of the variable and common requirement of the product line (Pohl et al., 2005).

3.1 Matching and Mapping Features of SPLs

Our method is based on comparison and analysis of the core and deltas to find the matching features. Since the data model contains the detail of the system at the database level, the result of matching the data model elements can make the matching of features more accurate. It is important to note that the features implementation at the data model level includes the details that make easier the identification of the differences and similarities of the features. For instance, the features *Registration* in *University* and *Register* in *Library* are matched just by looking at the feature

models. But in our method, in the core data model of the *University*, the *Registration* feature represents that the student takes some courses each term. In *Library* data model the *Register* feature corresponds to the membership of the students in library. As a result, the details of data models do not show any similarity between these features, also the *Registration* feature is implemented by a more precise relationship in *University* data model (*Takes*)[1]. Having a well-defined meta-model, the data model can be used as a guide to semantically differentiate between two features.

The Core and Delta Models Element Matching. At the first step, the core and delta models of the individual SPLs are compared and the set of matching schema elements are identified. We use name matching method to find the schema element with equal or similar names (Palopoli et al., 1998). In some cases, two elements have the same name but different concept in the core data models (called homonyms in database concept (Batini and Lenzerini, 1984)), so to prevent further conflicts in schema merging we rename one and modify all the deltas consisting the element (see Section 4.3).

Feature Matching. The set of features related to the matched elements are identified according to the mapping of the core and delta elements in data model to the features. As mentioned in Section 2, each data model element in the core or deltas is related to at least one feature. At the end of this step, a set of potentially matched features is generated: $\{(Student, Member), (Instructor, Member), (Admin, Admin), ...\}$

Feature Mapping. In this phase, the relation between the matched features is identified. We classify the relation between the features in three categories as listed below:

- Equivalent ($\equiv$): Feature F_1 from product family SPL_1 is to the feature F_2 from product family SPL_2 ($SPL_1.F_1 \equiv SPL_2.F_2$), if they represent the same concept. They can be renamed to have the same name.
- Generalized ($\sqsubseteq$): One feature can be generalized to the other one ($SPL_1.F_1 \sqsubseteq SPL_2.F_2$). It means that F_2 is a generalization of F_1. For instance, *"University.Student* $\sqsubseteq$ *Library.Member"* means that a *Student* in the *University* is a *Member* of *Library*.
- Undecided ($<>$): In some cases we cannot make a decision on the proper and reasonable relation between the matched features. For instance, the *Admin* in *University* and *Admin* in *Library* data models are mapped onto each other, but we cannot decide about these features, because they may or may not have the same functionality. In such cases, the customer's requirements of the MPLs define the relation between the features.

[1]University data model: http://khorshid.ut.ac.ir/~n.khedri/FigureUCore.png

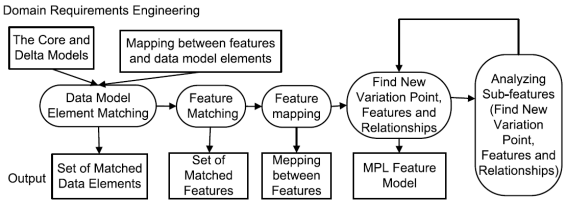

Figure 4: MPLs feature model methodology.

For example, analyzing the running example including *University* and *Library* shows a list of relations between the features including: (*University.Student* $\sqsubseteq$ *Library.Member*, *University.Instructor* $\sqsubseteq$ *Library.Member*, *University.Administration* $<>$ *Library.Admin*)

3.2 MPLs Feature Model

The goal here is to generate a data-focused feature model for MPLs containing all the features of the SPLs and their interrelations. To generate the feature model, a new feature model is created. The root feature is the MPLs, and its sub-features are the feature models of the product families in MPLs.

At first, we analyze each identified relation between features *A* and *B* (equivalent, generalized and undecided), to find a new possible variation point including new requirements or business rules and improve the MPLs feature model by adding integration features or new relations. For instance, when the *generalized* relation between *Instructor* and *Member* is identified (*Instructor* $\sqsubseteq$ *Member*), we can ask some questions as below: "Does the library lend books to Instructors?" "Is it necessary to have special rules to lend books to Instructors (for example, they can lend more books than general members)?" So, there is a variation point in *Library* feature model, and a new integration feature *InstructorMember* is added under the *Member* specifying whether the feature is mandatory, optional or alternative as well as its relationships to other features. For example, the *InstructorMember* is an optional feature (depicted in Figure 5). The interrelations between features of multiple systems are shown by "Requires" and "Excludes" relations between features.

We add the new variation point, features and relationships to the MPLs feature model. In some cases, the variation point and new features are added to the sub-tree of a product family. But in some other cases,

features are added under the MPLs' root which indicates a new variability that is created when combining the systems together. As a result, an integration feature is added to the MPLs feature model in accordance with the new variation points generated by integrating systems. An integration feature is added to the MPLs feature model in two levels: SPLs-level and MPLs-level.

Second, we iteratively analyze the sub-features and relationships of the matching features. For example, we should analyze the sub-features and relationships of *Instructor* and *Member* (*Instructor* $\sqsubseteq$ *Member*). *InvitedLecturer* is a sub-feature of the *Instructor*; the new questions are: "Does the library lend books to guest instructors?" "Is it necessary to have special rules to lend books to guest instructors?" As a result, a new feature *InvitedLecturerMember* is added under the *InstructorMember* with the new relation "*InvitedLecturerMember* $\Rightarrow$ *InstructorMember*".

Admin is a sub-feature of the *Member* in *Library* feature model; the questions are: "Can the library admin be a student at the university?" "If yes, is there any difference between a student and a librarian (as a student) for the University?" As there is no difference for the university between a library admin and a student, no new business rules is added and the MPLs feature model is not changed. The resulting feature model related to our case study including the university education and library systems is depicted in Figure 5.

In the case of an undecided relation between two features, the domain experts of the systems must analyze the systems and study the features, their functionality and role in each system and multi-system. In the case study, handling the "*University.Administration* <> *Library.Admin*" implies that administration operations are not the same in these systems. In this case, the feature model of the individual systems is modified and a new variation point is identified in the MPLs-level, which implies the variability on handling administration operation in the whole system (*Administration*, *Centralized* and *PerProduct*, depicted in Figure 5).

4 MPL DATA MODEL

Our method to build the data model of the MPLs is based on delta-oriented techniques (Khedri and Khosravi, 2013). The obvious method of designing data model for the whole MPL from scratch is not practical, since we lose the reuse obtained from applying SPL method on the individual subsystems. Furthermore, designing the MPLs core requires a team of different domain experts, not always available, and MPL is a(n) (ultra-)large scale system, the design and implementation of the core and all the deltas of which are complicated due to the interrelations between the systems.

In the presented method, we suppose to have the set of cores and deltas related to each family. First, we build a modified version of the core data models of each product, consisting of the interdependency between the core features of them. Also, we create a set of integration delta models related to the integration features of MPLs feature model. Then, the modified core data models are merged to create a core related to the MPLs. Finally, the data model of the MPLs is created by adding deltas associated with the selected features to the MPLs core. Given that the interrelations between systems are handled in modified core and MPLs-deltas, the presented method is more practical and understandable in comparison with the integrating data models method (mentioned above).

4.1 Integration Delta Models

In section 3, we present a method to generate a feature model consisting of all the features of MPL system and their relations in one feature model. In some cases, a new integration feature is added to the feature model to represent an interrelation between two systems. For instance, *StudentMember* is added to the *Library* feature model, indicating that student can be a member of library. In this step, we provide integration delta models for each integration feature.

The integration delta related to MPLs-level integration feature, named MPLs-delta, may integrate more than two systems in the MPLs and consequently, can affect more than two data models. Hence, the integration delta model generation is an important phase in our method possibly preventing further conflicts in data model generation.

4.2 Modified Core Delta Models

In the delta-oriented approach, we first build a core for MPLs and then apply delta models on it to generate the MPLs data model. In our method, the core of each system consists of the mandatory features of the family. We cannot merge the cores of the multiple systems without considering the interdependencies. Consequently, each core should be modified to include the interdependencies. At this step, the core features of each product based on the related sub-root in MPLs feature model are defined. For example, *StudentMember*, added to the *Library* product is

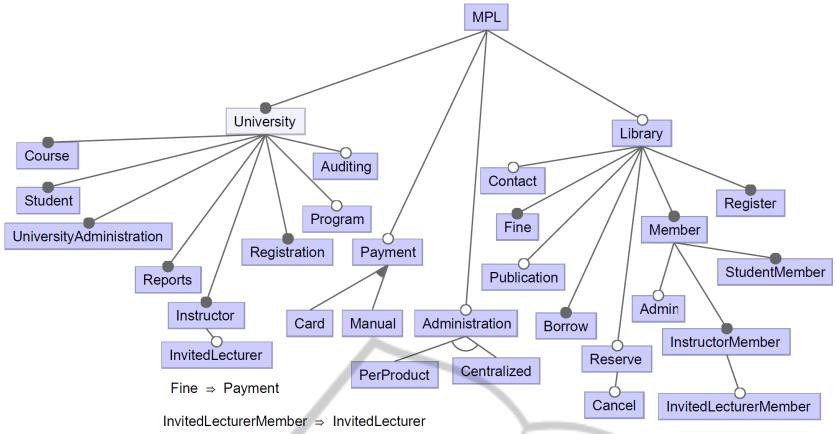

Figure 5: MPLs feature model.

a mandatory integration feature and is selected in the set of core features. We have "*Student IS – A Member*" as the relationship between the entities.

Then, we apply the new integration features and new relations to the related product core by applying the related DDL script to the core and resolve probable conflicts (Khedri and Khsoravi, 2013). As a result, each core is modified to include its integration features.

4.3 Generating the MPLs Data Model

In this step, the core data model of the MPLs is generated by integrating the core data models of the products, using integration (and merge) schema methods in database context (Pottinger and Bernstein, 2003). Some conflicts may arise during the integration process. These conflicts are resolved by database solutions (Pottinger and Bernstein, 2003). The output of this phase is the core data model of the MPLs.

Our method to merge the schemas, based on the schema integration methodology in the ER models, is explained in (Batini and Lenzerini, 1984). The method contains three main phases: conflict analysis, merging and restructuring. In conflict analysis the naming conflicts and modeling compatibility is checked. The names of the elements are compared in order to set a unique name to the matched elements in the integrated schema. Two possible cases can be identified here: synonyms and homonyms. Homonym is the case in which two unmatched elements have the same name, handled initially by renaming one element to another name during the element matching step (Section 3.1). Modeling compatibility analysis contains the type checking analysis, transforming the element type to another (for instance transform an entity to an attribute). Then the integrated schema is enriched and restructured to find new properties in integrated schema. Note that we keep track of all changes to the schemas in conflict analysis and restructuring phases to do the same changes on related deltas. After that schemas are merged. At last, the SQL script of the MPLs Integrated core data model is generated for the next phase.

In this section, the MPLs data model is generated by applying DDL delta scripts related to the MPLs selected configuration on the integrated core data model. The sequence of applying deltas is defined according to the relation between features in MPLs feature model by creating a dependency graph of selected features and then running topological sort on it.

4.4 Methodology

Software product line engineering (SPLE) has two concurrent processes: domain engineering and application engineering (Pohl et al., 2005). The commonalities and variabilities of the products are defined in domain engineering and a reusable platform is developed. The real products based on the customer requirements are generated during the application engineering. MPL is a collection of interconnected product families, so our methodology can be presented based on the processes of the SPLE mentioned above (Figure 6 illustrates our methodology).

As we present a method to generate the data model of the MPLs in database implementation level, all the activities in our method are considered as parts of the domain realization and application realization sub-processes of domain and application engineering processes respectively.

In domain realization sub-process of the domain engineering process, the detailed design and implementation of the software are provided to be reused

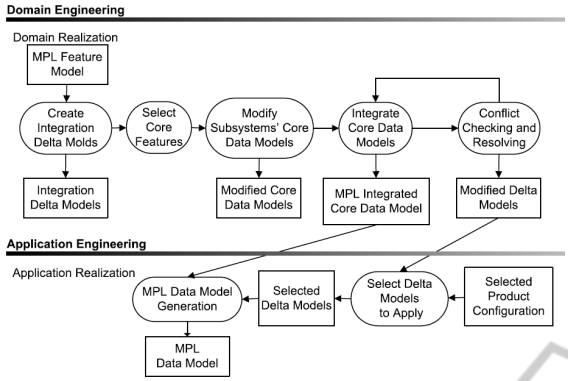

Figure 6: The presented methodology.

in application realization (Pohl et al., 2005). As a result, the conclusion can be drawn that the high level structure of the data model of the MPLs is generated in domain realization sub-process too. The core and deltas of the MPLs are implemented in domain realization.

5 DISCUSSION

Applying the Method on Global Schema. In the case of using global schema instead of the core and deltas, the mapping between global schema elements and features must be identified too. Our approach to find the mapping is to identify the database operation for each feature. For example, feature *Register* in *University* indicates that the courses for the new term are shown first (*select* operation on *Course* and *Term* tables), and then each student choose the desired courses (insert operation in the table which show the relation between *Student*, *Course* and *Term*). Analyzing the database operation performed in each features can result in a mapping between the database objects and the features, which in turn can be used as a guide to match features as described.

5.1 MPLs Feature Model

Integrating Feature Models. In the method presented, the mechanism to generate a universal feature model for the MPLs is based on matching feature models incorporating the data models of the product families in MPLs. This requires a teamwork between the domain experts of the individual SPLs, as well as their data modelers. Also, in the case that the method is applied to global schemas, the presented method requires added work on the schemas to find out the relationship between features and schema elements.

As the main focus of our method is to develop the data model of the MPLs, the universal feature model may miss some interdependencies and new features that do not affect the data access layer, and hence, cannot be considered as a complete variability model for the MPL. In case such a model is needed, separate effort based on other viewpoints (e.g. user interface) is needed.

5.2 MPLs Data Model

Conflict Resolution. As mentioned before, it is not possible to merge the data model of the product families in MPLs due to the (ultra-) large size of the data models and the inconsistency among data models. The inconsistency among data models results in conflicts in schema integration and the large size of the data models reduces the possibility of finding the source of the conflict. The data model of the MPLs is, as a result, generated in three steps and the emerging conflicts are resolved in each step.

The conflicts in the first step can be handled by reviewing the deltas in conflict and build a new delta for them. The goal of the second step is generating the MPLs core data model. In conflict analysis and transforming phases, the data model may change due to the conflict or inconsistencies between schemas. Moreover, the delta models are changed to be consistent to the MPLs core data model. In the last step, the data model of the MPLs is created by applying deltas to the core data model of MPLs. The conflicts in this step usually have two reasons. First, the dependency among the deltas models causes the conflict. Such conflicts can be handled by deriving a new delta for the features in conflict. Second, the changes on the modified cores cause an inconsistency between the MPLs core and deltas. We can solve such conflicts by transforming the deltas to a new one that changes the core data model to implement the related feature.

Data Model Completeness. The MPLs data model consists of all the elements needed to implement the related features in each product family as well as the interconnections among families. There is no element in the data model, redundant due to the selected feature model.

Data Integrity. The integrity of the MPLs data model is handled through defining the primary and foreign keys and constraints between database elements (collectively named constraint scripts). In the proposed method, we define constraint scripts in the related DDL scripts such as the core and deltas. In the first applying of deltas to the family core and in the last phase consisting of applying deltas to the MPLs core to generate the MPLs data model, delta scripts including the constraint scripts are applied and the database

checks the correctness of the constraint scripts and their references. As a result, the database itself checks the integrity of the data.

Complexity of the MPLs Schema. In the proposed method, the complexity of the MPLs meta-model is decreased because the MPLs data model consists of the required database elements and there is no redundant element in the MPLs data model.

6 RELATED WORK

The capabilities to support the MPLs are reviewed in (Holl et al., 2012a) such as sharing and distribution of variability models, defining and managing the different types of the dependencies and consistency checking as we face with them in this paper.

6.1 Variability Modeling in MPLs

Some existing work have studied merging feature models. Cross product line analysis proposed a method based on similarity metrics and clustering to analyze the commonalities and variabilities of various related SPLs (Wulf-Hadash and Reinhartz-Berger, 2013). Another approach for merging feature models based on graph transformation is discussed in (Segura et al., 2008), relying on having features with the same name and the same roots in the feature models. In MPLs, the systems do not necessarily have the same root and features with the same name. Merge and insert operators on feature models are developed as composition operators (Acher et al., 2009) and then a domain specific language named FAMILIAR presented to support the large scale management of feature models (Acher et al., 2011). In our method, a new feature model with new features and relations is generated for the MPLs in contrast to the merging feature models method (Wulf-Hadash and Reinhartz-Berger, 2013; Segura et al., 2008).

In (Hartmann and Trew, 2008), the context variability model meaning the variability of environment is combined to the feature model to support dimensions in context space and proposes a consistent context feature model. In (Rosenmüller et al., 2008), dependent SPLs are modeled in a class diagram that shows the SPLs and their dependencies. Then, in (Rosenmüller and Siegmund, 2010), a method to design and configure these systems is presented which automatically creates the configuration generators. In (Hartmann and Trew, 2008; Rosenmüller et al., 2008; Rosenmüller and Siegmund, 2010), the consistency and interdependencies between various SPLs are kept in a single model for the MPLs similar to our method. In our method, our focus is on data dependency and data consistency, whereas in (Hartmann and Trew, 2008) and (Rosenmüller et al., 2008; Rosenmüller and Siegmund, 2010), the focus is on the context and the relationship between different SPLs such as use, respectively.

In (Holl et al., 2012b), the MPLs is modeled and deployed by product line bundles. Product line bundles consist of the table of contents and meta-data, the variability model, organizational policies, dependencies to other bundles, and an expiry date. In (Holl et al., 2012b), the set of dependencies between product lines is defined as *inferred*, *confirmed* and *formalized*. In this work, the dependencies between systems are defined and modeled like in our method. In our method, the dependencies between systems are handled by the feature model relations *require* and *exclude*, and we concentrate on the data interdependencies between systems. Moreover, we identify the new integration features, especially at the MPLs-level that show new services, added to the whole system by using MPLs. It is important to note that our method is not in contrast with existing techniques on MPL feature modeling and other methods to integrated the feature models mentioned above can be employed in our method during the development of the universal feature model.

6.2 Managing Variability in Data Model

There are some studies on modeling data variability in SPLs (Zaid and Troyer, 2011; Bartholdt et al., 2009; Siegmund et al., 2009; Schäler et al., 2012). A method to manage data variability in domain model layer (not database design and implementation) is presented in (Bartholdt et al., 2009). Managing variability in the domain layer makes database designer check the database constraints in domain or application layer probably leading to data inconsistency. Also, in (Siegmund et al., 2009), a global consistent schema is created out of different local schemas for users by using view integration techniques. In (Schäler et al., 2012), the work is extended to modeling the variable schemas based on features and then generates the tailored schemas automatically by superimposing variable schemas, namely entity-relationship diagrams. The schema composition methods have the limitation on removing some parts of the database elements as well as the transforming database element type, but using the delta-oriented method proposed, we can apply the mentioned changes to database. For example, attribute *A* changes to a new table *A*. This change is impossible in schema composition approaches, but in our method, a delta script can handle it.

In (Zaid and Troyer, 2011), data model variability is managed in a single data model. The proposed method leads to a complicated data model when the size of the product family increases. Note that in (Zaid and Troyer, 2011), the entities and attributes are the variable database elements, but in our method there is no limitation on database elements. As mentioned in (Holl et al., 2012a), MPLs representation in a single model is hard due to its size and complexity. Consequently, we build the data model of the MPLs by generating the data model of the mandatory part of each data model, merging them, and applying the variable part at the end.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the feature model representation and data model variability problem in MPLs. First, we introduced a method for generating a universal feature model according to the data model interdependencies, extracted from data model element matching. Second, we propose a three-phase method to generate the MPLs meta-model based on delta-oriented programming and schema integration methods. The proposed method has been applied on the *University Education Systems*, *Reserve Food Systems* and *Library Systems*. As the proposed method is in progress, we plan to analyze the influences of the software evolution, adding feature and modify existing ones on both techniques: generating a universal feature model and a single data model for the MPL.

Our method is planned to support the interdependencies in the application and presentation layers and propose a universal feature model for MPLs in future. Furthermore, future work includes tool support for automatic generation of the MPLs data model that visually represents the cores and delta models related to each product family.

REFERENCES

Acher, M., Collet, P., Lahire, P., and France, R. B. (2009). Composing feature models. In *SLE '09*, pages 62–81.

Acher, M., Collet, P., Lahire, P., and France, R. B. (2011). Managing feature models with familiar: a demonstration of the language and its tool support. In *VaMoS '11*, pages 91–96.

Bartholdt, J., Oberhauser, R., and Rytina, A. (2009). Addressing data model variability and data integration within software product lines. *International Journal On Advances in Software*, 2:84–100.

Batini, C. and Lenzerini, M. (1984). A methodology for data schema integration in the entity relationship model. *IEEE Trans. Software Eng.*, 10(6):650–664.

Hartmann, H. and Trew, T. (2008). Using feature diagrams with context variability to model multiple product lines for software supply chains. In *SPLC '08*, pages 12–21.

Holl, G., Grnbacher, P., and Rabiser, R. (2012a). A systematic review and an expert survey on capabilities supporting multi product lines. *Information and Software Technology*, 54(8):828–852.

Holl, G., Thaller, D., Grünbacher, P., and Elsner, C. (2012b). Managing emerging configuration dependencies in multi product lines. In *VaMoS '12*, pages 3–10.

Khedri, N. and Khsoravi, R. (2013). Handling database schema variability in software product lines. In *APSEC '13*, pages 331–338.

Palopoli, L., Saccà, D., and Ursino, D. (1998). Semi-automatic semantic discovery of properties from database schemas. In *IDEAS '98*, pages 244–253.

Pohl, K., Böckle, G., and Linden, F. J. v. d. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag.

Pottinger, R. A. and Bernstein, P. A. (2003). Merging models based on given correspondences. VLDB '03, pages 862–873.

Rosenmüller, M. and Siegmund, N. (2010). Automating the configuration of multi software product lines. In *VaMoS '10*, pages 123–130.

Rosenmüller, M., Siegmund, N., Kästner, C., and ur Rahman, S. S. (2008). Modeling dependent software product lines. In *McGPLE '08*, pages 13–18.

Schaefer, I., Bettini, L., Bono, V., Damiani, F., and Tanzarella, N. (2010). Delta-oriented programming of software product lines. In *SPLC '10*, pages 77–91.

Schäler, M., Leich, T., Rosenmüller, M., and Saake, G. (2012). Building information system variants with tailored database schemas using features. In *CAiSE '12*, pages 597–612.

Segura, S., Benavides, D., Ruiz-Cortés, A., and Trinidad, P. (2008). Generative and transformational techniques in software engineering II. chapter Automated Merging of Feature Models Using Graph Transformations, pages 489–505. Springer-Verlag.

Siegmund, N., Kstner, C., Rosenmller, M., Heidenreich, F., Apel, S., and Saake, G. (2009). Bridging the gap between variability in client application and database schema. In *German Database Conference '09*, pages 297–306.

Wulf-Hadash, O. and Reinhartz-Berger, I. (2013). Cross product line analysis. In *VaMoS '13*, pages 21:1–21:8.

Zaid, L. A. and Troyer, O. D. (2011). Towards modeling data variability in software product lines. In *BMMDS/EMMSAD '11*, pages 453–467.