Case Tool Support for Variability Managing in Database Schemas

Nesrine Khalfallah[1*], Sami Ouali[2], Naoufel Kraiem[3]

[1] RIADI Laboratory, National School of Computer Science, Campus of Manouba, Manouba 2010, Tunisia.
[2] College of Applied Sciences, Ibri, Oman.
[3] Computer Science Department, College of Science, SQU, Oman.

*Corresponding author. Tel.: +21695740444; email: khalfallahnesrinee@gmail.com
Manuscript submitted August 27, 2018; accepted October 10, 2018.
doi: 10.17706/jsw.13.11.600-612

Abstract: In the software product lines (SPL), variability has been extensively studied to ensure the continuous evolution of the source code. This study still superficial and insufficient on the database side of the product lines. Accordingly, tool support for variability management has been gathering increasing momentum over the last few years and can be considered a key success factor for developing and maintaining database schemas. While several studies have already been conducted on variability management, none of these analyzed the available tool support in detail. Therefore, we worked on this research field to ensure a continuous evolution of database schemas through the modelling and the implementation of variability in these schemas. Thus, we proposed an approach for modelling and implementing variability in database schemas in database product lines (VariaLBD). In this paper, we present a tool support for variability managing in database schemas. It is a web site which enables the automation of the VariaLBD approach that allows the modelling of variability in the different database schemas along its life cycle. Thus, this tool allows the automatic generation of a logical, relational, evolutive, and variable schema of a given database. We also present our results of applying tool support to a case study, a web Content Management System (CMS). In addition, we present a quantitative evaluation of our experimentation which leads to a validation of the latter to show the reliability of VariaLBD.

Key words: Database product lines, database schema, software product lines, variability management.

1. Introduction

Variability is an important concept of software product line engineering. Variability can be applied to every element of a software system. Unfortunately, the previous studies focused on managing variability in source code side and his continuous evolution[1] . So the study of the variability in the database side has been neglected. However, at the socio-economic and industrial level, there would be many interests if the schemas of the databases in the software product lines (SPLs) [2] have been deeply studied such as: flexibility, genericity, better adaptation to users' needs, better interoperability, etc. Therefore, it creates a very backward database evolution [3], [4] rhythm compared to that of the code in software systems. That's why, our research focuses on proposing a new approach VariaLBD for modelling and implementing variability in database schemas in SPLs. Thus, it ensures the evolution of all parts of software systems in the same rhythm.

In fact, the variability managing in database schemas indirectly ensures the database evolution. Indeed in our proposed approach VariaLBD variability managing begins with the variability modelling until its implementation as without a reliable variability modelling, an implementation of the variability cannot take place. Therefore, it becomes crucial, first of all, to model variability in a high level of abstraction. Then, by a

refinement, we manage to model the variability at a lower level (model) which allows us to produce a conceptual and variable database schema. Thus after a normalization of this schema, we obtain a logical and variable database schema. Moreover, we success to implement a physical and variable database schema. It is important to note that throughout the database life cycle, the variability is well modelled and implemented. Even if the SPL is updated, database schema adaptation takes place from the conceptual level to the implementation level. In addition, we proceed to automate VariaLBD by proposing a complete tool that supports the variability managing in the different schemas of the database along its life cycle.

In this paper, we present our approach VariaLBD and the tool that supports the variability management in database schemas and allows the automatic generation of variable and evolutive database schemas. Indeed, this tool allows the execution of VariaLBD on case studies of the real world. Also we present as a case study the web Content Management System (CMS). In fact, we present an overview of VariaLBD in section 2. It's based on feature-oriented modelling analysis (FODA) [5], [6], [7] and relies on UML class diagrams. We explain how to use Model-driven engineering (MDE) to transform models in order to model variability in target models. This allows users to create a variable and conceptual database schema and to generate a variable and relational database schema based on the conceptual one after automatic normalization. Thus, VariaLBD has proposed a metamodel [8] that models variability at an abstract level (CIM) in order to generalize the variability concepts proposed by our approach. In section 3, we present the context of the VariaLBD experiment. In sections 4 and 5, we illustrate the functioning of our tool support. Then in section 6, we propose a quantitative evaluation of the experiment. In section 7, we present the treats of validity relative to our experiment. Moreover, we present some related work in section 8. Finally, we discuss the possible future directions of our research in section 9.

2. Overview on VariaLBD

VariaLBD [8] is an approach for modelling and implementing variability in database schemas in SPLs. VariaLBD used MDE to model variability in target models through a models' transformation. Indeed, VariaLBD work on both levels of the software product line engineering. The first level is the domain engineering where we work on the conceptual level by proposing transformation rules that allow transforming a source model to a target model which models the variability. This target model is the conceptual and variable schema of a given database. This schema uses new stereotypes proposed by VariaLBD to model the variability. These stereotypes are used to classify the classes of the diagram into three types. First, we propose *the stereotyped classes <<normal>>* which represent the classes in common between the source and target diagrams. They represent real-world entities. Secondly, we propose *the stereotyped classes <<type>>*. At logical and physical levels, these classes produce *parameterization tables* responsible for generating variability in the physical schema of the database according to the features chosen by the user. Third, we propose *the stereotyped classes <<administrator>>* which play the role of the conductor as they synchronize between the different stereotyped classes << type >>. In fact, in logical and physical levels, it generates a table that controls the different parameterization tables. Then after normalization a logical and variable schema will be generated. The second level is the application engineering where the logic model generated in the first level will be implemented and configured to ensure the user's needs.

In general, the behaviour of VariaLBD for variability management differs according to the localization of the variability within the diagram components. Indeed, VariaLBD defines and distinguishes two types of variability: *variability in attributes* and *variability in classes*. For each of them, specific transformation rules are defined to ensure the modelling of the variability in the conceptual database schema according to the variability type.

Moreover, we extend the class diagram's metamodel proposed by UML to support the new variability concepts suggested by VariaLBD. Hence, we obtain a metamodel [8] which models variability in an abstract level and which allows the creation of a variable model at a lower level.

Finally, VariaLBD ensures the evolution of database schemas in a product line on conceptual, logical, and physical levels. This enables to create synchronization between the various components of the software system in an SPL. In fact, VariaLBD treats variability in database product lines from zero. So, VariaLBD always starts by creating the conceptual schema of a database product line as it manages the variability at all levels of abstractions based on this conceptual schema. Therefore, every change in the database context must be necessarily presented in advance at the level of the conceptual schema of the database product line to be later considered in the logical and physical schemas. This ensures the continuous evolution of the database schemas because it can adapt to the changes of the requirements of database context.

3. Context

In this section, we briefly describe the case study used for experimenting VariaLBD. This case study focuses on modelling variability in a Content Management System (CMS). It is a web platform for building and updating websites using a web interface. The main objectives of the CMS are to allow a simple user to create and administer his website through simple web interfaces. So, CMS allows a user to update his NEWS, his pages and/or his product catalogue, modify his template, configure his site and consult statistics, and the help.

In this experiment, the input models for VariaLBD are the feature model FODA and the class diagram modelling the CMS database (Fig. 1 and Fig. 2).

4. Tool Support

We have developed a website for executing VariaLBD. This site provides an open online space for users to create their own variable and relational schema which is adapted to their requirements for any database in a product line. Indeed, our tool supports online modelling of FODA models and class diagrams. Even if the user has forgotten to enter the feature diagram FODA, our tool drives it and generate it automatically from previously entered class diagram. Next, our tool transforms the entered class diagram into a target one which models the variability in a conceptual level thanks to the implementation of transformation rules. Besides, it normalizes the transformed and variable class diagram to generate automatically its logical, variable, and relational database schema. Thus, it treats the variability within classes and the variability within attributes. In fact, our tool is simple and practical to use. And it has clear and readable interfaces. Finally, we used HTML 5, PHP 5, Bootstrap 3, Wamp Server, and SQL as technologies for the implementation of our support tool.

Briefly, our tool offers to the user the automation of several functionality of VariaLBD such as:

- The automatic generation of a relational and variable database schema.
- The automatic transformation of a source class diagram to another target and variable one by respecting a set of proposed and implemented transformation rules.
- The automatic switch from the conceptual level to a logical level ensured by the automation of normalization.
- The automatic modelling of user requirements at the conceptual level via the class diagram and the feature model and at the logical level via the relational schema.

Finally, it should be mentioned that our tool support offers two-level of automation. The first is at the level of domain engineering through a switch between different data models and between different levels of abstraction. The second is at the level of application engineering through the switch from a variable data

model to a particular configuration of its relational schema obtained after automatic normalization, using multiple implementation platforms.

Fig. 3 presents the home interface of our tool support. It offers many actions for each class diagram entered by the user such as: the display of the class diagram entered by the user, the transformation of the class diagram entered by the user, the creation or the automatic generation of the FODA model, the modification and suppression of the class diagram entered by the user.

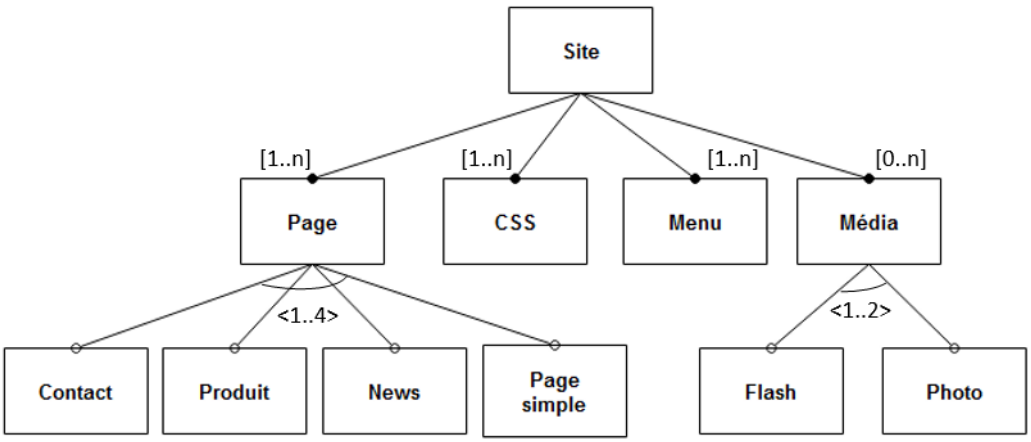

Fig. 1. FODA model related to CMS.

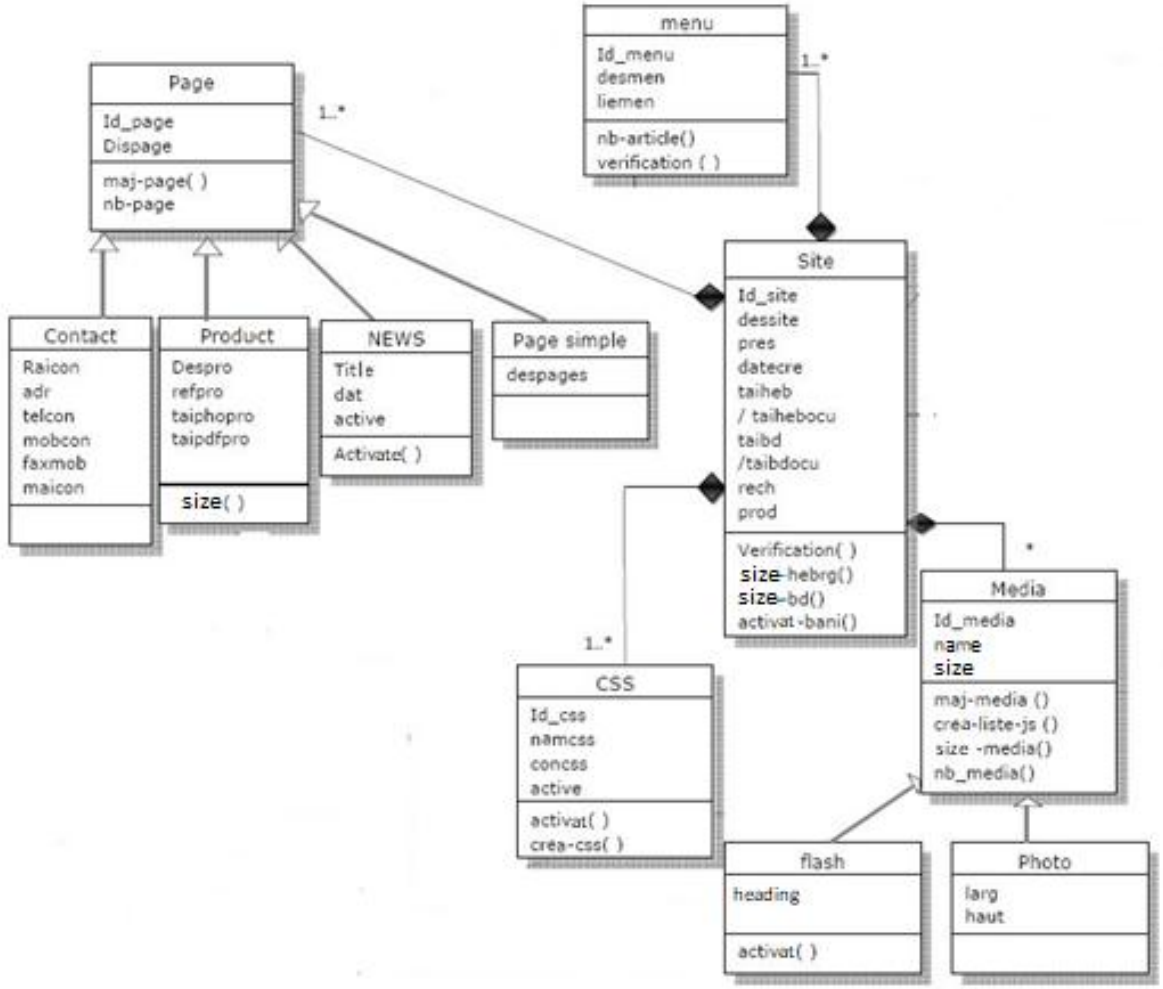

Fig. 2. CMS's class diagram.

5. Experimentation of VariaLBD

First of all, this experimentation starts with the input of the input model (the class diagram and the variability model). Then, an automatic transformation is carried out on the input class diagram thanks to the implementation of the transformation rules defined during the construction of VariaLBD. This transformation respects the existing constraints in the class diagram and in the variability model FODA. As a result of the execution of VariaLBD using this tool support, we obtain a variable class diagram automatically which represents a conceptual and variable database schema (Fig. 4). In fact, our tool support proposes a specific formalism to VariaLBD presented in Fig. 4 for the display of the class diagram: the green icons represent the names of the classes and the orange icons represent their attributes. The attributes are organized under the name of the class one below the other. Thus, changing the colour of the icon to green declares a new class. In addition, the associations are presented in a table where there is all the necessary information about each association such as: the association name, the cardinalities, the association type (nature), and the participating class names. At the same time, our tool support generates a relational and variable model of the database that represents her logical and variable schema. Thus, an automatic normalization of the conceptual and variable schema generated at the beginning enables to have a logical and variable database schema (Fig. 5) which in turn allows generating a physical and variable schema after the implementation of the database. Indeed, in Fig. 4 and Fig.5 the red box is the part added after transformation compared to the source class diagram entered by the user. In fact, this part is responsible for the modelling and the implementation of the variability in the database schemas whatever at the conceptual or logical level. Hence, Fig. 6 shows the CMS's FODA model entered by the user or automatically generated by our support tool from the class diagram entered by the user at the beginning. Thus, models in Fig. 4, Fig. 5 and Fig. 6 represent the experimental results of our experimentation (VariaLBD's outputs).

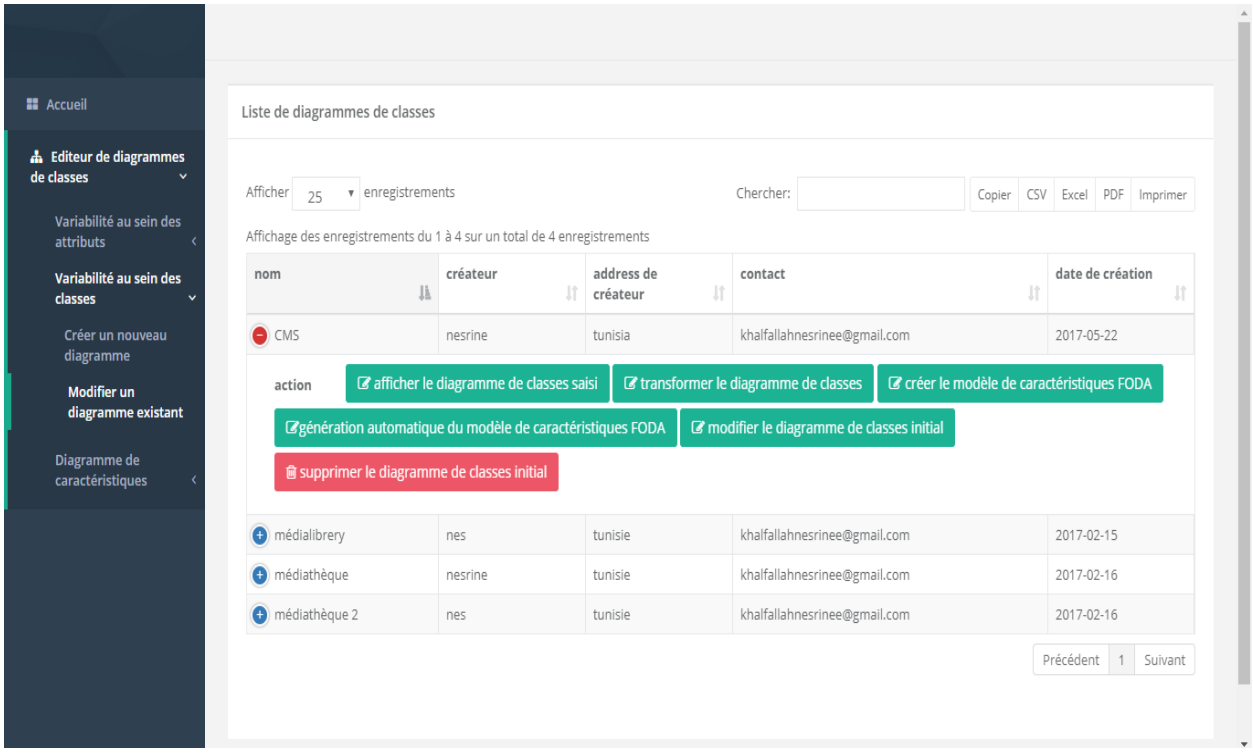

Fig. 3. Tool support's home interface.

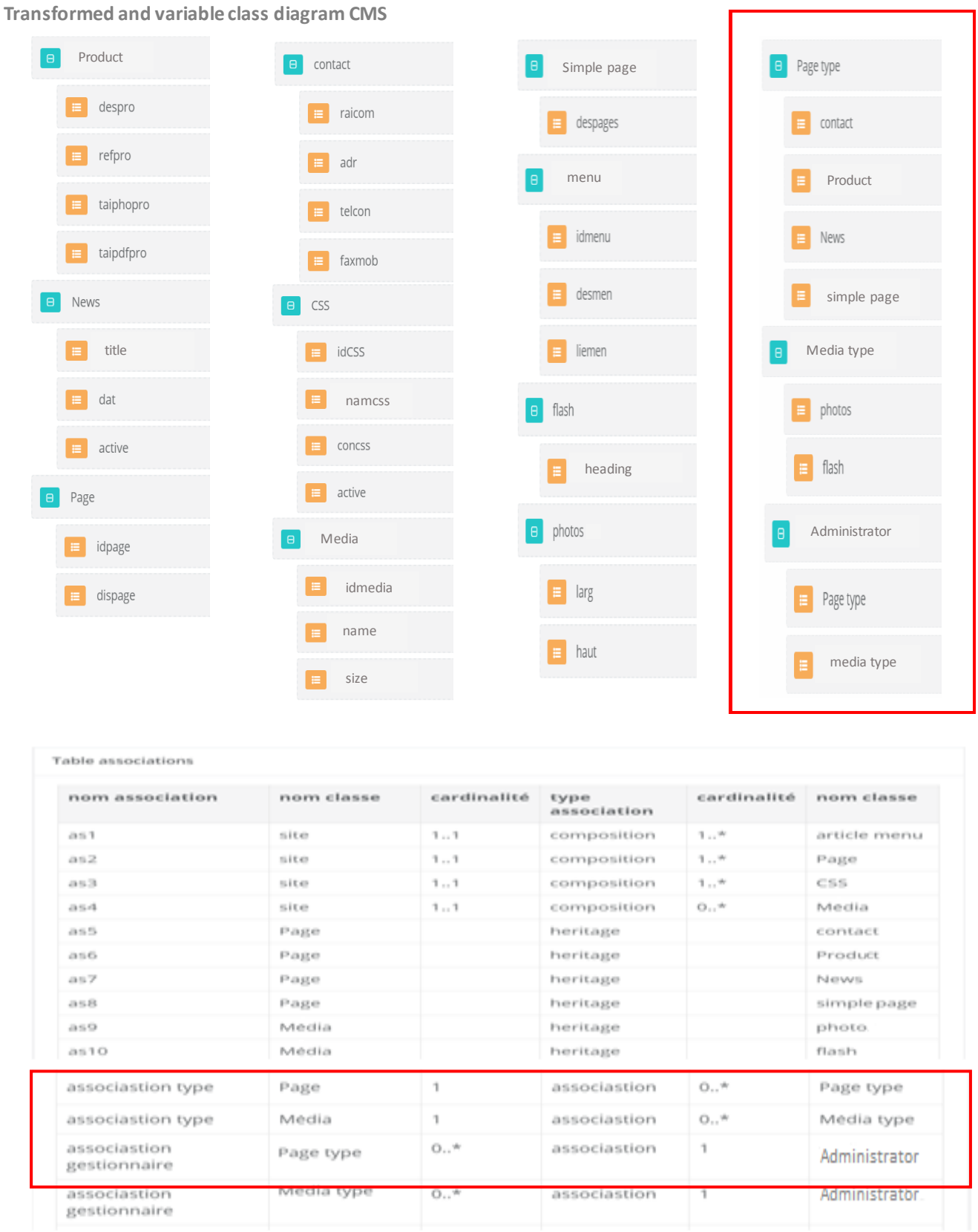

Fig. 4. Transformed and variable class diagram.

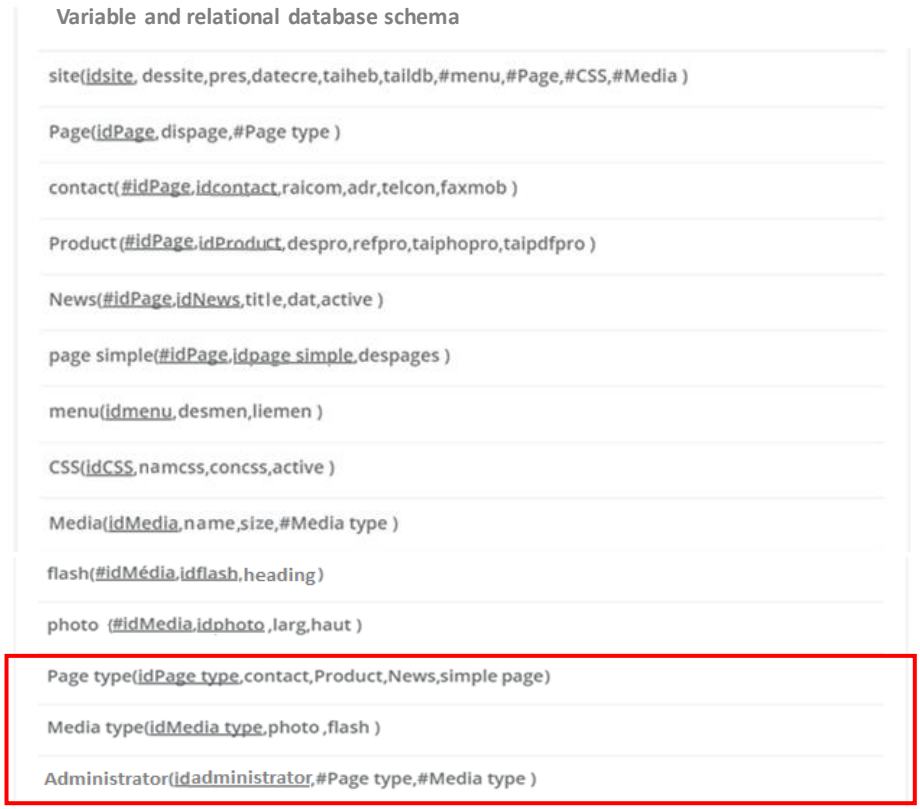

Fig. 5. Variable, logic, and relational database schema.

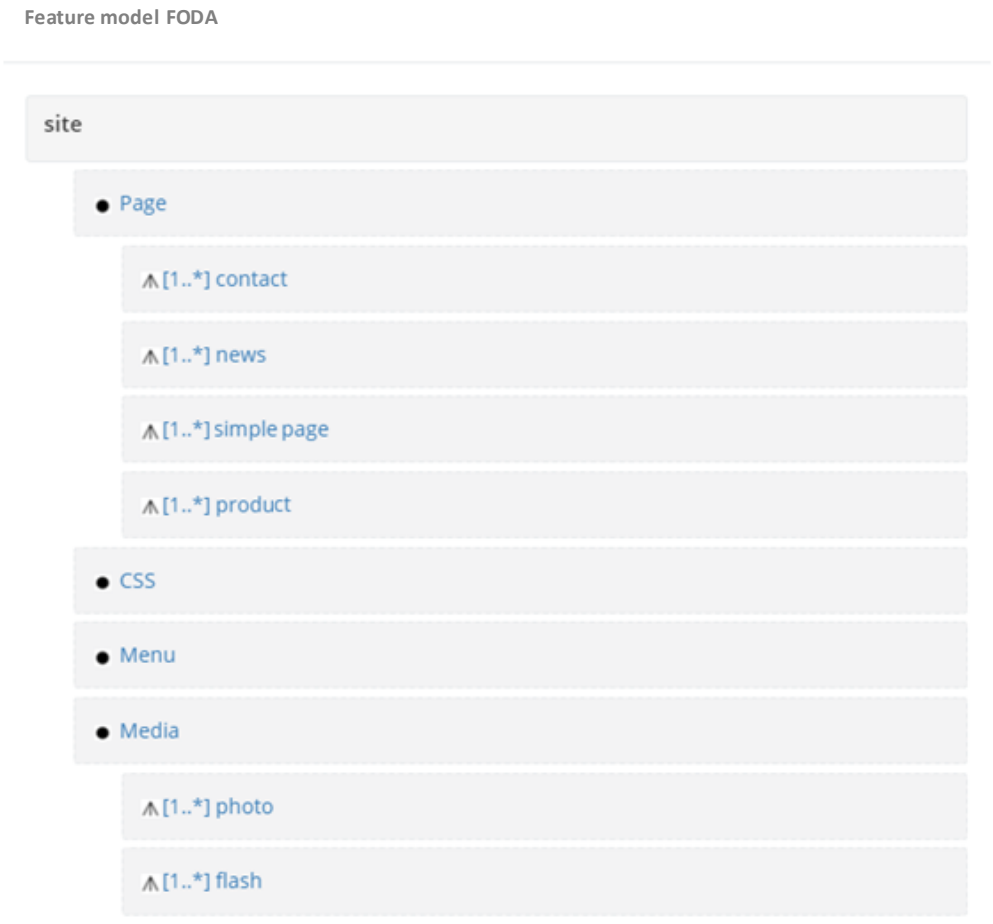

Fig. 6. Foda model generated by our tool.

6. Evaluation of VariaLBD

After the experimentation of VariaLBD, we realize in this section a quantitative evaluation based on numerical metrics of the generated database schema. We choose the most relevant metrics when running the database schema by DBMS which are: CPU execution time, input/output cost, operator cost, and used size, in order to enhance the contributions of VariaLBD. These metrics are chosen because the execution of a request means the execution of its various operators arranged in an execution plan. To evaluate VariaLBD, we must first implement the database schema generated by it and by each existing approach. Then, the necessary metrics are extracted to compare them. The database schema generated by the view approach and the variable schema approach are chosen to be implemented as they are the closest approaches to VariaLBD. Thus, Microsoft SQL server management studio 2008 is used as a database management system (DBMS) on a local workstation. This DBMS returns these metrics each time a query is executed in an execution plan. An overview of the execution plan generated by SQL Server when executing a query is presented in Fig. 7.

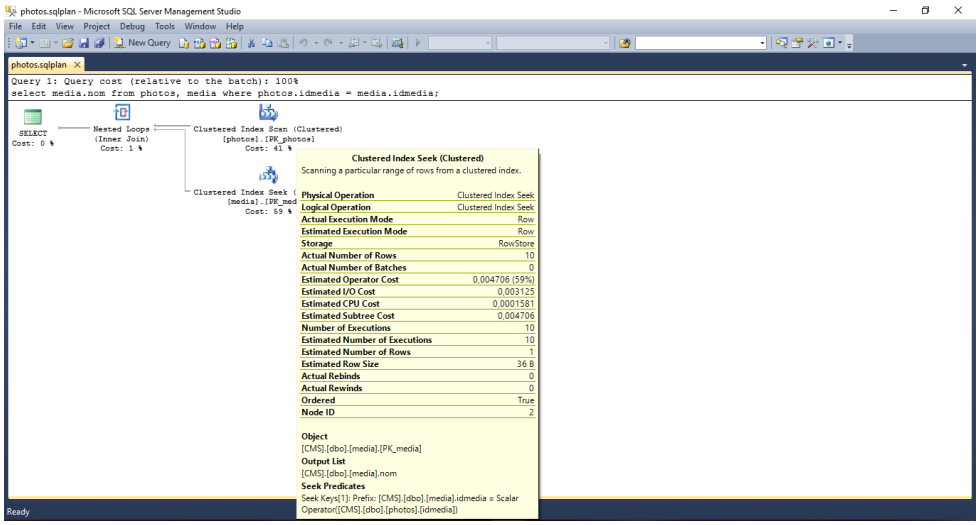

Fig. 7. Execution plan generated by SQL Server.

Indeed, the metrics related to the execution of each feature selection request are extracted for each schema. At the level of this evaluation, we work on the selection of the following optional features: photo, product, and contact for each different schema. Tables 1, 2, and 3 present the values of the metrics for each execution of the same query for each different schema.

Table 1. The Metrics for the Photo Feature Selection Query

Metrics / Approaches	CPU execution time	Input/output cost	Operator cost	Used size
VariaLBD	0.0001581	0.003125	0.003293	16B
View approach	0.000168	0.003125	0.004706	36B
Variable schema approach	0.000168	0.003125	0.004706	36B

Table 2. The Metrics for the Product Feature Selection Query

Metrics / Approaches	CPU execution time	Input/output cost	Operator cost	Used size
VariaLBD	0.0001581	0.003125	0.003293	16B
View approach	0.000168	0.003125	0.0061289	27B
Variable schema approach	0.000168	0.003125	0.0061289	27B

Table 3. The Metrics for the Contact Feature Selection Query

Approaches \ Metrics	CPU execution time	Input/output cost	Operator cost	Used size
VariaLBD	0.0001581	0.003125	0.0032985	9B
View approach	0.0001735	0.003125	0.0054965	27B
Variable schema approach	0.0001735	0.003125	0.0054965	27B

We use Values in Tables 1, 2, and 3 for the tracing of Fig. 8, Fig. 9, Fig. 10, Fig. 11, Fig. 12, and Fig. 13.

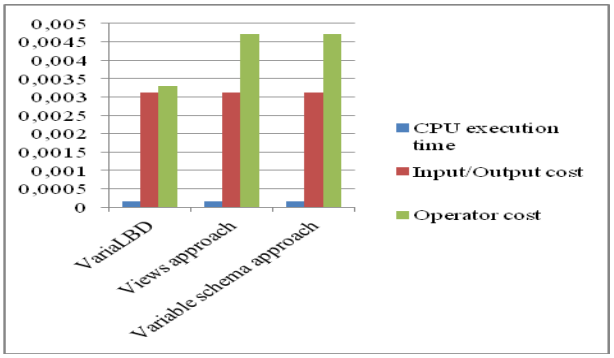

Fig. 8. Illustrative graph of metrics: CPU execution time, input/output cost, and operator cost related to the select request of the photo feature.

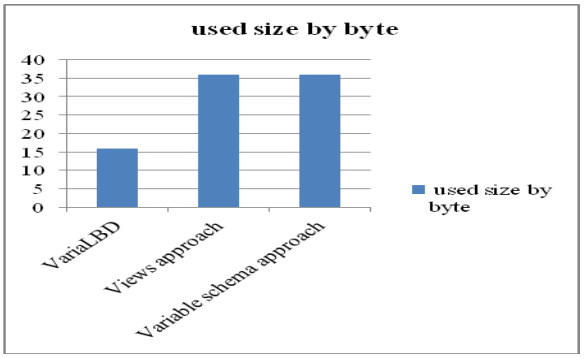

Fig. 9. Illustrative graph of the used size metric related to the select request of the photo feature.

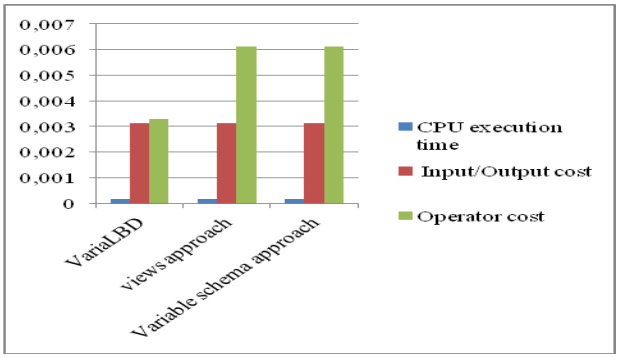

Fig. 10. Illustrative graph of metrics: CPU execution time, input/output cost, and operator cost related to the select request of the product feature.

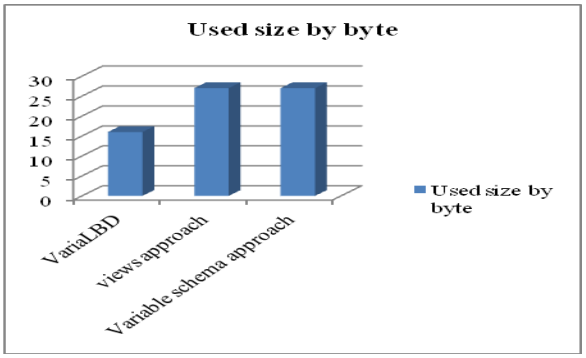

Fig. 11. Illustrative graph of the used size metric related to the select request of the product feature.

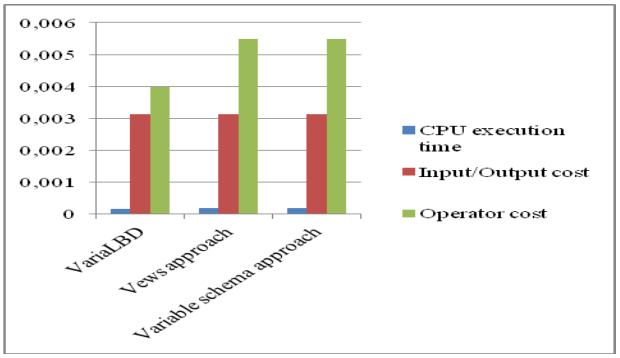

Fig. 12. Illustrative graph of metrics: CPU execution time, input/output cost, and operator cost related to the select request of the contact feature.

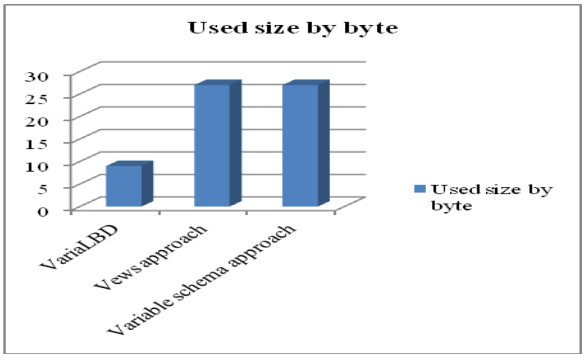

Fig. 13. Illustrative graph of the used size metric related to the select request of the contact feature.

Fig. 8, Fig. 10, and Fig. 12 present the estimated values of the metrics CPU execution time, input/output cost, and operator cost related to the selection of the optional features photo, product, and contact. It is noted that the estimated input/output cost is the same for the different approaches, even if the selected feature is changed. Then, the values of CPU execution time and operator cost generated by VariaLBD are better than those generated by the two other approaches, even if the selected feature is changed. So, VariaLBD is more efficient compared to other approaches because it allows saving time during the execution of queries.

Fig. 9, Fig. 11, and Fig. 13 present the estimated values of the metric used size related to the selection of the optional features photo, product, and contact. So, it is clear that the same size is used for the view approach and the variable schema approach, even if the selected feature is changed. However, VariaLBD uses a fewer size than the other approaches, even if the selected feature is changed. Thus, VariaLBD saves not only time but also space. It is important to note that the number of tuples is the same in the different tables in common between the different relational schemas generated by the various approaches.

7. Threats to Validity

This section presents a validity analysis [9] of the experiment of VariaLBD. Indeed, there are different schemas of classification for different types of threats to validity of an experiment [10], [11]. In this work, the validity analysis proposed by Cook and Campbell [11] is adopted with its four types of threats to show that the results obtained from the experiment have a satisfactory validity for the chosen population of interest. Thus, to draw conclusions, four stages must be realized where in each of them there is a type of threats to validity of the results. The threats to validity are: conclusion, internal, conceptual, and external.

7.1. Conclusion Validity

This validity concerns the relationship between the treatment and the result. In addition, it ensures a statistical relationship between them. In the case of VariaLBD, the evaluation is mainly based on the following measures: the CPU execution time, the input/output cost, the operator cost, and the used size. These are objective measures, which can be repeated with the same result. They are more reliable than subjective measures. But it is more accurate to calculate these measures. Also taking into account the following factors: the size of the collected corpus of execution (as the evaluation is carried out on the same corpus for all the approaches), the size of the studied database product line, the number and the nature of constraints, and the number of clients using the system to analyse the effect on performance.

7.2. Internal Validity

This validity concerns the relationship between the treatment and the result. Indeed, it ensures a causal relationship between them. It resumes all factors which can cause the experiment to show behaviour that is not due to treatment, but to the disruptive factor. In this experiment, not all data stored in the DB is collected from actual websites (CMS). This does not negatively affect the results of VariaLBD, but does not reflect the results of real data.

7.3. Conceptual Validity

This validity concerns the relation between the theory and the observation. If the relationship between cause and effect is causal, two things must be assured. First, the treatment reflects well the construction of the cause design. Second, the result reflects the effect design. Indeed, conceptual validity concerns the generalization of the result of the experience to the theory behind the experience. VariaLBD is applied only to simple academic case studies. So this research needs to be pushed to be applied to real-world scenarios in order to deal with more complex database product lines and to improve the generalizability of VariaLBD.

7.4. External Validity

This validity concerns the possibility of generalizing the experimental results outside the experimental setting. The quantitative evaluation of VariaLBD was based mainly on the following measures: CPU execution time, input/output cost, operator cost, and the used size. These metrics are considered to compare VariaLBD with other existing approaches. If the experimental context changes then other measures could be added or eliminated. For example, are these evaluation measures always remain sufficient for validation, if the population concerned by this experiment is modified or if the experimental tools are modified, especially in this experimentation these tools represent an individual choice?

8. Related Work

In the literature, many tools support are proposed to managing variability in the database schema. Indeed, there are tools that are interested in the adaptation of the conceptual database schema and others that are interested in the adaptation of the logical and physical database schema. Hence, the lack of a complete tool that manages the adaptation of the database schema along its life cycle. At the conceptual level, Siegmund *et al.* [12] propose an Eclipse plugin to provide a virtual annotation of the database elements belonging to each characteristic. But this tool allows only a virtual decomposition into characteristics of the conceptual database schema and not a physical decomposition. Recently, Humblet *et al*. present in [13] SVL tool. It is a plugin for Case Tool DB-Main that allows the modelling of the characteristic models and the mapping of these models to the elements of the database schema. As a result, this tool produces a new database schema that includes only selected features. But this tool lacks heuristics and detection algorithms for the features of variability extraction. In addition, Khedri *et al.* [14] propose an incremental method for variability modelling based on delta-oriented programming. Indeed, the conceptual database schema is built from a SQL script of data definition. In fact, it is not an automatic tool. It is just a script to describe the conceptual model created manually. At the logical and the physical level, for example, Rosenmüller *et al.* in [15] develop tailored DBMS using SQL dialect families. Then, they propose in [16] the customization of the DBMS for embedded systems. In addition, there are the plugins [17] which are extensions added to the database schema implementation. Plugins add additional features and constraints to the database schema that cannot be implemented in the main schema to manage variability. Thus, this plugin replies the requirement of completeness. Unfortunately, the use of plugins can lead to a problem of partitioning of tables. As a result, this has a negative impact on the complexity of the schema variants and limits the modelling expressivity. Hence, the effort provided for database modelling increases because new challenges have emerged such as the conflicts of naming. Moreover, Khedri *et al.* [18] propose a technique based on delta-oriented programming to manage variability when implementing the schema of the database. In fact, it is not a real and automatic tool but it is just a technique to implement variability in the script of the physical database schema. Thus, Hermann *et al.* [19] present a tool support DAVE to manage the database evolution in the product lines. Otherwise, DAVE treats the weaving problem only at the logical level. So, it cannot address the physical optimization of a global evolution scenario. Simply, DAVE is a tool support for managing manually the evolution and the migration of a database. It is far from being an automated and controlled process for the database evolution in SPLs.

9. Conclusion and Future Work

In this paper, we introduce VariaLBD and the proposed tool support. Then, we show the reliability of this tool by applying a CMS case study. Indeed, this tool allows the execution of VariaLBD and the automatic generation of a relational and variable database model. Then, we present a quantitative evaluation of VariaLBD compared to some existing approaches. This evaluation is based on numerical metrics derived from the execution of the database schemas of different approaches such as: CPU execution time,

input/output cost, operator cost, and used size. In addition, the VariaLBD experiment is validated using the validity analysis proposed by Cook and Campbelle [11] which is based on four types of threats to validity in order to prove that the results obtained by VariaLBD are reliable. Finally, this paper is closed by an overview of some existing tools in order to highlight the originality of our tool support as it is a complete tool which treats the variability in all the databases schemas along its life cycle. In fact, this experiment is not the only one that is done on VariaLBD. So, the approach evaluation was carried out on several academic case studies and the obtained results are always in favor of VariaLBD. Indeed, our tool support allows, first of all, the modelling of the variability at a conceptual level by an automatic transformation of the source class diagram to a target class diagram which models the variability and which represent the conceptual database schema. Besides, it generates a logical and variable database schema after an automatic normalization. Then, the implementation of the logical schema using a DBMS produces a variable and evolving physical schema. Therefore, every change in the conceptual level of the database leads to the regeneration of a new variable and conceptual schema, as well as new variable, logical and physical schemas after automatic normalization. One possible improvement of our tool support, once it will be an automatic update of the conceptual schema from the existing as it still manual, it's up to the designer to do it. The next step of this work would be to adapt VariaLBD to develop a tool working directly into the Java code of applications. This brings a whole integrated tool support for managing variability in database applications.

References

[1] De, S. S., & G. J. (2017). Assessing and improving code transformations to support software evolution. PhD dissertation, University of Science and Technology, Lille 1, France.

[2] Ben, R. T. (2014). Composition des modèles de lignes de produits logiciels. PhD dissertation, University of South Paris, France.

[3] Vassiliadis, P., Zarras, A. V., & Skoulis, I. (2015). How is life for a table in an evolving relational schema? birth, death and everything in between. *Proceedings in the 34 th Int. Conf. on Conceptual Modeling (ER)* (pp. 453–466). Stockholm, Sweden.

[4] Skoulis, I., Vassiliadis, P., & Zarras, A. (2014). Open-source databases: Within, outside, or beyond Lehman's laws of software evolution?. *Proceedings of the Int. Conf. on Advanced Information Systems Engineering (CAiSE).* (pp. 379–393). Thessaloniki, Greece.

[5] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (Technical report, CMU/SEI TR-21). USA.

[6] Pohl, K., Böckle, G., & Linden, F. V. D. (2005). *Software Product Line Engineering. Foundations, Principles, and Techniques*. Springer, Verlag Berlin Heidelberg, Germany.

[7] Mazo, R. (2014). Avantages et limites des modèles de caractéristiques dans la modélisation des exigences de variabilité. *Journal Génie logiciel,* 111, 42-84, Paris, France.

[8] Khalfallah, N., Ouali, S., & Kraiem, N. (2016). Managing variability in database context using an MDE approach. *Proceeding of the 4th Int. Conf. on Control Engineering & Information Technology (CEIT)* (pp. 1-6). Tunisia: Hammamet.

[9] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering.* London: Springer Heidelberg New York Dordrecht.

[10] Campbell, D. T., & Stanley, J. C. (1963). Experimental and Quasi-experimental Designs for Research. (pp. 1-88). USA: Houghton Mifflin Company Boston.

[11] Cook, T. D., & Campbell, D. T. (1979). *Quasi-experimentation: Design and Analysis Issues for Field Settings.* Houghton Mifflin Company Boston.

[12] Siegmund, N., Kästner, C., Rosenmüller, M., Heidenreich, F., Apel, S., & Saake, G. (2009). Bridging the gap between variability in client application and database schema. *Proceedings of the 13 GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)* (pp. 297–306). Germany.

[13] Humblet, M., Weber, J. H., & Cleve, A. (2016). Variability management in database applications. *Proceedings of the 1st International Workshop on Variability and Complexity in Software Design* (pp. 21-27). USA: Austin, Texas.

[14] Khedri, N., & Khosravi, R. (2015). Incremental variability management in conceptual data models of software product lines. *Proceedings of the.22nd Asia-Pacific Software Engineering Conference (APSEC)* (pp. 222-229). India.

[15] Rosenmüller, M., Kästner, C., Siegmund, N., Sunkle, S., Apel, S., Leich, T., & Saake, G. (2009). SQL à la Carte-Toward Tailor-made Data Management. *Proceedings of the 13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)* (pp. 117-136). Germany.

[16] Rosenmüller, M., Apel, S., Leich, T., & Saake, G. (2009). Tailor made data management for embedded systems: A case study on berkeley db. *Journal Data & Knowledge Engineering (DKE),* 68, 1493–1512.

[17] Johnson, R., & Foote, B. (1988). Designing Reusable Classes. *Journal of Object-Oriented Programming,* 1(2), 22-35.

[18] Khedri, N., & Khsoravi, R. (2013). Handling database schema variability in software product lines. *Proceedings of the 20th Asia-Pacific Software Engineering Conference (APSEC): Vol. 1.* (pp. 331-338). Thailand: Bangkok.

[19] Herrmann, K., Reimann, J., Voigt, H., Demuth, B., Fromm, S., Stelzmann, R., & Lehner, W. (2015). Database evolution for software product lines. *Proceedings of the 4th International Conference on Data Management Technologies and Applications (DATA). Vol. 1.* (pp. 125-133). France: Colmar, Alsace.

Nesrine Khalfallah, is a PhD student at RIADI Lab, ENSI, Campus of Manouba, Manouba, Tunisia. She obtained her maitrise degree in Computer Science applied to management in 2010 and his research master's in software engineering in 2012 and subscribed in the first year of thesis in 2014. She is working on this subject: The modelling and implementation of the variability in the database. She works in the public sector as a teacher and researcher.

Sami Ouali is an assistant professor in the College of Applied Sciences of Ibri in Oman. He is a member of the RIADI labs. His research interests lie in the areas of software engineering and software product line.

Naoufel Kraïem is an associate professor in the Department of Computer Science in Sultan Qaboos University. He is a member of the RIADI labs. His research interests include IT adoption and usage Information modelling, software engineering, software product lines, method engineering, web services and CASE tools.