Using Moose platform for the implementation of a Software Product Line according to model-based Delta-Oriented Programming*

Boubou T. Niang[1,2], Giacomo Kahn[1], Nawel Amokrane[2], Yacine Ouzrout[1], Mustapha Derras[2] and Jannik Laval[1]

[1]*Univ Lyon, Univ Lumière Lyon 2, INSA Lyon, Université Claude Bernard Lyon 1, DISP, EA4570, 69676 Bron, France*
[2]*Berger-Levrault, 1 Pl. Giovanni da Verrazzano, 69009 Lyon, France*

Abstract
Software product line engineering allows common features to be reused to implement a set of products. Implementing a product line can be done using several paradigms, including delta-oriented programming. However, due to a lack of available tools, most works are limited to theory without concrete implementation on real case studies. This paper proposes a framework that lays the foundations of a tool for implementing software product lines according to the delta-oriented programming paradigm. The framework is developed using the Moose platform and will support model-based engineering to manage better the variability of product lines at a high level of abstraction. An industrial use case is proposed to illustrate the interest of the tool whose prototype is presented.

Keywords
Delta-Oriented Programming, Model-Based Engineering, Software Product Lines.

1. Introduction

A software product line (SPL) [1] is a set of products with common characteristics to which variability is applied to create software that meets a specific need. SPL-based architectures have the advantage of being designed to increase the reusability of functionalities across multiple products. The literature presents several paradigms for implementing a software product line, classified into compositional and annotative approaches [2]. Although research focuses mainly on compositional approaches such as feature-oriented programming (FOP) [3], aspect-oriented programming (AOP) [3] or delta-oriented programming (DOP) [4], as they allow for feature traceability and modularity, in practice, annotative approaches such as pre-processors are more common because they are more accessible to adopt [5]. Moreover, Model-Driven Engineering (MDE) [6] is a discipline that considers models as first-class entities to facilitate the development and analysis of complex software systems. As such, the adoption of MDE can provide more abstraction and the ability to automatically generate source code for software products and thus reduce the effort required to develop, maintain and evolve them. Thus, even if many research

IWST'22: International Workshop on Smalltalk Technologies
boubou-thiam.niang@univ-lyon2.fr (B. T. Niang); giacomo.kahn@univ-lyon2.fr (G. Kahn); nawel.amokrane@berger-levrault.com (N. Amokrane); yacine.ouzrout@univ-lyon2.fr (Y. Ouzrout); mustapha.derras@berger-levrault.com (M. Derras); jannik.laval@univ-lyon2.fr (J. Laval)

works on software product lines using compositional approaches and tools have proposed their implementation according to certain paradigms, few of them propose an implementation according to the DOP paradigm, especially at the model level.

This article aims to introduce the prototype of a tool under development that supports the implementation of software product lines following the DOP paradigm. The choice of DOP is motivated by its ability to introduce variability into an existing software product [7], which is very interesting for ensuring the scalability of software and systems. With DOP, a product line is represented by a core module and a set of delta modules. The core module provides an implementation of a valid product. The delta modules contain operations called delta actions to specify the changes applied to the core module to create a derived product by adding, modifying, or removing features. However, these changes can be seen at different levels of abstraction. At a low level of abstraction, delta modules can represent modifications that can be applied to the source code of a core product. In this case, creating delta modules can be laborious and time-consuming. In this paper, we focus on model-based DOP for more abstraction and, therefore, more flexibility, scalability, and better management of change operations. To do so, we rely on the metamodeling possibilities of Moose [8], an open-source platform for software and data analysis.

The paper is organized as follows: In section 2 we discuss related work. The prototype of the tool is described in section 3. An illustrative case study is provided in section 4. The section 5 outlines the future works. Finally, we present perspectives and a conclusion.

2. Related Works

Delta-oriented programming [4] is a recent paradigm for the implementation of software product lines. Following this paradigm, an SPL is implemented as a core module with a set of delta modules. The core module contains a complete product implementation for some valid configuration, which conventional single-application engineering techniques can develop. The delta modules specify changes to be applied to the core model to implement other products. However, there are not many examples of concrete implementations of product lines following the DOP paradigm in the literature. This section presents the main existing tools and frameworks.

DeltaJ [9] is a prototype-oriented programming language that supports the basic operations of DOP. DeltaJ allows adding, modifying, and deleting methods and classes' fields with different operations. However, DeltaJ requires manual development such as core modules, delta modules, and decorators. Thus, although DeltaJ can cover all stages of product line implementation, leveraging MDE engineering will improve the expected time savings through reuse to motivate the choice of the SPL approach. Moreover, the implementation of DeltaJ is done at the code level, which may require making some choices beforehand. For example, applications derived with DeltaJ generate java code. This can be seen as a limitation of the need to create derived applications in another language.

DeltaEcore [10] is a suite of tools for the rapid creation of delta languages that can be seamlessly integrated into the variant derivation process of a software product line. Indeed, delta modules define changes associated with different configurations in realization artifacts, such as source code, by adding, modifying, or removing relevant elements. This is where a dedicated delta language is required for each realization language, for example, DeltaJava for Java [1].

SiPL [11] is a model-based delta-oriented framework built on the Eclipse Modeling technology stack. SiPL offers several functions such as the manual specification of delta modules using a domain-specific language or automatic derivation of delta modules from model differences, automatic generation of products from a given configuration, and analysis of a set of delta modules. However, we can identify some additional features that our tool should consider. Indeed, SiPL allows to automatically calculate delta modules based on the modifications applied to a core model, i.e., the difference between an original model and a modified version of the model. These tasks are performed manually by an operator who prepares some delta operations by making the possible modifications. It can therefore be difficult to predict all possible delta modules, limiting the number of reusable operations to create a derivative, especially for large-scale models. Moreover, even if the configuration is done manually, it could be interesting to integrate a decision aid to choose the best configuration or choose between the best possible configurations according to the specification.

3. The Model-Based DOP Framework

The proposed framework is inspired by existing solutions such as DeltaJ, and SiPL introduced in Section 2 that have already made a considerable effort to implement a software product line according to the DOP paradigm, to which we want to add additional functionalities to simplify or automate steps in the software product derivation process. For this, we chose to develop the tool with the Pharo language. Pharo is an object-oriented programming language influenced by Smalltalk. The choice of Pharo is motivated by the fact that it comes with an extensible and flexible programming environment. Thus, it will be easier to create the different bricks of the framework, such as parsers and importers, according to the need. On the other hand, as our objective is to implement the SPL following the DOP programming at the model level, we use the Moose platform [8]. The Moose platform allows us to manipulate metamodels flexibly and to perform operations such as adding and deleting entities and calculating differences between versions of the metamodels, which is necessary for the DOP paradigm.

3.1. The Perimeter of the Model-Based DOP Framework

Software product line engineering consists of two sub-processes: Domain Engineering (DE) and Application Engineering (AE) [12]. The DE sub-process allows the implementation of software product lines. It concerns feature detection, variability modeling, and the implementation of reusable artifacts. The AE sub-process specifies an expected product, chooses the corresponding configuration by selecting the required features, and derives the desired outcome by reusing artifacts implemented in the DE sub-process. Each sub-processes is split into two parts, the problem space, and the solution space. The problem space deals with variability analysis and software variability modeling for the DE sub-process, while the AE sub-process deals with the specification of the target software product and corresponding configuration. As far as the

[1]http://deltaecore.org/

solution space is concerned, it allows the implementation of the product line itself for the DE sub-process and the use of the product line to obtain the target product through derivation. Our proposal focuses on the product line's implementation and operation. It does not support the definition of the problem space except for the configuration in the AE. However, the tool uses the problem space, which represents the entry point of our tool. Figure 1 The figure shows the software product line engineering process, highlighting the steps our tool should cover.

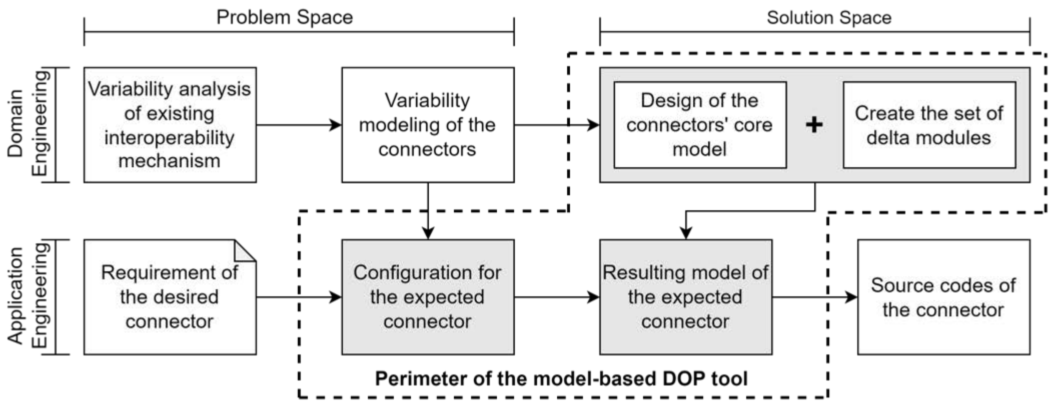
Problem Space
Solution Space
Domain Engineering
Application Engineering
Variability analysis of existing interoperability mechanism
Variability modeling of the connectors
Design of the connectors' core model
+
Create the set of delta modules
Requirement of the desired connector
Configuration for the expected connector
Resulting model of the expected connector
Source codes of the connector
Perimeter of the model-based DOP tool

Figure 1: Perimeters of the model-based DOP tool for the software product line implementation - here with interoperability connectors as a SPL use case .

3.2. Description of the Framework Process

The proposed framework requires external input such as a feature model (FM) [13], which is a model representing all possible features to create a product, and a software specification that computes the expected software model. The specification is currently not formalized, and human intervention is required to understand it and select the configuration that matches the specified software. The tool allows performing several intermediate operations to provide the expected software product. The tool includes an importer and a parser, which reads the feature model in XML format. The parser produces an Expression Product line (EPL) corresponding to a logical equation establishing the relationships and constraints between the features. A validator is applied to the EPL. The configurator selects the features required to create a specific software product based on the validated constraints and the product specification. The base model of the product line, a metamodel, must also be created manually based on the feature model. Thus, several versions of the metamodel are created manually from the original metamodel representing the product line or from another version of the metamodel. A difference calculation engine allows the preparation of ready-to-use delta actions from the different models. The features chosen in the configurator determine the delta actions. Then we can apply deltas to the metamodel referenced in the derivation to obtain a product model corresponding to the specification automatically.

Figure 2 summarizes the different steps of the framework prototype.

- Importer and Parser This step consists in importing the feature model into the tool. To

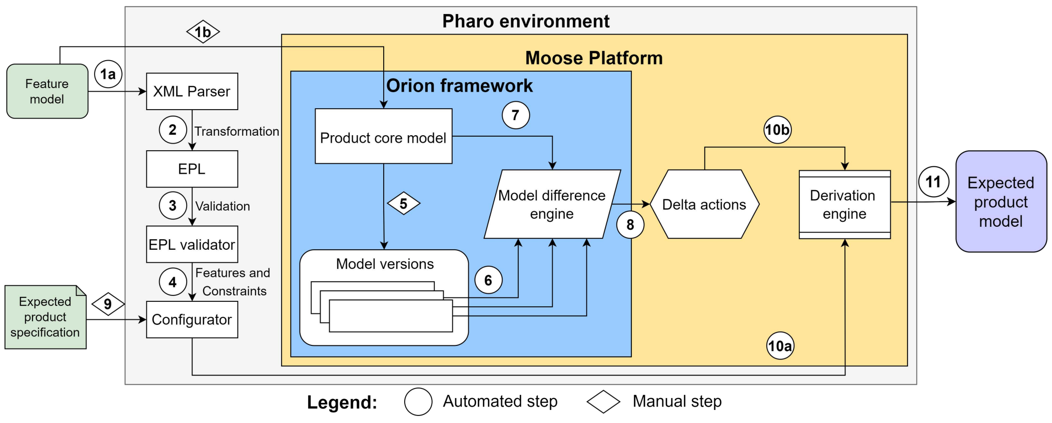

Figure 2: Overview of the prototype tool. It shows the steps required to obtain an expected product model from a given feature model and a product specification. The prototype uses Moose platform.

create a software product, we must first parse the feature model that represents all possible product configurations. We consider the XML representation of the feature model as available in featureIDE [2] tool. The parser parses the input XML file into an EPL. The EPL is a textual language and grammar representing the feature model, features, constraints, and relations, using logic functions such as and, or, not, implies operations. Once the feature model is imported, validation is required to ensure that the imported feature model respects the grammar defined for the EPL.

- Metamodel and delta modules creation To implement the SPL using model-based DOP paradigms, we need first to create the metamodel that represents a valid product of the product line. Our tool allows for creating a metamodel, which is initially possible in Moose. However, using Moose is not enough to implement our product line since it is also necessary to create delta modules. To this end, the present tool relies on Orion [14], an interactive prototyping tool for reengineering, which allows to simulate changes on different versions of software source code models and compare their impact. The philosophy of Orion is that each modification triggers an Orion action which is in charge of adding the modification to the data model. The use of Orion is motivated by creating different versions of a metamodel and to keep the link between the original metamodel and the new metamodels. Maintaining these links between the different versions of the metamodel allows Orion to define the modifications made to the original metamodel. This allows us to have the necessary information about the modifications and thus information to create our delta modules. The delta actions are possible operations to bring to the core model for the derivation, such as removing an entity from the model name, adding an entity from the model name, removing an attribute from the entity name, and adding an attribute from to name.
- Product configuration and generation using human-machine interaction To exploit the software product line implemented in the DE sub-process to derive a product in the

[2]https://featureide.github.io/

AE sub-process, we need some interfaces, *e.g.* for configuration or product generation purposes. For this, we will set up a graphical interface to interact with the tool. To do so, we opted for Spec2 [3] a framework in Pharo for describing user interfaces.

4. Use Case

In order to demonstrate the usefulness of the framework whose prototype we are proposing today, we have considered a case study reflecting the needs of our partner Berger-Levrault (BL)[4]. Our case study focuses on implementing a software product line for interoperability connectors in this context. Connectors are components that enable interaction between applications, regardless of their heterogeneity [15]. Indeed, the information systems of companies such as our industrial partner BL have different types of connectors. Some connectors allow the transfer of files, others allow asynchronous communication, and still, others are a mixture of synchronous and synchronous communication. Connectors share common characteristics while each may have its variants, and we can therefore consider them as software product lines as stated [16]. In addition, component evolutions are ubiquitous. To create a new connector, developers copy an existing interoperability mechanism and adapt it to the new connector, removing and adding new features that look like a semblance of the manual of delta-oriented programming in this article. Figure 3 illustrates an example of interoperability connectors that present some variability.

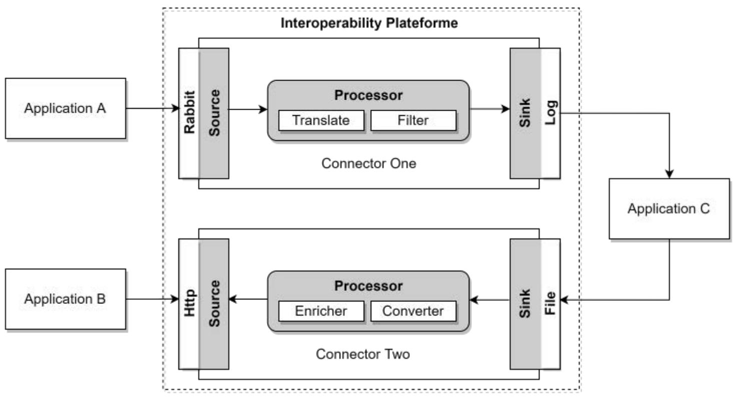

Figure 3: Industrial case study that represents two interoperability connectors that establish communication between several heterogeneous application.

In the Figure 3, we notice the presence of two interoperability connectors that share some common features, such as a source that receives messages from applications, a sink that transmits messages to other applications, and eventually a processor if the information that passes through the connector need to be processed, the processor is a common feature but is not required. We also identify some variability depending on the communication need. We note, for example,

[3]https://github.com/pharo-spec/Spec
[4]Berger-Levrault is a software provider specialized in the fields of education, health, sanitary, social and territorial management.

that the ”Connector One” has a Rabbit Source, while the ”Connector Two” has an HTTP source. Figure 4 show the feature model that presents all the possible configurations to create connectors. The feature model presented in Figure 4 is used as one input for the tool, the other being the

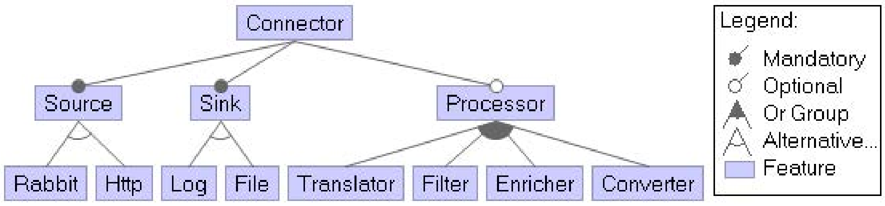

Figure 4: The feature model of the use case, created with the FeatureIDE tool. It presents all possible configurations to create a valid interoperability connector product.

target connector specification.

The next step is to create the software product line, *i.e.,* the solution space of the DE sub-process. This will consist of creating the connector core model and required delta modules to prepare delta actions that will provide the expected product model at the output and solution space of the AE sub-process.

The metamodel shown in Figure 5 upper box represents the core model that allows the creation of a valid connector core model with respect to the feature model.

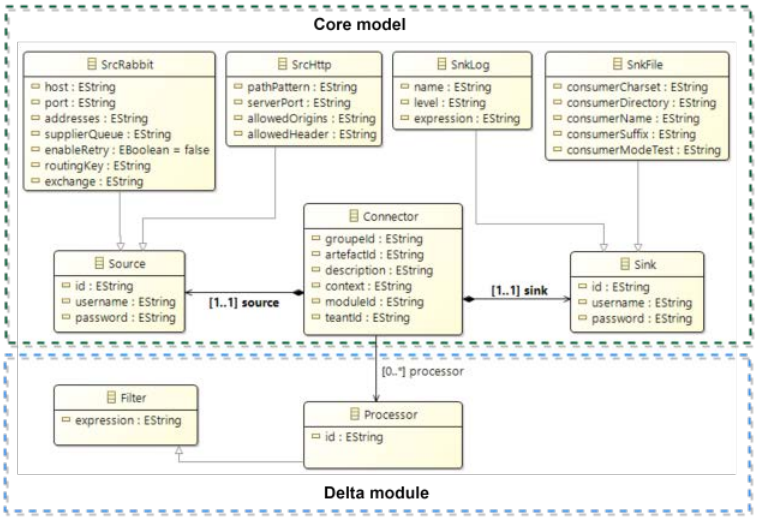

Figure 5: The connector metamodel of the use case is presented in the upper box. The box below represents an example of a delta operation that consist in creating a delta module.

Once the base model is available, we can prepare to apply delta modules, *i.e.,* modification operation to change the core model. Delta modules are applied to the core model according to constraints defined by the feature. The lower box on Figure 5 shows an example of delta module that modifies a core model.

As presented in Figure 5, applying a delta module to a core model generates a delta action for a given configuration. The resulting delta actions are added to the delta action set for further use, *i.e.,* in application engineering. Then, several delta actions can be applied to a core model to produce the model of an expected product. The proposed prototype is designed to cover the requirements of the realization process.

5. Future Works

The first stage of reflection allows us to identify the needs and the different functional blocks required for our tool. First of all, the parsing stage of a feature model in XML format is well advanced but still in progress. The XML file is provided to extract each listed feature using the XMLDOMParser class available in Pharo. Indeed, in addition to isolating all the functionalities, we have to keep the relationships and dependencies between features to propose a product line expression, a textual representation corresponding to the feature model as a logical equation. In the short term, we want to work on the EPL validator, which allows us to validate the imported feature model. For future work, in the short term, we want to work on the product line expression validator, which allows us to validate the import feature model and the configurator. The configurator will be a graphical interface proposing a visual representation of the different features in the form of a box with hierarchical links between them, allowing the user to select features according to an expected specification of the software product while respecting the constraints imposed by the EPL validator. Then we will work on the difference calculation engine between models based on the Orion framework and the automatic recovery of all delta modules with the action deltas that contain them. Here we want to enable the tool to compute all possible module deltas exhaustively based on the relationships between entities in an original base model with mandatory and optional entities, e.g., compositional relationships and cardinalities. However, some tasks are planned in a second step. These are the semi-automatic transformation of the feature model into a base model representing the product line and the formalization of the expected product specification for a semi-automatic configuration.

6. Conclusion

In this paper, we present the prototype of a future tool that aims to implement a software product line using the model-based delta-oriented paradigm. An industrial case study describes the different steps of the framework on which the tool is based. The article provided the theoretical and technical basis for the realization of the tool. The next step will be the concrete implementation of the tool using the Moose platform and the Orion frameworks. This environment offers complete control of the steps and underlying meta-modeling and transformation techniques that will allow us to practice our SPL approach.

References

[1] P. C. Clements, L. M. Northrop, Salion, inc.: A software product line case study, Technical Report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 2002.

[2] G. C. S. Ferreira, F. N. Gaia, E. Figueiredo, M. de Almeida Maia, On the use of feature-oriented programming for evolving software product lines—a comparative study, Science of Computer programming 93 (2014) 65–85.

[3] D. Batory, Feature-oriented programming and the ahead tool suite, in: Proceedings. 26th International Conference on Software Engineering, IEEE, 2004, pp. 702–703.

[4] S. Schulze, O. Richers, I. Schaefer, Refactoring delta-oriented software product lines, in: Proceedings of the 12th annual international conference on Aspect-oriented software development, 2013, pp. 73–84.

[5] C. Kästner, S. Apel, Integrating compositional and annotative approaches for product line engineering, in: Proc. GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering, 2008, pp. 35–40.

[6] D. C. Schmidt, Model-driven engineering, Computer-IEEE Computer Society- 39 (2006) 25.

[7] M. Nieke, A. Hoff, I. Schaefer, C. Seidl, Experiences with constructing and evolving asoftware product line with delta-oriented programming, in: Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems, 2022, pp. 1–9.

[8] N. Anquetil, A. Etien, M. H. Houekpetodji, B. Verhaeghe, S. Ducasse, C. Toullec, F. Djareddir, J. Sudich, M. Derras, Modular moose: A new generation of software reverse engineering platform, in: International Conference on Software and Software Reuse, Springer, 2020, pp. 119–134.

[9] J. Koscielny, S. Holthusen, I. Schaefer, S. Schulze, L. Bettini, F. Damiani, Deltaj 1.5: delta-oriented programming for java 1.5, in: Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java platform: Virtual machines, Languages, and Tools, 2014, pp. 63–74.

[10] C. Seidl, I. Schaefer, U. Aßmann, Deltaecore-a model-based delta language generation framework, Modellierung 2014 (2014).

[11] C. Pietsch, T. Kehrer, U. Kelter, D. Reuling, M. Ohrndorf, Sipl–a delta-based modeling framework for software product line engineering, in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2015, pp. 852–857.

[12] R. Mazo, C. Salinesi, D. Diaz, O. Djebbi, A. Lora-Michiels, Constraints: The heart of domain and application engineering in the product lines engineering strategy, International Journal of Information System Modeling and Design (IJISMD) 3 (2012) 33–68.

[13] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Technical Report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[14] J. Laval, S. Denier, S. Ducasse, J.-R. Falleri, Supporting simultaneous versions for software evolution assessment, Science of Computer Programming 76 (2011) 1177–1193.

[15] M. C. Kaya, M. S. Nikoo, S. Suloglu, B. Tekinerdogan, A. H. Dogru, Managing heterogeneous communication challenges in the internet of things using connector variability, in: Connected Environments for the Internet of Things, Springer, 2017, pp. 127–149.

[16] B. Niang, G. Kahn, N. Amokrane, Y. Ouzrout, M. Derras, J. Laval, Towards the generation of interoperability connectors using software product line engineering, 2021. Presented at 9ème Conférence en IngénieriE du Logiciel CIEL 2021. Available on HAL archive, no proceedings.