Multiple Software Product Lines: applications and challenges

Guadalupe Isaura Trujillo-Tzanahua[1] Ulises Juárez-Martínez[1], Alberto Alfonso Aguilar-Lasserre[1] and María Karen Cortés-Verdín[2]

[1]Division of Research and Postgraduate Studies. Instituto Tecnológico de Orizaba, Veracruz, México.
[2]Facultad de Estadística e Informática. Universidad Veracruzana, Xalapa, Veracruz, México.
[1]gtrujillot@ito-depi.edu.mx, ujuarez@ito-depi.edu.mx, aaguilar@itorizaba.edu.mx
[2]kcortes@uv.mx

Abstract. The goal of a software product line is to create a suitable platform for fast and easy production of software for same market segment. However, a software product line is limited because it needs to meet new stakeholder requirements either through upgrades or the introduction of new technologies. A Multi Product Line aims at deriving new software products from reuse of a set of features provided by several heterogeneous software product lines without modifying or altering the independent operation of the same. This paper presents a study about the application of Multi Product Lines in the software development process. It shows some domains that illustrate applications of multi product lines principle in the process and the product. Also, the main current challenges in applying multi product line in software engineering are described. This paper aims to show the importance and usefulness of applying multi product lines approaches in Software Engineering.

Keywords: Software Engineering, Software Product Lines, Multi Product Lines.

1 Introduction

Product lines are successfully used in both software and non-software domains such as automotive, metallurgy and manufacturing to support systematic reuse. A classic example is automobile manufacturing consisting of creating variations of a single car model with a set of parts and a factory specifically designed to configure and assemble such parts. Software Product Lines (SPL) are analogous to industrial manufacturing, in which similar products are configured and assembled from reusable prefabricated parts for fast and easy software development focused on a specific market.

The Software Engineering Institute (SEI) defines a Software Product Line as a "set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed

J. Mejia et al. (eds.), *Trends and Applications in Software Engineering*,
Advances in Intelligent Systems and Computing 688,
https://doi.org/10.1007/978-3-319-69341-5_11

from a common set of core assets in a prescribed way" [1]. Software Product Lines is a means to improve the processes of software development by reducing the cost and substantially improving the productivity and the quality of the products developed. For the development of a Software Product Line, a key element is the analysis, specification and management of common and variable elements within the set of products produced by the SPL.

Frequently, Software Product Lines need to meet the changing requirements of the market either by functionality, focus or technology and consequently, it is necessary to add and configure new products in the SPL. However, experts in the development of SPL recognize that they are limited and that in most cases, it is impossible to extend or adapt the platform. An alternative solution is a Multiple Software Product Line which derives new software products from the reuse of a set of features provided by different heterogeneous software product lines without modifying or altering the independent operation of the same. The purpose of this study is to examine Multi Product Lines applications in software engineering.

This document is organized as follows. Section 2 presents the research methodology. In section 3, basic concepts on Multi Product Lines are explained. Application of Multi Product Lines on different domains is shown in section 4. Challenges faced by multi product lines in the field of software engineering are presented in section 5. Section 6 presents the discussion. Finally, conclusions are presented in section 7.

2 Research methodology

The methodology is composed of three stages. The first stage was the research of works related to Multiple Software Product Lines in several databases of scientific journals. The second concerned the classification of these works in the different fields of knowledge. Finally, the third stage of the methodology involved the report of detailed literature review that identifies challenges addressed and technologies, tools and programming languages used in the implementation of the proof of concept applications in some of the reported works. For this review, we first searched in the major databases of electronic journals for a comprehensive bibliography of relevant research of MPL. The digital libraries considered were: (1) ACM Digital Library, (2) CiteSeerX, IEEE Xplore Digital Library, IGI Global, ScienceDirect (Elsevier), Semantic Scholar and SpringerLink.

3 Application of Multi Product Lines on different domains

Nowadays, there are many examples that illustrate the original idea of using MPLs, among which stand out mainly in the metallurgy, steel industry and mechatronics systems.

3.1 Metallurgy

In Metallurgy, alloys are a good example to achieve special desired properties by combining different metals. For copper to be versatile, its characteristics are modified through mixing with other metals depending on the desired end use. From this mixture, it is possible to obtain more than 400 alloys, such as bronze, brass, alumni bronze. Another alloy identified is derived from nickel and steel and is known as Invar or FeNi36. Invar is used in the manufacture of precision parts such as watchmaking, apparatus of physics, the valves of engines, among others and in instruments for measuring length such as those used in topography due to its small coefficient of expansion. This notion of combining, integrating or composing different approaches motivates the approach called Invar (Integrated View on Variability) [2, 3] for the development of software using multiple product lines. Invar facilitate the exchange of heterogeneous models of variability during the configuration of the product regardless of the techniques, notations and tools used in the organization.

3.2 Steel Industry

Another example of implementation of MPLs occurs in the Steel industry, specifically in the mini-mills that make up the product portfolio of SIEMENS VAI [4–7]. Unlike the integrated steel mill, the mini-mill is a facility that produces steel products using scrap steel as an iron source. A mini-mill is integrated by several product lines such as the electric furnace, the caster, the rolling mill, and the maintenance and setup system (MSS), a software tool used by customers for customizing the mini-mill software solution during operation. Although the mini-mill is subject to the same requirements that the integrated steel mill differs in that the plant is flexible with ability to be upgraded technically, diversity in the styles of management, labor relations and markets for the product. A mini-mill and the different subsystems can be customized in terms of the amount of iron, furnace type, number of filaments, type of lamination train or lamination capability.

3.3 Mechatronics systems

Mechatronics combines various disciplines as mechanical engineering, electronics, automatic control and software for the design of products and processes. Mechatronics systems have several applications: robotics, aeronautics, automotive industry, medical industry, home automation, among others. In the field of mechatronics, products are described by multiple models belonging to different engineering domains such as mechanical, electrical, and electromechanical. In medical domain, multiple product line development is identified for Philips Medical imaging systems through hierarchical product lines [8]. Typically, several product lines are available because products are developed in different parts of the world, and within different product groups such as magnetic resonance, X-ray, and ultrasound tomography. To handle the complexity in product lines for mechatronic systems, in [9] is proposed a MPL ap-

proach to distinguish between software and hardware by using different feature models for each.

Table 1 shows the analysis of four different domains of MPL application reported in the literature. This table highlights the main similarities and differences in the MPL applied to the industry for the manufacture of physical products and software products. It is important to note that in an MPL applied to the manufacturing industry, production capacity is considered because it defines the competitive limits of the company since the quantity of products or services that can be obtained by a period of time depends on demand and company infrastructure. While the capacity of an MPL in the context of software is unlimited because it can generate n software products. Both industrial or software MPLs agree that it is possible to involve different suppliers, obtain different versions of products (variability) and reuse processes and tools. Another aspect to highlight in MPL in the context of software is that similar to Software Product Lines not only takes care of generating software product but also generates requirements, code, architecture, tests, documentation.

Table 1. Analysis of application domains

Element	Domains			
Domain	Metallurgy	Steel Industry	Mechatronics systems	Software Engineering
Assets	Metals	Scrap steel	Hardware Software	Software artifact and such as requirements, feature model, architecture, libraries, test.
Reuse	Process, machines and tools	Process, machines and tools	Process, machines and tools	Process and software tools
Providers	Several	Several	Several	Several
Variability	Yes	Yes	Yes	Yes
Production Capacity	Limited by infrastructure	Limited by infrastructure	Limited by infrastructure	Unlimited but depends on product configuration
Line Balancing	Yes	Yes	Yes	Yes
Application	Alloys	Mini-mill	Philips Medical Imaging Systems, Aselsan REHIS	SECO, SOS, ERP, Software Supply Chain

MPL is a multi-domain approach that arises to develop large and complex systems through several independent product lines which are developed by several organizations with different approaches and technologies for different geographic areas and in any context. This approach is investigated for the development of Systems of Systems (SoS) [5, 10] and Software Ecosystem (SECO) [11, 12] that result from the inte-

gration of several operationally independent systems. The feasibility of the approach and its implementation could be useful for implementation of ERP systems [2] or software supply chains[13].

4 Multiple Software Product Lines

Software artifact reuse from different software product lines, referred to as multi product lines [14, 15] is addressed by several authors [16–18]. The reuse and composition of multiple software product lines is also known as Nested Software Product Lines [19, 20], Hierarchical Product Lines or Composite Product Lines.

A Multiple Software Product Line (MSPL) also called Multi Product Line (MPL) is a software product line that results from combining components or products developed from several independent and heterogeneous software products lines [15]. It means that software product lines are provided by different organizations and use diverse approaches and technologies.

According to Holl [14], an MPL is ”a set of several self-contained but still interdependent product lines that together represent a largescale or ultra-large scale system”.

Multi Product Line configurator is an assembly entity that is responsible for controlling and reusing the artifacts of software product lines according to the needs of stakeholders (see Fig1). Given the need to combine, integrate or compose different software product lines, the inclusion of different approaches to model variability, annotations and tools is detected.

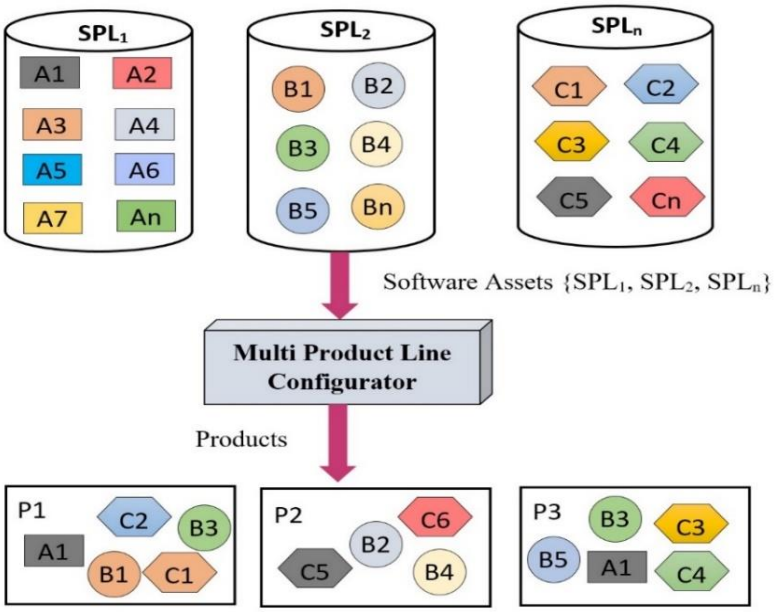

Fig. 1. Multi Product Lines

The decision of using a traditional approach or mass production through SPL or MPL for software development depends on requirements such as: reusing components, implementing non-functional requirements, launching or modifying a product to focus on a different market segment, acquisition of a competitive advantage, incorporation of new technologies (hardware, software), resource sharing, among others [21].

5 Challenges

Some of the general challenges faced by multi product lines in the field of software engineering are presented below.

5.1 Software Product Lines Reuse

Reuse in Multi Product Lines refers to the process of deploying or upgrading software systems using existing software assets across multiple software product lines. This results in a composition of SPLs that facilitates to reuse SPLs within other SPLs. Consequently, techniques, methods or approaches are required to facilitate and maximize the reuse of artifacts from software product lines to generate valid product families in a Multi Product Line [22–24]. Likewise, the fusion and reuse of feature models provided by different companies or suppliers are required. The degree of reuse depends on the scope of the available SPLs, so for the development of an MPL not only reuse the implementation but it is possible to reuse processes, tools, requirements, tests, technologies.

5.2 Interoperability between software product lines

Interoperability refers to the ability of two or more systems or components to exchange information and use the information exchanged. In an MPL, the interoperability and the integration of software product lines and their products must be facilitated [24, 25]. An adequate interoperability is useful to promote cooperation between independent systems in order to integrate them as a System of Systems [5, 10] or Software Ecosystem [11, 12] and provide more complex functions.

5.3 MPL Reference Architecture

Reference Architecture is a type of software architecture that captures knowledge and experience about how to structure architectures of software systems in a domain. Its purpose is therefore to be a guidance for the development, standardization, and evolution of systems of a single domain or neighbor domains for example Autosar (AUTomotive Open System Architecture Open Systems) for the Automotive sector. One or more reference architectures could be used as a basis of MPL. In an MPL, is necessary to define a reference architecture or consistent MPL architecture [26] that repre-

sents the common and variable artifacts of the software product lines available for incremental product development.

5.4 Automating product derivation in MPL Engineering

Product derivation in an MPL refers to complete process of building a product through the software assets of many software product lines. Automating the derivation of the product means automating this building process. The subprocesses of product derivation are product configuration and product generation.

Product configuration in an MPL is a process of collaboration between different people and teams with different knowledge regarding the domain, notations and programming languages used in the development of software product lines. Problems arise at the moment of making the decisions to assign the interested parties in the right order and at the right time based on the knowledge of the domain of the people.
Generating products in an MPL relies on a configurator to make decisions, taking as input the configuration to build the final product (or parts of it, such as executables, documentation, tests, and so on).

5.5 User Guide on Product Derivation

During the derivation of products in an MPL, it is necessary to ensure that the multiple users involved in the configuration and generation of products are aware of the chosen decisions [4]. For this reason, users need to be aware of the variability and dependencies between software product lines.

5.6 MPL tools

Supporting tools offer the developers a complete environment for development and maintenance of software product line, aiming at facilitating its adoption. Although there is a huge variety of tools for software product lines development, it is not possible to ensure that all needs of SPL engineers are being fulfilled. It is necessary to better investigate the scope, the availability and the utility of these tools for MPLs development. For this reason, SPL engineers need to adapt and extend of software product line tools for feature modeling, SPL composition, variability management, configuration and derivation of products in an MPL. In addition, tools are required to facilitate automated analysis of dependent features models.
Another aspect that stakeholders need to consider for MPL implementation is the use of different approaches in individual software product lines such as Feature-oriented Programming (FOP), Aspect-Oriented Programming (AOP) and Delta-oriented Programming (DOP).

6 Discussion

Multi Product Lines are successfully used in domains such as metallurgy, steel industry, mechatronics systems and software (Section 3). The results of this study confirmed that Multi Product Lines approach is a feasible option for software mass customization. This is because the principles used in other areas mainly in the metallurgical industry, steel industry, mechatronic systems are completely applicable to software development in order to optimize the individual systems development by taking advantage of their common features and managing their differences.

In the software industry, MPLs emerge as a flexible and viable development paradigm that enables companies to enhance their products from reusing and mass customization to a rapid market introduction, reducing costs and maximizing quality of products.

Currently, several approaches and proposals that support the development of software using MPLs are reported. However, as mentioned in Section 5, MPLs in software context present certain limitations and challenges that need to be addressed. Also, several areas of opportunity were identified in this field of research since the use of MPL depends mainly on the capabilities of the development team, market and technologies.

7 Conclusion

This study addresses the main aspects necessary to understand the importance of multi product lines and its application in the field of software engineering. Multi product line is an area with a great potential of applications in the field of software engineering, particularly in the study of software development processes. In addition, the different challenges and problems faced by multi product lines are identified.

The current study on Multi Product Lines allowed to detect that despite the experience that has in the industry, the Software Engineering requires to meet certain challenges that arise when increasing the complexity of the systems, by addition of new products, make updates by technology or change of requirements, all this in order to meet the needs or expectations of the market. Also, several areas of opportunity are detected. This work serves as a basis for future research on Multi Product Lines.

For the future work, we aim at contributing to mature the area of Multi Product Lines and propose means to better explore the software product development using this approach.

References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. (2001).
2. Dhungana, D., Seichter, D., Botterweck, G., Rabiser, R., Grünbacher, P., Benavides,

D., Galindo, J.A.: Configuration of Multi Product Lines by Bridging Heterogeneous Variability Modeling Approaches. In: Proceedings of the 2011 15th International Software Product Line Conference. pp. 120–129. IEEE Computer Society, Washington, DC, USA (2011).

3. Dhungana, D., Seichter, D., Botterweck, G., Rabiser, R., Grünbacher, P., Benavides, D., Galindo, J.A.: Integrating Heterogeneous Variability Modeling Approaches with Invar. Proc. Seventh Int. Work. Var. Model. Software-intensive Syst. 8:1--8:5 (2013).
4. Rabiser, R., Grünbacher, P., Holl, G.: Improving Awareness during Product Derivation in Multi-User Multi Product Line Environments. In: Proceedings 1st Int'l Workshop on Automated Configuration and Tailoring of Applications (ACoTA 2010) in conjunction with 25th IEEE/ACM Int'l Conference on Automated Software Engineering (ASE 2010), Antwerp, Belgium, September. pp. 1–5 (2010).
5. Holl, G., Elsner, C., Grünbacher, P., Vierhauser, M.: An Infrastructure for the Life Cycle Management of Multi Product Lines. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. pp. 1742–1749. ACM, New York, NY, USA (2013).
6. Holl, G., Thaller, D., Grünbacher, P., Elsner, C.: Managing Emerging Configuration Dependencies in Multi Product Lines. In: Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems. pp. 3–10. ACM, New York, NY, USA (2012).
7. Holl, G., Grünbacher, P., Elsner, C., Klambauer, T., Vierhauser, M.: Constraint Checking in Distributed Product Configuration of Multi Product Lines. In: 20th Asia-Pacific Software Engineering Conference (APSEC). pp. 347–354. IEEE Computer Society, Washington, DC, USA (2013).
8. van der Linden, F., Wijnstra, J.G.: Platform Engineering for the Medical Domain. Presented at the (2002).
9. Brink, C., Peters, M., Sachweh, S.: Configuration of Mechatronic Multi Product Lines. In: Proceedings of the 3rd International Workshop on Variability & Composition. pp. 7–12. ACM, New York, NY, USA (2012).
10. Klambauer, T., Holl, G., Grünbacher, P.: Monitoring System-of-Systems Requirements in Multi Product Lines. In: Doerr, J. and Opdahl, A. (eds.) Requirements Engineering: Foundation for Software Quality. pp. 379–385. Springer Berlin Heidelberg (2013).
11. Urli, S., Blay-Fornarino, M., Collet, P., Mosser, S., Riveill, M.: Managing a Software Ecosystem Using a Multiple Software Product Line: a Case Study on Digital Signage Systems. In: Euromicro Conference series on Software Engineering and Advanced Applications(SEAA'14). pp. 344–351 (2014).
12. Schmid, K., Eichelberger, H.: EASy-Producer: From Product Lines to Variability-rich Software Ecosystems. In: Proceedings of the 19th International Conference on Software Product Line. pp. 390–391. ACM, New York, NY, USA (2015).
13. Hartmann, H., Trew, T.: Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains. In: Proceedings of the 2008 12th International Software Product Line Conference. pp. 12–21. IEEE Computer Society, Washington, DC, USA (2008).
14. Holl, G., Grünbacher, P., Rabiser, R.: A Systematic Review and an Expert Survey on

Capabilities Supporting Multi Product Lines. Inf. Softw. Technol. 54, 828–852 (2012).
15. Rosenmüller, M., Siegmund, N.: Automating the Configuration of Multi Software Product Lines. In: Proceedings of Fourth International Workshop on Variability Modelling of Software-Intensive Systems. pp. 123–130 (2010).
16. Aoyama, M.: Continuous and Discontinuous Software Evolution: Aspects of Software Evolution Across Multiple Product Lines. In: Proceedings of the 4th International Workshop on Principles of Software Evolution. pp. 87–90. ACM (2001).
17. Aoyama, M., Watanabe, K., Nishio, Y., Yasuyuki, M.: Embracing requirements variety for e-Governments based on multiple product-lines frameworks. In: Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International. IEEE (2003).
18. Bühne, S., Lauenroth, K., Pohl, K.: Why is it not Sufficient to Model Requirements Variability with Feature Models? Work. Automot. Requir. Eng. AURE04. 4, 5–12 (2004).
19. Krueger, C.W.: New methods in software product line development. In: Software Product Line Conference, 2006 10th International. pp. 95–99. IEEE (2006).
20. Marinho, F.G., Andrade, R.M.C., Werner, C., Viana, W., Maia, M.E.F., Rocha, L.S., Teixeira, E., Filho, J.B.F., Dantas, V.L.L., Lima, F., Aguiar, S.: MobiLine: A Nested Software Product Line for the domain of mobile and context-aware applications. Sci. Comput. Program. 78, 2381–2398 (2013).
21. Savolainen, J., Mannion, M., Kuusela, J.: Developing Platforms for Multiple Software Product Lines. In: Proceedings of the 16th International Software Product Line Conference - Volume 1. pp. 220–228. ACM, New York, NY, USA (2012).
22. Rosenmüller, M., Siegmund, N., Kästner, C., Ur Rahman, S.S.: Modeling Dependent Software Product Lines. Engineering. 13–18 (2008).
23. Altintas, N.I., Cetin, S.: Managing Large Scale Reuse Across Multiple Software Product Lines. In: High Confidence Software Reuse in Large Systems. pp. 166–177 (2008).
24. Schröter, R., Siegmund, N., Thüm, T.: Towards modular analysis of multi product lines. Proc. 17th Int. Softw. Prod. Line Conf. co-located Work. - SPLC '13 Work. 96 (2013).
25. Nakagawa, E.Y., Oquendo, F.: Perspectives and Challenges of Reference Architectures in Multi Software Product Line. In: Proceedings of the 17th International Software Product Line Conference Co-located Workshops. ACM, New York, NY, USA (2013).
26. Tekinerdogan, B., Erdoğan, Ö.Ö., Aktuğ, O.: Chapter 10 – Archample—Architectural Analysis Approach for Multiple Product Line Engineering. In: Relating System Quality and Software Architecture. pp. 263–285 (2014).

Automated software generation process with SPL

Jesús-Moisés Hernández-López[1,1], Ulises Juaréz-Martínez[2,2] and Ixmatlahua-Díaz Sergio-David[1,3]

[1] Instituto Tecnológico Superior de Zongolica, Veracruz, México,
[2]Instituto Tecnológico de Orizaba, Veracruz, México
[1]jesus_hernandez_pd197@itszongolica.edu.mx, [2]ujuarez@ito-depi.edu.mx,
[3]sergio.ixmatlahua.pd169@itszongolica.edu.mx

Abstract. Software Product Line (SPL) is a set of applications with a common architecture and shared components, with each application specialized to reflect different requirements. SPLs manage a large number of artifacts, for each of them it is necessary to define *commonality* and *variability*. Consequently, the applications generation from SPL becomes complicated and it often must be done manually. This paper shows the development of a SPL with an automated software generation process, in which *commonality* and *variability* have been defined for each artifact (i.e., product management, requirements, design, realization and test). Some technologies used were: XML files, Scala, AspectJ, Apache Maven and Junit. To automate the applications generation process, we have developed a configurator that enables *features selection* for each application and generate it in an automated way from SPL. The software generated includes: executable and documentation. Further, we propose a model-driven architecture for support the evolution in the SPL.

Keywords: variability, MDA, inmotic, traits, aspects, mixins

1 Introduction

Nowadays, there is a demand for enhancing the quality of software, reducing costs and accelerating their development processes [1]. SPLs are focused on these aspects (i.e., quality, cost and time).

A SPL is a set of applications with a common architecture and shared components, with each application specialized to reflect different requirements [1]. SPL development requires generating a large number of artifacts, such as: product management, requirements, design and test [2]. For each artifact generated, *variability* and *commonality* must be defined and managed. Variability defines the flexibility of a SPL to generate products with different features and commonality defines artifacts

J. Mejia et al. (eds.), *Trends and Applications in Software Engineering*,
Advances in Intelligent Systems and Computing 688,
https://doi.org/10.1007/978-3-319-69341-5_12

that will be reused in each application from SPL. Due to the number of artifacts, the generation process becomes in a complicated task for developers, which many times must be done manually. The drawback of doing it manually is that it increases the time of software generation and there is a risk of making mistakes in the configuration.

In this paper, we show the development of a SPL based on *features selection* with an automated software generation process. The development of this SPL was carried out through the classic methodology for Software Product Line Engineering, in which 2 processes are distinguished: Domain engineering process.– *This process is responsible for establishing the reusable platform and thus for defining the commonality and the variability of the product line*. Application engineering process.- *This process is responsible for deriving productline applications from the platform established in domain engineering* [2]. *Commonality* and *variability* were defined with different technologies, such as: XML files, Scala, AspectJ, Apache Maven and Junit. To automate software generation process, we developed a configurator, which is in charge of reusing common artifacts and exploiting variability defined in the SPL, in order to generate complete applications (i.e., documentation and/or executable application). The process is executed in an automated manner, based only on *feature selection.* Further, we propose a model-driven architecture for support the evolution in the SPL. To verify our approach, we choose inmotic domain as case study.

Document is organized as follows: Section 2 presents related works. Section 3 explains the automated software generation process with a simple example. Section 4 presents a case study. Section 5 shows a proposal to support evolution in SPLs. Section 6 summarize conclusions and future work.

2 Related works

In [3], variability was introduced through integration of Model-Driven Software Development (MDSD) and Aspect-Oriented Software Development (AOSD). Combination of MDSD and SPL facilitates traceability from problem domain to solution domain. Aspect-oriented language is useful to generate code where an architecture doesn´t provide links.

In [4], an Aspect Oriented Analysis (AOA) on product requirements was presented to design Product Line Architectures (PLA). AOA scheme consists of: (1) requirements are separated in each aspect of original requirements, (2) requirements of each aspect are analyzed and the architecture is examined for each of them, (3) results and design options are analyzed.

In [5], an MDE (Model-Driven Engineering) and Aspect-Oriented Software Development approach was presented to facilitate implementation, management and traceability of variability. Features are separated into models, which are composed by AO techniques at model level. By integrating MDSD into SPLE (Software Product

Line Engineering), DSL (Domain Specific Language) manages the variability with respect to its structure or behavior.

In [6], a new approach was presented to implement SPL by fine-grained reuse mechanisms. Featherweight Record-Trait (FRTJ) was introduced and product functionality units are modeled with traits and records. Reuse degree of traits and records is higher than the potential to reuse hierarchies based on standard static classes.

In [7], an approach to facilitate variability management was proposed to model architectures of SPLs. Domain requirements and architecture are captured into models. For application engineering, DSL was used to specify requirements of particular applications. AO techniques were used during the domain engineering to modulate concerns in models, transformers, and generators.

In [8], delta-oriented programming (DOP) was proposed. DOP is a programming language designed to implement SPLs. A delta module can add classes for products implementation or remove classes from them. A SPL implemented with DOP is divided into a core module and delta modules. The core module comprises a set of classes to implement a complete product with valid feature configuration. It enables a flexible modular implementation for product variability, starting from different core products.

In [9], an automated assembly for domain components of a PLA was presented at low level abstraction. To address this problem, Model-Driven Engineering (MDE) was used. Benefits obtained with MDE are: (1) improving development of PLA with integration of modeling tools and specific domain components, (2) model-based structures help keep stability of domain evolution in MDE-based systems, (3) improving robustness and ability of models transformation, further debug support to correct errors in transformation specifications.

Related works show different approaches to manage variability and commonality at different levels of abstraction. There are some approaches that enable managing variability at code level, such as: *aspects*, *traits* and DOP [1] [3] [4] [5] [6]. We have also researched approaches to manage variability at high level abstraction: for example: MDA and AOA approaches [2] [7]. In conclusion, approaches based on *aspects* enable code manipulation at compile time and *traits* have a higher degree of reuse than classic class inheritance. MDA approaches support the evolution on PLA.

3 Automated software generation process

The automated application generation process is described step by step in Figure 1 through a simple example. Step (1), the administrator of the SPL makes *features selection* from *features model*, which is shown with a graphic interface in the configurator. The *features model* was defined through Common Variability Language (CVL) [10]. This example contains a *features model* with 3 features (A, B and C).

The Variability Specification Tree Resolution (VSpectree Resolution) is generated in step (2), applying CVL rules [10] defined in the configurator. The example of Figure 1 shows a VSpectree Resolution with two features (A and C), which is a valid selection for the features model.

VSpectree Resolution is verified in step (3) with the *configuration file* based on XML. This file contains features defined in the SPL, features definition for each application, and location of each artifact. The example of Figure 1 shows a *VSpectree Resolution* is equal to "Product 1", defined in the *configuration file*.

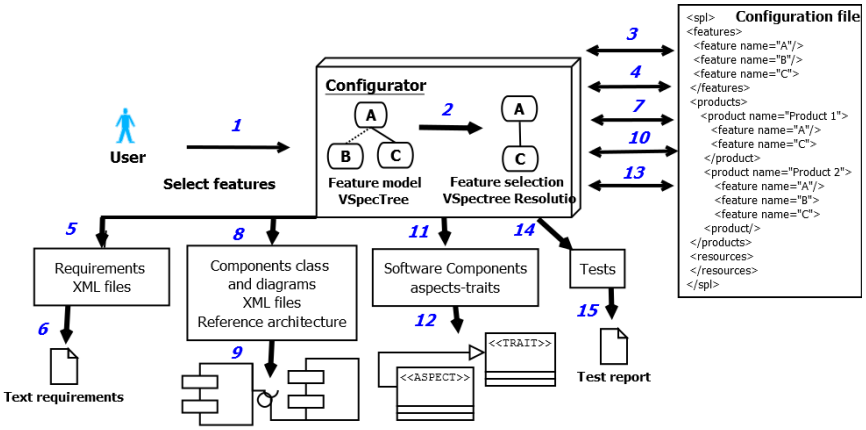

Figure 1. Automated application generation processes

Requirements artifacts (i.e., requirements on XML files) are reused in steps (4), (5) and (6). Common requirements for all applications are reused directly, and specific requirements are added depending on the application that will be generated. *Requirements artifacts* are result of *domain engineering process [2]*, which it defines communality and variability. Figure 1 shows the configurator working like a traceability link mechanism, linking features between "Product 1" from *configuration file* until *requirements artifacts.* The configurator generates application requirements in the step (6), reusing requirements with the value "Base" for the attribute condition and adding requirements where the value is equal to features name in "Product 1". . This process is observed in Figure 2.

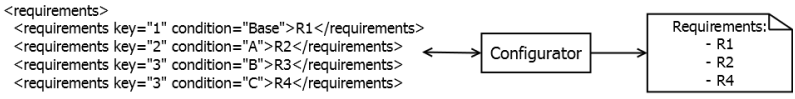

Figure 2. Requirements generation for "Product 1"

Like steps (4) and (5), steps (7) and (8) work with design artifacts developed on XML files, such as: components and class diagrams. *C*ommunality and variability were defined using a tool called PlantUML [11]. The artifact generated by *domain*

engineering is the *reference architecture.* This artifact is reused for all applications of the SPL in order to generate complete architectures for applications that will be generated. *To* generate a complete architecture in step (9), *application engineering process is* carried out *as* observed in Figure 3. The process is the following: Configurator chooses components, interfaces and relations with value "Base" in order to generate *reference architecture*. Subsequently, the configurator chooses and assembles components, interfaces and relations, verifying that the value in *condition* is equal to features name from "Product 1" from Figure 1.

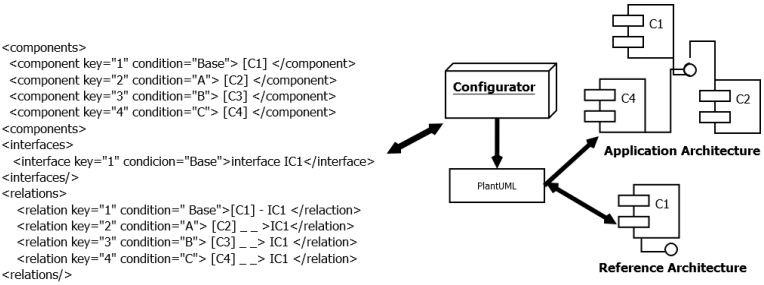

Figure 3 Example of architecture generation

Steps (10), (11) and (12) of Figure 1 are carried out in order to generate executable applications. In step (10), the configurator is responsible for verifying whether each *trait* is available, comparing *traits* and *features* defined for "Product 1" *in file configuration.* If there is no problem then the process continues. The application generation works in a systematized way from step 12. This process is shown more broadly in Figure 4. Before starting compilation, the configurator identifies the product that will be generated at time compile to weave *aspects* and *traits* through Maven [12], using command "*mvn install"*. "*AddFeatures"* aspect is responsible for assembling features "A" and "C" in *the reference architecture* during compilation. Components assembly is performed by *application engineering process.* Finally, whether there are no faults in the testing phase, then executable application (.jar) is generated.

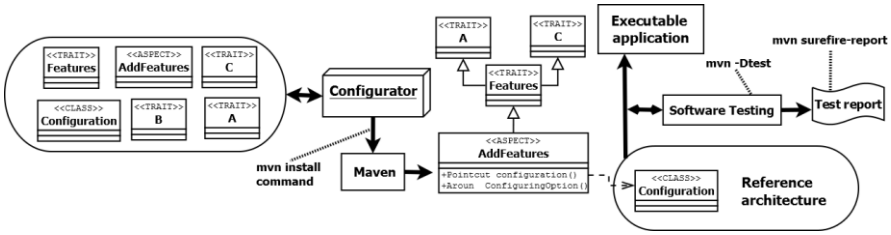

Figure 4 Applications generation process

Steps (13), (14) and (15) are carried out in order to apply software tests for the application configured in step (12). We have developed test suites with JUnit framework to distinguish domain and application tests. Before executable application

(.jar) is generated, tests are executed by the configurator with "*mvn –Dtest"* command, as observed in Figure 4. Whether there are no faults in the testing phase, then executable application (.jar) is generated with its test report, using "mvn surefire-report" command. Whether there are faults, the test report is also generated but the application generation process (.jar) is aborted until it is corrected. The following is an example of *Test Suite* for "Product 1" from Figure 1, in which will only execute tests for Product 1, specifically test for A and C features.

```
@RunWith (classOf [Categories])
@IncludeCategory (classOf[Product1])
@SuiteClasses(Array(classOf[TestA],classOf[TestC))
class SuiteProducto7Test {}
```

4 Case study

The case study chosen for this SPL was the inmotic domain. Inmotic is the incorporation of numerous subsystems in installations of tertiary or industrial use, in order to optimize resources, reduce costs and unnecessary energy consumption. The reason for dealing with this domain was to get a domain in which the variability is necessary. Examples of variability about inmotic domain, such as: sensors (e.g., flame, light or temperature), actuators (e.g., relays, valves or motors) and user interfaces (e.g., TV interface, a web-based interface or mobile interface).

Figure 5 shows features model defined with CVL [10] for the SPL. Lights feature and device feature define a large part of *commonality* in the SPL. They are defined with continuous line (i.e., mandatory features), therefore, applications generated will always have LIGHTS feature and DEVICE feature (i.e., a microcontroller). Variability is defined at different levels VSpectree. The most notorious variability is defined in device feature . There are applications that work with ARDUINO and others with RASPBERRY, but they don't work in the same application.

.

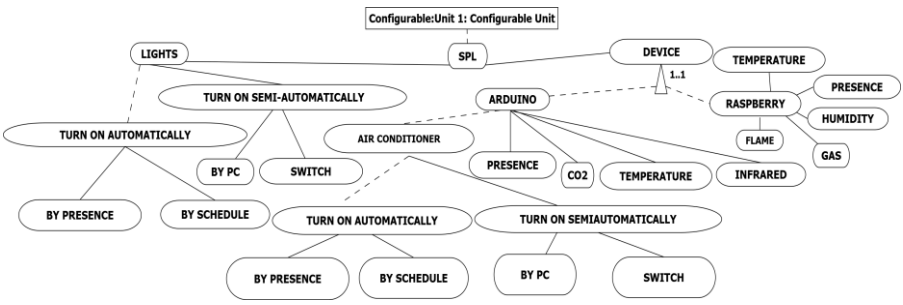

Figure 5 Features model defined with CVL (VSpectree)

The SPL is able to generate 8 applications for inmotic domain, which include executable applications, requirements, design and test report. Different artifacts are

generated for each application, however, some components are shared between them. Two valid aplications are shown in Table 1, in which there are notable differences. For example: application 1 doesn´t has any automation features, such as: Turn on Lights automatically by presence sensor or turn on lights automatically by schedule. Aplication 2 have features for light automation and other sensors, such as: flame, humidity or gas.

Table 1. Valid applications obtained from VSpetree of Figure 5

Application 1	Application 2
Lights	Lights
Turn on lights semi-automatically	Turn on lights semi-automatically
Lights control by pc	Lights control by pc
Lights control by switch	Lights control by switch
Arduino	Turn on lights automatically
Air conditioner	Turn on lights automatically by schedule
Turn on air conditioner semiautomatically	Turn on lights automatically by precense sensor
Air conditioner control by pc	Raspberry
Air conditioner control by switch	Presence sensor
Presence sensor	Humidity sensor
CO2 sensor	Gas sensor
Temperatura sensor	Flame sensor
Infrared sensor	

The *reference architecture* of this SPL is shown in Figure 6. *Reference architecture* is an incomplete architecture that will be reused and completed to generate each application of this SPL, as is explained in Section 3. Architectural style of the *Reference architecture* is the classic client-server.Therefore, after every automated application generation process, SPL in conjunction with the configurator will deliver two executables (i.e., client.jar and server.jar).

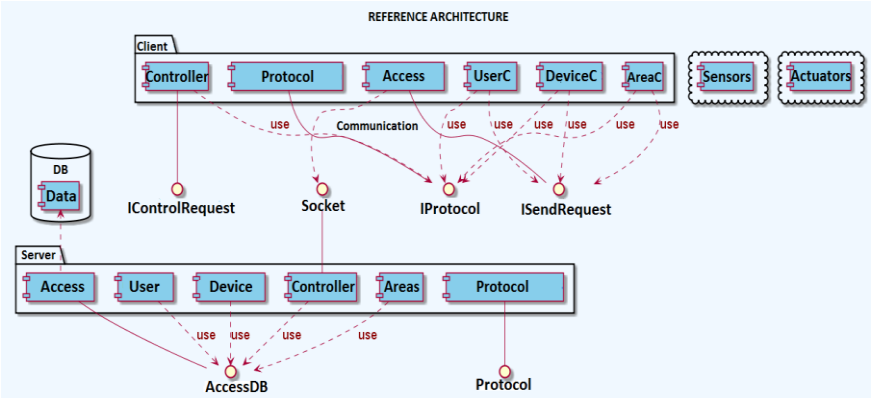

Figure 6. Reference architecture of the SPL

Design artifacts and realization artifacts for the applications described in Table 1 are shown to observer the reuse of the same components and differences between them caused by variability. Figure 7 shows a diagram class in which application 2 is configurated, as observed in Table 1. Trait called "*Features"* inherits functionality from traits called "*Light", "allData", "Presence","Temperature", "Flame", "Gas"and "Humidity"* creating mixin composition . Subsequently, the *aspect* called "*Access*" inherits the *trait* "*Features*" configured to add it in the *reference architecture* to generate the *application architecture of the application 2.*This process is generated at binary level by the configurator, which is explained in Section 3.

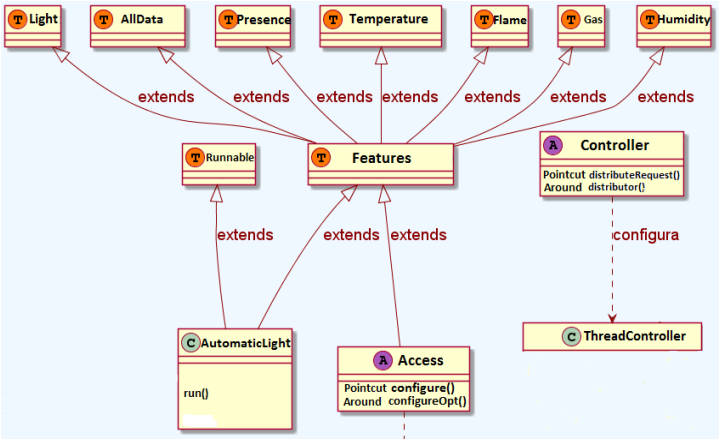

Figure 7. Combination of *aspects* and *traits* to configure Application 2

Coming up next is a fragment of the configuration for some features in Application 1. In this code is shown the reuses of "Presence", "Light" and "Temperature" traits. The "*AirConditioner", and "Infrared"* assembly is equally performed for the 8 applications from SPL. The *"AccessAspect"* aspect inherits "*Features*" trait and with a cutting weaves the full functionality where the *Pointcut* has indicated.

```
trait Features extends Light
with AirConditioner
with Presence
with Temperatura
With Infrared
with CO2
with Arduino{}

@Aspect
class AccessAspect extends Features {}
@Pointcut("call()) && args()
def configure() = {}
@Around()def configureOpt() = {}
```

5 MDA to Support Evolution in SPLs

Our proposal to address problems in terms of evolution was represented through two models of MDA approach: Platform-Independent Model (PIM) and Platform-Specific Model (PSM). PIM will represent all elements of the SPL at high level abstraction, without indicating technologies. PIM can be observed in Figure 9. Coming up next PIM elements are described:

Domain. - Corresponds to market segment for one SPL.
Features model. - There are different languages to model features. PIM does not indicate any specific language.
Features selection. - This concept will get a model features instance.
Platform. – It contains artifacts obtained from domain process and application process
Configurator. - It's responsible to generate applications using automatically artifacts (i.e., domain artifacts and application artifacts). Also, it allows verify all domain artefacts, before generating applications.
Domain or application artifacts. - They are artifacts generated by the configurator.

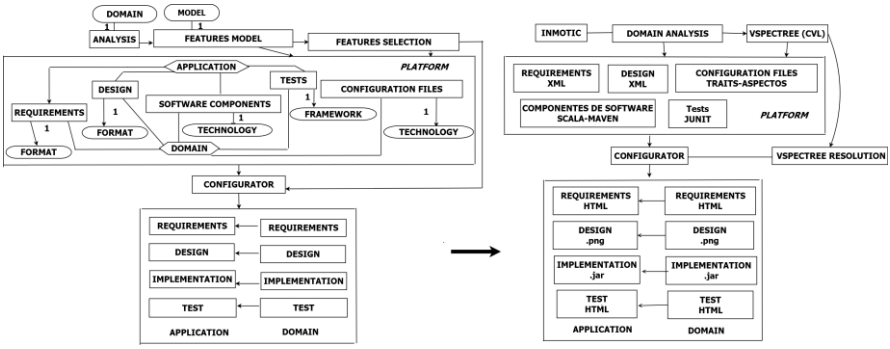

Figure 8 PIM to PSM

From PIM, is possible obtaining one or more PSMs, showing a projection of the PIM. PSM generation is based on transformation rules established in the PIM. The PSM generated in Figure 8 shows a model with elements of a SPL for inmotic domain.
In this way is possible generating others PSMs for different domains.

6 Conclusions and future work

The *application engineering process* [2] was automated by the configurator, which is responsible to generate artifacts for 8 applications from SPL. Commonality and

variability of the SPL was managed at different levels of abstraction with technologies quite flexible. *Aspects* and *traits* allow weaving binary code at specific points of the reference architecture to configure applications. MDA was used to propose a PIM to support evolution in SPLs, thinking not to depend on technology, but rather to adapt to unexpected changes. PSMs generated from PIM will be constantly changing at technological level but without modifying the PIM, this way, the evolution will be in both models. Although the goal of this paper neither comparative nor statistical, it is quite remarkable that the automated software generation process from SPL reduces time to generate software instead of manually configuring it. An automated software generation from this SPL takes about 10 minuts, depending the number of features chosen for one application. The software quality increases also due to the constant work with the same software components, further, test reports generated indicate whether the software has bugs and where they were located. Future work will be intended to perform a comparative and statistical analysis of advantages and disadvantages between manual processes and automated process to generate applications, and to evaluate attributes focused on time and quality. It will also increase the number of applications to the product portfolio [2] of the SPL.

Reference

1. Sommerville, I.: Software Engineering 9th ed, (2010)
2. Van Der Linden, F., Pohl, K.: Software Product Line Engineering: Foundations, Principles, and Techniques. In: Springer Science Business Media, (2005)
3. Voelter, M., Groher, I.: Product line implementation using aspect-oriented and model-driven software development. In: Software Product Line Conference, 2007. SPLC 2007. 11th International, pp. 233-242 IEEE, (2007)
4. Kishi, T., Noda, N.: Aspect-oriented analysis for product line architecture. In: Soft-ware Product Lines pp. 135-145, Springer, (2000)
5. Groher, I., Voelter, M.: Aspect-oriented model-driven software product line engi-neering. In: Transactions on aspect-oriented software development VI pp. 111-152, Springer, (2009)
6. Bettini, L., Damiani, F., Schaefer, I.: Implementing software product lines using traits. In: Proceedings of the 2010 ACM Symposium on Applied Computing pp. 2096-2102, ACM, (2010)
7. Groher, I., Voelter, M., Schwanninger, C.: Integrating Models and Aspects into Product Line Engineering. In: SPLC pp.355, (2008)
8. Schaefer, I., Bettini, L., Bono, V., Damiani, F., & Tanzarella, N. (2010, September). Delta-oriented programming of software product lines. In *International Conference on Software Product Lines* (pp. 77-91). Springer Berlin Heidelberg.
9. Deng, G., Schmidt, D. C., Gokhale, A., Gray, J., Lin, Y., Lenz,G.: Supporting Evolution in Model-Driven Software Product-line Architectures. In: ACM SIGSOFT Software Engineering Notes, ACM, (2007)
10. CVL Submission Team: Common variability language (CVL), OMG revised sub-mission ,(2012)
11. PlantUML, http://plantuml.com/
12. Maven, I.: Apache maven project, (2011)