A COMPARATIVE STUDY OF SOFTWARE PRODUCT LINES AND DYNAMIC SOFTWARE PRODUCT LINES

Reeta Kumari, Dr. Ashish Kumar Sinha and Dr. Mahua Banerjee

ABSTRACT

With invent of new technologies the world is becoming complex and competitive requiring rapid changes. The impact is there in software industry also. Software products with new features need to be introduced on the market with the same pace as the changing demands. Two methodologies namely Software Product Lines (SPLs) & Dynamic Software Product Lines (DSPLs) have been emerged to fulfill the needs. Both have their own features, advantages & disadvantages. This paper describes a comparative study of SPLs & their features, strengths & limitations along with DSPLs. Various tools to develop DSPLs are also discussed.

Index Terms: *Software Product Lines, Dynamic Software Product Lines, Self-Adaptive Systems, dynamic binding, Delta Oriented Programming, etc.*

Reference to this paper should be made as follows:

Reeta Kumari, Dr. Ashish Kumar Sinha and Dr. Mahua Banerjee, (2021), "A COMPARATIVE STUDY OF SOFTWARE PRODUCT LINES AND DYNAMIC SOFTWARE PRODUCT LINES" Int. J. of Electronics Engineering and Applications, Vol. 9, No. 2, pp. 01-10, DOI 10.30696/IJEEA.IX.I.2021.01-10.

Biographical notes:

Reeta Kumari *completed her B Level (MCA) from DOEACC Society and C Level (M.Tech.) from (National Institute of Electronics and Information Technology (NIELIT). Currently she is a Research Scholar in Jharkhand Rai University, Ranchi, India. She is working as Asst. Professor in Mrs. KMPM Vocational College, Jamshedpur, India. She has published 8 papers in various National & International Seminar and Conferences. Her area of interest is Software Engineering.*

Dr. Ashish Ku. Sinha *completed his MCA from Magadh University, Bodhgaya, India. He obtained his Ph.D. from the same university in 2015. He is working as Associate Professor in Jharkhand Rai University, Ranchi, India. He has authored various research papers.*

Dr. Mahua Banerjee *completed her B.Sc. from Calcutta University, MCA from IGNOU. She obtained her Ph.D. from ISM Dhanbad (IIT). She is working as Associate Professor in Xavier Institute of Social Services, Ranchi, India. She has over 20 publications in reputed National & International journals& conferences.*

1. INTRODUCTION

In current technology software pervades every sector of business. Even for the organizations which do not directly deals with the software, software has become the bottom line for the success of the business. For example Mobiles, Various electronic embedded devices like washing machine, refrigerator, car etc. need support of software. The universal business goals include improved, efficient and high productivity. Reusability has become main strategy for software development.

Generally it is found that few systems are unique but most of the products are the members of a product family with some different features with a core or common feature. In this scenario strategy of reuse makes sense. Reusability is an old trend. It started in 1960s with the reuse of subroutines, then in 1970s with modules, in 1980s object, in 1990s with components and 2000 onwards with services. This leads to the various development technologies that use the concept of reusability. SPLs and DSPLs are among these technologies. [1]

2. BACKGROUND

Since 2000 software product line (SPL) has emerged as an efficient development technology which is suitable for mass production and mass customization at the same time. SPLs have been used successfully in industry to build product families consisting of related products with common core and additional variable features. In a fast moving world the modern software demand more and more adaptive features, runtime or dynamic binding capabilities, post deployment activities, autonomous decision making capabilities, etc. Some of the systems like Software Ecosystems, Service Based Systems, Self-Adaptive Systems, Surveillance Systems, Robotics, etc. demand the above features. SPLs are unable to support runtime adaptations as they support static binding. Dynamic Software Product Lines (DSPLs) are one of the recent approaches that attempt to face the adaptation challenges of changing environment. Although DSPLs are proving an efficient approach to build Adaptive Software, its solution architecture is not very well defined and matured. It is used in various Service Oriented Systems like Robotics, Automated Houses & Hotels, Mobile Systems, Wireless Sensor Networks, etc. It can be used to develop business applications also to support unexpected dynamic changes.

3. SOFTWARE PRODUCT LINES (SPLs)

The methodology adopted to develop software with reusability is Software Product Line (SPL). A Software Product line is a new application of proven concept. It is an innovative, growing concept in software engineering. [1] Building software systems from the previously developed components saves costs and time of redundant work and improves the system and its maintainability. [2]

A software product line is 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 from a common set of core assets in a prescribed way. The approach uses a common set of core assets to modify, assemble, instantiate, or generate multiple products and is referred to as a product line. Such a product line approach involves building a product line as a product family. [3]

SPL methodology allows building a software family with less effort, time and cost with high quality. It satisfies individual customer’s needs. [4] It supports incremental development by taking a common core product and changing the additional features according to the requirements of various groups of users. In this methodology the core product remains same and only the features

The main limitation of Software Product Lines is its incapability to support runtime variability that is to bind and adopt the changes during runtime. It is not very efficient in changing the structural variability at runtime that is to select the variants dynamically. As it supports static binding, it is able to supports to make the changes and to select the variants at design time. It becomes difficult and cumbersome to make frequent changes at design time and develop it. This decreases the importance and benefits of SPLs.

Due to this limitation of SPLs a new methodology having the same commonality and variability concept but able to support runtime adaptation of the variability and binding the changes at runtime (dynamic binding) is required. One such emerging technology is DSPL.

3.1 Challenges

The domains of modern software systems are changing rapidly and tend to be extremely long lived. Some of the emerging domains are ubiquitous computing, service robotics, house automation or ambient intelligence, self-managed and autonomous systems etc. require a higher degree of adaptability from their software system due to their high complexities. During this period the requirements of customers can be changed dynamically hence the resource constraints will also vary and need to be changed at runtime. The SPLs building such types of systems will have to face these challenges. Hence the modern software has to evolve to meet the changing requirements. Another need of modern software is to meet runtime variability that is to support dynamic binding.

The main limitation of Software Product Lines is its incapability to support runtime variability that is to bind and adopt the changes during runtime. It is not very efficient in changing the structural variability at runtime that is to select the variants dynamically. As it supports static binding, it is able to supports to make the changes and to select the variants at design time. It becomes difficult and cumbersome to make frequent changes at design time and develop it. This decreases the importance and benefits of SPLs.

Due to this limitation of SPLs a new methodology having the same commonality and variability concept but able to support runtime adaptation of the variability and binding the changes at runtime (dynamic binding) is required. One such emerging technology is DSPL.

4. DYNAMIC SOFTWARE PRODUCT LINES (DSPLs)

Dynamic Software Product Line is emerging as an efficient methodology for the same that produce software capable of adapting the changing requirements and fluctuations. Since the development of runtime configuration assets is still innovative and not fully investigated in SPL area, there is an important need to support the dynamic properties of systems and post deployment capabilities as a new promising area in research and development area using DSPL methodology. [5]

Another problem is impossibility to foresee all the functionality or variability an SPL requires. In contrast with traditional SPLs, dynamic SPLs bind variation points at runtime, where software is launched to adapt to the current environment, as well as during operation to adapt to changes in the environment. Building a product line that dynamically adapts itself to changing requirements implies a deployment of the product configuration at runtime [6]. Such systems also require monitoring capabilities for detecting changes in the environment. As a response to these changes, the system adapts by triggering a change in its configuration, providing context-relevant services or meeting quality requirements. It means the system has runtime capabilities for flexible adaptation, reconfiguration and post deployment activities.

Dynamic software reconfiguration is concerned with changing the application configuration at runtime after it has been deployed. In terms of features, dynamism means that both configuration of features and their quality constraints vary at runtime [7]. Reconfigurations are also needed to implement feature availability and qualities. Dynamic addition, deletion or modification of product features is some examples of dynamic reconfiguration for SPLs. Other issues concerning DSPLs concern the commonality and variability analysis from the quality point of view. Identifying them at the software quality level is much more difficult than in the traditional case. Generally speaking, a DSPL may have many of the following properties [8]:

Dynamic variability: configuration and binding at runtime,

- binding changes several times during its lifetime,

- Variation points change during runtime: variation point addition (by extending one variation point),

- deals with unexpected changes(in some limited way),

- deals with changes by users, such as functional or quality requirements,

- context awareness and situation awareness,

- autonomic or self-adaptive properties

Interest in DSPLs is growing as more developers apply the SPL approach to dynamic systems. They represent an efficient way for modeling adaptive capabilities in software product lines. Adaptive capabilities mean the capabilities of the systems to deal with a continuously changing environment and emerging requirements that may be unknown at design time.

5. COMPARISON OF SPLs WITH DSPLs

Both SPLs and DSPLs are based on the concept of reusability as they deal with variability. SPLs manages static variability that is binding at design time where as DSPLs manages dynamic variability that is binding at runtime.

SPLs support reusability in two phases: [4]

- Domain Engineering

- Application Engineering

In Domain Engineering phase the SPL infrastructure is developed for the specific product family which basically includes the variability model, reference architecture and other product line assets.

In Application Engineering phase specific product is built using a variability model per customer's needs to give the final product.

Variability features of SPLs:

- The goal of variability is to satisfy individual customer's need.
- Binding time of variability is pre runtime mostly, sometimes runtime.
- Variants are decided and designed by Application Engineer.
- Variants are selected by reuse of shared assets conforming a variability model.
- Life time of variability model is pre runtime.

On the other hand DSPLs builds the product in two layers:

-Adaptation Logic

-Application Logic

Adaptation is change of system properties and behavior at runtime in response to dynamically

Varying user needs and resource constraints.
Using Adaptation logic:

- the context of the system is monitored internally and externally.
- the context for conditions is analysed that where adaptation is needed.
- the response plan following adaptation policies is decided.
- the decided plan is carried out by dynamic addition, deletion, or modification of product features, or introducing changes in the architectural structure of the system.
- monitoring of the system's internal state is done and needed interface is provided for adaption manager.
- provides the needed interface for adaptation manager for runtime adaptation.
- perform the adaptation2

Using Application Logic specific product is built using a variability model as per changing environment.

Variability features of DSPLs:

- The goal of variability is to maintain service in changing environment.
- Binding time of variability is runtime mostly, sometimes pre runtime.
- Variants are decided and designed by Application Engineer or user or application itself.
- Variants are selected by use of SPL assets at runtime.
- Life time of variability model is pre runtime / runtime.

Properties of DSPLs

Runtime reconfiguration

- allows for dynamic variability: configuration and binding at runtime.
- changes binding several times during its lifetime.
- introduces variation points [which] change during runtime: variation point addition (by extending one variation point).
- deals with unexpected changes (in some limited way).
- deals with changes by users, such as functional or quality requirements.

Self-management *(Optional)*:

- context awareness
- autonomic or self-adaptive properties
- automatic decision making

6. VARIOUS TOOLS FOR DEVELOPMENT OF DSPLs.

As SPLs and DSPLs are becoming popular and handy in various areas, a number of development tools are also developed and used. Some of them are Feature Oriented Programming (FOP), Aspect Oriented Programming (AOP), Delta Oriented Programming (DOP) etc.

6.1 Feature-Oriented Programming primarily focuses on the features of a system, instead of the objects that comprise it. [9]

Feature–Oriented programming is a programming methodology which creates a large number of minor features to remove redundancy and improving efficiency. These minor features are then linked together in the functions/methods/procedures which take care of the core functionality.

It is a modular methodology which promotes small, tightly focused, but still general purpose, functions, in place of large, specific, functions. Most functions should be seen as a single specific tool to complete a single general task. Although the program itself should have a specific goal, modularization supports to recognize more specific tasks within the functions. For example a function for sorting or searching.

Feature-oriented programming would create greater consistency in programming, since the programs would only depend on code being written once, and then referred to (though the code might contain features to change it behavior under certain conditions). Another benefit is that most functions would be placed centrally, which would mean, unlike Object-Oriented Programming which spread their features across several objects – even though the chain of program calls would always be the same, the functions would be readily available to most of the code. [10]

6.2 Aspect Oriented Programming is a programming methodology which enables a clear separation of concerns in software applications. [11]

In software development a "concern" can be understood as"*a specific requirement that must be addressed in order to satisfy the overall system goal"*. [12]

That is, *concerns* can be subdivided in two subgroups:

Core concerns Specify the main functionality of a software module, such as account withdrawing or a button click behavior. We can define this as the *business logic* of our implementation.

Crosscutting concerns specify the peripheral requirements that cross multiple modules, such as transaction, threading or authentication.

The idea is to be able to model, or visualize the goal system as a combination of core and crosscutting concerns in a multi-dimensional space implementation rather than a mixture of all of them along a one-dimensional space implementation. Because concrete software are one-dimension implementation, all concerns are then mapped into it.
An interesting research question for aspect-oriented programming (AOP) [13] is how far dynamic AOP is capable as a variability mechanism for dynamic SPLs.
For example, dynamic aspects have been used to implement business rules. Although dynamic AOP solutions provide a flexible variability mechanism, they do not provide appropriate support for declaring, detecting, and resolving dynamic interactions. Most dynamic AO solutions only provide support for defining the precedence of aspects i.e., defining the execution order of their advice. But these precedence relations are inappropriate for expressing exclusions, i.e., one cannot declare that one aspect does not allow another aspect to be present. Furthermore, dynamic dependency relations between the features implemented by aspects cannot declare that one aspect needs another aspect to work correctly. The works in support static declaration of exclusions and dependencies between aspects but do not address dynamic interactions.

The limitation of FOP is that it implements software product lines by composition of feature modules. It relies on the principles of stepwise development. Feature modules are intended to refer to exactly one product feature and can only extend existing implementations. Even AOP is basically suitable to handle crosscutting & tangling problems.

6.3 Delta Oriented Programming: To overcome these limitations another programming approach to develop Dynamic Software Product Lines is Delta Oriented Programming which can be further extended as Dynamic Delta Oriented Programming. [14]
Delta-oriented programming (DOP) is a novel approach for implementing software product lines. Delta-oriented programming offers an expressive and flexible programming meta-language for specifying a set of products.
Its aim is to relax the restrictions of currently established SPL description formalisms such as feature-oriented programming (FOP) by adding the explicit possibility to remove parts of a program. In delta-oriented programming, an SPL is implemented as a core module together with a set of delta modules. The core module contains a complete product implementation for some valid feature configuration, which can be developed by conventional single-application engineering techniques. Delta modules specify changes to be applied to the core module in order to implement other products. [14]
DOP handles changes to the set of selected features by triggering the application of deltas; this results in a program transformation, as deltas can add, modify and also remove code. The lower level support required for such dynamic transformations is available and some of this has crept into features relevant for SPL engineering. [15]
DOP can be extended with the capability to switch the implemented product configuration at runtime and present a formal foundation for dynamic DOP. A dynamic delta-oriented SPL is a delta-oriented SPL with a dynamic reconfiguration graph that specifies how to switch between different feature configurations [16]. Dynamic DOP also supports (unanticipated) software evolution such that at runtime, the product line declaration, the code base and the dynamic reconfiguration graph can be changed in any (unanticipated) way that preserves the currently running product. [16-17].
To implement a DSPL using Dynamic Delta Oriented Programming there are various plug-in integrated to Eclipse IDE. Two such plug-INS is discussed below:

- DeltaJ
- DeltaJava
-

6.3.1 DeltaJ

DeltaJ is available as an Eclipse plug-in. It is based on the Xtext Framework therefore it provides good integration into Eclipse.
DeltaJ 1.5 with Java 1.5 provides full access of the Java 1.5 syntax and corresponding extensions and improvements of the delta-orientated operations. This prototype is available in **https://www.isf.cs.tu-bs.de/cms/research/deltas/downloads/plug-in/.**
DELTAJ 1.5 is a perfect language which supports DOP for complete JAVA 1.5 [18].

6.3.2 DeltaJava with DeltaEcore

DeltaJava is a commanding delta-oriented programming language [58]. It is designed to produce Software Product Lines easily and efficiently inside the Java environment. DeltaJava was made with EMFText - "an Eclipse plug-in which lets the prorammer to outline text syntax for languages designated by an Ecore metamodel" [18,19].

Its base is an EMFText Execution of Java, named as JaMoPP (Java Model Printer & Parser), on top of which DeltaJava defines its actions and components by enfolding JaMoPP's Java code within them [18]. In DeltaJava, JaMoPP is also used to define a DeltaEcore dialect which conveniently provides a solid procedure for generation of various products or variants. When DeltaJava delta modules are saved, DeltaJava performs a model transformation to a DeltaEcore delta module using the dialect created for JaMoPP. These DeltaEcore delta modules are then used for the variant generation.

CONCLUSION

The concept of reusability is there from beginning from subroutines to modules to objects to components to services and this lead to the Software Product Line development technology. SPLs is based on design of a common core product, then deciding the variants and designing the variability models as per customer's needs. In changing world where software needs to show random behavior at runtime and needs self adaptive features, SPLs are unable to cop up as they are static in nature. DSPLs are used to solve these problems. DSPLs utilize part of SPLs infrastructure to adapt at runtime. It uses adaptation logic along with application logic. A thorough study of capabilities of DSPLs, various tools to develop them and a mature infrastructure model which is in fancy in current scenario will lead DSPL technology to bring a revolution in the development area of software for all types of applications including business applications like production, inventory management, sales order management etc.

FUTURE SCOPE

Research shows that application domain of DSPLs mostly included smart and automatic homes, robotics, mobile software, web application, service based systems etc. Its use in business processing areas is infancy. In near future, the Dynamic Software Product Lines can be and will be used in almost all types of software systems like production, marketing, financial, inventory etc. and will be adopted by almost all leading industries. An efficient DSPL can designed & developed for sales processing and inventory management systems for different types of analysis like ABC, VED analysis etc. which include random requirements, dynamic selection of quantities, self-adaptation etc. There will be a huge requirement for the tools for the design & development of DSPLs. Delta-oriented programming can be a useful tool to develop DSPLs.

REFERENCES

[1] Cesar Andres, Carlos Camacho and Luis Llana, “ A Formal Framework for Software Product Lines”, Publication: Information and Software Technology, November 2013, http://doi.org/10.1016/j.infsof.2013.05.005.

[2] Shamim Ripon, Sk. Jahir Hossain and Touhid Bhuiyan, “MANAGING AND ANALYSING SOFTWARE PRODUCT LINE REQUIREMENTS”, International Journal of Software Engineering & Applications (IJSEA), Vol.4, No.5, September 2013, DOI : 10.5121/ijsea.2013.4505 63.

[3] A. Chaudhary, B.K. Verma and J.L. Raheja, “Product Line Development Architectural Model”, In proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, China, July,2010, pp.749-753.

[4] Mike Hinchey, Sooyong Park and Klaus Schmid, “Building Dynamic Software Product Lines”, In:IEEE Computer 45.10 (2012), pp. 22-26. DOI:10.1109/MC.2012.332.

[5] Paul Istoan, Gregory Nain, Gilles Perrouin and Jean-Marc Jean-Marc Jezequel, “Dynamic Software Product Lines for Service-Based Systems”, In proceedings of the Ninth IEEE International Conference on Computer Science and Information Technology-Volume 02, October 2009, pp.193-198. http://doi.org/10.1109/CIT.2009.54.

[6] H. Gomaa and M. Hussein, “Dynamic software reconfiguration in software product families”, Conference: Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena, Italy, November 2003, Revised papers. DOI:10.1007/978-3-540-24667-1_33.

[7] J. Andersson and J. Bosch, “Development and use of dynamic product line architectures”, Software Engineering. IEE Proceedings-, vol. 152, no. 1, pp. 15–28, Feb. 2005. DOI:10.1049/ip-sen:20041007, Corpus ID:16166712.

[8] Mahua Banerjee and Reeta Kumari, “Dynamic Software Product Line: An Approach to Dynamic Binding”, In. Proceedings of the 3rd International Conference on Recent Advances in Information Technology | RAIT 2016, 978-1-4799-8578-4/16/$31.00©2016IEEE, organized by ISM, Dhanbad, India

[9] S. Apel, C. Lengauer, B. Moller and C. Kastner, “An Algebra for Features and Feature Composition”, In Proceedings of the International Conference on Algebraic Methodology and Software Technology (AMAST 2008), vol. 5140 of LNCS, pp. 36–50, Springer-Verlag, 2008.

[10] S. Apel, “The Role of Features and Aspects in Software Development”, PhD thesis, School of Computer Science, University of Magdeburg, 2007.

[11] T. Elrad, R. E. Filman and A. Bader, “Aspect-Oriented Programming: Introduction”, Communications of the ACM (CACM), vol. 44, no. 10, pp. 29–32, 2001, DOI:10.1145/383845.383853.

[12] Erik Hilsdale and Jim Hugunin, “Advice Weaving in AspectJ”, Conference: Proceedings of the 3rd Internatinal Conference on Aspect-Oriented Software Development, AOSD 2004, Lancaster, UK, March 2004, DOI:10.1145/976270.976276.

[13] W. Abdelmoez, Hatem Khater and Noha El-shoafy, “Comparing Maintainability Evolution of Object-Oriented and Aspect-Oriented Software Product Lines”, The 8th International Conference on INFOrmatics and Systems (INFOS2012) – 14-16 May.

[14] Daniel Bruns, Vladimir Klebanov and Ina Schaefer, “Verification of Software Product Lines with Delta-oriented Slicing”, FoVeOOS’10: Proceedings of the 2010 International Conference on Formal Verification of Object-Oriented Software, 2010, pp.61-75, DOI:10.1007/978-3-642-18070-5_5.

[15] Nelly Bencomo, S. O. Hallsteinsen and E. S. de Almeida, “A View of the Dynamic Software Product Line Landscape”, published by the IEEE Computer Society, 2012, DOI: 10.1109/MC.2012.292

[16] Ferruccio Damiani and Ina Schaefer, “Dynamic Delta-Oriented Programming”, SPL’11, Proceedings of the 15th International Software Product Line Conference, Vol. 2, Article no. 34, pp. 1-8, 2011, DOI:10.1145/2019136.2019175.

[17] Ferruccio Damiani, Luca Padovani and Ina Schaefer, ”A Formal Foundation for Dynamic Delta-Oriented Programming”, ACM Transactions on Programming Languages and Systems, Vol. 0, No. 0, pp.1-2, 2013.

[18] https://www.deltajava.org/

[19] https://www.isf.cs.tu-bs.de/cms/tools/deltajava/