Architecture Conformance Analysis Approach within the Context of Multiple Product Line Engineering

Bedir Tekinerdogan
Department of Computer Engineering
Bilkent University, Ankara, Turkey
e-mail: bedir@cs.bilkent.edu.tr

Evren Çilden, Özgü Özköse Erdoğan, Onur Aktuğ
Aselsan, PO Box. 1, 06172 Yenimahalle,
Ankara, Turkey
e-mail: {ecilden | ozkose | oaktug}@aselsan.com.tr

***Abstract*— One of the important concerns in software product line engineering is the conformance of the application architecture to the product line architecture. Consistency with the product line architecture is important to ensure that the business rules and constraints that are defined for the entire product family are not violated. Usually, the conformance checking to the product line architecture is a manual and tedious process. A popular approach for ensuring architecture conformance is reflexion modeling which has been primarily used to check the consistency between the architecture and the code. In this paper we present an approach for product line conformance analysis based on reflexion modeling. We consider conformance analysis in product line engineering and extend our discussion to multiple product line engineering. Our study shows several important challenges regarding reflexion modeling within the context of product line engineering.**

Keywords—architecture conformance analysis, design structure matrix, architecture views

I. INTRODUCTION

Unlike earlier software reuse approaches, software product line engineering (SPLE) aims to provide pro-active, pre-planned reuse at a large granularity (domain and product level) to develop applications from a core asset base. In general the SPLE process consists of the two basic activities *domain engineering* and *application engineering*. The domain engineering process is responsible for establishing the reusable platform and thus for defining the commonality and the variability of the product line. In the application engineering process the applications of the product line are built by reusing the artefacts of the reusable platform. An important artefact in SPLE is the *product line architecture* which represents the common architecture for the products in the selected product line. Based on the product line architecture and the application requirements the *application architecture* is developed. Hereby, it is important that the application architecture is consistent with the product line architecture. Unfortunately, both the product line architecture and the application architecture can change due to various reasons such as bug fixes or new requirements. As such, the product line architecture and application architecture might evolve separately from each other leading soon to inconsistencies between both architectures. For very large systems the scope of the product line can extend even further and the product can be built using sub-products from multiple product lines. In that case the system will include a system-of-systems architecture and conformance analysis need to be considered from a further broader perspective.

A related problem to the inconsistencies of the corresponding architectures in the SPLE context, is usually defined as *architectural drift* problem. This problem defines the general case of the discrepancy between the architecture description and the resulting implementation. To detect the inconsistencies among the code and the architecture, various architecture conformance analysis approaches have been proposed. A popular conformance analysis is reflexion modeling in which the architecture design is compared with the derived abstract model of the code. A reflexion model highlights the differences between the code and the architecture and as such defines the extent of the architectural drift problem.

In this paper we provide an approach for checking the conformance within the multiple product line engineering context. We discuss the new challenges and the required conformance analysis steps at different levels and pave the road for further research.

The remainder of the paper is organized as follows. In section 2 we provide the background on multiple product line engineering and reflexion modeling. Section 3 presents the industrial case study that we will use to illustrate the problem statement and the approach. Section 4 presents the overall reflexion modeling approach. Section 5 discusses the tool that implements the approach. Finally, section 6 presents the conclusion.

II. BACKGROUND

A. Reflexion Modeling

Architecture consistency implies that the architecture design elements can be mapped to the implementation elements. In case the relationships between the architecture and implementation do not correspond then these are called *architectural violations*. If the relations that are present in the architecture are also found in the implementation then this is *convergent relation*. In case the architecture relation is not present in the implementation then this is called an *absence relation*. Absence relations occur of course during the initial development of the system in which the architecture is defined but the implementation is not ready yet. As such, in the early

978-1-4799-3149-1/14 $31.00 © 2014 IEEE
DOI 10.1109/ASWEC.2014.23

phases of the development these absence relations might be a lesser concern. Finally, if the implementation includes relation that is not present in the architecture, then this is called *divergence relation*. Architectural violations are due to absence or divergence relations.

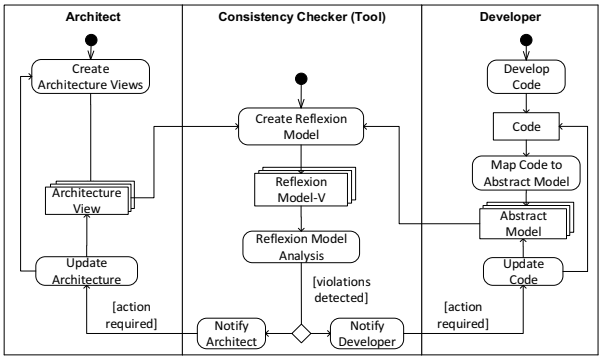

Fig. 1. Activity diagram showing the steps in reflexion modeling for architecture-to-code conformance

A successful design recovery technique that is used for architecture consistency checking is the *reflexion modeling* approach as proposed by Murphy et al. [4]. In principle, a reflexion model allows a software developer to view the structure of a system's source through a chosen high-level (often architectural) view. To check the consistency between the architecture model and the code, an abstract model of the code is derived. The two models are then compared to each other with respect to earlier defined mapping rules between the code and the implementation. The results of the comparison are presented to the user through a *Reflexion Model*. The reflexion model explicitly represents the convergence (solid edge), the divergence (dashed edge), and the absence relation (dotted edge). By analyzing the reflexion model, the architecture, the code or the mapping rules can be altered. Usually architecture conformance analysis approaches that apply reflexion modeling include tools for modeling the architecture, modeling the mappings, deriving the abstract model from the source code, the consistency analysis checker, and the generator of the resulting reflexion model.

B. Multiple Product Line Engineering

When reuse is an important concern a system can be built based on product line approach. The relation between these concepts is illustrated in Fig. 2. For very large systems the scope of the product line can extend further and the product can be built using sub-products from multiple product lines. The notion of multiple product lines has been addressed earlier by different authors [6][11][13][15]. In this context, the notion of *multiple product lines*, *nested product lines* or *product lines of product lines* have been used to denote the same concept. In [15] the authors define multiple product lines as "a set of interacting and interdependent SPLs".

In principle we can consider the composition of product lines as the application of a composite pattern as shown in Fig. 2. A large and complex *System* could be build using non-Product Line Units and/or using a *Product Line*. *Product Line* could be either a flat *Software Product Line (PL)* or a *Composite Product Line (CPL)*. *CPL* itself could contain other product lines and likewise the product line can be built in a nested manner. Alternatively, the CPL could include only flat product lines leading to a multiple product line consisting of independent product lines.

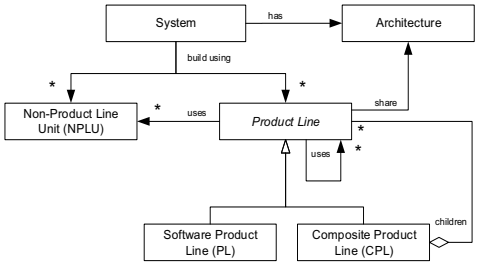

Fig. 2. Conceptual Model for System Development using Product Lines, Multiple Product Lines and non-Product Line Units

An example of a multiple product line architecture is shown in Fig. 3 which is designed within the context of Aselsan REHIS, a leading high technology company in defense systems development in Turkey.

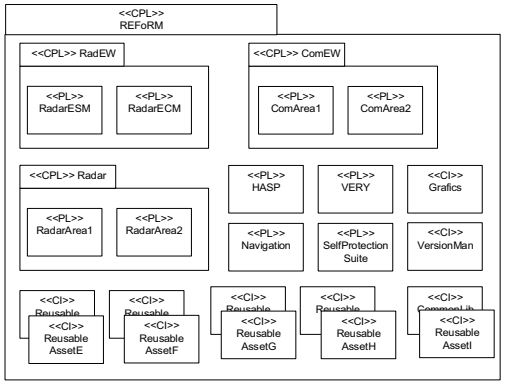

Fig. 3. Multiple Product Line Architecture

Due to the large scope of the required products a multiple product line engineering approach is adopted in which products are composed from different but related product lines. Fig. 3 represents an example of the product line decomposition view for the given case study that is based on the adapted decomposition viewpoint [2]. The stereotypes <<CPL>> indicate a composite product line, <<PL>> a flat product line and <<CI>> a configuration item. In the given example, the system has been defined as one composite product line (CPL) that contains three separate CPLs (*RadEW, ComEW,* and *Radar*), four PLs (*HASP*, *VERY*, *Navigation*, and *SelfProtectionSuite*), and 12 CIs (libraries). The CPLs each consists of two PLs. An important concern in the development of products is the consistency of the products on this overall system-of-systems architecture, as well as the consistency between the product line architecture and the application architecture, and the consistency between application architecture and its corresponding code. We elaborate on these in the following section

III. REFLEXION MODELING APPROACH IN MPL ARCHITECTURE

Obviously, the scope of the architecture drift problem and the required architecture conformance analysis techniques need to be considered from a broader scope. We distinguish two basic categories of conformance analysis techniques with respect to the considered scope: (1) *conformance analysis from single system perspective* (2) *conformance analysis from system-of-systems perspective.* We discuss these two approaches in the following two subsections.

A. Single System Conformance Checking

Fig. 4 shows conformance analysis from a single system perspective. In this case the product is developed from a single product line. The artefacts involved are *Product Line Architecture, Application Architecture* and *Code.* Hereby we consider the following levels of architecture conformance analysis:

- *Application Architecture-Code Conformance Analysis*

This level considers the ACA of a single application architecture with the code. This is the general view of ACA as shown in the approach of Fig. 1. Here, it is usually expected that the consistency relation is bidirectional, that is, the code should conform to the architecture and vice versa. This ACA can be done using conventional approaches in the literature.

- *Product Line Architecture-Application Architecture Conformance Analysis*

This level considers the consistency between the product line architecture and application architecture for a separate sub-product. The application architecture is derived from the product line architecture. Inconsistencies might occur in the initial design or the evolution of the product line architecture and the application architecture. Usually, the *conforms to* relation is from the application architecture to the product line architecture. However, one might also decide to adapt the product line architecture to align it for the different application architectures. The reflexion modeling approach for this case is shown in Fig. 5.

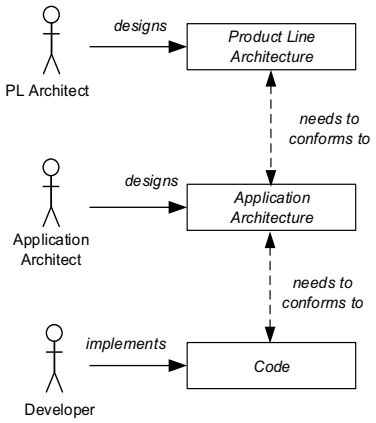

Fig. 4. Conformance analysis from single system perspective

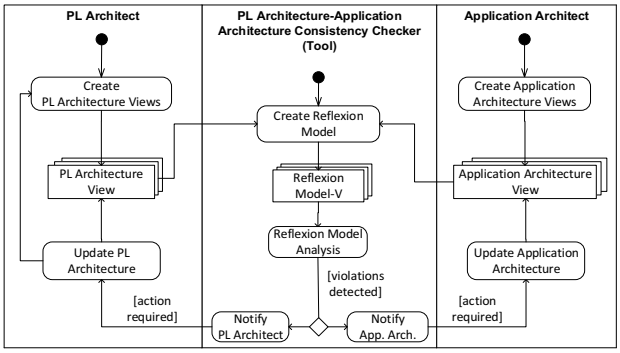

Fig. 5. Activity diagram showing the steps in reflexion modeling for product line architecture-to-application architecture conformance analysis

Note that in this case the creation of reflexion model will be different than in the case of conformance checking of the architecture with the code. This is because in general the application architecture will include so-called *deltas,* i.e. unique entities that are required for a particular application but which are not part of the product line architecture. To make a difference of deltas with real inconsistencies the reflexion model should include separate notations for these.

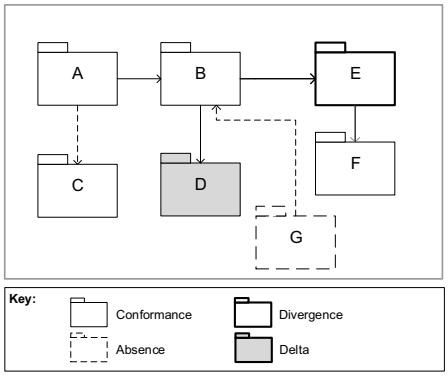

Fig. 6. Reflexion Uses View

Fig. 6 shows for example a possible reflexion model as a result of the comparison of the product line architecture uses view with the application architecture view. The model shows in addition to convergence, absence and divergence entities also delta entities. From the figure we can observe that the uses relation B to E (bold) is missing in the application architecture although it was defined in the product line architecture uses view (absence). The uses relations A to C, and G to B (dashed) is added to the implementation although this was not present in the architecture (divergence). The module D is denoted as a delta module that is added to the application and which is allowed by the product line architecture. This implies that for conformance checking in the product line architecture it is also important to define the possible violation or allowance rules for the deltas.

B. System-of-Systems Conformance Checking

Fig. 7 shows the conformance analysis levels from the system-of-systems perspective that typically applies to the multiple product line engineering context. Hereby, the product is developed from sub-products that are developed in separate product lines. The artefacts involved are *Multiple Product Line Architecture, System-of-Systems Architecture and System-of-Systems Code.*

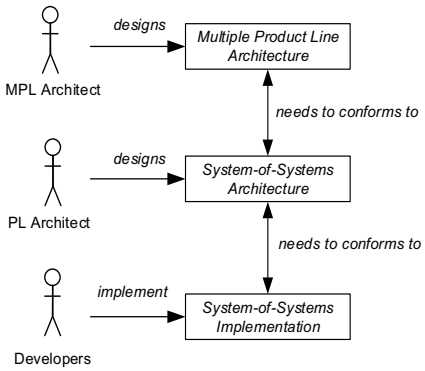

Fig. 7. Conformance analysis from system-of-systems perspective

Here, we distinguish among the following levels of conformance analysis:

- *System-of-Systems Architecture - System-of-Systems Implementation Conformance Analysis*

This ACA considers the conformance between the system-of-systems architecture and its implementation. In fact the conformance analysis will be typically similar as in architecture to code conformance analysis.

- *Multiple Product Line Architecture - System-of-Systems Conformance Analysis*

This ACA considers the consistency of the multiple product line architecture with the System-of-Systems Architecture. The process is similar as defined in Fig. 5. The consistency analysis in this case checks whether the final product adheres to the design and configuration constraints as defined at the MPL level.

IV. CONCLUSION

In this paper we have described an approach for reflexion modeling within the context of multiple product line engineering. The need for this was derived from a real industrial context in which consistency of the multiple product line architecture, the product line architecture, and the code is important. In our analysis we distinguish between the system level and system-of-systems level of reflexion modeling. Based on this we could define four different levels of architecture conformance analysis. The conformance analysis for product line architecture and application architecture imposes new requirements on the generation of the reflexion model as well as the violation rules that define the inconsistencies. In our future work we will detail each approach and provide an integrated tool for coping with these four different levels of architecture conformance analysis. Further we will also look at the implications for architecture conformance analysis techniques other than reflexion modeling.

REFERENCES

[1] M. A. Babar, L. Zhu, and R. Jeffery. A framework for classifying and comparing software architecture evaluation methods. aswec, 00:309, 2004.

[2] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley, 2010.

[3] J. Knodel, D. Popescu. A comparison of static architecture compliance checking approaches. Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture, Mumbai, India, 2007; 12.

[4] G. Murphy, D. Notkin, K. Sullivan. Software reflexion models: Bridging the gap between design and implementation. IEEE Transactions on Software Engineering 2001; 27(4):364–380.

[5] J. Rosik, A. Le Gear, J. Buckley, M. A. Babar, and D. Connolly, "Assessing architectural drift in commercial software development: a case study," Software: Practice and Experience, 41:1, pp. 63–86, 2011.

[6] M. Aoyama, K. Watanabe, Y. Nishio, and Y. Moriwaki: "Embracing Requirements Variety for e-Governments Based on Multiple Product-Lines Frameworks". Proceedings of the 11th IEEE International Requirements Engineering Conference, 2003.

[7] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. *Documenting Software Architectures: Views and Beyond.* Second Edition, Addison-Wesley: Reading, MA, 2011.

[8] P. Clements, L. Northrop. *Software Product Lines: Practices and Patterns*. Boston, MA:Addison-Wesley, 2002.

[9] http://en.wikipedia.org/wiki/Electronic_warfare_support_measures, accessed January 2011.

[10] ISO/IEC, ISO/IEC 42010 Systems and Software Engineering -- Recommended Practice For Architectural Description Of Software-Intensive Systems, 2007.

[11] C.W. Krueger, *New Methods in Software Product Line Development*. BigLever Software, Austin, TX; in Proc. of 10th Software Product Line Conference, 2006

[12] K. Lee, L.C. Kang. Feature Dependency Analysis for Product Line Component Design, in: J. Bosch and C. Krueger (Eds.): ICSR 2004, Springer LNCS 3107, pp. 69–85, 2004.

[13] R. van Ommering, Widening the Scope of Software Product Lines – from Variation to Composition, proceeding of the Software Product Lines 2nd International Conference, SPLC 2, San Diego, CA, Aug 2002, Springer-Verlag LNCS 2379, p 328.

[14] K. Pohl, G. Böckle, F. van der Linden. Software Product Line Engineering – Foundations, Principles, and Techniques, Springer, 2005.

[15] M. Rosenmüller and N. Siegmund. Automating the Configuration of Multi Software Product Lines. In Proceedings of the International Workshop on Variability Modelling of Software-intensive Systems (VaMoS). Linz, Austria, Jan. 2010.

[16] K. Schmid, M. Verlage. *The Economic Impact of Product Line Adoption and Evolution*. IEEE Software, Vol. 19, No. 4, July/August 2002, 50-57.