Guaranteeing Correct Evolution of Software Product Lines: Setting Up the Problem

Maurice H. ter Beek[1], Henry Muccini[2], and Patrizio Pelliccione[2]

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa, Italy
maurice.terbeek@isti.cnr.it
[2] Università degli studi dell'Aquila, Dipartimento di Informatica, L'Aquila, Italy
{henry.muccini,patrizio.pelliccione}@univaq.it

Abstract. The research question that we posed ourselves and which has led to this paper is: *how can we guarantee the correct functioning of products of an SPL when core components evolve?* This exploratory paper merely proposes an overview of a novel approach that, by extending and adapting assume-guarantee reasoning to evolving SPLs, guarantees the resilience against changes in the environment of products of an SPL. The idea is to selectively model check and test assume-guarantee properties on those SPL components affected by the changes.

1 Introduction

Product Line Engineering (PLE) aims to develop product families using a common platform and mass customization. Software Product Lines (SPLs) are part of an SPLE approach to develop, in a cost effective way, software-intensive products and systems that share an overall reference model of a product family [10]. Variety is achieved by identifying variation points as places in the Product Line Architecture (PLA) where specific decisions are reduced to the choice among several *features*, but the feature to be chosen for a particular product variant is left open (like optional, mandatory or alternative features). Variability management is the key aspect differentiating SPLE from 'conventional' software engineering.

Several approaches aim to verify correctness of SPLs (w.r.t. constraints and properties) and the conformance of products w.r.t. a family's variability [1, 2, 3, 4, 5, 6, 11]. SPLs however are subject to evolution, and their complexity makes it difficult to ensure the resilience of each product, i.e., the ability of the system to persistently perform in a trustworthy way even when facing changes. Apart from some work on regression testing, we are not aware of any existing approach on guaranteeing the correctness of products whose SPL or PLA is subject to evolution.

In this paper, we propose a step towards a solution to this problem by reusing, adapting, and combining currently available state-of-the-art techniques. More specifically, we intend to use assume-guarantee reasoning, which represents the environment as a set of properties that should be satisfied for it to interact correctly with the component. These properties are called *assumptions*, intended

E.A. Troubitsyna (Ed.): SERENE 2011, LNCS 6968, pp. 100–105, 2011.

to be the assumptions that the component makes on the environment. If these assumptions are satisfied by the environment, then components behaving in this environment will typically satisfy other properties, called *guarantees*. By appropriately combining the set of assume and guarantee properties, it is possible to prove the correctness of an entire system without actually constructing it.

Our idea is then to annotate components of a PLA with pairs of assumption and guarantee properties, considering a component's environment as the composition of the remaining components, thus enabling assume-guarantee compositional verification to deal with evolution. The idea underlying *compositional verification* is to decompose a system specification into properties that describe the behavior of a system's subset, to check these properties locally, and to deduce from the local checks that the complete system satisfies the overall specification. The main advantage is that one never has to compose all subsystems, thus avoiding the state explosion problem. On the downside, however, checking local properties over subsystems in general does not imply the correctness of the entire system, due to the existence of mutual dependencies among components. More precisely, a single component cannot be considered in isolation, but as behaving and interacting with its environment (i.e., the rest of the system).

In the future, we moreover intend to adapt the assume-guarantee testing approach of [9] to evolving PLAs, to complement architecture-level verification with code-level testing. This would allow one to guarantee the resilience of products of an evolving SPL by selectively model checking and testing affected components.

2 Problem Setting

Figure 1 shows the context of our work. During *domain engineering*, the whole SPL is taken into account: commonality and variability of the software family are analyzed in order to identify both the common components and the variation points. Domain assets are produced during the domain engineering phase to describe the requirements, architecture and tests of the entire family of applications. This current work focusses on architectural assets, thus focussing on the Product Line Architecture (PLA) of the SPL.

During the *application engineering* phase, instead, a decision process (more or less explicit) is then typically employed to decide which optional, mandatory or alternative features need to be selected in order to produce a certain software system from the product line. The application level product reflects development constraints and decisions made during the decision process, and its architecture is denoted as a Product Architecture (PA).

In this context, we consider the problem of guaranteeing the correct evolution of the SPL upon changes in components of the PLA. Figure 2 illustrates our problem setting by means of the following model problem,[1] which is due to Paul Clements et al. at the SEI:

[1] A model problem has been defined as a problem that, if solved, would result in a significant decrease in project resources devoted to testing and analysis and/or a significant increase in system quality given an expenditure level.

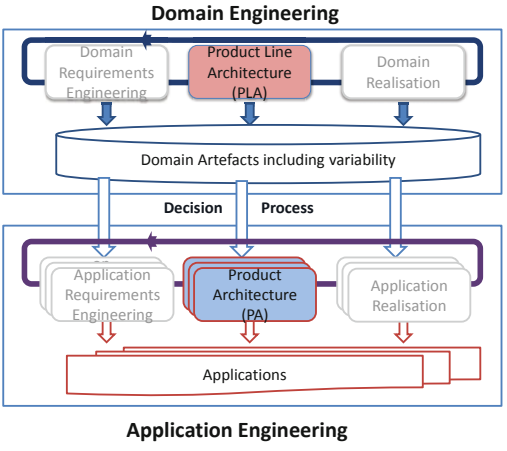

Fig. 1. Software Product Line

> *"I run the same software in different kinds of helicopters. When the software in a helicopter powers up, it checks a hardware register to see what kind of helicopter it is and starts behaving appropriately for that kind of helicopter. When I make a change to the software, I would like to flight test it only on one helicopter, and prove or (more likely, assert with high confidence) that it will run correctly on the other helicopters. I know I can't achieve this for all changes, but I would like to do it where possible."*

In the context of SPLE, this model problem can be re-phrased as follows: assuming various products (helicopters) have been derived from an SPL — which have moreover been formally certified — what can be concluded for new products obtained from the SPL by modifying one or more core components?

We assume that all products of the product line must be guaranteed to conform to a specific standard. Furthermore, we assume that there is a policy according to which any change to a core component requires all products containing that core component to be rebuilt. The question is whether it is necessary to re-validate all the products of the SPL or whether a subset can suffice. For instance, in the aforementioned model problem, when we change the software of the helicopter line, such as installing a new kind of radio across the fleet, we would like to flight-test it only on one helicopter, and prove or — more likely — assert with high confidence, that it will run correctly on the other helicopters. While it is impossible to achieve this for all kind of changes, we would like to pinpoint the conditions under which this is feasible.

Based on this problem, our research question becomes: *After a component C of a product is modified and the resulting product is guaranteed correct, what can be guaranteed for the other products of the product family that also contain C?* Different scenarios can be distinguished, and in this paper we sketch a solution for modifications resulting from changing, adding or removing such components.

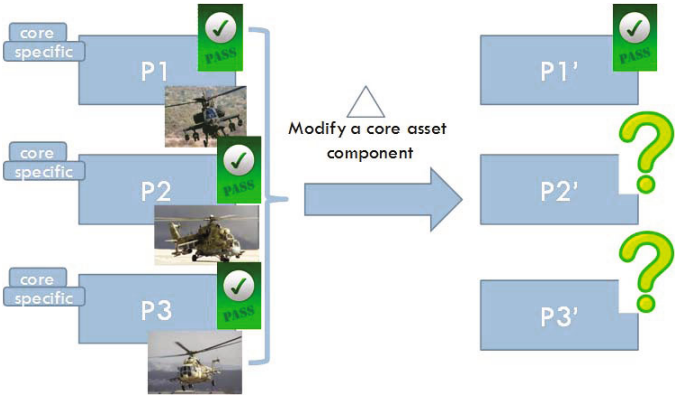

Fig. 2. Helicopter problem

3 Proposed Solution

While much work has been proposed for analyzing and testing SPLs, we are not aware of any approach for guaranteeing their products' correctness upon evolving components. Techniques for the formal verification of SPLs require a complete re-run of the verification after evolution, whereas regression testing of SPLs is not able to provide many guarantees for success. Finally, while compositional verification techniques are well suited to be applied to evolving systems, we know of no approach that is directly applicable to SPLs.

Figure 3 illustrates our idea of formally verifying that modifications to a component that is part of a PLA do not have unforeseen consequences. Each component is enriched with assume and guarantee properties, by properly calculating assumptions characterizing exactly those environments in which the component satisfies its required properties. In [8], this is done by means of the automatic generation of assumptions implemented in the LTS Analyser. We intend to adapt and extend this approach to consider evolving product line architectures.

When the PLA of the product family of interest is specified, assume and guarantee properties are contextualized: given a component C with its assume-guarantee pair, denoted by $C(a, g)$, its environment becomes the subarchitecture connected with C. What is challenging in PLA-based assume-guarantee reasoning is that the reasoning is performed on the PLA, including all variation points, rather than on each single concrete product, which means that the assumptions need to be calculated considering the variability management of components. In other words, once a component evolves this modification will potentially impact several PAs, namely each PA containing the modified component.

For instance, when a component B evolves in a component B' (see Fig. 3), the assumption and guarantee pairs of B and B' must be checked:

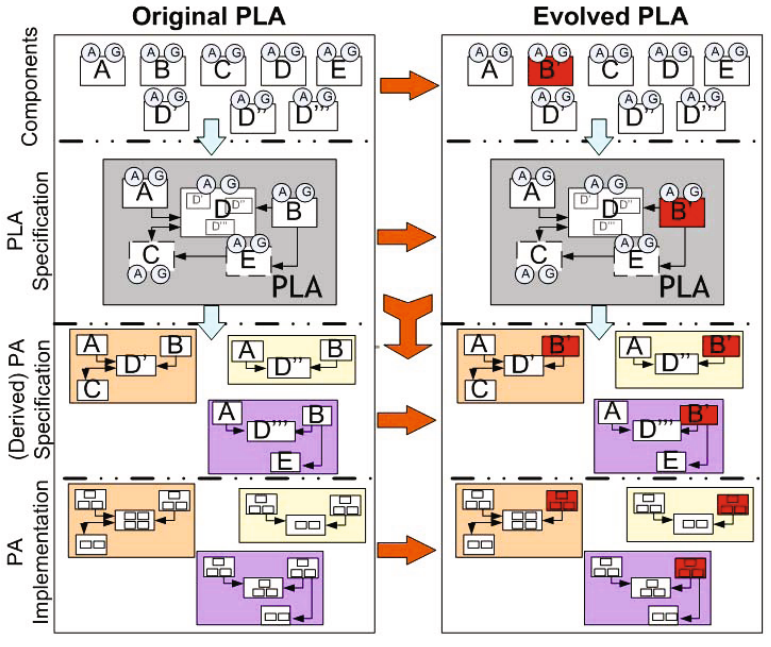

Fig. 3. The original SPL (left) and the evolved one (right)

– If $B'(a, g)$ exactly matches $B(a, g)$, then B' can safely substitute B.

– If $B'(a, g)$ is less restrictive than $B(a, g)$, in the sense that the assumption of B' is less restrictive than that of B, then it is possible to restrict the behavior of B' to that of B by means of suitable wrappers, thus forcing B' to behave as requested by the assumption of B.

– If B' is not able to behave as expected, according to $B(a, g)$, then assume-guarantee reasoning can be used to understand the effect of the evolution. In fact, since the components composing the PLA are organized in a chain of assumptions and guarantees (through which the composition of the PLA is realized) it is possible to understand the effect of changes in a component.

Components may moreover represent various alternatives, all of which share some basic properties while they differ in specific details. Formally, we intend to adapt the approach of [1] and define a component C as a Modal Transition System (MTS) that represents its n different variants $C_1, \ldots, C_n$ as Labelled Transition Systems (LTSs). Then there exist assume and guarantee properties such that for all $i \in \{1, \ldots, n\}$, $C_i(a, g)$ either exactly matches, or is less restrictive than, $C(a, g)$. When C is composed with another component D according to $C(a, g)$, then also each C_i can be composed with D. Obviously, it may be the case that $C(a, g)$ does not provide enough guarantees to D, but that there

exists a component C_j, with $i \neq j \in \{1, \ldots, n\}$, which does provide the required guarantees, in which case only this variant C_j can be composed with D.

As soon as the PLA components are defined as MTSs, the MTS Analyser [7] — a tool built on top of LTSA to deal with MTSs rather than LTSs — can be used in order to properly adapt the approach in [1] to PLAs. The attained result is that only parts of the assume-guarantee chain need to be re-verified, thus minimizing the amount of re-verification needed after a component evolves.

4 Conclusion and Future Work

In this paper, we presented a preliminary approach to guarantee the resilience of products of an SPL when facing changes in some of its components. The approach is based on assume-guarantee reasoning and permits to selectively model check and test only those components affected by evolution. Our expectation is that this considerably reduce the effort needed to re-analyze products of an SPL upon a change in that SPL. As future work we plan to implement the overall approach and to experiment its effectiveness by applying it to several case studies.

References

1. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: Formal description of variability in product families. In: SPLC 2011. Springer, Heidelberg (to appear, 2011)
2. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: Design and validation of variability in product lines. In: Proceedings PLEASE 2011, pp. 25–30. ACM, New York (2011)
3. Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A.: Symbolic model checking of software product lines. In: Proceedings ICSE 2011, pp. 321–330. ACM, New York (2011)
4. Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A., Raskin, J.-F.: Model checking lots of systems: efficient verification of temporal properties in software product lines. In: Proceedings ICSE 2010, pp. 335–344. ACM, New York (2010)
5. da Mota Silveira Neto, P.A.: A Regression Testing Approach for Software Product Lines Architectures: Selecting an efficient and effective set of test cases. LAP (2010)
6. da Mota Silveira Neto, P.A., do Carmo Machado, I., McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R.: A systematic mapping study of software product lines testing. Inf. Softw. Technol. 53(5), 407–423 (2011)
7. D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S.: MTSA: The modal transition system analyser. In: Proceedings ASE 2008, pp. 475–476. IEEE, Los Alamitos (2008)
8. Giannakopoulou, D., Pasareanu, C., Barringer, H.: Component verification with automatically generated assumptions. Autom. Softw. Eng. 12(3), 297–320 (2005)
9. Giannakopoulou, D., Pasareanu, C., Blundell, C.: Assume-guarantee testing for software components. IET Softw. 2(6), 547–562 (2008)
10. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Heidelberg (2005)
11. Thüm, T., Schaefer, I., Kuhlemann, M., Apel, S.: Proof composition for deductive verification of software product lines. In: Proceedings VAST 2011 (to appear, 2011)