Guaranteeing Correct Evolution of Software Product Lines

by Maurice ter Beek, Henry Muccini and Patrizio Pelliccione

Researchers from the Software Engineering and Architecture group, University of L'Aquila, together with the Formal Methods and Tools group of ISTI-CNR are developing a novel approach that extends and adapts assume-guarantee reasoning to evolving SPLs in order to guarantee resilience against changes in the product environment. The proposal is to selectively verify and test assume-guarantee properties over only the components affected by the changes.

Software Product Lines (SPLs) are part of an SPL Engineering approach aimed at cost effective development of software-intensive products that share an overall reference model. Variety is achieved by identifying variation points as places in the Product Line Architecture (PLA) where specific products are built by choosing between several optional or alternative features.

While many approaches have been proposed to economize on the validation of SPL products by exploiting product similarities and proper variability management, ensuring the resilience of products of an evolving SPL is an ongoing problem. This is illustrated in Figure 1 which shows a model problem, proposed by Paul Clements and colleagues from Carnegie Mellon University.

"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 an SPL context, this problem can be rephrased as: assuming various products (eg helicopters) derived from an SPL have been formally certified, what can be guaranteed for new products obtained from the SPL once one or more core components have been modified?

We took a first step towards a solution by adapting assume-guarantee reasoning, which sees the environment of a component as a set of properties, called assumptions, that should be satisfied for its functioning. If these assumptions are satisfied by the environment, then components in this environment will typically satisfy other properties, called guarantees. By appropriately combining these properties, it is possible to prove the correctness of a system before actually constructing it. Our idea is to permit selective (re-)testing and model checking of assume-guarantee properties on only those SPL components and products affected by the evolution.

Figure 2 contextualizes our work. Components used in the PLA of the SPL of interest are enriched with assume and guarantee properties (top left). The PLA configuration provides context to the assumptions and guarantees: given a component C with its assume-guarantee pair, its environment becomes the sub-architecture connected with C. The process is complicated by the fact that the reasoning is performed on the PLA, including all variation points, rather than on each indivdual product. This means that the assumptions need to be calculated in a smart way taking into consideration the variability management of the components. Once a component evolves (top right), this modifi-

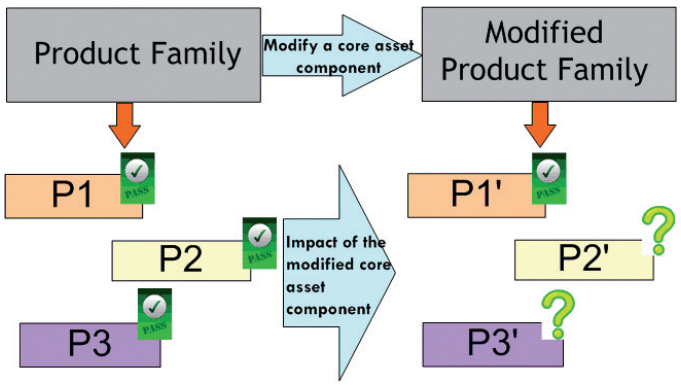

Figure 1: Evolution requires quality re-evaluation.

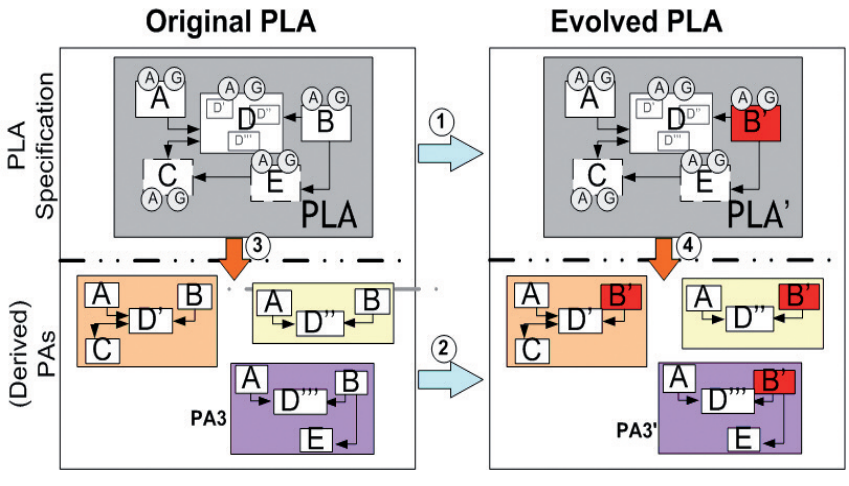

Figure 2: Contextualization of our work

cation is expected to have an impact on several Product Architectures (PAs), namely each PA that contains the modified component. For instance, when B evolves into B', the assumption and guarantee pairs of both B and B' must be checked.

Our solution thus envisages a combination of regression testing and assume-guarantee testing applied to evolving PLAs. Conceptually, this requires us to better understand two issues.

First, how to extend assume-guarantee testing to evolving architectures. The assume-guarantee pair associated with each component in an architecture is normally used to generate component-specific testing traces which, once evaluated against the appropriate assume-guarantee premise, can show whether the composition of components can produce failures. Assuming a PLA/PA has been tested, the challenge becomes how to apply assume-guarantee reasoning for regression testing the modified PLA/PA (see (1) and (2) in Figure 2, respectively). Second, how to use the relationships between a PLA and its PAs (see (3) in Figure 2) to apply regression testing to the evolved PAs.

In summary, we currently envisage what we call a double regression testing approach, in which an evolved product (eg PA3' in Figure 2) can be tested based on how it regressed from PA3, and based on its relationship with the architecture of its family, PLA' (cf. Figure 2). Our expectation is that this considerably reduces the effort needed to re-analyze products of an SPL upon its evolution.

We are working out the details of our approach, after which we intend to implement it and investigate its effectiveness by applying it to case studies.

Please contact:
Maurice ter Beek
ISTI-CNR
E-mail: maurice.terbeek@isti.cnr.it

Software Evolution in Model-Driven Product Line Engineering

by Silvia Abrahão, Javier González, Emilio Insfrán and Isidro Ramos

New requirements and technology changes lead to continuous changes of the assets comprising a software product line. Since the product line represents a large number of potential products (or already deployed products) in a given domain, managing these changes becomes a key issue when dealing with evolution. We present a framework to support the development and evolution of high-quality software product lines. The framework is based on several interrelated models or system views (eg, functionality, variability, quality) and a production plan defined by model transformations that generate a software system that meets both functional and quality requirements. We used our framework to develop a software system for the automotive domain.

Software Product Lines (SPLs) are families of products that share common functionality but also have variations tailored for different customer needs. Many of the benefits expected from SPLs are based on the assumption that the additional investment in setting up a product line pays off later when products are created. However, product line assets have to evolve continuously in order to keep the economic benefits of a product line at an optimal level. Managing this evolution therefore becomes a key issue when maintaining a product line.

In the MULTIPLE project, we defined a framework to support the development and evolution of high-quality SPLs. This framework is based on the definition of a multimodel that represents the different system views and the relationships among them. The variability management involves the manipulation of features, represented as cardinality-based feature models, and the support of such variability in the so-called product line core assets. Specifications of the system variability, functionality, and quality can be dealt with models that are independent from each other but where their inter-consistency is assured by means of the relationships defined in this multimodel.

The SPL production plan is parameterized by means of the multimodel which specifies the corresponding model transformations that are needed to obtain a specific product with the desired features, functions, and quality.

Figure 1 shows the multimodel playing a pivotal role in the SPL production plan. In the domain engineering phase it is used to express the impact and constraints among features, functional components and quality attributes, describing in this way, the extension of the product line. In the application engineering phase, it guides the product configuration, allowing the selection of features and functional components that meets the quality attributes selected by the application engineer. This approach leads us to re-consider the problems related to the intra (eg internal consistency of the feature model) and inter-model consistency (eg correspondences among elements of the feature and quality models) in a broader and realistic context.

Model-based evolution of software systems implies the evolution by using models, eg applying model-driven techniques to support product evolution, or the evolution of models, ie the evolution of the models/metamodels that describe the product. In our framework these evolutions are done by means of reification of software artifacts at the metalevel. When the evolution affects the types then the metamodels need to be reified and evolved. A reflection process translates to the source level the evolved artifact. At a model level, the process is similar. The models are reified at the metalevel, evolved, and then