Refactoring Product Line Architectures

Matt Critchlow, Kevin Dodd, Jessica Chou, and André van der Hoek
Department of Informatics
School of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
{critchlm,kdodd,jychou,andre}@ics.uci.edu

Abstract

In this position paper we explore the application of refactoring to product line architectures (PLAs). A PLA is a common architecture for a set of closely related products. As the set of products and their features changes, the PLA must evolve as well. A typical problem in managing such evolution is that the overall structure of the PLA slowly but surely degrades. This is caused by the fact that a set of individual, localized changes does not necessarily result in the best structure for the overall PLA. We discuss our ongoing research in addressing this problem. In particular, we present our metrics for diagnosing structural problems in a PLA, and introduce our set of architectural refactorings that can be used to resolve those problems—thereby improving the overall structure of the PLA.

1. Introduction

Refactoring has established itself as an effective technique for improving the structure of source code [4]. Its use, however, can be beneficial in other domains as well. In this paper, we explore and demonstrate the benefits of refactoring in one such domain: product line architectures.

A product line architecture (PLA) specifies the architecture for a set of closely related software products [1,2] in terms of components (responsible for all computational aspects), connectors (providing all communication among components), and configurations (specifying the particular topologies of individual products) [6]. The use of PLAs has been on the rise in the past few years [3,7,9,10], which can be attributed to their ability to promote reuse—not just of the individual components and connectors, but also of the overall architectures themselves. As such, a PLA is a critical asset to an organization, and its quality must be maintained over time.

Unfortunately, doing so is difficult. A PLA is fluid, and must constantly be updated when new products are introduced, old products are retired, and existing products are modified. Of particular interest to this paper are the modifications that change the variation points of a PLA. Complementing the core components and connectors (i.e., the components and connectors that are present in all configurations of the PLA), variation points designate the components and connectors in the PLA along which individual configurations differ. These variation points are specified as either *optional elements* (i.e., components and connectors that may or may not be present in a particular configuration) or *variant elements* (i.e., components and connectors that always are present, but are selected to be one of multiple alternatives). Not surprisingly, a PLA can have many variation points. Isolated changes to one or more of these variation points may have structural implications for the remainder of a PLA. In fact, anecdotal evidence suggests that the combined result of multiple changes virtually always leads to structural degradations in the PLA, irrespective of how well-defined the PLA was at its inception.

In this position paper, we present initial results of our research that is specifically aimed at addressing this problem. In particular, we leverage the service utilization metrics defined in our previous work [8] in building two new tools: ARCHMETRIC, which automatically calculates and presents the service utilization metrics as applied to a particular PLA, and ARCHREFACTOR, which provides support for refactoring a PLA in response to problems identified by the metrics. Below, we first introduce an example PLA that we use throughout the remainder of the paper. We then briefly reiterate our metrics, introduce the two new tools, and conclude with an outlook at the work that remains to be done.

2. An Example PLA

Figure 1 introduces an example PLA consisting of four components (for brevity, we omit any connectors from the ensuing discussion; they would be treated in similar fashion with respect to the metrics and refactoring). Components A and B are core components of the PLA, regardless of which product is selected, they are always incorporated. As indicated by the dashed lines, component C is an optional component. In some configurations, it will be included; in others, it will not. Finally, component D is a variant component consisting of four different variants. Depending on the desires of the architect when they instantiate a particular product, one of the four variants will be included in the configuration of that product.

Each component is labeled with both its provided and required services. For instance, component B provides 2 services, `par()` and `paraboo()`, and in turn requires three services, `foo()`, `bar()`, and `foobar()`. Arrows represent usage of services from other components. For instance, component B uses services from both component C and whichever variant is instantiated of component D.

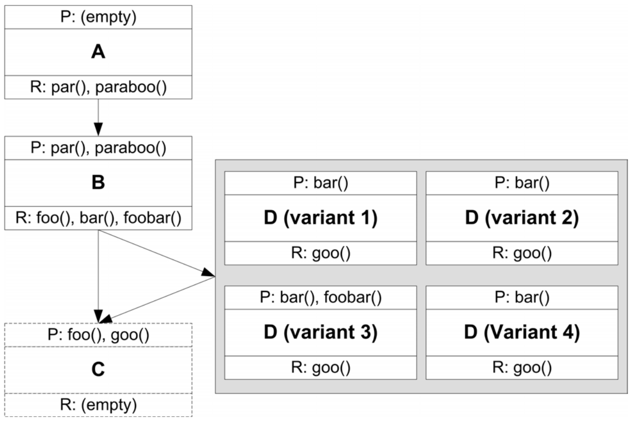

Figure 1. Example PLA.

3. ARCHMETRIC

ARCHMETRIC is the first of our two new tools that, together, address the structural quality of a PLA. The purpose of ARCHMETRIC is to support an architect in identifying and locating potential structural weaknesses. To do so, ARCHMETRIC calculates and visually presents a span of the following two metrics for each component as it appears in each of the different configurations [8]:

$$PSU_X = \frac{P_{actual}}{P_{total}} \qquad RSU_X = \frac{R_{actual}}{R_{total}}$$

PSU represents the *Provided Service Utilization*, and calculates the percentage of provided services (public functions, methods, etc.) of a component that are actually used by other components in a given configuration. RSU stands for *Required Service Utilization*, and complements PSU in calculating the percentage of required[1] services of a component that are actually provided by other components in a given configuration. By calculating the PSU and RSU values for a component as it appears in each configuration (and, thus, each product), we can construct a span of PSU values and a span of RSU values. In essence, these spans provide a graphical visualization of the degree of usage of the services of a component. Through this graphical visualization, an architect can discern such issues as underuse of services, bloated components, particular usage patterns, and other patterns that may indicate the need for potential improvements, etc.

For the PLA presented in Figure 1, the PSU and RSU spans of the various components would be calculated by instantiating each of the possible 8 (2 x 4; 2 for optional component C and 4 for variant component D) configurations. Two of the resulting spans are shown in Figures 2 and 3 as they would be displayed by ARCHMETRICS, namely the PSU span of component D and the RSU span of component B.

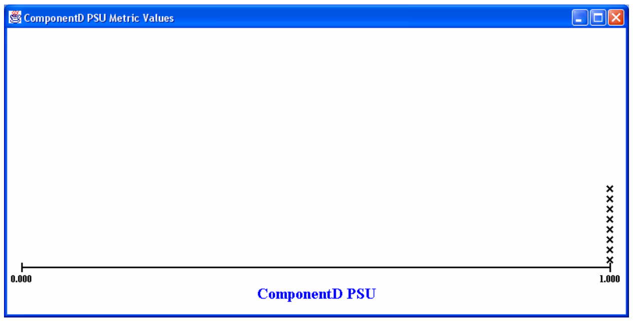

Figure 2. PSU Span of Component D.

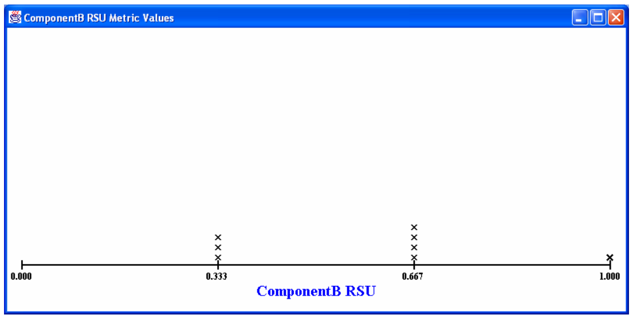

Figure 3. RSU Span of Component B.

[1] While it is clear that P_{actual} may be less than P_{total}, the fact that R_{actual} can be less than R_{total} is counter-intuitive. Required services normally must be met for a system to properly operate. In a PLA, however, it is common to design components with "required" services that, if not provided by other components, will not lead to system failure. When matched, however, these services enhance overall system functionality [11].

The combination of these two spans is interesting: it demonstrates that, for each variant, all of its provided services are always used (all 1's in the span of the PSUs for D), but that the services needed by B are certainly not always provided—except for in one configuration in which both the optional component C and variant 3 of component D are included. This is due to the fact that variant 3 is the only variant that provides the service *foobar()*. While this does not have to be a particular problem, it is possible that this extra service could be split off into a new optional component E. Because this optional component would be a separate element within the overall PLA, it could be used as desired, making the PLA structurally more versatile and all four variants more equitable.

Space prohibits us to go into more detail about the metrics and the potential improvements they can suggest to improve the structure of a PLA. We refer to our previous work for this discussion as well as demonstrations of how we used the metrics to improve the structural quality of three separate PLAs. From the brief discussion here, however, it should be clear that the metrics can be helpful, albeit that they definitely require human interpretation and human investigation of the validity of the symptoms they indicate.

4. ARCHREFACTOR

In our previous work, any improvements to the structure of the PLA needed to be performed manually using, for instance, our design environment Ménage [5]. Clearly this is not a desirable situation, since similar patterns in the metrics indicate similar kinds of improvements in the structure of the PLA. Therefore, we are now in the process of exploring ARCHREFACTOR, a tool that complements Ménage in implementing a set of pre-defined refactoring strategies. An architect simply provides ARCHREFACTOR with input on which refactoring algorithm to apply on which part of the architecture. ARCHREFACTOR then coordinates with Ménage in performing the actual work of doing so.

Tables 1 and 2 list the refactorings that ARCHMETRICS currently supports. These refactorings focus on a specific aspect of PLAs: optional and variant components. While many other refactorings apply as well among, for instance, core components, we chose to focus on options and variants since these have not been addressed with refactoring before and since they play a key role in PLAs. We intend to examine the application of other, more regular refactorings for OO systems in our future work.

The refactorings in Table 1 are for a single component. Most of these refactorings are trivial (e.g., remove an optional component that is never used, turn an optional component that is always used into a core component, etc.). Nonetheless, it is important to explicitly identify them, since they form the core of our approach and, moreover, since they can be composed into more complicated strategies. For instance, the last row shows an optional variant component, which by its very nature can exhibit any of the situations listed in the other five rows of the table. In fact, it could even exhibit both one of the optional situations and one of the variant situations. In such cases, both associated refactorings should be applied (in a desired order), resulting in their combined changes.

Table 1. Refactorings for a Single Component (O = optional, V = variant, OV = optional variant).

Type	***Situation***	***Refactoring***
O	Optional never used	Remove optional component
O	Optional always used	Make optional component a core component
V	One variant used, all others not	Make the one variant a core component and remove other variants
V	A few variants used, all others not	Remove the unused variants
V	One variant has more functionality than the others	Split off an optional component, thereby equalizing all variants
OV	(any combination of an optional situation and a variant situation)	(the matching refactorings, applied one after the other)

Table 2 lists refactorings that involve making changes to two components at the same time. Based on the metrics, we have identified four the different refactorings as shown in the table. An interesting aspect about these refactorings is that they represent interrelationships between optional and variant components. For instance, the second row lists a case in which two optional components are turned into a variant component, and the third row shows a case involving the creation of a new optional component that contains an original optional component and a small set of variants. This shows that optional and variant elements are closely related, and that their use can significantly change over time during the evolution of a PLA.

The refactorings listed in Tables 1 and 2 exhibit a critical property: they are compositional. That is, in situations in which three or more components could be refactored in more than one way (per the rules in the tables), application of "smaller" refactorings leads to the desired "larger" results by first applying any refactorings for two components and then applying any remaining refactorings for a single component. The result is a PLA that incrementally improves in structure.

Finally, we note that all of our refactorings are rooted in the metrics presented in Section 3. The situations as described in each of the rows can be deduced by studying the output of ARCHMETRIC. For instance, the situation in row 5 of Table 1 was used as an example in the discussion of ARCHMETRIC. As a second example, the fact that an optional component is always used is shown in a span for that component by having as many marks on the span as the number of configurations in the PLA. More complicated situations, as those situations listed in Table 2, are presented by ARCHMETRIC as more complicated patterns that are annotated with information about each of the marks on the span.

Table 2. Refactorings for Two Components (O = optional, V = variant).

Type	*Situation*	*Refactoring*
O+O	Span contains either both optionals or neither one	Combine the two optionals in a new optional component
O+O	Only one of them is selected at the same time	Turn both optionals into a variant component
O+V	Optional always selected in combination with one or more variants	Create a new optional component consisting of the existing optional component and its matching variants, thereby splitting the variant component in two variant components
V+V	Specific variants are always selected in matching pairs	Reorganize the two variant components to be a single variant component in which each variant consists of the matching variants of the original components

5. Conclusions

We have started addressing the problem of maintaining the structural quality of PLAs. Thus far, we have implemented two important steps towards a comprehensive solution: (1) a diagnostic tool, ARCHMETRIC, which can detect a variety of problematic situations in the structure of a PLA, and (2) an improvement tool, ARCHREFACTOR, which relies on refactoring to restructure a PLA in response to problems that ARCHMETRIC identifies.

Clearly, our solution is not complete yet. We intend to design additional refactorings and additional mechanisms of examining the structural quality of PLAs. Moreover, we wish to connect ARCHMETRIC and ARCHREFACTOR in an automated fashion: rather than requiring the human step of interpreting the results of ARCHMETRIC and translating that interpretation into instructions for ARCHREFACTOR, we wish to implement a bridge between the two tools that automatically translates results from ARCHMETRIC to suggested refactorings. This still requires human input, but only to approve suggested changes—a significant reduction in effort.

Acknowledgments

Effort funded by the National Science Foundation under grant numbers CCR-0093489 and IIS-0205724. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the National Science Foundation.

References

[1] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison Wesley, 2000.

[2] P. Clements and L.M. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, New York, New York, 2002.

[3] F. de Lange and T. Jansen. *The Philips-OpenTV Product Family Architecture for Interactive Set-Top Boxes*. Proceedings of the Product Family Architecture Workshop, 2001: p. 177-190.

[4] M. Fowler, et al., *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[5] A. Garg, et al. *An Environment for Managing Evolving Product Line Architectures*. Proceedings of the International Conference on Software Maintenance, 2003.

[6] N. Medvidovic and R.N. Taylor, *A Classification and Comparison Framework for Software Architecture Description Languages.* IEEE Transactions on Software Engineering, 2000. 26(1): p. 70-93.

[7] L.M. Northrop. *Reuse That Pays: ICSE Keynote Presentation*. Proceedings of the 23rd International Conference on Software Engineering, 2001: p. 667.

[8] A. van der Hoek, E. Dincel, and N. Medvidovic. *Using Service Utilization Metrics to Assess the Structure of Product Line Architectures*. Proceedings of the METRICS 2003, 2003.

[9] J. van Gurp and J. Bosch, eds. *Proceedings of the Workshop on Software Variability Management*. 2003.

[10] R. van Ommering. *Building Product Populations with Software Components*. Proceedings of the Twenty-fourth International Conference on Software Engineering, 2002: p. 255-265.

[11] R. van Ommering, et al., *The Koala Component Model for Consumer Electronics Software.* Computer, 2000. 33(3): p. 78-85.