

# Multi-Tiered Design Rationale for Change Set Based Product Line Architectures

Scott A. Hendrickson, Swaminathan Subramanian, and André van der Hoek

Department of Informatics  
University of California, Irvine  
Irvine, CA 92697-3440 U.S.A.

{shendric,ssubrama,andre}@ics.uci.edu

## ABSTRACT

Knowledge of *why* some product line architecture (PLA) consists of the specific features and feature interactions that constitute its overall structure is an invaluable resource for an architect. However, most PLA modeling techniques do not model these concepts explicitly. This requires that the architect express rationale separately, which increases the likelihood that it diverges and slowly but surely loses its value. To address this, we present an approach to capturing rationale that relies on a particular form of PLA modeling, namely PLAs modeled using change sets and relationships. In so doing, we enable the architect to express rationale concerning the PLA at three different tiers, covering individual elements in the PLA, change sets and their *raison d'être*, and the relationships that govern how individual product architectures are composed.

## Categories and Subject Descriptors

D.2.11 [Software Engineering]: Software Architectures: *Languages*

## General Terms

Documentation, Design

## Keywords

Architectural knowledge, design rationale, product line architectures

## 1. INTRODUCTION

Product line architectures (PLAs) capture multiple product architectures in a single representation by explicitly managing the elements that are common to these architectures as well as the elements by which these architectures vary from one another. In our previous work [5], we presented an approach to capturing PLAs that uses *change sets* and *relationships*. In this approach, modeling a PLA is done through composition instead of selection. In traditional approaches, a single representation is created in which

variability is captured through variation points inside that representation. From this representation, individual product architectures are selected by resolving the variation points. This approach, however, leads to the modeling of features in a scattered manner.

In our approach, individual features map onto individual change sets that are composed together into different product architectures. The rules by which these compositions may be performed are captured as relationships, which form the basis for the modeling of variants, options, includes, excludes, and other concepts common to PLA modeling [11]. The key benefit of our approach is that it closes the gap between how an architect logically views and interprets a PLA (i.e., in terms of the features that determine differences among individual products) and the actual modeling constructs the architect must use to express those differences.

As with any PLA, a PLA modeled as change sets and relationships requires additional documentation to capture the knowledge that went into the construction of the PLA. In this paper, we explore how the nature of change sets and relationships lends itself to doing so in a more effective way than traditional PLAs. Particularly, we argue that the composition model facilitates a multi-tiered modeling of design rationale that naturally integrates with the way in which the PLA itself is modeled. In fact, we propose to use the change set concept itself to add design rationale, so it is tied to the PLA and always present in the modeling environment. Three levels of design rationale are considered: (1) rationale attached to elements inside change sets to highlight fine-grained observations, (2) rationale attached to change sets to explain their contribution to the overall PLA, and (3) rationale attached to relationships to detail how individual architectures can be composed.

## 2. BACKGROUND

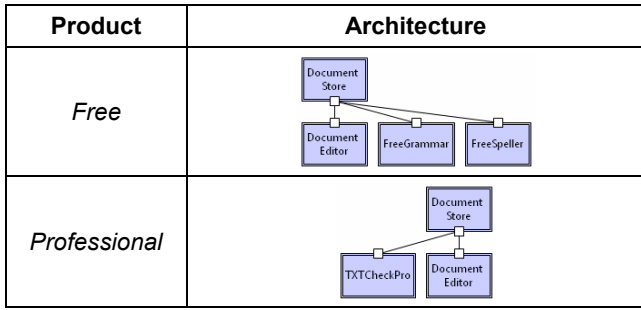
Traditional approaches to product line architecture (PLA) modeling define a single, overarching architectural structure for a set of closely-related products with languages that focus on expressing explicit variation points to differentiate among the various product architectures that make up the PLA (e.g., Koala uses switches [8], Ménage uses optional, variant, and optional variant elements [4], and COVAMOF utilizes optionals, alternatives, optional variants, variants, and values [9]). However, this approach results in a sizeable mismatch between conceptual variability (the logical variability in product architectures that the architect wants to express) and actual variability (the physical variability modeling features of the language via which the architect expresses logical variability). For instance, an optional feature is conceptually considered as one or more architectural elements that are included or excluded in

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SHARK'08, May 13, 2008, Leipzig, Germany.

Copyright 2008 ACM 978-1-60558-038-8/08/05...\$5.00.

**Table 1. Word Processor PLA Products**



unison. However, each and every component, connector, and link that together form the optional feature has to be modeled as a separate variation point, each governed by its own, typically Boolean clause to determine its inclusion or exclusion. This means that the same clause is repeated throughout the PLA representation and as variabilities start to interrelate (which they often do [1]), keeping track of which individual variation points belong together and coherently updating them when the features and their conditions of inclusion and exclusion change quickly becomes a nightmare of repetitive, brittle, and non-intuitive expressions.

We take an *intensional* [2] approach towards modeling PLAs, that is, we model the *changes* in the PLA as opposed to the overall outcome. Rather than following a model in which variation points are resolved to select a product architecture, we *compose* a product architecture by merging together a particular selection of *change sets*. Each change set consolidates the additions and removals that together form a logical increment into a single, conceptual unit that is applied or unapplied to a base architecture. Using change sets, for instance, an optional feature is modeled as a single change set that adds all of the elements for that feature to the architecture.

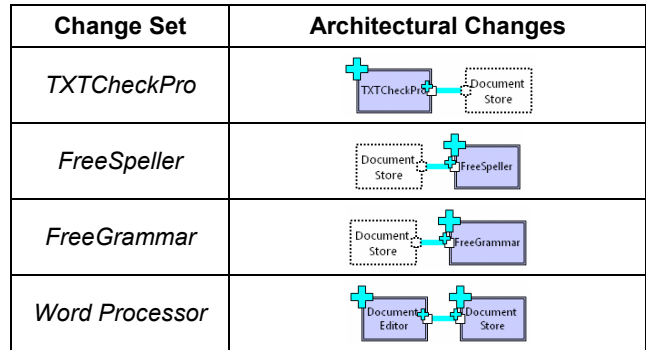
To illustrate the use of change sets, consider the two product architectures shown in Table 1 for a Word Processor PLA. Rather than combining them in to a single representation in which their variations are explicitly modeled, in our approach the architectures are broken down into units from which they can be composed. Naturally, there are many ways in which to do so (from very fine grained to very coarse grained), but we choose to represent them using the change sets presented in Table 2, which represents a middle ground in granularity and is more flexible.

For each of the elements in a change set, an annotation with a “+” means that it is being added by the change set and a “x” that it is being removed. The presence of a clear, dashed, “ghost” element indicates that the change set references but does not actually modify that particular element. For instance, the *FreeSpeller* change set adds the *FreeSpeller* component, its interface, and a link to the interface on the *Document Store* component, which is referenced by the added link but not otherwise modified by the change set.

Using these change sets, the *Free* product can be composed from the *Word Processor*, *FreeSpeller*, and *FreeGrammar* change sets, and the *Professional* product is composed from the *Word Processor* and *TXTCheckPro* change sets.

Figure 1 shows EASEL, our change-set based architecting environment. It is split up in to two parts, a drawing canvas for specifying PLAs and a variability spreadsheet for managing change sets

**Table 2. Word Processor PLA Change Sets.**



and relationships. The content that the drawing canvas presents is the result of merging the change sets selected in the second column of the variability spreadsheet (three change sets are select as such, as highlighted by the three highlighted purple “delta” signs). Any subsequent changes are stored in the currently highlighted change set (*Word Processor* in this case). As with the notation we used in Tables 1 and 2, an architect may choose to explicitly see the “contribution” of a change set. In this case, the changes stored in the *FreeSpell* change set are annotated in the drawing canvas.

The rest of the variability spreadsheet shows relationships among the individual change sets. In this case, two relationships are represented graphically in the last two columns. The first relationship indicates that when the *TXTCheckPro* change set is included, the *FreeSpeller* and *FreeGrammar* change sets should not be and the second relationship indicates that when including any of the three change sets at the top of the list, the *Word Processor* change set should also be included. All of the relationships are continuously verified for their validity with respect to the current state of the PLA, and any violated relationships are highlighted to indicate that an inconsistency has arisen [5].

### 3. PROBLEM

We note that, in some ways, change sets and relationships are already better than a simple flat document in expressing – though implicitly – some knowledge about *how* a PLA was constructed in terms of layers of change sets. But *how* is not sufficient, we also need to understand *why* [7, 12, 14].

As an example, a number of questions concerning the Word Processor PLA are not answered by the model presented thus far. For example, concerning the fine-grained architectural elements, there are for instance other spelling and grammar checkers that could have been used. *Were they considered? Why were these particular choices made?* Concerning entire change sets, as we already indicated, the change sets listed in Table 2 are but one possibility. It would be possible, for instance, to combine the *FreeSpeller* and *FreeGrammar* change sets into a single change set. *Why was that not done? Would it be more appropriate to do so?* And concerning relationships, it seems that one of the relationships is trying to express a variant relationship between features (i.e., either include the commercial component, *TXTCheckPro*, or the open source components, but not both). *Is this so? Would it be appropriate to re-express this relationship as a variant relationship?* This type of knowledge is not captured in the approach as presented thus far, yet clearly represents architectural knowledge that is important to capture and preserve.

## 4. APPROACH

Modeling PLAs as change sets and relationships provides us with a unique opportunity for including rationale within the PLA itself. Our approach focuses on capturing three distinct tiers of rationale: (1) rationale attached to *architectural elements* inside change sets, (2) rationale attached to *change sets* to explain their contribution to the overall system, and (3) rationale attached to *relationships* to explain why the system is composed in a particular manner. Being able to express rationale at these three levels allows an architect to answer the types of questions raised in Section 3.

### 4.1 Rationale for Architectural Elements

We first extended our environment with the ability for designers to specify rationale for their choices of architectural elements, to address the first questions that we brought up in Section 3: *Why these elements? Why not others?*

In traditional modeling environments, such rationale is typically present as notes on the canvas. We essentially adopt the approach of notes, but the use of change sets creates some special affordances. Specifically, just like the “regular” architectural elements being modeled, we store notes in individual change sets. That means that each change set can add, remove, or modify rationale. For instance, some rationale for an element may only be relevant if a particular change set is applied—that rationale should be added by that change set. Other rationale for a particular element may be invalidated by the inclusion of a particular change set—that change set could remove this rationale.

Overall, this means that rationale takes on a historical and contextual meaning. It is historical, because each element can have a trace of rationale that incrementally builds up the knowledge for its existence (or in some cases, non-existence!). It is contextual, as the rationale is specific to the particular product architecture formed by the chosen composition of change sets.

To view and manipulate rationale at this level, we have added a new *Rationale View* to EASEL, shown in Figure 2. The rationale view has two panes. The left pane shows rationale associated with any elements that are selected in the drawing canvas and the right pane shows to which element(s) the rationale is attached. In this pane, the architect can add, modify, and remove rationale – the result of which would be stored in the currently active change set.

In the example shown, the architect has associated rationale to the *FreeSpeller* component stating: *FreeSpeller: open-source project, faster than OpenSpeller*. This explains both why this component was selected and what other component was considered. Similarly, the architect has associated rationale to the *FreeGrammar* component stating: *FreeGrammar: developed by FreeSpeller group, thus easy integration, but not best - Grammatic is best*. This indicates that *FreeGrammar* was chosen because it easily integrated with its companion component, *FreeSpeller*. However, one of the different components tested, *Grammatic*, would be best in isolation. We note that rationale is annotated with the familiar “+” and “x” signs to indicate whether it was added or removed by a specific change set.

### 4.2 Rationale for Change Sets

While capturing rationale for individual architectural elements is useful, it cannot adequately express rationale for higher level con-

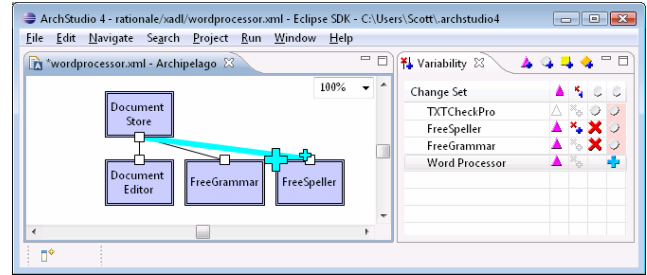


Figure 1. The EASEL Architecting Environment

cepts. For instance, whether it is better to keep the *FreeSpeller* and *FreeGrammar* change sets separate or to combine them into a single, larger change set has no impact on the architectural elements that constitute the two, but is a decision at a higher level.

To facilitate expressing rationale as to why the set of change sets is as it is, EASEL supports a second tier of rationale capture that is associated with entire change sets. Instead of storing this rationale in the change set, it is necessary to store it separately, so the rationale is independent from the applying or unapplying change sets and can be examined separately.

The *Rationale* view is again used as a means for visualizing and manipulating rationale, but this time at the level of change sets. By selecting change sets from the variability spreadsheet, rather than selecting architectural elements from the drawing canvas, an architect is presented with the rationale associated with each selected change set. As before, an architect can add and remove rationale for the current selection of change sets.

Returning to the questions we posed in Section 3, the answer as to whether or not the *FreeSpeller* and *FreeGrammar* change sets should be merged is quite different depending on what rationale is associated with them. For example, rationale associated with these two change sets reading: *Adds open-source components that perform various checks on the document*, might indicate that merging them is okay. Alternatively, if rationale for *TXTCheckPro* reads: *We cannot split TXTCheckPro into separate spelling and grammar features because it is only available as a single package that does both*, then one should conclude that merging the *FreeSpeller* and *FreeGrammar* change sets is not a good idea.

### 4.3 Rationale for Relationships

A third tier of rationale is necessary to capture knowledge about why some compositions of change sets are valid and others invalid. Rationale attached to individual change sets should not stipulate this kind of knowledge; instead, rationale attached to relationships should explain this.

While the function of a relationship is clear (testing which change set compositions satisfy it), why each relationship exists in a PLA and tests for its particular condition is not necessarily so. To manage rationale for relationships, we use the same view once again, but this time reflecting a selection of relationships. As with change sets, rationale for relationships is expressed independently so that applying or unapplying change sets does not affect rationale associated with a relationship (and relationships of course are never applied or unapplied, they are always active in verifying the current composition). Overall, then, the *Rationale View* becomes a central clearinghouse for all management of rationale, with a consistent behavior across all three tiers of rationale. Simply by con-



Figure 2. EASEL With Rationale For Two Architectural Elements

text of use of EASEL, i.e., is the designer working with elements, change sets, or relationships, the *Rationale View* switches among the three tiers.

## 5. CONCLUSION AND FUTURE WORK

This work represents initial work in closely associating rationale with product line architecture concepts. Previous work in this area has generally relied on attaching notes to the PLA that provide a historical trace [13], but cannot contextualize this trace with respect to individual product architectures. In our work, we provide a multi-tiered system of rationale capture that relies on the unique representation of PLAs as change sets and relationships to find natural places to associate rationale – ranging from detailed information regarding individual elements to high-level structural descriptions governing the change sets and relationships.

Our next step will be to extend our work to incorporate support for modeling design decisions. We view decisions as strictly different from rationale, as decisions provide motivation for rationale but generally lead to rationale “spread” throughout the PLA. This is a different view from existing work (e.g., Archium [6] models design decisions as containing rationale and architectural changes, Sinnema et al. [10] additionally models design decisions interaction as first class entities using COVAMOF [9], FeatureKing [3] maps design decisions to architectural features), but one that critically enables “crosscutting” design decisions, i.e., design decisions that influence changes over multiple features, or across multiple products in the PLA, or just across multiple components.

## 6. REFERENCES

- [1] Batory, D. Feature Models, Grammars, and Propositional Formulas. Ninth International Software Product Line Conference. p. 7-20, 2005.
- [2] Conradi, R. and Westfechtel, B. Version Models for Software Configuration Management. ACM Computing Surveys. 30(2), p. 232-282, June, 1998.
- [3] Dhungana, D., Rabiser, R., et al. Architectural Knowledge in Product Line Engineering: An Industrial Case Study. 32nd EUROMICRO Conference on Software Engineering and Advanced Applications. p. 186-197, 2006.
- [4] Garg, A., Critchlow, M., et al. An Environment for Managing Evolving Product Line Architectures. IEEE International Conference on Software Maintenance. p. 358-367, 2003.
- [5] Hendrickson, S.A. and van der Hoek, A. Modeling Product Line Architectures through Change Sets and Relationships. 29th International Conference on Software Engineering. p. 189-198, 2007.
- [6] Jansen, A. and Bosch, J. Software Architecture as a Set of Architectural Design Decisions. 5th IEEE/IFIP Working Conference on Software Architecture. p. 109-119, 2005.
- [7] Lago, P. and Avgeriou, P. First Workshop on Sharing And Reusing Architectural Knowledge. ACM SIGSOFT Software Engineering Notes. 31(5), p. 32-36, 2006.
- [8] Ommering, R.v., Linden, F.v.d., et al. The Koala Component Model for Consumer Electronics Software. IEEE Computer. 33(3), p. 78-85, March, 2000.
- [9] Sinnema, M., Deelstra, S., et al. COVAMOF: A Framework for Modeling Variability in Software Product Families. Third International Software Product Lines Conference. p. 197-213, 2004.
- [10] Sinnema, M., van der Ven, J.S., et al. Using Variability Modeling Principles to Capture Architectural Knowledge. ACM SIGSOFT Software Engineering Notes. 31(5), September, 2006.
- [11] Svahnberg, M., van Gorp, J., et al. A Taxonomy of Variability Realization Techniques. Software Practice and Experience. 35(8), p. 705-754, July, 2005.
- [12] Tang, A., Babar, M.A., et al. A Survey of the Use and Documentation of Architecture Design Rationale. 5th Working IEEE/IFIP Conference on Software Architecture p. 89-98, 2005.
- [13] Trujillo, S., Azanza, M., et al. Exploring Extensibility of Architectural Design Decisions. Second Workshop on SHaring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent. 2007.
- [14] Tyree, J. and Akerman, A. Architecture Decisions: Demystifying Architecture. IEEE Software. 22(2), p. 19-27, 2005.