

From Requirements Documents to Feature Models for Aspect Oriented Product Line Implementation

Neil Loughran, Américo Sampaio and Awais Rashid

Computing Department, InfoLab 21, Lancaster University, Lancaster LA1 4WA, UK

(loughran | a.sampaio | awais) @comp.lancs.ac.uk

Abstract. Software product line engineering has emerged as an approach to developing software which targets a given domain. However, the processes involved in developing a software product line can be time consuming and error prone without adequate lifecycle tool support. In this paper we describe our approach, NAPLES, which uses natural language processing and aspect-oriented techniques to facilitate requirements analysis, commonality and variability analysis, concern identification to derive suitable feature oriented models for implementation.

1 Introduction

Software product line engineering [19] promotes an architecture centric approach to developing software, which targets a particular domain or market segment. Software can then be created that is customizable to the particular requirements of different customers. Utilizing such an approach can yield high quality, optimized software with an increase in productivity and consistency as well as reductions in time to market, costs and error rates. Among the influencing factors that may necessitate the move to a product line architecture are identification of new market trends and their domains, business mergers and the encapsulation of multiple but overlapping existing products that a company may have in their itinerary. However, software product lines are difficult to develop with many of the activities (e.g., domain analysis, modeling and implementation) being time consuming and error prone. Lifecycle tool support and an associated methodology which aims to address these problems are therefore essential.

In this paper we present our approach, NAPLES (Natural language Aspect-based Product Line Engineering of Systems), a product line engineering approach that provides life cycle tool support for taking requirements documents and other textual assets (e.g. documentation and user manuals) and analyzes them for potential features and aspect candidates as well as commonalities and variabilities within a given domain. We demonstrate the approach by taking an existing set of requirements and performing analysis on them in order to mine for key domain concepts, viewpoints and aspects as well as variabilities, which can then be mapped to a suitable feature model.

The next section provides an overview of NAPLES. Section 3 demonstrates how NAPLES provides guidance and tool support to effectively mine concepts (e.g., concerns, commonalities, variabilities) as well as facilitates to produce the models in further phases. Section 4 demonstrates how framed-aspect models can be systematically delineated from the previous identified concepts and how they can map to a modularized implementation. Section 5 provides discussion on the process and explains how it could be utilized for different contexts such as building a product line from scratch or from existing systems. Section 6 briefly describes related work while section 7 concludes the paper.

2 Natural language Aspect-based Product Line Engineering for Systems (NAPLES)

The NAPLES approach addresses product line (PL) engineering throughout the lifecycle by using different techniques, e.g., natural language processing (NLP) and aspect-oriented software development (AOSD), to provide automated support and separation of concerns during the PL lifecycle. For example, during requirements activities, tool support based on natural language processing techniques and aspect-oriented requirements engineering (AORE) [1, 2] is provided to mine for different concepts and help the developer to build models from these concepts. The tools used with the approach do not automate 100% of the engineering tasks but they aim to provide effective support by automating time-consuming activities which is vital for product line engineering. Figure 1 presents the NAPLES approach showing its activities and input/output artifacts.

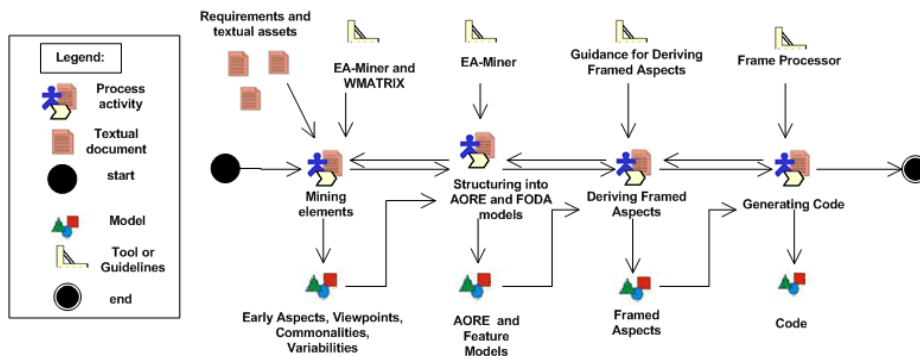


Figure 1 - NAPLES approach

The approach starts with the *Mining Elements* activity which identifies important concepts (e.g., early aspects [1-3], viewpoints [4], commonalities and variabilities) from the requirements documents used as input, and presents them to the user in a format that can be used to produce a structured model (AORE model [1, 2] and feature model). The EA-Miner [3, 5] tool uses the WMATRIX [6, 7] natural language processor to pre-process the input documents and get relevant information.

WMATRIX provides part-of-speech and semantic tagging, frequency analysis and concordances¹ to identify concepts of potential significance in the domain. Part-of-speech analysis automates the extraction of syntactic categories from the text (e.g., nouns and verbs).

The information produced by the NLP processor is then used by EA-Miner to help list possible key domain concept candidates. For example, for the identification of viewpoints, the tool lists the most frequently occurring nouns in the text, and for Early Aspects it lists words whose meaning resembles a broadly scoped concern (e.g., security, performance, parallel, logon, authorize, and so forth). Details on how EA-Miner performs its identification can be found in [3, 5]. Commonalities and variabilities are also identified in a similar fashion, and this is detailed in Section 3.

After the software developer has identified and selected the concepts of interest in the previous activity, EA-Miner helps to build structured models during the *Structuring into Models* activity. The tool enables the application of *screen out* functionalities (e.g., add, remove, check synonyms) to discard irrelevant concepts, add new ones and check if the same concepts are identified as different ones. The output is an AORE model showing the viewpoints, early aspects and composition rules as well as a feature model showing features alongside their commonalities and variabilities.

The *Deriving Framed Aspects* activity uses the previous models (AORE and feature model) and provides guidance on how to delineate an aspect-oriented model based on framed aspects. The framed classes and aspects are then used by the frame processor in the *Generating code* activity to create the code in a specific language (e.g., AspectJ [17]). More details are given in Section 4.

3 Commonality and Variability Analysis

The process for identifying commonality and variabilities is similar to what we have previously done with success to identify viewpoints and early aspects in [3, 5]. We will explain the procedure using an example of a requirements description for a product line of mobile phones.

Example:

A mobile phone company XYZ wants to create a product line for its products aiming at reducing the costs of its operations. The phones will have similar features that can vary according to the model. Some details of the product line are:

- The phone models are: Model A, Model B and Model C.
- The feature game will be present in all models. For models A and B the games are already installed and the difference is that model B has games G3 and G4 in addition to G1 and G2 also found in A. Model C has all the previous games and also the option to download more games. For all gaming features, it is important

¹ Concordance is a way of presenting a list of chunks of text containing a specific word and its surrounding text. The word of interest is highlighted and separated from the rest of the text (e.g., centered). Example: text... vehicletext.

to offer good performance to the users. Each game provides some facility to store high scores.

- The feature list of contacts is offered in all 3 models and varies in the capacity of the numbers of contacts that can be stored. For models A and B up to 50 contacts can be saved while for model C up to 100.
- Model C is the only one to offer a web browser and chat functionalities for real-time communication over the web. There is provision for the user to store the URLs of their favorite web pages.
- All mobile phones offer a password protection mechanism that is requested when the phone is turned on. The password is stored in the phone in an encrypted manner.

The identification of commonalities is based on a lexicon of relevant domain concepts for mobile phones (e.g., model, game, contacts, chat and calendar). The task of the tool is to compare if each word in the document is “equalTo” a domain concept. The “equalTo” procedure is defined as: *if a word is lexically equal, ignoring case and suffixes, to the word in the lexicon AND the word has same semantic class as a word in the lexicon*. The comparison after the AND avoids identifying words in the text that have the same spelling but are used in a completely different meaning (e.g., the word *performance* can be used to indicate a constraint on a software or to indicate the act of a dancer or artist in a show). The meaning of the words are attributed by the tagging feature of WMATRIX which tags each word in the input file with its part-of-speech and semantic categories (e.g., <w id="8.25" pos="VVN" sem="A9+> stored </w>). This means that the word *stored* is a verb (VVN) whose meaning belongs to the class of “Getting and giving; possession” (A9).

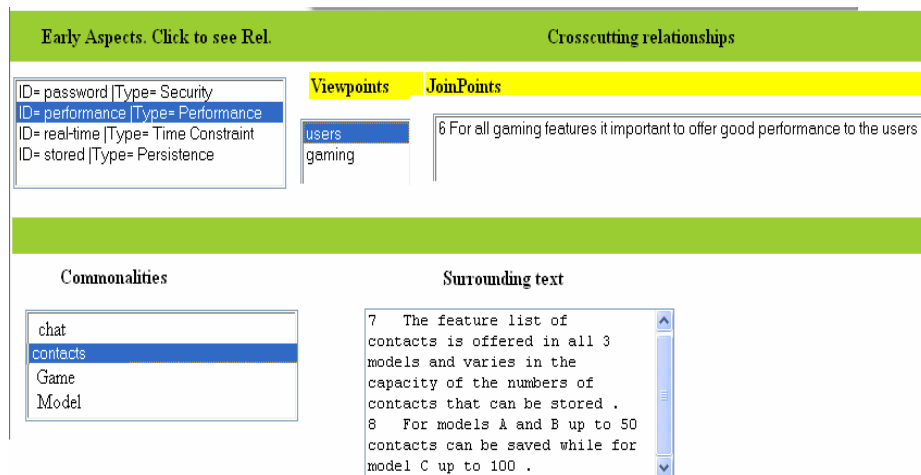


Figure 2 - EA-Miner tool

The EA-Miner tool helps the user to identify variabilities by providing the surrounding text in which the word occurs. In figure 2, after the user selects the “contacts” commonality, the right-hand side shows in which sentences (sentences 7

and 8) of the document the word appears. The rules of thumb for identifying commonalities and variabilities are:

- The tool lists the commonalities on the left-hand side and the user searches for possible variabilities by looking at the surrounding context of a specific commonality (e.g., *contacts*);
- The user looks at the details on the right-hand side and identifies concepts that modify the commonality in some way (e.g., the *size* of the list of contacts is variable depending on the model).

Another example would be to select the word ‘game’ which would result in the display of the following sentences:

- ⇒ 4 *Some details of the product line are: The phone models are: Model A, Model B and Model C. The feature game will be present in all models.*
- ⇒ 5 *For models A and B the games are already installed and the difference is that model B has games G3 and G4 in addition to G1 and G2 also found in A. Model C has all the previous games and also the option to download more games.*

This information then helps to identify ‘game’ as a commonality and that each model can offer different games. Figure 3 shows a feature model based on FODA [18] which can be easily built from the information previously described.

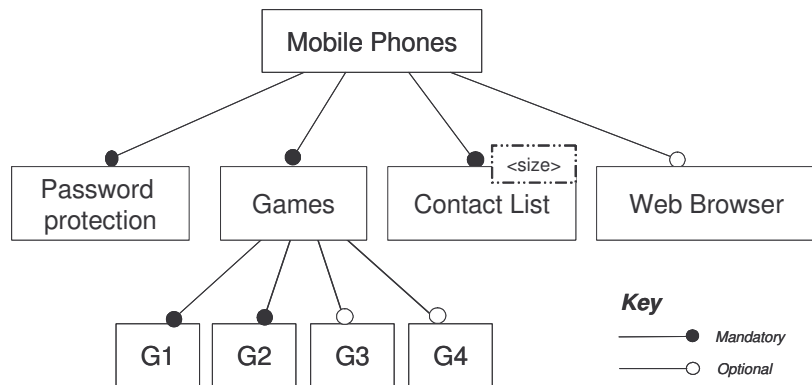


Figure 3 – Part of the FODA model for mobile phone domain

It is a relatively simple process to map the variabilities identified to the appropriate features. For example the *Contact List* feature is parameterized with a *<size>* attribute to indicate that this a fine grained variability point contained *within* the feature itself. An optional feature such as *Web Browser* is coarser grained in nature so is given its own feature space.

The benefits of using the tool are more evident when considering situations where the input files contain masses of information of a varying nature such as manuals, legacy documents, and previous requirements. This kind of situation is common to occur in company mergers where a wide variety of documents exist. In such situations, tool support would be very helpful in order to mine for the key concepts

that will aid the construction of assets for the merged product line. Another benefit of the tool is that it is scalable. We have previously run the tool with several requirements documents in [5] and the tool takes just a few minutes to process and display the results even considering large documents (tens of thousands of words).

The process for identifying early aspects and viewpoints, shown in figure 2, is detailed in [3, 5] and is briefly mentioned here. Viewpoints represent stakeholders' intentions and also are the base decompositions of our RE model. They are identified by getting a list of the most frequent nouns from WMATRIX and their structure groups the requirements needed to satisfy specific stakeholder goals.

The early aspects seen on top of figure 2 represent crosscutting concerns (e.g., system properties such as security) that crosscut the viewpoints. The crosscutting points are also listed by the tool (e.g., "performance" crosscuts viewpoint "users" at sentence 6 in figure 2).

The identification of viewpoints will drive the implementation of the functional requirements of the features and the early aspects will help to point out some crosscutting concerns (as we will discuss for persistence in Section 4) that can affect multiple features and are not captured by the FODA model in figure 3. The next section shows how the list of viewpoints, early aspects and the FODA model built by the EA-Miner tool will be useful for delineating the models in the next stages.

4 Delineating Implementation from Feature Models

The previous stages involved searching for key domain concepts, identifying possible aspects and their variabilities and modeling them appropriately. In this section we describe how we might go about implementing the various concerns in our product line.

In [15] a process called *framed aspects* is described which unifies frame technology [16], a language independent meta-language for implementing variability, with aspect languages such as AspectJ [17]. Frame technology implements useful conditional compilation, parameterization and code refactoring techniques along with a configuration language. The framed aspects process uses AOP to modularize crosscutting and tangled concerns and utilizes framing to allow those aspects to be parameterized and configured to different requirements. An important contribution of framed aspects is the provision of a methodology for development.

The framed aspect approach involves a number of key stages:

- 1) Variability modeling using FODA (as done in figure 3).
- 2) Frame delineation of feature model (as done in figure 4)
- 3) Creation of parameterized, generalized aspects within the delineated frames.
- 4) Creation of composition rules for composing the required frames together and imposing constraints.
- 5) Development of specification templates for developers.

From the derived feature model in figure 3 we can now clearly delineate *coarse grained* common features in our product line from the variant ones. This is illustrated in figure 4. Mandatory features are always included in every product instantiation so

these form part of the commonality set and are delineated as such. Optional features are variants and are therefore delineated separately. The fine grained variability point `<size>` is a simple parameter which is *internal* to a common feature, but the feature *Contact List* itself is mandatory, and therefore delineated as a common feature. In other words, at this level of abstraction we are only really interested in delineating *coarse grained features*.

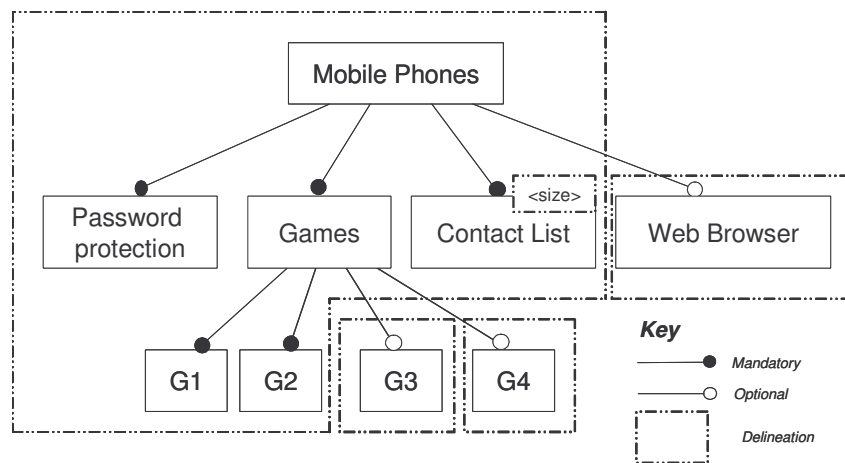


Figure 4 - Delineating common and variable parts

The feature diagram approach, used alone in isolation, does not provide an obvious implementation strategy, as it focuses on providing a commonality and variability model, which can be used to delineate features and create the appropriate configuration rules and constraints. However, by using the guidelines previously identified in the mining and structuring activities, as described in section 3, we can make a sound judgment as to what aspects and classes will be used in our architecture. Then we can combine this with the feature diagrams in order to generalize those concerns using frame technology.

An example of a concern we identified at the mining activity was *persistence*. This was identified due to the words ‘storage’ and ‘stored’ being used in the document. Because these words crosscut multiple features (e.g. password protection, games, contact list and web browser) we knew that this was a potential candidate for an aspect. Moreover, we could identify from the feature model that some of these features, where the persistence aspect crosscut, were optional (e.g. web browser), therefore the persistence aspect itself needed to be generalized using the variability mechanisms available in framed aspects. Figure 5 illustrates how an implementation of the persistence concern might proceed with this in mind. Note that in the figure we have simply mapped non crosscutting features to a single class implementation although in reality each feature may consist of multiple classes.

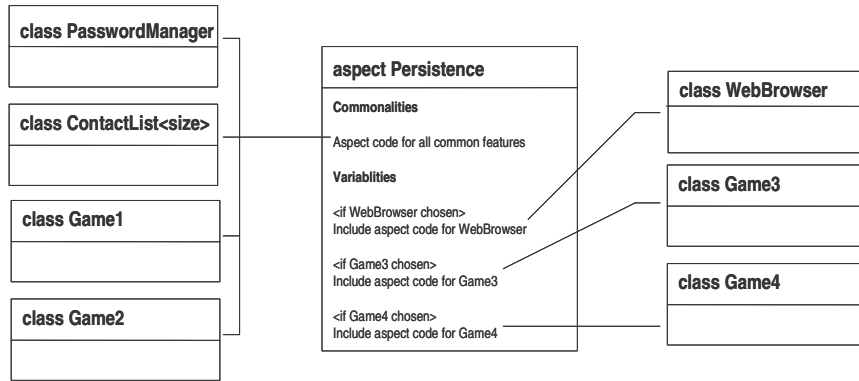


Figure 5 - The persistence concern crosscutting common and variable features

5 Discussion

The process we have described in this paper considered an example where a product line would be created from scratch. However, in many cases this approach might not be the most practical. A product line could be the merging of existing overlapping products in order to better manage them. Alternatively a product line may be taking an existing product and generalizing so that it can be adapted to different requirements. The question is, *can we still apply the processes of NAPLES to these situations?* We believe the answer is yes.

With respect to multiple overlapping systems, textual assets (e.g. the original requirements documents, user manuals, legacy documentation and so forth) can be processed with EA-Miner in order to find common and variable parts which will facilitate redesign. The redesign can enable the construction of a new product line since early stages of development (e.g., requirements, design) mining the existing assets and restructuring them according to NAPLES models (e.g., AORE, framed aspects). The approach will minimize the effort in structuring the existing information into a new product line by offering the automated support described before and also facilitate further evolution by means of separation of concerns.

Similarly, in the process of making a single existing product into a software product line, domain concepts can be mined using the domain lexicon and can also help with the identification of possible future places for variability. The viewpoint and aspect identification models can lead to better modularized designs which are more conducive to software product line development. For example, considering the description of a specific mobile phone we could see that possible candidates for common features would be games, list of contacts, chat, web browser, video, camera, etc. Each of these features would be identified by the tool alongside the possible aspects that affect them such as security, persistence and performance. The processes described in sections 3 and 4 would then help to restructure the current assets into the basis of a product line architecture with less effort compared to manual approaches.

6 Related Work

The use of natural language processing techniques in automation of some tasks in requirements engineering has been discussed in [8-10]. Most of these approaches focus on identifying abstractions in requirements documents and building models from them. NAPLES differs by adding semantic tagging and AORE to address crosscutting concerns and issues pertinent to product lines e.g. domain analysis, commonality and variability and lifecycle variability management.

One approach that addresses crosscutting concerns throughout the lifecycle is the Theme [11] analysis and design method; Theme/Doc [12] is the requirements modeling approach within this method while Theme/UML is the design mechanism. Some support for package parameterization exists in Theme/UML and one can also map Theme/UML designs to multiple AOP languages. However, Theme has not been designed for product line engineering and as such does not support variability and commonality identification and analysis. Like our approach, Theme/Doc has support for lexical analysis to identify aspects in requirements. However, the developer has to read through the whole set of input documentation and manually identify a list of action words and entities that are provided as input to the Theme/Doc tool. This has the potential of becoming a bottleneck in case of large documents used for input (e.g., tens of thousands of words). Our approach, however, is based on the WMATRIX NLP tool suite which has been shown to work for large document sets [7]. The approach in [13] considers the identification of variabilities in code using concern graphs [14]. After the variabilities are found a list of refactorings are applied in the code to factor out the variabilities into aspects. A key difference that NAPLES addresses is that product lines are not only about code level assets and therefore other assets such as requirements documents and design models can also be considered and receive their proper treatment.

7 Conclusions

In this paper we have described NAPLES, our approach for taking textual assets (e.g. requirements documents, user manuals, interview transcripts, legacy documents and so forth) and deducing concerns, aspects, feature commonalities and variabilities so that an implementation can follow. Utilizing tool support saves time and effort, reduces errors and provides a holistic treatment of concerns and variability across the software lifecycle aiding traceability of requirements to their implementation. The semi-automated approach we have outlined provides guidelines for designs, which will inevitably help to drive the implementation phase and ease the creation of generalized code assets. Improving the identification of early aspects, commonalities and variabilities, as well as evolution issues pertaining to the product line, will be a focus of future work. Additionally, the guidelines applied to the delineation of feature models and their realization with the framed aspects approach will also be investigated in greater depth. The processes in NAPLES provide a systematic approach to creating a software product line architecture from requirements through

to implementation. The approach automates many of the tasks which consume time and effort thus cutting costs.

Acknowledgments

This is supported by European Commission grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe), 2004-2008.

References

1. Rashid, A., A. Moreira, and J. Araujo. Modularisation and Composition of Aspectual Requirements. in 2nd International Conference on Aspect Oriented Software Development (AOSD). 2003. Boston, USA: ACM.
2. Rashid, A., et al. Early Aspects: a Model for Aspect-Oriented Requirements Engineering. in International Conference on Requirements Engineering (RE). 2002. Essen, Germany: IEEE.
3. Sampaio, A., et al. Mining Aspects in Requirements. in Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (held with AOSD 2005). 2005. Chicago, Illinois, USA.
4. Finkelstein, A. and I. Sommerville, The Viewpoints FAQ. BCS/IEE Software Engineering Journal, 1996. 11(1).
5. Sampaio, A., et al. EA-Miner: A tool for automating aspect-oriented requirements identification. in 20th IEEE/ACM International Conference on Automated Software Engineering (ASE2005) 2005. Long Beach, California, USA.
6. Rayson, P., UCREL Semantic Analysis System (USAS). 2005: <http://www.comp.lancs.ac.uk/ucrel/usas/>.
7. Sawyer, P., P. Rayson, and R. Garside, REVERE: Support for Requirements Synthesis from Documents. Information Systems Frontiers, 2002. 4(3): p. 343-353.
8. Ambriola, V. and V. Gervasi. Processing natural language requirements. in International Conference on Automated Software Engineering. 1997. Los Alamitos: IEEE Computer Society Press.
9. Burg, F.M., Linguistic Instruments in Requirements Engineering. 1997: IOS Press.
10. Goldin, L. and D. Berry, AbstFinder: A Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation. Automated Software Engineering, 1997. 4.
11. Baniassad, E. and S. Clarke. Theme: An Approach for Aspect-Oriented Analysis and Design. in International Conference on Software Engineering. 2004. Edinburgh, Scotland, UK.
12. Baniassad, E. and S. Clarke. Finding Aspects in Requirements with Theme/Doc. in Workshop on Early Aspects (held with AOSD 2004). 2004. Lancaster, UK.
13. Alves, V., et al. Extracting and Evolving Mobile Games Product Lines. in 9th International Software Product Line Conference (SPLC-EUROPE 2005) 2005. 26-29 September 2005 Rennes, France
14. Robillard, M. and G. Murphy. Concern graphs: Finding and describing concerns using structural program dependencies. in 24th International Conference on Software Engineering. 2002.
15. Loughran, N., Rashid A. (2004) *Framed Aspects: Supporting Variability and Configurability for AOP*. International Conference on Software Reuse, Madrid, Spain.
16. Bassett, P. *Framing Software Reuse: Lessons From the Real World*: Prentice Hall, 1997.
17. "AspectJ Home Page <http://www.eclipse.org/aspectj/>," 2005.
18. Kang, K, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature Oriented Domain Analysis Feasibility Study," SEI Technical Report CMU/SEI-90-TR-21 1990.
19. Clements, P and L. Northrop, "Software Product Lines - Practices and Patterns," Addison Wesley, 2002.