Squid: An Extensible Infrastructure for Analyzing Software Product Line Implementations

Alexandre Vianna[1], Felipe Pinto[1, 2], Demóstenes Sena[1, 2], Uirá Kulesza[1]
Roberta Coelho[1], Jadson Santos[1], Jalerson Lima[1,2], Gleydson Lima[1]

[1]Department of Informatics and Applied Mathematics (DIMAp)
Federal University of Rio Grande do Norte (UFRN), Brazil

[2]Federal Institute of Education, Science and Technology of Rio Grande do Norte (IFRN)
Natal, Rio Grande do Norte, Brazil

strapacao@gmail.com, {felipe.pinto, demostenes.sena, jalerson.lima}@ifrn.edu.br,
{uira, roberta}@dimap.ufrn.br, {jadson, gleydson}@info.ufrn.br

ABSTRACT
Software product line engineering is about producing a set of related products that share more commonalities than variabilities. This approach promotes benefits such as cost reduction, product quality, productivity and time to market, but it brings new challenges that must be considered during the evolution of the software product line. In this context, recent research has explored and proposed automated approaches based on code analysis and traceability techniques for change impact analysis. This paper presents Squid, an extensible infrastructure for analyzing software product line implementations. The approach uses information from variability modeling, variability mapping to code assets, and dependency relationships between code assets to perform analysis of SPL implementations. A Squid instantiation example is presented to illustrate the usage of the tool in practical scenarios.

Categories and Subject Descriptors
D.2.13 [**Reusable Software**]: Domain engineering, Reusable models

General Terms
Design, Languages

Keywords
Software Product Line, Software Analysis, Software Product Line Evolution

1. INTRODUCTION
Software product line engineering [1] is about producing a set of related products that share more commonalities than variabilities. A software product line (SPL) defines a family of systems that share common features and differ in other features according to the requested software systems – products. The SPL approach promotes benefits such as cost reduction, product quality, productivity and time to market, but it brings new challenges that must be considered during the SPL evolution.

SPLC'12-Workshops, September 02-07, 2012, Salvador, Bahia, Brazil.

Nowadays, the software market requires a massive development of new features to existing software systems. These changes impact on the evolution of software systems and create challenges to the software engineers. However, the development of SPLs presents specific particularities, such as the modeling and implementation of variabilities, and the modeling of configuration knowledge [2] that defines the mapping between variabilities and code assets.

Therefore, the management of SPL evolution presents specific peculiarities that involve dealing with assets that represent not just single software, but families of related systems, requiring appropriate approaches for this context. Techniques to support the SPL evolution [3][4] are very useful to help and reduce the effort of maintenance activities and avoiding adding unwanted side effects due to changes, for example, adding new features. The SPL development involves the elaboration of many assets of diverse types, such as, implementation, documentation and modeling. Furthermore, there are complex relationships involving these assets, which increase the difficulty to manage them.

When there is an evolution in a SPL, the changed assets can potentially affect other assets that are directly or indirectly related. In such cases, the impacts resulting from these changes need to be identified and analyzed. The change impact analysis is not a trivial task to be performed manually by software engineers. It requires specialized supporting tools. Over the last years, several techniques and tools have been developed to perform change impact analysis in the SPL evolution context [5][6][7]. Besides the fact that these research works bring new advances to the area, there are still limitations to allow the customization of existing strategies, and to add new types of analysis to be performed. For example, the proposed tools are not flexible enough to allow: (i) the incorporation of new types of assets and dependencies between them; (ii) the definition of specific and customizable analysis considering information of variabilities and their dependencies; and (iii) the implementation of techniques that support the analysis of dependencies among code assets.

In this scenario, this paper presents Squid – an extensible infrastructure for analyzing software product lines implementations. It works with information from: (i)

commonalities and variabilities extracted from variability management tools; (ii) mapping between variabilities and code assets extracted from variability management tools; and (iii) dependencies and relationships among code assets extracted using static analysis tools. The Squid infrastructure defines an extensible model of analysis, named Squid Analysis Model, which is used to store assets and respective relationships that model and implement the SPL.

The remainder of this paper is organized as follows: Section 2 presents an overview of Squid infrastructure, architecture and main modules. Section 3 describes an example of Squid infrastructure instantiation. Finally, Section 4 concludes the paper and presents directions for future work.

2. Squid Analyzer

This section presents Squid Analyzer – an extensible infrastructure for analyzing assets from software product lines implementations. Section 2.1 gives an overview of the Squid infrastructure. Section 2.2 illustrates the Squid architecture by describing its main components. Finally, Section 2.3 shows how to instantiate the Squid infrastructure to promote different kinds of analysis in SPL implementations.

2.1 Approach Overview

Our approach aims to provide an extensible infrastructure to the development of SPL analysis tools, which focuses on the relationships among assets from problem and solution space. The Squid infrastructure is implemented as an Eclipse plug-in that provides extension points to support the implementation of SPL analysis tools through the extraction of information from reusable assets, and thereafter the analysis of relationships and properties of these assets using query and visualization functionalities.

Figure 1 presents an overview of our approach. Initially (step 1), SPL existing assets are processed by our infrastructure to extract relevant information to be used in the subsequent analysis of its properties. Examples of information that can be extracted are: (i) code assets that implement the SPL; (ii) commonalities and variabilities specified, for example, using existing feature model specifications; and (iii) configuration knowledge that maps abstractions from problem space (e.g. features) to solution space (e.g. code assets). Most of this information can be extracted from existing models and artifacts specified using variability management tools, such as pure::variants [8], Gears [9] or GenArch [10]. The Squid infrastructure provides extension points to implement different kinds of extractors to get information from existing source models. At the end of these steps, all the extracted information is stored in the Squid Analysis Model.

In the step 2 of our approach, the Squid infrastructure can proceed with the source code static analysis in order to extract detailed information about the dependencies relationships among code assets. It enhances the information stored in the Squid Analysis Model, thus contributing to improve the subsequent analysis of SPL properties. Our infrastructure also provides extension points to connect with existing static analysis tools in order to promote the extraction of information about the code assets implemented in different programming languages and using a variety of libraries and frameworks.

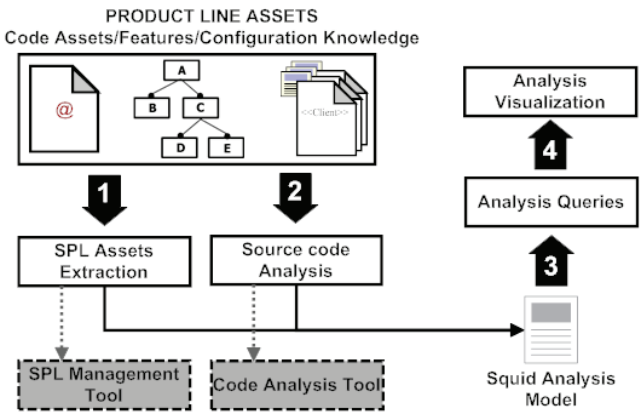

Figure 1. Squid Approach Overview.

After the creation of the Squid Analysis Model using different asset extractors and code analysis tools (steps 1 and 2), several analysis can be conducted to verify properties or search for specific information from the SPL assets by implementing or reusing specific queries and viewers for the infrastructure (steps 3 and 4). Examples of such analysis are: tracing from features to code assets, and change impact analysis of specific features or code assets.

2.1.1 Squid Core

The Squid infrastructure focuses on a flexible implementation that allows SPL engineers to develop their own extractors, code analyzers, queries and visualizers in order to address their project specificities and needs. The infrastructure also promotes the reuse of previously developed modules by other engineers, thus contributing to the instantiation of new SPL analysis tools.

The flexibility of the Squid infrastructure is addressed by means of extension points provided by classes from the Squid core module. Figure 2 shows a class diagram containing the Java classes that implement the infrastructure core, and interfaces that define the extension points. The `SquidAnalyserFacade` class represents the main core class that maintains references to objects of types defined by the following interfaces: `Extractor`, `CodeAnalyser`, `Query` and `Visualizer`. Each interface describes an extension point, which specifies the methods that engineers must implement in order to be explicitly called by the `SquidAnalyserFacade` class. The Extractor interface specifies an extension point to extracting SPL asset information, such as code assets, common and variable features, and respective mapping from features to code assets. The `CodeAnalyzer` interface defines the extension point to extract detailed information about the relationships and dependencies between code assets. The instance of the `CodeAnalyzer` interface must explore the SPL code assets searching for information about code assets dependencies. The `Query` interface describes an extension point to the development of analysis queries on the Squid Model Analysis Model. Finally, the `Visualizer` interface is an extension point to the construction of visualizations of analysis queries results.