On the Problems with Evolving [redacted] Software Product Line

Bartosz Michalik, Danny Weyns
DistriNet Labs, Departement of Computer Science
Katholieke Universiteit Leuven, Belgium
{bartosz.michalik,danny.weyns}@cs.kuleuven.be

Wim Van Betsbrugge
Resesarch & Development, [redacted] Automation
Antwerp, Belgium
[redacted]

ABSTRACT

[redacted] an industrial manufacturer of logistic systems is adopting a Software Product Line (SPL) approach to manage the development of their product portfolio. However, due to the intrinsic complexity of the logistic systems and lack of explicitly documented architectural knowledge evolution of the products is error-prone. Faulty updates increase maintenance costs and harm the company's reputation. Therefore, [redacted] searches for a systematic solution that can improve their SPL evolution strategy.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Maintenance

General Terms

on-line, evolution, SPL, software produt line

Keywords

Evolution, SPL, software product line

1. INTRODUCTION

[redacted][1] is an industrial manufacturer of logistic systems. The company is adopting a Software Product Line (SPL) approach. Their SPL evolved from previous separate solutions. Over time [redacted] widened its SPL scope by offering customers new features. However, the heritage of previously developed systems led to severe difficulties with the evolution of their SPL. In this paper, we report on the problems analysis that resulted from a joint project between [redacted] and DistriNet Labs at K.U.Leuven.

The key problem [redacted] encounters can be formulated thus:

> *How to correctly and efficiently evolve the Software Product Line in the context of incomplete architectural knowledge?*

[1]http://www.[redacted]

PLEASE '11, May 22-23, 2011, Waikiki, Honolulu, HI, USA

[redacted]s products are distributed systems deployed in production environments, therefore correct and efficient updates are key. A correctness of a system evolution means that the maintainer has to perform an accurate sequence of update steps to evolve the deployed system to the new version when it is updated. Efficiency of the evolution refers to the availability of the logistic system during the update. Typically, a logistic system has to operate 24/7, therefore minimal interruption of the running system should be ensured.

Managing the evolution of SPL and ensuring consistency of the changes in all affected products are both recognized as the key research challenges [10]. However, the evolution of SPL is a broad term so researches address different aspects. Bosch discusses the evolutionary and revolutionary approaches for SPL adaptation in an organization [3]. CompAS [5] is a method to analyse evolution in a features space. Krueger [8] discusses the 3-Tiered Methodology for SPL evolution. The base tier provides basic infrastructure for first-class variation management. The middle tier focuses on organizing the assets and the development teams around the reusable components and subsystems. Evolution is addressed in the top tier (Feature Based Portfolio Evolution) which manages the portfolio by features, not products. Dhungana [4] proposes a decision-oriented solution for evolving model-based SPL. This solution is based on the use of model fragments, and attempts to synchronize models with asset base elements automatically. Clone Miner [2] is an example of a tool that helps evolution at the code level. Despite these approaches, Babar [1] convincingly claims that systematic and sufficient support of evolution is limited.

In this paper, we focus on the evolution of deployed products, in particular the execution of the maintenance tasks to update deployed products.

The remainder of this paper is structured as follows. Section 2 gives a brief overview of [redacted]s SPL. Section 3 outlines the problems with its evolution. Section 4 shows the consequences of evolution of SPL with incomplete knowledge. A discussion follows which outlines future work.

2. [redacted]S SPL SETTINGS

[redacted] is a leading company that provides full life cycle support for logistic systems. Such systems are used for warehouse automation, e.g., for distributing manufactured products to storage locations or as an interprocess system between various production machines. A logistic system customized and optimized for specific customer needs is a *product.* [redacted] maintains more than 200 logistic systems de-

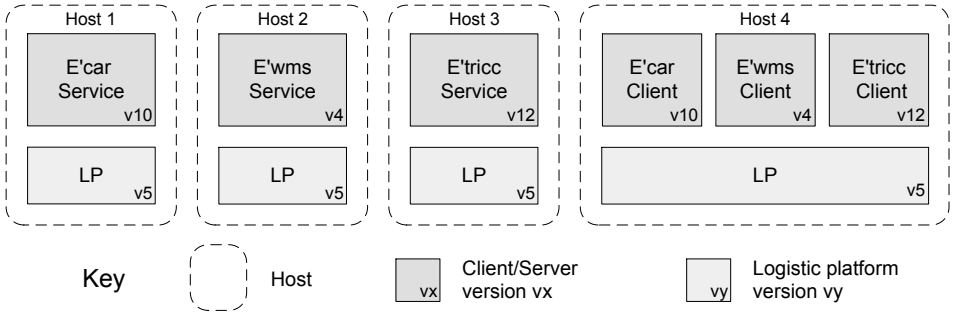

Figure 1: Typical configuration of a product

ployed at their customers sites across Europe. This number grows significantly each year.

A typical configuration of a logistic system is presented in Figure 1. The software is deployed on four hosts. Each logistic subsystem comprises a service and a client that makes use of the distributed logistic platform. The service offers the functionality of the subsystem, the client offers a graphical interface to access the service. The installation includes a warehouse management system ([redacted] warehouse management system) that is responsible for managing tasks in the system, and control software for various transportation subsystems such as automated guided vehicles ([redacted] transport intelligent control center), and cranes ([redacted] crane automatic storage and retrieval system). All subsystems consist of multiple interrelated components that can communicate.

Logistic systems are long-lived (typically 10+ years). During this lifespan, product evolves as a result of SPL assets evolution or because of changing customer requirements or environmental settings. Examples of typical evolution scenarios, that can be observed in the [redacted] maintainers daily work, are:

S1 A security submodule of a logistic platform is updated to remove a problem with the cipher mechanism found in version v_5.

S2 For reasons of performance, a collision avoidance component of the E'ttric subsystem is split into two parts starting from version v_{13}.

S3 A new crane device is bought from a 3^{rd} part provider by a customer and the control software of this device must be integrated with the existing configuration.

S4 A new statistical module for E'wms is offered to the clients that improves throughput. The module can be integrated with E'wms from versions v_4 on.

S5 Customers want to evolve their operating systems which may affect their deployed installations.

The goal of system evolution is to migrate the logistic system from its current version to the new version in a series of update steps. These steps include component additions, removals and replacements, as well as stops and restarts of affected processes. The required changes have different impact on a running configuration. For instance, only the server part of the E'ttric module needs to be updated in scenario S2, whereas the implementation of scenario S5 may require a full system restart.

Unfortunately, without a detailed architecture knowledge about the product under update performing the evolution is error-prone. In the next sections, we pinpoint the problems of [redacted]'s SPL evolution and discuss the consequences of imprecise knowledge for the evolution tasks.

3. [redacted]'S EVOLUTION PROBLEMS

[redacted]'s deployed products were derived at different moments in time and composed from variants of the same component. Therefore, knowledge about the exact configuration of a product is critical to ensure a correct and efficient evolution process. In fact, during the evolution of [redacted] products maintainers have varying information needs.

In scenario S1 the key issue is to identify the products that include the buggy security component and its dependencies with other components. Scenario S2 requires changes in the component structure of a deployed system. Therefore maintainers must have proper knowledge to integrate the new components in a correct way. With the right update strategy the interruption of the running system can be reduced significantly. Scenario S3 emerges from a customer requirement. The main issue is to determine how the new software components have to be integrated with the running system. Additionally, the integration of the new feature in the SPL may be considered. In scenario S4, a new feature is offered to [redacted]'s customers. Therefore, SPL-wide knowledge about existing installations is required to identify the customers for which the feature could be useful. Finally, in scenario S5 the runtime platform evolves. The impact of this change on the deployed systems has to be determined and addressed accordingly.

Unfortunately, in the current situation precise documentation of the deployed systems and traceability between deployed and assets base components is not available.

[redacted] SPL is an example of a Ploughed Fields Adoption of SPL Development [6]. The SPL emerged from a set of products previously offered by the company. As a consequence, [redacted] SPL contains a lot of legacy code which is not fully documented. Moreover, only a coarse-grained mapping between features and the code artifact is maintained.

[redacted] maintains more than 200 of deployed products, which generates more than 300 of the maintenance tasks each year. The subsystems are developed by different teams that work relatively independently. Moreover, most of the

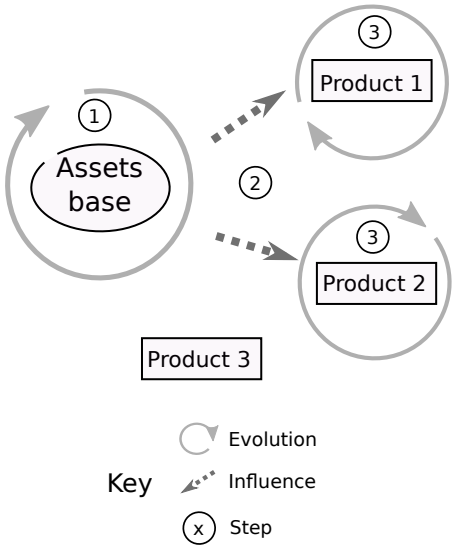

Figure 2: Evolution in [redacted]s SPL

architectural knowledge is personalized among team engineers. Therefore the knowledge of deployed products and their execution environment required to perform evolution tasks is not available to the maintainers team. Knowledge about the traceability between the SPL asset base and the product's components is also imprecise.

There are two evolution stages (see Figure 2) of [redacted]'s SPL in which accurate knowledge about a product under update is required.

One stage is *SPL-wide* evolution. During this stage products that are affected through the change have to be identified (step 2 in Figure 2). Knowledge about dependencies between deployed (customer optimized) assets and asset base (generic) components help to identify the affected products. For example, when a new statistical module is developed (S4) it should be offered only to the customers with the proper version of the E'ttric subsystem.

However, to evolve the product, the new components must be integrated with currently deployed product. This is a *product-specific* evolution (step 3 in Figure 2). The focus of this evolution's stage is to determine the correct and efficient sequence of update steps. It can happen that in the realization of scenario S1, a single component replacement is required in one product, while in another product, multiple subsystems have to updated. Precise knowledge about the deployed components and their interrelations is a prerequisite to evolve the product correctly.

In the following section we discuss the consequences of SPL evolution in the context of incomplete knowledge.

4. CONSEQUENCES OF IMPRECISE KNOWLEDGE

The problems with evolution of their SPL are known by [redacted] Lack of accurate knowledge of the deployed products of the SPL results in poor analysis with respect to product-wide evolution. Lack of the required knowledge of deployed products leads to ad-hoc update practices that are error-prone. Moreover, uncertainty about the product configuration make it difficult to achieve the availability and correctness requirements discussed in Section 2. Restarting a system in incorrect configuration may lead to the inconsistency and in extreme cases even to serious damage to industrial installations. That can harm the company's reputation.

To reduce the risk of incorrect update, maintainers team consist of highly skilled engineers from all production teams. Nevertheless, maintainers typically follow a defensive update approach which implies unnecessary shutdowns. This results in the higher cost of the update and deflects the experienced employees from their daily tasks. Moreover, this approach does not guarantee correctness of the update.

5. TOWARDS A SOLUTION

As presented in Section 3 we identified two stages of [redacted]s SPL evolution, namely the *SPL-wide* and *product-specific* evolution. Whereas we addressed the later, the former awaits a workable solution.

5.1 Product-Specific Evolution

The goal of the product-specific evolution is to migrate deployed product from the current version (*as-is*) to the expected one (*to-be*). The knowledge about the deployed components and their dependencies is a prerequisite to evolve the product correctly. Nevertheless, this information is missing. To address the problem we have built the EvoLine tool.

EvoLine address two important quality requirements inherent to the logistic product updates. First is correctness which means that a deployed system must be transformed to the new configuration without compromising its consistency. Second is availability of the system under update.

Our approach benefits from the research in the domain of software architecture reconstruction (SAR). SAR tools use source code [7, 9], historical information [9], or application management API [11] to construct various architectural views.

EvoLine reconstructs architectural models from the deployed resources. Central to our approach is an evolution viewpoint that defines four model kinds that deal with the various stakeholder concerns. As-Is (M1) and To-Be (M2) Product Deployment models allow stakeholders to browse the locations and structure of the deployed product (as-is) and the future product (to-be) respectively. Update Procedure Model (M3) shows the update steps that maintainers have to perform to evolve a deployed product, dealing with the availability and correctness concerns. Finally, Update Inconsistencies Model (M4) shows inconsistencies of the product, dealing with the correctness concern. The four model kinds are based on an integrated meta-model that offers the basis for an architectural repository.

The repository is populated with the information collected by the harvesters. A harvester is a small subprogram that extracts architectural knowledge from specific source(s). In the EvoLine tool three types of harvesters are used: (1) Assembly Harvester, that gathers information about components including their dependencies and versions; (2) Configuration Harvester collecting information about dynamic dependencies between deployed components; and (3) MSI File Harvester, that gathers information about to-be components from installation bundles.

The collected knowledge is analysed and visualized. Model M3 is obtained by comparison of the models M1 and M2 [12]. Second type of analysis identifies inconsis-

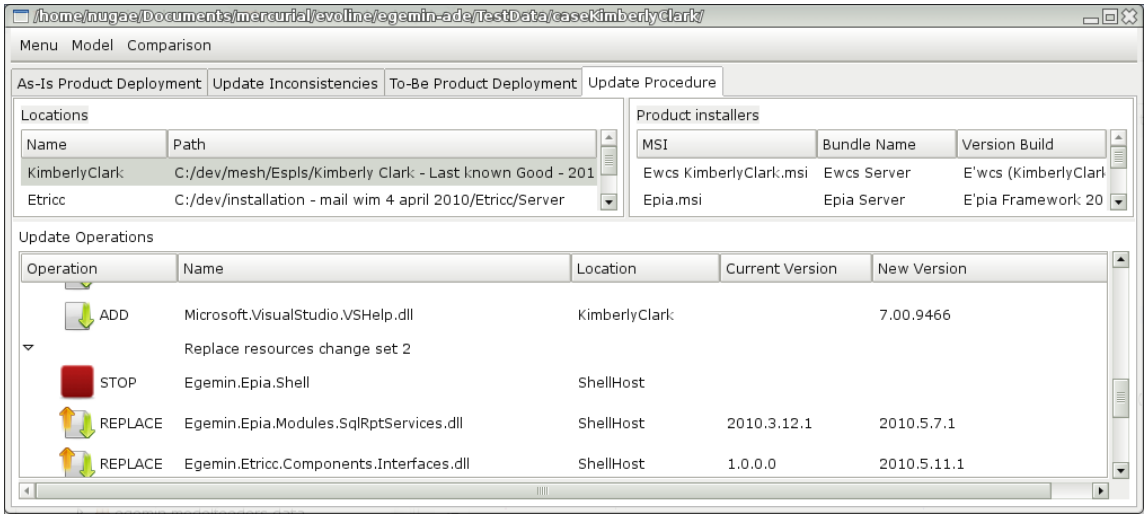

Figure 3: Example of an update procedure model.

tencies of the updated product (M4).

Figure 3 shows a snapshot of the EvoLine tool. In the figure an Update Procedure model for one of the products of [redacted]s SPL is presented. The box top left shows the different locations on which the product is deployed. The box on the right hand side shows the installation bundles (product installers) that have to be deployed on the selected location. The box at the bottom shows the update script that resulted from the analysis. The update script shows the subsequent update steps the maintainer has to perform to realize the update.

5.2 SPL-wide Evolution

The focus of SPL-wide evolution phase is on the change impact analysis of assets on the deployed products. Our preliminary idea is to extend our meta-model with the information about variability and configuration used to build a given variant of the logistic product. This information can be stored as a product meta-data, and generated during the product derivation phase. When one of the assets is changed the variability and configuration information can be used to determine affected products.

Unfortunately, the current variability models of [redacted]s SPL are not sufficient to be used in this approach. Therefore, refinement of these models as well as support for automatic analysis of change impact of the products are prerequisites to support SPL-wide evolution.

6. CONCLUSIONS AND FUTURE WORKS

In the paper, we pinpointed the problems [redacted] faces with the evolution of their SPL. Next, we characterize two evolution stages. In the SPL-wide evolution stage, the candidate products for updates have to be identified. In the product-specific evolution stage, a sequence of evolution steps has to determined to migrate the deployed product for its current configuration to the new, fully functional version. Unfortunately, accurate knowledge to support maintainers with both these evolution stages is not available. Being aware of the consequences of ad-hoc evolutions, [redacted] searches for a systematic solution to improve the evolution of their SPL.

Currently, we are looking at architecture reconstruction techniques that can be used to generate the required architecture models of the system under evolution. We have built a prototype tool that supports [redacted]s maintainers in the product-specific stage of the evolution. Developing a solution to support the SPL-wide evolution stage is our future work.

7. REFERENCES

[1] M. A. Babar, C. Lianping, and F. Shull. Managing variability in software product lines. *Software, IEEE*, 27(3):89–91, 94, 2010.

[2] H. A. Basit and S. Jarzabek. A data mining approach for detecting higher-level clones in software. *IEEE Transactions on Software Engineering*, 35:497–514, 2009.

[3] J. Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. *Software Product Lines*, pages 247–262, 2002.

[4] D. Dhungana, P. Grönbacher, R. Rabiser, and T. Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, 83(7):1108–1122, 2010.

[5] G. Douta, H. Talib, O. Nierstrasz, and F. Langlotz. Compas: A new approach to commonality and variability analysis with applications in computer assisted orthopaedic surgery. *Information and Software Technology*, 51(2):448–459, 2009. doi: DOI: 10.1016/j.infsof.2008.05.017.

[6] I. Gorton and I. Books24x7. *Essential software architecture*, volume 11. Springer Berlin, 2006.

[7] J. Knodel, M. Lindvall, D. Muthig, and M. Naab. Static evaluation of software architectures. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 10 pp. –294, Mar. 2006.

[8] C. Krueger. The 3-tiered methodology: Pragmatic insights from new generation software product lines.

In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 97 –106, 2007.

[9] M. Pinzger. *ArchView-Analyzing Evolutionary Aspects of Complex Software Systems*. PhD thesis, Vienna University of Technology, 2005.

[10] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer-Verlag New York Inc, 2005.

[11] H. Song, G. Huang, F. Chauvel, Y. Xiong, Z. Hu, Y. Sun, and H. Mei. Supporting runtime software architecture: A bidirectional-transformation-based approach. *Journal of Systems and Software*, In Press, Corrected Proof:–, 2010.

[12] D. Weyns and B. Michalik. Codifying Architecture Knowledge to Support Online Evolution of Software Product Lines. In *Sharing and Reusing Architectural Knowledge, 2011. SHARK'11. ICSE Workshop on.* IEEE, 2011.