The Journal of Systems & Software 144 (2018) 84–105

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

# Self-adaptation of service compositions through product line reconfiguration

Mahdi Bashari[a], Ebrahim Bagheri[*,b], Weichang Du[a]

[a] Faculty of Computer Science, University of New Brunswick ,Canada
[b] Department of Electrical and Computer Engineering, Ryerson University, Canada

A R T I C L E I N F O

*Keywords:*
Service composition
Feature model
Software product lines
Self adaptation

A B S T R A C T

The large number of published services has motivated the development of tools for creating customized composite services known as *service compositions*. While service compositions provide high agility and development flexibility, they can also pose challenges when it comes to delivering guaranteed functional and non-functional requirements. This is primarily due to the highly dynamic environment in which services operate. In this paper, we propose adaptation mechanisms that are able to effectively maintain functional and non-functional quality requirements in service compositions derived from software product lines. Unlike many existing work, the proposed adaptation mechanism does not require explicit user-defined adaptation strategies. We adopt concepts from the software product line engineering paradigm where service compositions are viewed as a collection of features and adaptation happens through product line reconfiguration. We have practically implemented the proposed mechanism in our*Magus tool suite* and performed extensive experiments, which show that our work is both practical and efficient for automatically adapting service compositions once violations of functional or non-functional requirements are observed.

## 1. Introduction

Many service and API sharing platforms such as ProgrammableWeb index thousands of web services which are readily available to be used by developers. The growing number of such services and their relatively easy utilization has motivated researchers to create methods and supporting tools to build composite services (Lemos et al., 2016). Given the high variability of services and the abundance of their variations, researchers have proposed that the development of service compositions can, among other ways, be performed through Software Product Line (SPL) engineering techniques in two lifecycle phases, namely *domain engineering* and *application engineering* (Pohl et al., 2005). In the domain engineering phase, a domain expert would organize the functional aspects of the domain through an SPL variability modeling mechanism such as a *feature model* (Lee et al., 2002). This will include the definition of the domain functionality and the services that can implement it. In the application engineering phase, the user specifies her requirements by selecting a valid subset of the features from the variability model. On this basis, in our previous work (Bashari et al., 2016), we have proposed a method to facilitate the composition of services by enabling the user to express her requirements in terms of software product line features. Our method would then automatically build service compositions represented in the form of executable BPEL code based on the selected features.

In this paper, we are focusing on another aspect of service composition, which deals directly with the practical execution of service compositions at runtime. Considering that service compositions often rely on open API and online services, their functional availability and non-functional guarantees are highly dependent on the availability and performance of the services that were used to build them. Therefore, changes or failures in the constituent services can affect the functional and non-functional guarantees of the service composition. In order to handle such situations, we suggest enabling self-adaption. Self-adaptation is the ability of a system to react to changes in its environment to maintain service and is used in different problem domains such multi-agent (Jiao and Sun, 2016), cyber-physical (Gerostathopoulos et al., 2016; Chen et al., 2017), and even industrial software systems (CÃ!'mara et al., 2016). We propose enabling self-adaption in order to allow a service composition to *self-heal* (Kephart and Chess, 2003) in response to such failures. Although researchers have been working on various methods for enabling self-adaptation in the BPEL domain, our work is still timely considering that BPEL is currently extensively used for defining business processes in industry and many recent work are focused on enabling adaption for BPEL processes (ai Sun et al., 2018; Alfrez and Pelechano, 2017; Margaris et al., 2016; 2015). The proposed work is also relevant considering that it addresses limitations in existing work by enabling service compositions to *autonomously* adapt at runtime to recover from failure. More specifically, our work addresses the

* Corresponding author.
*E-mail addresses:* mbashari@unb.ca (M. Bashari), bagheri@ryerson.ca (E. Bagheri), wdu@unb.ca (W. Du).

https://doi.org/10.1016/j.jss.2018.05.069
Received 25 October 2017; Received in revised form 2 April 2018; Accepted 31 May 2018
Available online 06 June 2018

following challenges in the state of the art:

- In some of the existing self-healing methods (Cetina et al., 2009; Subramanian et al., 2008), it is the developer's responsibility to design adaptation strategies. Designing adaptation strategies tend to be complicated and prone to error given the diversity of ways or circumstances under which a service or a collection of services can fail (Lemos et al., 2013).
- Many of the existing work adopt an *all-or-none* approach in healing functional failures where they try to fully recover functionality and fail when full recovery is not possible (Hristoskova et al., 2013; Angarita et al., 2016).
- Some of the methods only focus on maintaining either functional or non-functional requirements (Hristoskova et al., 2013; Canfora et al., 2008). However, an effective mechanism should be able to consider both types of requirements at the same time when performing adaptation since both of these types of properties can be critical for the system's operations (Tan et al., 2014; Angarita et al., 2016).

A real-life example which can show the shortcomings of existing approaches is a flight booking website. Such websites often provide additional features such as hotel and car rental services, which provide discounted price based on the destination and the selected airline. These additional features are not critical although being desirable. These features are typically implemented as a holistic process, which will break if the services realizing the desired but not critical features do not perform properly. To address such scenarios, existing self-adaptive approaches focus either on finding alternate ways to realize these features, which is not always possible or leave the task of devising appropriate mitigation strategies to the developer, which could be complex and labour-intensive. In our work, we address failure by removing the feature(s) which caused the failure if they are non-critical (such as hotel booking or car rental) while ensuring integrity of the whole process. As a further example, it might be expected that the ticketing process finishes in less than a specific amount of time in order to guarantee customer satisfaction. Existing approaches focus on optimizing the time-to-completion of the process through alternative services or processes which may or may not result in meeting the specified constraint. In our proposed approach, we satisfy such constraints by removing the minimal set of non-critical features which guarantee the time-to-completion constraint of the system being satisfied (e.g., automatically removing hotel booking and/or car rental when they take much longer than expected).

More concretely, we propose a *Dynamic Software Product Line (DSPL)* engineering-based method (Hallsteinsen et al., 2008; Bosch and Capilla, 2012), which enables a service composition to adapt automatically and recover from violations of functional and/or non-functional requirements without the need for the adaptation strategies to be explicitly defined by the experts. Dynamic software product line engineering methods use software product line models and methods at *runtime* to satisfy or maintain requirements (Montalvillo and DÃaz, 2016). We propose a method based on feature model reconfiguration techniques in software product line engineering to enable automated re-selection of features such that all critical functional and non-functional requirements are subsequently recovered after failure. The concrete contributions of our work are enumerated as follows:

- We propose an automated failure mitigation method, which focuses on finding an alternate feature model configuration for the service composition that recovers critical functional and non-functional requirements. This method is able to find an alternate service composition to replace the failed service composition.
- In order to enable finding an optimal feature model configuration with desired non-functional properties at runtime, we propose a method which is able to estimate the effect of each selected feature on the non-functional properties of the service composition. Additionally, the proposed method is able to update its estimates at runtime as the non-functional properties of the constituting services of the service composition change.
- We have implemented the proposed method and added it to our existing service development suite, called *Magus*. The tool suite allows for the specification of the product line representation of the domain, as well as the modelling of the desired functional and non-functional requirements. Magus will automatically generate executable BPEL code, continuously monitor the execution of the generated service composition and adapt it as necessary.

It should be noted that the work in this paper is an extension of our earlier work (Bashari et al., 2017b) and extends it in the following ways that are not addressed earlier: (1) In this paper, we propose a systematic approach for calculating how software product line features can impact the non-functional properties of a service composition and how they can be continuously monitored, estimated and maintained; (2) We present an algorithm, and formally prove its desirable characteristics, for finding linearly independent feature subsets within a software product line feature model. Linearly independent feature sets are important since a non-functional property of a service composition can be estimated as a unique linear equation over the availability of one of the features in each of these subsets; (3) We extend the formal representation of constraints within the context of psuedo-boolean optimization to cover three distinct types of constraints, namely the constraints defined over non-functional properties of service composition by the user, the constraints defined over combination of features which describe a valid service composition, and the assumptions made over input data by the failed service; and (4) We introduce our fully functional publicly available online tool suite and also provide an extensive comparative analysis of the literature beyond what was covered earlier.

The rest of this paper is organized as follows: Section 2 provides a general overview of the techniques that are used in this paper along with an introduction to the running case study. This is followed by Section 3, which provides an overview of the proposed approach. The details of this approach is presented in two sections. In the first section (Section 4), we discuss how features and non-functional properties are related to each other while the adaptation mechanism is proposed and discussed in a subsequent section (Section 5). In Section 6 the architecture used to implement the proposed approach has been discussed. This section is followed by Section 7 in which we go through the functionality provided by our tool suite. In Section 8, the design details of the experiments for evaluating the proposed approach have been presented and our findings have been reported. In Section 9, the proposed work is compared with existing works in both self-healing software systems and dynamic software product line engineering. The paper is finally concluded with a discussion of lessons learnt, threats to validity as well as a summary of the findings and directions for future work.

## 2. Background

In the following, background on feature models, how service compositions can be contextually modeled, and how automated composition of services can be performed, will be provided.

### 2.1. Feature models

Feature models are among the more popular models used in the SPL community for representing the variability of the problem domain (Benavides et al., 2010). Feature models allow for hierarchical representation of features that are related to each other through structural and/or integrity constraints. The structural constraints relate features to their parents through Mandatory, Optional, Alternative, and Or relations. Mandatory children of a feature must be selected when their