A Model-Driven Approach for Managing Variability in Service-Oriented Environments

Sandile W. Dlamini, Paul Tarwireyi and Matthew O. Adigun

Department of Computer Science
University of Zululand, Private Bag X100
1 Empangeni 3886, South Africa
sandilewilmoth@gmail.com

Abstract:

The service-oriented architectural (SOA) style is becoming one of the preferred distributed computing paradigms for designing and developing systems that are characterized by coarse-grained services and service consumers. One of the key motivating factors behind the adoption of this paradigm is the ability to deal with service changes and variability. In SOA, services could be consumed by many consumers with different functional and quality requirements, thus they (services) cannot remain static or unchanged. Services in SOA environments have to constantly evolve in order to address different consumers’ requirements. Thus, the need for agile solutions that react to requirements and context changes is becoming more fundamental; to guarantee an increase in usability and applicability of services to their consumers. However, due to the distributed and dynamic nature of SOA environments and the fact that SOA requires a fine-grain solution, the development of tools and mechanisms for managing service changes and variability is still at an early stage (a topic of ongoing research). This paper presents our proposed model-driven approach for managing service variability in SOA.

Keywords-
GUISET (Grid-based Utility Infrastructure for SMME Enabling Technology), Managing Variability, Model-Driven Development (MDD), Service-Oriented Architecture (SOA), Software Product Line Engineering (SPLE)

I. INTRODUCTION

Service-Oriented Architecture (SOA) is an architectural style that has become one of the ideal paradigms for designing and developing systems that are characterized by coarse-grained services and service consumers [1]. Services are self-contained, loosely-coupled business functions or software artifacts that can be advertised, discovered and invoked in order to deliver certain functionality as standalone entities or combined together to produce value-added applications [2]. According to [3], the purpose of SOA is to address the requirements of loosely-coupled, standard-based and protocol independent distributed computing, by mapping enterprise information systems to the overall business process flow. One of the key known advantages of this (SOA) paradigm is to deliver flexible IT solutions that have the ability to react quickly and economically to changing business and/or consumer requirements. In SOA, services could be consumed by many consumers that have different requirements, thus they cannot remain static or unchanged. Services in SOA environments have to constantly evolve in order to address different consumers’ requirements; some consumers may want stability in services while others may want services to change as their business changes or because of their application and business context.

The Grid-based Utility Infrastructure for SMME Enabling Technology (GUISET) research infrastructure proposed by [4], is an infrastructure which aims at addressing the problems of software and
hardware acquisition experienced by resource constrained enterprises such as Small Medium and Micro Enterprises (SMMEs) in Africa. The concept of the GUISET infrastructure is based on the idea that there is the need for a technology or a platform that would enable SMMEs to have access to IT services on-demand, without owning the infrastructure on which the services are running. GUISET leverages on service-oriented computing to enable services to be discovered and invoked in order to support the business processes of SMMEs. In this infrastructure services are shared among consumers with different needs, thus as mentioned previously, services in SOA environments have to continuously evolve because of the variability in service consumers’ requirements. Hence, the need to support and manage service changes and variability in service-oriented environments like GUISET is strongly required, with a view to increase the usability and applicability of services to their consumers.

Reference [5] defines service variability as “the ability of a service to be efficiently changed, extended, customized or configured for use in a particular context”. Variability has been studied and/or relates to many research disciplines such as, Object-Oriented Databases (OOD), Component-Based Engineering (CBE), Semantic Web, and Software Product Line Engineering (SPLE), but relates the most to SPLE discipline. SPLE is a discipline for building a diversity of software-intensive products that share a common set of features while allowing a specific margin for differentiation to satisfy diverse customer needs [6]. The main goal of this engineering discipline is to reuse assets from all stages of the development life-cycle. This ambition is parallel to the one of SOA, which is to promote flexible deployment and reuse. However, despite the fact that SOA and SPLE share a common goal, they differ in various ways. First, in SPLE components are developed a priori and in most cases by the same developing organization, whereas services in SOA are usually developed and deployed by external service providers who are not aware of their potential consumers. Second, SPLE employs reuse in all the stages of the development life-cycle using all kinds of assets, while in SOA, only services are usually reused and in most cases when composing a composite service. SOA also relies on industry best practices in order to model variability, while SPLE have explicit techniques with regard to modeling variability [7]. Examining these differences, it is apparent that in order to deal with variability in SOA, inspiration and valuable techniques can be drawn from SPLE discipline. Hence, existing works such as [8-10] have already applied SPLE concepts to address variability in SOA.

One of the widely used approaches for dealing with service variability in SOA is service versioning. According to [11], versioning in SOA is very important because of re-usability and evolving services over time. Service versioning equates to the co-existence of multiple variants of the original service, which enables consumers satisfied with the original service to continue using it unchanged, while ensuring that a new variant of a service is created to meet the needs of consumers with different requirements. Another approach for managing service variability is through service customization; whereby consumers are permitted to perform customizations in order to generate specific service variants suiting their application and business context. In the former approach, different services satisfying specific consumers’ requirements are deployed as individual services. As a consequence, consumers are the one responsible for searching and invoking services suitable for their application and business context. The latter allows service consumers to perform run-time customizations in order to generate specific services satisfying their needs.
This work is funded by The Technology and Human Resources for Industry Programme (THRIP) and Telkom Centre of Excellence for Mobile e-Services under the project GUISET Service Evolution & Deployment Framework

Looking at these two approaches, if there exist a small number of service consumers requiring different services, the first approach is considered to be simpler and does not require much effort from the service consumers’ side. However, if the number is very large, the second approach becomes more appropriate, due to the fact that developing and deploying individual service variants for each service consumer or for meeting all possible consumers’ requirements will result in computing resources being wasted. Also, the development effort will be wasted because the commonality of the core requirements will not be exploited to support reuse [12]. Consequently, economies of scale will not be achieved. The rest of this paper is ordered as follows: In Section II we provide a short discussion on model-driven developmental methodology and feature modeling. Section III discusses related works. In Section IV, we give an overview of our approach and conclude the paper in Section V.

II. BACKGROUND

A. Model-Driven Development

Model-Driven Development (MDD) is a methodology in software development which focuses on making software artifacts more abstract. The main goal of this developmental methodology is to increase automation, simplify the process of design, increase re-usability, and productivity by maximizing compatibility between systems [13]. One of the most popular realizations of MDD is the Model-Driven Architecture (MDA). MDA is one of the Object Management Group (OMG) initiatives, which provide standards and guidelines for supporting model-driven construction of software artifacts.

B. Feature Modeling

Feature modeling is a variability modeling technique for determining prominent and distinctive features of software products in software product lines. A Feature is defined as any software product characteristic that is present in one or more members of a product family. In product line engineering, feature models are at present the de-facto standard for capturing and representing software product features [14]. A more comprehensive discussion regarding feature modeling and feature models can be found in [15].

III. RELATED WORKS

There are various number of works that have been proposed in the literature for dealing with variability in SOA. Reference [16] proposed an aspect-oriented approach for the adaptation of services. A framework for customizing Web service compositions was proposed by [17]. The authors in [18] introduced a policy-based framework for web service customization. In [19-20], the authors borrowed SPLE variability modeling concepts in order to facilitate service customization. However, their approach focused only on modeling variability at a technical level, not considering variability at requirements level. Also, their approach does not take into account adapting the internal implementation of services. Reference [12] also adopted variability modeling concepts from SPLE to facilitate service customization. An approach to extend SoaML (Service-Oriented Architecture Modeling Language) was introduced by [21] in order to support the notion of variability. Reference [22] also
extended SoaML for service adaptation based on consumers’ views. A multiple-view modeling and meta-modeling approach for service-oriented product lines using UML was described in [9].

After reviewing all these works, we found that these works only focused on managing variability in service consumers’ functional requirements, not taking into account the non-functional requirements. Service consumers usually don’t have varying functional requirements, their non-functional requirements also vary; and according to [23] a reasonable service must meet both the functional as well as the non-functional requirements of its consumers. Moreover, non-functional properties give consumers assurance and confidence that they are using the best service. These works also do not consider managing the life-cycle of service variants, thus they do not reuse service capabilities (i.e. operations, business protocols, etc) hence re-usability is not promoted. In SOA the demanding requirements of service consumers increase the cost involved in building or generating variants of a service, thus it is important to reuse service capabilities in order to support different application scenarios. Our work proposed to take into account both the functional and non-functional properties of a service in managing service variability. The approach proposed in our work is also exploiting techniques from SPL E, particularly feature modeling techniques, and the use of SoaML, a standard language for modeling SOA solutions.

IV. APPROACH OVERVIEW

This section gives an overview of our model-driven customization framework (see Fig.1 below). We illustrate how customized services meeting specific consumers’ requirements are delivered at run-time. Our solution adopts the MDD approach to automate almost all the parts and/or operations of the framework. The steps involve in our framework for the provisioning of customizable services and promoting reuse are described below:

a) **Service Providers**: the first step involves the service provider creating and publishing service descriptions (e.g. service interfaces) into the service registry. In this case, the service description artifacts are considered as feature models, which incorporate both the functional and non-functional characteristics of a service. Feature models are used to facilitate the customization steps and to capture the core and varying

![Figure 1. Framework Overview](image-url)
requirements of service consumers. The other reason feature models are considered description artifacts is that the service interfaces that would be used by service consumers would not be the same, this is due to the fact that service interfaces are allied to business requirements and they would be the product of the customization process.

b) Service Consumers: the second step involves service consumers searching for services they need to consume in the service registry. Once a service of interest is discovered and located, service consumers then retrieve a feature model of that particular service from the service registry and start selecting or providing all the functional and non-functional features they want their services to include. This information is then sent to the Feature Management Engine (FME).

c) Feature Management Engine: this engine uses analysis techniques of feature models [15] [24] and is responsible for checking whether the provided customization information by service consumers through feature models, does not violate the functional and non-functional properties provided by service providers in their feature models. If the customization information is valid, it is forwarded to the service providers’ side else an invalid message is sent to the service requesters. Before a service is created, a service provider first check on the Variants Registry (VR) if there exist any service variant matching the requesters’ customization information. If there is any and is still running that service’s endpoint would be communicated to the service requesters and the service requesters would then bind, invoke and start using the features of the service in their usage scenarios. Otherwise, a different service variant would be derived and before it is deployed it is registered in the VR.

d) Variants Registry: this registry is responsible for storing and keeping track of the already existing and deployed service variants. VR manages the life-cycle of existing service variants by containing information about their life-time status for example, running, expired, and so forth.

V. CONCLUSION AND FUTURE WORK

SOA is emerging as one of the ideal computing paradigms for building large and dynamic distributed systems. However, due to variability in consumers’ requirements and with the continuing advancement of this paradigm, service providers are facing a lot of problems. First, in order to draw a significant number of consumers, they need to take into account the varying requirements. Second, to achieve economies of scale, they need to ensure that different services preserve enough commonalities. Therefore, the need of tools and mechanisms for supporting and managing service changes and promote reuse in service-oriented environments is very important. This paper gives an overview of our model-driven customization framework for managing variability in consumers’ functional and non-functional requirements; so as to deliver adaptive services and promote reuse in service-oriented environments. Since our research is ongoing, the next steps involve providing our detailed service variability model, meta-modeling and implementation, which would be validated by a case study.

REFERENCES


