Towards a Model Driven Approach for variability management in SOA

B. CHAKIR, M. FREDJ
ENSIAS, Mohammed V Souissi University, Rabat, Morocco.
c_boutaina@yahoo.fr
fredj@ensias.ma

Abstract— Service Oriented Architecture (SOA) is an emerging architectural style for the development of interoperable, large scale and distributed software systems. But with the continuing evolution of information systems, the need of agile systems that respond to the requirements, conditions of execution and context changes is becoming crucial. Hence, the need of mechanisms that support change and that encourage the reuse in SOA systems is essential. So, it is important to consider the concept of variability. Indeed, the variability refers to the ability of a system to adapt, specialize and configure itself with the context of use. Several proposals have been submitted in this sense, but they are still immature and incomplete. Our proposal is to provide a complete approach that enables the development of SOA solutions taking into account the management of variability in all lifecycle stages. The principle of our approach is to divide the SOA development process into two sub-processes: basic process and variability process. Besides, to improve our method, we adopt a model driven approach, which adds formalism and offers models transformation to our method. Especially, we propose a UML profile in order to model functional and non-functional variability of SOA systems.

Keywords— Adaptable, Model Driven Architecture, Model Driven Development, Service modeling, Variability.

I. INTRODUCTION

A service-oriented architecture (SOA) is a style of architecture based on the description of services and their interactions. The main properties of a SOA solution are the loose coupling, independence from the technological aspects and scalability [1]. The property of loose coupling implies that a service does not call directly another service. Therefore, it is easier to reuse a service since it is not directly linked to others. The independence from the technological aspects is achieved through the use of contracts, which are independent from the technical platform used by the service provider. Finally, scalability is made possible by the possibility to invoke and discover a new service at runtime [2].

Moreover, with the continuing evolution of information systems, the need of agile systems that respond to the requirements, conditions of execution and context changes is becoming necessary. Hence, it is essential to dispose of mechanisms that support changes specification which is called variability. Thus, the aim of service variability is to provide one central technique for better supporting the reusability of services in different application scenarios and to simplify the service consumption by consumers [3].

In this paper, we propose a model driven engineering method for the management of service variability. Such an approach is mainly based on a UML profile that we call VarSOAML (based on SOAML profiles [4]). It takes into account functional and non-functional variability.

The paper is structured as follows: Section II introduces model driven engineering. Section III explains variability management principles. In Section IV, we list some related works. Our method and also the UML profile VarSOAML, are presented in Section V. Finally, we conclude and give some perspectives to our work in Section VI.

II. MODEL DRIVEN ENGINEERING

Model-driven engineering (MDE) is a software development methodology which focuses on creating models. Its purpose is to increase productivity by maximizing compatibility between systems, simplifying the process of design, and promoting communication between individuals and teams working on the system [5]. In our work we adopt the Model Driven Architecture (MDA) introduced by the Object Management Group (OMG) [6]. To proceed to the MDA transformations, we propose a UML profile based on SOAML [4], which is an OMG proposal for modelling SOA solution.

A. MDA

The architecture of MDA is organized into four levels, which are respectively [6]:

- M0 level: corresponds to the real world. This represents the real information of the user.
- M1 level: consists of information models, which allow the description of information M0. Any model is expressed in one language, whose definition is explicitly provided in M2 level.
- M2 level: is composed of meta-models. A meta-model can be defined as a model of a modeling language; which serves to well express the common concepts of all models of a domain [7].
- M3 level: consists of the MOF, also called meta-meta-model which is the language for defining meta-models [6].

MDA proposes the following models [6]:
• CIM (Computation Independent model): considers the system as a black box and allows the description of the organization, the flow and the actions.
• PDM (Platform Description Model): a PDM contains information for the model transformation to a specific platform.
• PIM (Platform Independent Model): it is business-oriented and describes the processing.
• PSM (Platform Specific Model): describes the technical details of a PIM, related to the implementation.

In order to proceed to MDA models transformations, we can use the concept of UML profile which will be explained in the next point.

B. UML profile

A UML Profile allows the adaptation of UML to a domain. It is based on stereotypes, values marked and constraints [8]. Stereotypes allow adding new elements to the meta-model. Moreover, values marked (tagged values) enable the addition of properties to a meta-class and constraints allow the addition or modification of rules [8]. In our work, we based our proposition on the SOAML profile, which was introduced by the OMG as a future language for SOA modeling as explained in the next point.

C. SOAML profile

The SoaML [4] profile proposed by OMG (Object Management Group) extends the UML2 meta-model to support the explicit service modelling in distributed environments. This extension aims to support different service modelling scenarios such as single service description, service oriented architecture modelling, or service contract definition.

The construction of SoaML uses the UML package merge relationship which is a direct relationship between two packages [9]. This relationship indicates that the contents of the two packages are to be combined, which means that the source package conceptually adds the characteristics of the target package to its own characteristics resulting in a package that combines the characteristics of both. Also, it is introduced by UML2 as the way to extend an existing meta-model with additional information.

Furthermore, modeling concepts in UML are partitioned into four layers called compliance levels [9]:
• “L0”: is the result of the merge of the basic package from the UML infrastructure and the primitive types. Package L0 contains basic concepts such as Class, Package, DataType, Operation, etc.
• “L1”: adds to level 0 additional packages like interfaces, BasicBehaviours or Dependencies.
• “L2”: level 1 and their contents are extended with additional packages like BasicComponents, profiles or ports.
• “L3”: adds to L2 language units for information flows, templates and model packaging.

Therefore, SoaML is an empty package that merges the UML L3 package with the service package defined in SoaML standard, see (Fig.1).

![Fig.1 UML/SOAML Meta model approach [10]](image)

After having seen the principal concepts of MDE used in our work, we will detail the concept of variability in the following section.

III. VARIABILITY MANAGEMENT

A. Definitions

The variability has been studied in several areas such as engineering domains, product lines, components, semantic web, etc. Also, according to each application domain, this concept takes different dimensions. For instance, in the engineering of product lines, the notion of variability is used to group the features that differentiate the products from the same family (the management of this variability is indeed the first activity for the development of lines of products [10]).

Besides, according to [10], “Variability management encompasses the activities of explicitly representing variability in software artefacts throughout the lifecycle, managing dependencies among different variation points, and supporting the instantiation of the variability”.

B. Properties

For the management of variability, we must take into account the following properties:

• Variability identification: according to [11], the identification of the variability is related to the definition of the characteristics that differentiate similar applications. Two concepts are used by [12]: variation point and variant. A variation point is a place of the system where choices must be made to identify variants to use [13].

• Variability types: there are several types of variability. In [14], the authors identify four types of variability of Services: Workflow, Composition, Interface and Logic.

• Variability scope: the scope of variability implies the scope of variation points. The authors of [15] define four types of variation: option, alternative and optional set of alternatives.
C. Stages of variability management

Generally, there are five stages in the management of variability [10]:

• The definition of variation points in the basic artefacts.
• The definition of elements that may potentially be related to these variation points.
• The definition of the relations between these elements and the variation points.
• The definition of constraints between potential variation points that may affect the instantiation.
• The specification of the mechanism of resolution of different variants.

We can categorize these stages into two main categories: modeling and representation as well as resolution and adaptation.

In the next section we will list some works related to the variability in SOA and their limits in order to introduce our proposal.

IV. RELATED WORK

Several works have dealt with the variability in SOA, and this under two perspectives: i) the variability for reuse, where we try to make explicit the variability in design artefacts, to reuse at the service provider level, ii) the variability for adaptation, which focuses on the context of use, and allows the customization of the solutions offered by the system based on user needs at runtime.

Among the works reviewed, in the perspective of variability reuse, we find the representation of taxonomies of variability [14]-[16], the representation of variability in services interfaces [14], the variability requirements studies [17]-[18], and the proposals of development process that manage the service variability [19]-[20].

In the perspective of variability for adaptation, we mention [9], which proposes an extension of SOAML (UML Profile and Meta-model for Semantically-enabled Heterogeneous service oriented) that supports the notion of variability, integrating two aspects: the configuration and the extension. The configuration is the adaptation of the service by the client through selecting the appropriate features. The extension allows the client to add new functionalities and features. Reference [21] proposes a UML profile: VSOAML for the adaptation of services to the user view. Authors [22] propose an aspect oriented framework for the adaptation of services. In [23], a framework is proposed for the customization of web services compositions.

By reviewing these works, we found the absence of a comprehensive approach that supports variability in different lifecycle stages. Moreover, most approaches focus on one aspect of services variability: functional or non-functional, and respond only to one perspective: reuse at the provider level or adaptation by the user. So, our proposed method considers both the functional and non-functional aspects of variability, and this, throughout the lifecycle of SOA solutions.

V. METHOD OVERVIEW

A. Framework for the management of variability in SOA systems

In order to manage the variability, we propose a framework to the development of SOA solutions, that takes into account the variability (functional and non-functional), and this, in different lifecycle stages. Our framework will allow both reuse, and the adaptation of services.

The proposed modeling method (see Fig.2) is based on model driven architecture (MDA) which enables the automation and the formalization of the method. In our case, we use two models: the PIM (Platform Independent Model) which allows a neutral specification of the system and PSM (Platform Specific Model), which contains business process and services models depending on a specific platform. The main steps of the method have been detailed in [24]:

• Requirements specification: it is based on domain analysis and users requirements. Also, it allows the identification of business process and the separation between functional and non-functional requirements. Finally, it allows the development of the features model that contains common features and variable ones. The fixed elements trigger the “basic process” of the method and the variable elements are the subject of the “variability process”.

• Basic process: represents the process that provides the basic services (atomic or orchestrated), that don’t take into account the specification of the variability. The main activities are: services analysis and design, services realization and services implementation.

• Variability process: is the process that handles variability, in order to get reusable solutions. The principal activities are: variability analysis and design, variability realization and variability implementation.
In order to apply MDA to our framework, we propose a UML profile to provide common concepts for modeling services variability and also to be used in transformation mechanism. This profile will support functional and non-functional variability of services.

B. VarSOAML: a UML profile for variability management of SOA systems

We use the UML package merge relationship to create the VarSOAML profile. So, our profile is an empty package merging SoaML [4] with the extensions defined in two other packages: Variability package and NFR (Non functional Requirements) package (Fig.3)

- Variability package: describes the variability of services (Fig.4). We distinguish the following stereotypes:
  - BasicInterface: contains basic functionalities offered by the service as a set of operations.
  - VariantInterface: represents the variant capabilities of a given service as a set of variant operations.
  - ConfigurableInterface: composed of BaseInterface and a set of VariantInterface and represents a configuration of service capabilities.
  - VariableElement: specifies variation point. Scope attribute takes values: optional, required, and prohibited.
Rational attribute refers to the reason of choosing this VariableElement.

- Constraint: describes the different dependencies between variable elements. It can be required or optional.

- VariantOperation: contains operations that do not belong to the base interface.

- VariantType: contains types that do not belong to the basic interface.

- Fig 4. Variability package

- NFR package: contains elements that model non-functional requirements (Fig.5). The profile defines the following stereotypes:
  - QosCategory: classifies QoS characteristic. We adopt the proposition of (OASIS) [25], which distinguishes between twenty category (Business Value Quality, Service cost, Service suitability, Service After effect, Interoperability, Traceability, Performance, Security Quality…). 
  - QoSCharacteristic: represents a quantifiable aspect of a service (like latency).
  - QoSDimension: used in quantification of a quality of service characteristic. For instance, the quality of latency characteristic can be quantified as minimum latency.
  - QoSConstraint: a constraint that restricts a QoS characteristic. For instance, for a service, the maximum latency must be less than 0.02 seconds.
  - QoSStatement: groups a number of QoS constraints; for example, constraints related to connectors and constraints related to messages.
  - QoSProfile: specifies the qualities the service provides in real word.
  - QoSContract: presents the qualities the service should provide.
C. Case study

In order to illustrate our approach, we choose a simple example of payment services. As illustrated in (Fig.6), the payment service is structured of several services, like one service for encryption, and different services for payment (cash payment, card payment and cheque payment). The operations for accessing the payment service are defined in the “PaymentInterface” (Fig.7).
On the one hand, the service has operations that are almost optional. Indeed, the operation setAmount among <<BaseInterface>> and other ones are <<VariantOperation>>.

The <<SimpleVariantOperation>> as setCardId, SetCardCode and setOwnerName, permit to identify the card owner so we can declare a <<ComplexVariantOperation>> login, which contains the three operations. Besides, the use of the <<SimpleVariantOperation>> requestCardValidation needs the use of the <<SimpleVariantOperation>>: encrypt() and decrypt() for security.

On the other hand, we can apply some QOS properties to variant operations as shown in (Fig.8), where we declare a <<QosCharacteristic>>: LatencyForEncrypt applied for instance to encrypt() operation.

```
<<QosCharacteristic>>
<<QosDimension>>
   minumumLatency: real
   {unit(MaxLatUnit), direction('decreasing'),}
<<QosDimension>>
   maximumLatency: real
   {unit(MaxLatUnit), direction('increasing'),}
...

```

VI. CONCLUSION

In this paper, we have presented an effective model driven engineering approach for managing variability in SOA. Our method distinguishes between two sub-processes, “basic process” which provides basic services and “variability process” which takes into account the variation points, variants and dependencies in different basic models. In order to apply MDA to our method, we propose a UML profile called VarSOAML which considers functional and non-functional variability for the elaboration of the PIM model.

As future works, we plan to enhance our profile by the OCL constraints and validate our method by a detailed case study.

REFERENCES


M. Stollberg, M. Muth, Service customization by variability modeling, 5th International Workshop on Engineering Service-Oriented Applications (WESOA'09), (2009).


B. Chakir is a Software Engineer graduated from ENSIAS (2004) (National Higher School for Computer Science and System analysis), holder of an Extended Higher Studies Diploma from ENSIAS (2007) and “Ph.D. candidate” at ENSIAS. Her research focuses on management variability in SOA. She is a project engineer at the Ministry of Economy and Finance of the Kingdom of Morocco since 2004.

M. Fredj PhD in Computer Sciences. Professor in ENSIAS, (National Higher School for Computer Science and System analysis), Rabat, Morocco.