Variability as a Service: Outsourcing Variability Management in Multi-tenant SaaS Applications

Ali Ghaddar[1,2], Dalila Tamzalit[2], Ali Assaf[1], and Abdalla Bitar[1]

[1] BITASOFT, Nantes, France
{ali.ghaddar,ali.assaf,abdalla.bitar}@bitasoft.com
[2] Université de Nantes, LINA, France
ali.ghaddar@etu.univ-nantes.fr, Dalila.Tamzalit@univ-nantes.fr

Abstract. In order to reduce the overall application expenses and time to market, SaaS (*Software as a Service*) providers tend to outsource several parts of their IT resources to other services providers. Such outsourcing helps SaaS providers in reducing costs and concentrating on their core competences: software domain expertises, business-processes modeling, implementation technologies and frameworks etc. However, when a SaaS provider offers a single application instance for multiple customers following the multi-tenant model, these customers (or *tenants*) requirements may differ, generating an important variability management concern. We believe that variability management should also be outsourced and considered as a service. The novelty of our work is to introduce the new concept of *Variability as a Service (VaaS) model.* It induces the appearance of VaaS providers. The objective is to relieve the SaaS providers looking forward to adopt such attractive multi-tenant solution, from developing a completely new and expensive variability solution beforehand. We present in this paper the first stage of our work: the VaaS meta-model and the VariaS component.

Keywords: SaaS, multi-tenant, variability.

1 Introduction

In recent years, the tendency towards outsourcing IT resources that are not the key competencies of an enterprise has caused the appearance of new services providers type. These new services providers develop and maintain, on their infrastructures, such outsourceable IT resources while offering their access to enterprises through the web. The business model of such resources offering aims at providing a pay-as-you-go payment model for their use, where these resources can be provisioned and un-provisioned on demand. Such new outsourcing principle is gaining wide acceptance, especially in the small and medium enterprises (SME) segment. The reason for this acceptance is purely economic. Indeed, the majority of SME consider taking less risks by contracting the implementation of parts of their IT resources to a more experienced provider and find this option more cost effective. The term *Cloud Computing* summarises these outsourcing efforts and provisioning of services on demand. This term has emerged as the run-time platform to realise this vision [13]. More concretely, Cloud Computing is viewed as a stack containing these new types of services to be provided. That is to provide an Infrastructure as a Service (IaaS), also to provide a development Platform as a Service (PaaS), to finally provide a Software as a Service (SaaS). In the following, we will focus on this last service type.

SaaS can be defined as: "Software deployed as a hosted service and accessed over the Internet" [5]. In the model of this service, the provider offers a complete application, ready to be used by prospective customers. These customers subscribe to the application and use it via the web through a simple browser. Full details on the adopted implementation technology and the application deployment server are transparent. This type of solution is offered by SaaS providers as an alternative to the traditional software applications that require installation on the client side, as well as significant investments in terms of materials and physical resources. However, from a SaaS provider perspectives, the real benefits of SaaS begins when it is possible to host multiple customers on the same application instance, without the need of a dedicated application to be deployed and separately maintained for each customer. This approach is called *multi-tenant*, where tenant means a customer organisational structure, grouping certain number of users [3]. In fact, application tenants could be distributed across different geographic zones and belong to different industry verticals, and thereby, their requirements from the application may differ, increasing the need to support these specific requirements in addition to common ones. This inevitably generates an additional and important *variability* [15] management concern, which obviously distract SaaS providers focus from their core competences.

According to our experience, variability management in multi-tenant applications generally implies: (1) modeling such variability (explicitly documenting the variable features and the possible alternatives), (2) binding the variability model and storing each tenant customisations and choice of alternatives, (3) adapting the appropriate application behavior at run-time for a tenant, according to its stored customisations. This last key concern is tied to the application's architecture. Our work is conducted in a SOA context where the targeted multi-tenant applications are designed and developed following the service oriented architecture (SOA) model. So, such application adaptation usually concern its look and business-processes (services composition an orchestration). As these variability requirements may be distracting and disturbing issues for SaaS providers, we are convinced that they will gain to outsource, if their is a possibility, such variability management concern to a specific service provider. For this purpose, we introduce in this paper the concept of *Variability as a Service (VaaS)* model (in the same spirit of IaaS and PaaS models in terms of outsourced resources). The VaaS model implies the necessity of a new type of service providers: the VaaS providers. A VaaS provider that follows the VaaS model we propose, will permit to its customers (i.e. multi-tenant SaaS providers) to model and to resolve their multi-tenant applications variability on his infrastructure. We will explain our approach and its context in the following. The remainder of the paper is structured as follows.

Section 2 presents the needs of modeling variability in terms of concepts for modeling and managing; it mainly identifies the limitations that the variability modeling techniques from software product line engineering encounters to realise our objectives from variability outsourcing. Section 3 presents our suggested architecture, process and meta-model for the variability outsourcing and the VaaS model. Section 4 presents a case study from the food industry that we have revisit in order to apply the new approach. In Section 5, discussions and future works are detailed. Section 6 shows related works and Section 7 concludes the paper.

2 Needs of Variability Modeling

When many variable features in a software system are identified, variability modeling will become an important requirement to express these features. Generally, variability modeling consists on explicitly documenting, in the software system, what does vary and how does it vary, allowing thus to improve traceability of the software variability, as well as to improve variability communication with the customers [12]. However, since software systems are composed from different development artifacts, a variability concern may impact and cut across these different artifacts (i.e. in SOA-based systems, the same variability in the business-process may have effects on the GUI and may also requires the development and deployment of additional services). Modeling variability in each artifact separately and differently is a very bad idea because it makes us loose all the sense of variability tracking, and it makes representing dependencies between variability in these different artifacts very hard. In addition, the different way of modeling variability in each development artifact will complicates the reasoning about variability in the entire system. Therefore, in order to avoid these problems and to ensure strong expressiveness and management of variability, it is necessary to guarantee its representation in a separated, central and uniform model, while maintaining its relation with impacted artifacts.

2.1 Orthogonal Variability Models

Based on the need of tracing dependencies between variability in different development artifacts and to maintain the uniformity in variability definition and modeling, orthogonal variability models such as the OVM have been introduced in the software product line engineering (SPLE) [1, 12] domain. These orthogonal variability models represent variability in one central and separated model, and then link each variability information to one or many development artifacts. In [6], we have shown how OVM can be used to model variability in multi-tenant SOA-based applications. One of our main goals was to show that concepts for variability modeling from the SPLE can be used in a multi-tenant SaaS context. These OVM modeling concepts are resumed in the following:

- *Variation point (VP) and variant:* A *VP* represents one or many locations in the software where a variation will occur, indicating the existence of different alternatives, each of them may result in a different software behavior. A *Variant* represents an alternative of a VP.
- *Artifact dependency:* this concept relates the defined variability in the variability model to different development artifacts, indicating which artifacts are representing VPs and which ones are realising variants.
- *Constraints dependency:* constraints in OVM can be operated between variants, between VPs and between variants and VPs: an *exclude* constraint specifies a mutual exclusion; e.g., if variant1 at VP1 excludes variant2 at VP2, the variant2 can not be used at VP2 if variant1 is used at VP1. A *requires* constraint specifies an implication; i.e. if a variant is used, another variant has to be used as well.
- *Internal and external variability:* these two concepts separate between the variability that is only visible to the developers (*internal*), and the variability that is communicated to the customers (*external*).

However, according to our objectives from variability outsourcing, the OVM is not well-suited. In fact, OVM encounters certain limits that prevent its adoption as a variability modeling solution on a VaaS provider infrastructure. These limits result from the differences between the SPLE domain and the multi-tenant SaaS model. In SPLE, the software adaptation to customers requirements is realised through a customisation task that is usually done by the same organisation that has built the software. While in the multi-tenant SaaS model, since all tenants are using the same software instance, it is more practical to give these tenants the control to customise the software to their needs (interventions from the SaaS provider still always required to guide the tenants through the customisation of the application).

2.2 Enabling the Software Customisation by Tenants

The *direct* customisation of the software by tenants is one of our objectives since it relieves, in many cases, the SaaS providers from doing this heavy customisation tasks each time a new tenant subscribe to the application. However, such customisation capabilities implies that SaaS providers restrict, for certain tenants, the possibility to choose certain variants which, for example, cannot be selected in the application test period (tenants usually pass through an application test period to determine if the software can provide the service they expect). In addition, some variants may be highly expensive and they must be restricted for certain tenants (in use period) with limited budget. The possibility to define, depending on the tenants contexts, the required restriction conditions for variants and VPs is not the focus of OVM, as it assume that the software customisation will be always done by the software creator. In addition, another main reason to propose variability as a service is minimising the number of interventions that SaaS providers have to make for managing variability, and especially, providing concrete data values for variants. Therefore, our variability modeling solution gives SaaS providers the possibility to define variants with free values, thus tenants can provide the informations they like, such as their own application logo, title, background color etc., without having to always use those offered and predefined by the SaaS providers. Once again, this option is not supported by the OVM. Finally, since the application will be customised directly by individual tenants, it will be important to specify, in the variability model, the order in which the variation points must be bound (Some VPs may depend on each other, which is not always intuitive for tenants). In OVM, such ordering could be done by exploiting the constraints dependencies between VPs, but since constraints in OVM were not designed for this purpose, using them to order the VPs binding will be a workaround. We suggest in the next section, a new variability meta-model that covers the limitations of OVM and others SPLE variability modeling techniques, by bringing additional variability modeling concepts, allowing the SaaS providers to give their tenants certain customisation controls and capabilities in a simple, secure and flexible manner.

3 Variability as a Service: Architecture, Process and Meta-model

In this section we will explain in details the *VaaS model* and its core element: the *VariaS component*. The VariaS component provides different interfaces for SaaS providers to create and manage their applications variability models, as well as to resolve variation places in the applications. The VariaS component is deployed and maintained on a VaaS provider infrastructure. The high-level architecture and different actors of VaaS are presented in figure 1.

3.1 Architecture and Variability Outsourcing Process

As depicted in figure 1, the VaaS architecture defines certain steps that must be performed by SaaS providers as well as by tenants in the variability outsourcing process. These steps are ordered in time and they are divided under *Specification* and *Execution* steps. All mentioned steps ((step 1), (step 2)etc.) and elements ((a), (b), etc.) are shown in the figure.

Specification. First of all, the SaaS providers have to instantiate the variability meta-model (a) offered by the VaaS provider (step 1). The variability model resulting from such instantiation is stored in a variability models repository (b). Prospective tenants of the SaaS application (having outsourced its variability management) must bind its variability model (choosing the variants that meet their needs), before they can start its use (step 2). Such variability binding has to be made through the SaaS provider application and then redirected to the VariaS component, since it is more easy and safe that tenants continue to communicate only with their SaaS providers. In addition, such binding (through an integrated variability binding tool) has to be controlled thus tenants will not be able to select non authorised variants and will not have access to certain

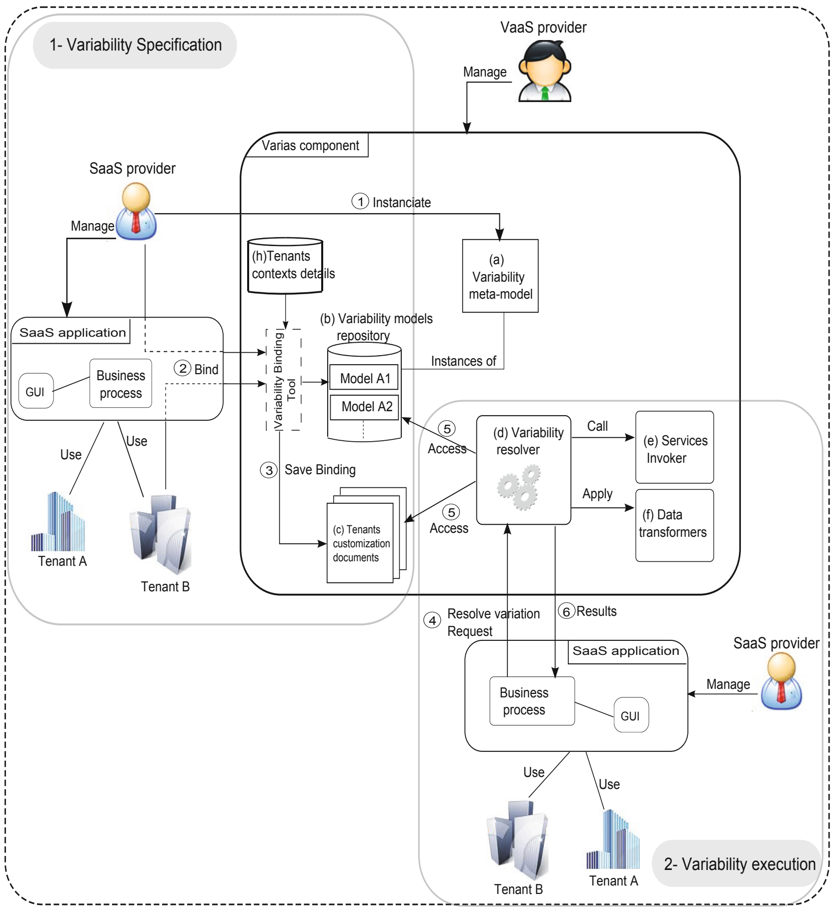

Fig. 1. High-level VaaS Architecture

VPs depending on their contexts and details (h). However, after binding the application variability model, tenants variants choices are saved as customisation documents (c) (step 3). Each customisation document concern one tenant, it indicates their variants chosen at the variation points. Having done so, the tenants can start using their SaaS application and expect from it to behave as they have customised it. Therefore, the application behavior must be adapted at run-time according to these customisations.

Execution. In order to adapt the multi-tenant application behavior according to its defined variability model and the tenants customisations, the SaaS developers must request the VariaS component at each variation place they have identified in the application. Each variation place must have a corresponding variation point in the application variability model that is stored on the VaaS provider side, thus the VariaS component can identify such variation (step 4). These variability resolution requests are in specific forms and use specific protocols defined by the VaaS provider. When a request is received, it will be handled by a variability resolver (d). Such resolver accesses the stored variability models as well as the tenants customisation documents in order to identify the variability action that must be performed (step 5). This action will result in a specific value which is either equivalent to the variant value (selected by the tenant) or deduced from it in case that additional computations based on the variant value must be performed. The variability resolution results will be returned to the application that sent the resolve variation request(step 6). These results must always have the same data format expected by the SaaS application at the corresponding variation place. The non respect of this format may break down the application. In this case, the additional computations to perform on a variant value would be, for example, to transform its data format to the one expected by the application. The expected data format from the resolution results of each variation point must be defined in the variability model.

3.2 Variability Meta-model

A variability meta-model is essential for the VaaS model, since it defines how SaaS providers describe their applications variable parts, and how they indicate restricted variants, those accessible to different customisations possibilities. In addition, the variability meta-model has to provide a technical and code level solution for SaaS providers, to help them resolving their application variations while still independent from these applications implementation details. Thus, the VaaS model, through such a meta-model, deals with applications based on standard technologies, such as web-services, and use common applications code artifacts such as texts, files and expressions. In section 2, we have discussed the need of an orthogonal variability model, and shown the limitations that the SPLE modeling techniques encounters for realising our variability outsourcing vision. In the following, new variability modeling concepts related to the VaaS model as well as reused concepts from OVM will be presented. Figure 2 shows our variability meta-model.

The variability modeling approach we present, aims to provide certain level of flexibility to SaaS providers, making them able to deal with complex variability modeling situations in a simple manner. Complex modeling situations such as constraining (by restricting or obligating) the choice of certain variants as well as the binding of certain VPs depending on the tenants contexts are needed. Being able to do so, the *Activation Condition* concept is introduced. Such concept allows SaaS providers to define conditions, which if they evaluate to *true*, their related *Constraints* will be activated. In this way, and depending on the conditions evaluation results, the same variant may be restricted for some tenants and enabled for others. A condition semantic is expressed by a SaaS provider the same as in *If-Then-Else* statements in programing languages. These conditions

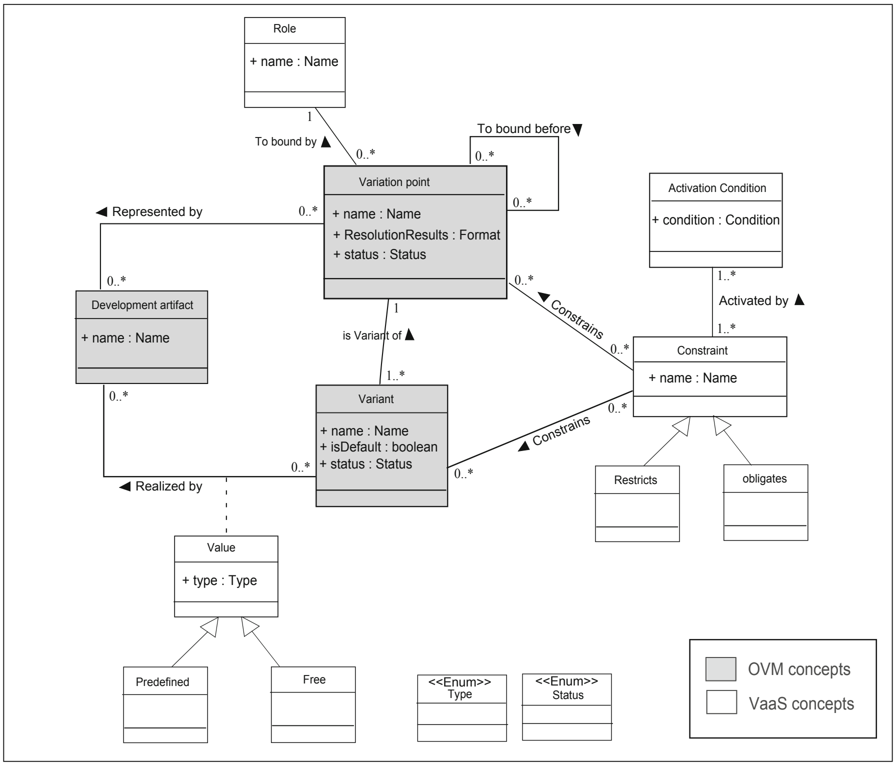

Fig. 2. Variability Meta-model of VaaS

evaluate on informations such as tenant identity, status, already chosen variants and other informations may be useful to limit or extend the software usability for tenants depending on the SaaS provider commercial offer.

The variation point, variant and development artifact concepts are reused from OVM. As mentioned above, variability informations (variants and VPs) are represented in an abstract and artifacts-independent manner, while their relations with the impacted development artifacts is maintained through so called *Artifact Dependency*. However, for each variant (realised by one or many artifacts) its important to indicate its corresponding *value* in each artifact, thus such value can be treated by our service and returned to the SaaS application when needed. The same variant can have different values in different artifacts, each value must have a *type*, thus our variability service can determine the variability action to perform on resolution requests, as well as the appropriate GUI in which the SaaS provider can provide these values. For example, if a SaaS provider creates a value instance for a particular variant with the *Web-Service* type, automatically, a relevant GUI will be opened to provide the web-service description, end point address, operation to invoke, input and output massages types etc. The same, if a value instance is created with a *File* type, the GUI opened would be a simple file input. These is beneficial for SaaS providers, as *free* values can be added by tenants through dedicated and simple GUI's. Finally, the *Role* concept has been added and associated to a variation point in order to differentiate between VPs to bound by the tenants and others to bound by the SaaS providers. In addition, such role concept is beneficial in case one or many application resellers exist. In fact, the SaaS provider may find more efficient that application resellers make decisions on certain VPs that are related to a given segment specificities which some prospective tenants are parts of. Generally, such resellers are more experienced in these segments standards and rules. In the following, we present a case study that shows an instantiation example of these variability concepts.

4 Case Study: A Food Industry Application

In this section, we will revisit a food-industry application (FIA for short) that we have developed as presented in [6]. FIA is a SOA-based multi-tenant application. It allows its food industry tenants looking for foods production and quality improvement to predict the future expenses and benefits from their potential recipes, before executing a real and expensive manufacturing process. FIA evaluates, by relying on an internally developed simulation service, the recipes manufacturing time and cost, as well as their quality characteristics such as nutritional value, taste and smell. FIA has also an integrated shipping service for foods, from the warehouse of the foods supplier to the tenants manufactures. Figure 3 left side shows the FIA business process.

The back-end business logic of the application is implemented using a Business Process Execution Language (BPEL) [9] based solution, and the front-end for the tenants is a Web application which gathers data and hands it over to the BPEL engine for processing the recipes simulation requests. When a tenant user accesses the application, he can manage his existing recipes or decide to create a new one. In this last case, the user has to provide the foods composing the new recipe as well as their respective percentages in it. The user must also describe the recipe cooking process, by sending a pre-negotiated XML structure, or he can rely on a FIA integrated cooking process drawing tool. When finishing, the user must click on simulate button and wait for results. When the application receives the recipe details, the BPEL process invoke the foods supplier service, asking for foods prices and quality informations. These informations, as well as the recipe cooking process allows the simulation service to calculate exactly the final recipe cost and quality. However, when the simulation ends the process sends back a simulation report to the user containing the recipe simulation results. In case user validate the results, the process saves the recipe details in the database and invoke the shipping service thus the foods (composing the recipe) will be shipped from the warehouse of the foods supplier to the tenant manufacture.

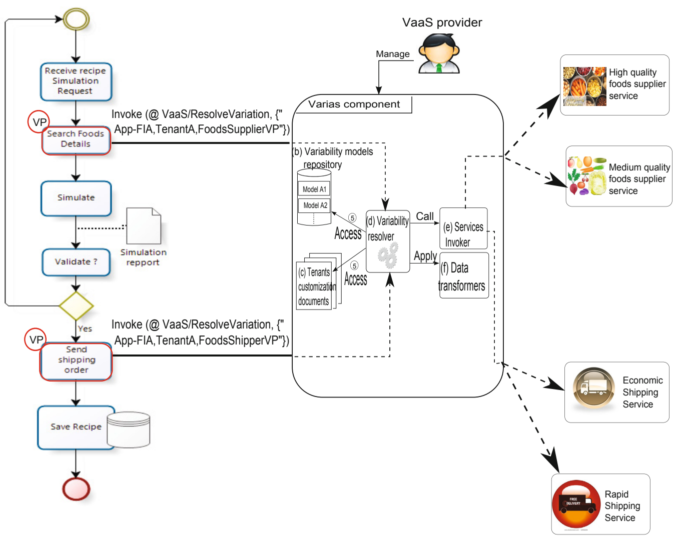
VaaS provider
Manage
Receive recipe Simulation Request
VP
Search Foods Details
Invoke (@ VaaS/ResolveVariation, {" App-FIA,TenantA,FoodsSupplierVP"})
Simulate
Simulation repport
Validate ?
Yes
VP
Send shipping order
Invoke (@ VaaS/ResolveVariation, {" App-FIA,TenantA,FoodsShipperVP"})
Save Recipe
Varias component
(b) Variability models repository
Model A1
Model A2
Access
(c) Tenants customization documents
Access
(d) Variability resolver
Call
Apply
(e) Services Invoker
(f) Data transformers
High quality foods supplier service
Medium quality foods supplier service
Economic Shipping Service
Rapid Shipping Service

Fig. 3. FIA Business-process and its Interaction with the Varias Component

4.1 Modeling FIA Variations Following the VaaS Approach

In order to detect possible variations in the application we have presented its business process to some selected prospective tenants. Three variations have been detected: 1) some tenants are interested by the software but they have asked for another foods supplier with higher foods quality. 2) Some other tenants asked for another foods shipper because the existing one is costly. Taking into account this shipping variation, we have decided to add an economic shipping service to the application but with a higher shipping time. The costly shipping service is very rapid, but it does not supports international shipments. On the other side, we have found that high quality foods suppliers have their own shipping services and that will excludes the need of the shippers proposed by our application. 3) Finally, some tenants want their own company-specific logos and specific titles for the application. Beside these tenant-oriented variations, a segment variation is added to indicate the different simulation models that the application supports. Currently, we are supporting the French and the American models which are the most used worldwide. However, these FIA variations was modeled in first time by relying on the OVM model, and the application customisation was always done by a member of our development team. In figure 4, the FIA variations are modeled following the VaaS approach.

4.2 Resolving FIA Variations

In this section we only focus on the business-process variability, as it requires additional computations to perform by the Varias component in order to resolve

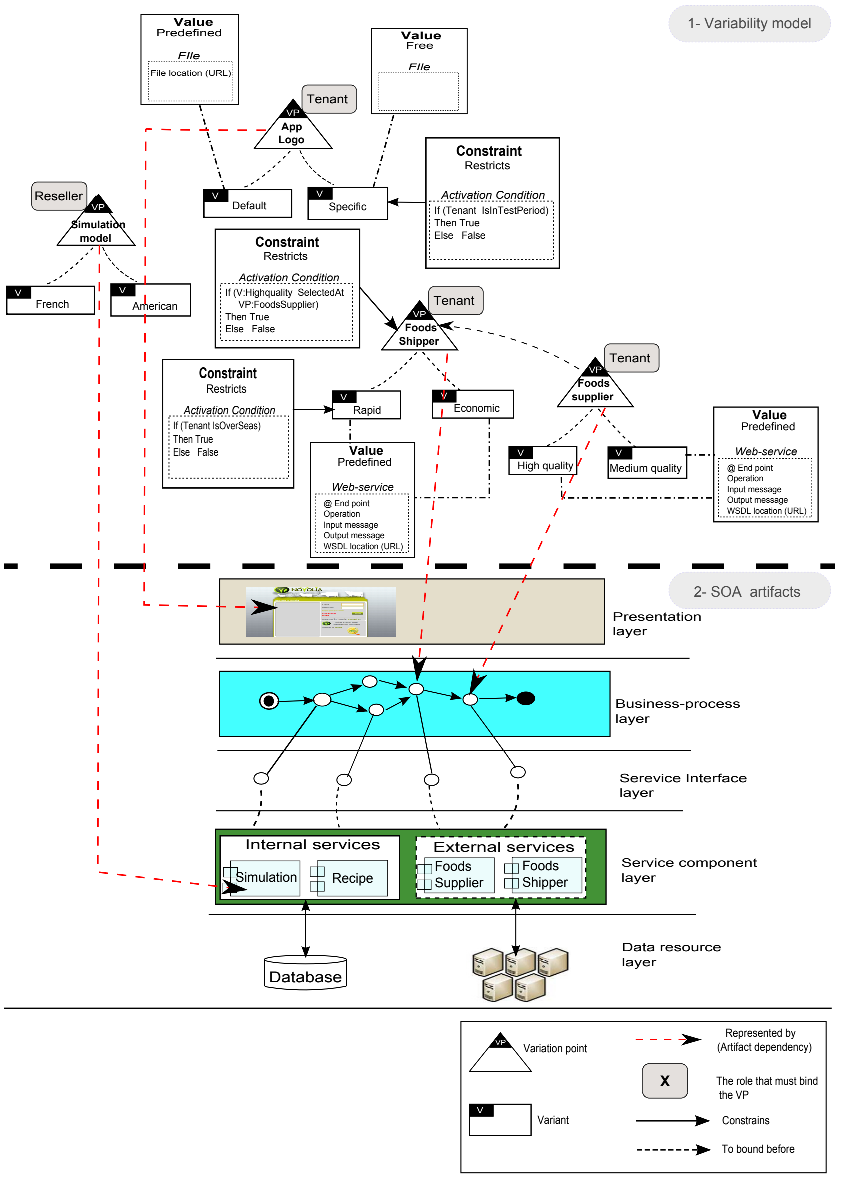

Fig. 4. Modeling FIA Variability Following the VaaS Meta-model

them. In fact, as previously mentioned, we consider multi-tenant applications as SOA-based. In SOA, the application business-process is realised by composing and orchestrating available web-services achieving the desired functionality. For each service in the process there may be different alternatives which implement the service with different implementation logics or quality attributes. In this case, the variability occurs on selecting the most appropriate service depending on the tenants requirements. If a SaaS provider defines these different services alternatives as variants at a particular variation point, the VariaS component will be responsible of invoking the appropriate service (being able to invoke the appropriate service variant requires providing the informations needed for its invocation, such as end point address, input and output massages, operations etc.) and compensating the data mismatch at the input and output messages of different services, if exists. The services invoker (e) and the data transformers (f) elements in the VariaS component (see figure 1) are designed for this purpose. As depicted in figure 3, the two variation points at different services invocations are resolved by sending requests to the Varias component. These requests are in a fixed form which abstract for SaaS providers the concrete services addresses. In addition, such abstraction is also beneficial for SaaS providers since they always interact with variable services in the same manner and expect the same output massages format.

5 Discussion and Future Work

In this paper, we presented the first stages of our work: VaaS or outsourcing *Variability as a Service* with its associated architecture and meta-model. We also presented the VaaS model in its specification and execution steps (see section 3.1) consider SOA applications. We see them as a set of several artifacts, possibly variable. It is important to outline two main hypothesis we voluntarily considered as implicit in this paper:

– *Specification:* a first implicit hypothesis of our work is that variability models of an application and its business design are defined at quite the same time. So the application and its variability models are both defined from scratch and the variability concern is considered at the first stages of the design of the application. This is what we called *early variability* in [6].
– *Execution:* a second implicit hypothesis of our work in its current stage is that the executed application has a stable design and a stable variability. This hypothesis implies that only variants need to be detected, uploaded, executed and potentially evolved. More precisely, in the execution step, we consider only source-code variable artifacts, those executed at run-time. The FIA application is an illustration example through its executable artifacts (i.e. Business-process, web-services, GUI etc.). We also consider that *(ii)* the application design and its variability models, essentially the View Points, are stable.

At this stage, we aim to stabilise the VaaS concept and VariaS component before exploring the following important future issues:

– *Validation:* we successfully developed multi-tenant applications by including and managing variability through VP and variants. This development has been successfully tested and achieved. The next validation step is to implement the VaaS architecture and its VariaS component.

- *Non-code artifacts:* rather than supporting only code , we aim to generalise the approach to other artifacts like requirements, design and test.
- *VaaS and existing applications:* the objective is to enable the VaaS architecture able to support variability as well for new applications as for existing ones. We thus have to look how one can re-engineer an existing application in order to inject variability in terms of specification and management.
- *Evolution:* applications are in essence evolutive. Managing evolution in a VaaS approach is our ultimate objective. It can be seen through three main concerns: *(i)* variability evolution, *(ii)* co-evolution of the application and its variability models and *(iii)* the co-evolution of the variability meta-model and its variability models.

6 Related Work

6.1 SOA Variability Management

Several authors studied the variability concern in the context of SOA systems. In [4] authors identify four types of variability which may occur on SOA. In [14] authors present a framework and related tool suite for modeling and managing the variability of Web service-based systems. They have extended the COVAMOF framework for the variability management of software product families. In [8] authors describe an approach to handle variability in Web services, while [10] focus on variability in business-process by proposing a VxBPEL language. However, all these approaches focus on variability management in SOA systems as a part of the software providers responsibilities which is different from our VaaS and variability management outsourcing approach.

6.2 SaaS and Multi-tenancy

In [7] authors provide a catalog for customisation techniques that can guide developers when dealing with multi-tenancy, they have identified two types of customisation: Model View Controller (MVC) customisation and system customisation. In [2] the authors propose some architectural choices to make when building multi-tenant applications, while in [3] authors discuss their experiences with re-engineering an existing industrial single-tenant application into a multi-tenant one. In [11], authors propose a variability modeling technique for SOA-based multi-tenant applications, they differentiate between internal variability only visible to the developers, and external variability that is communicated to the tenants of the application. Once again, they do not propose the variability as a service which we have treated in this paper.

7 Conclusion

In this work, we have shown the importance and the complexity of supporting variability in SOA-based multi-tenant applications. We have also motivated the need of outsourcing the variability management to a VaaS provider. Architecture and meta-model supporting our approach have been also provided. In addition, we have revisited a multi-tenant application case study and transforming its existing OVM variability model into VaaS model. Our approach aims to decrease the variability management complexity, and to relieve the SaaS providers looking forward to adopt a multi-tenant solution, from developing an expensive variability solution beforehand.

References

[1] Bayer, J., Gerard, S., Haugen, O., Mansell, J., Moller-Pedersen, B., Oldevik, J., Tessier, P., Thibault, J.P., Widen, T.: Consolidated product line variability modeling (2006)

[2] Bezemer, C.P., Zaidman, A.: Multi-tenant saas applications: maintenance dream or nightmare? In: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), pp. 88–92. ACM (2010)

[3] Bezemer, C.P., Zaidman, A., Platzbeecker, B., Hurkmans, T., 't Hart, A.: Enabling multi-tenancy: An industrial experience report. In: 2010 IEEE International Conference on Software Maintenance (ICSM), pp. 1–8. IEEE (2010)

[4] Chang, S.H., Kim, S.D.: A variability modeling method for adaptable services in service-oriented computing. In: 11th International Software Product Line Conference, SPLC 2007, pp. 261–268. IEEE (2007)

[5] Chong, F., Carraro, G.: Architecture strategies for catching the long tail. MSDN Library, Microsoft Corporation, pp. 9–10 (2006)

[6] Ghaddar, A., Tamzalit, D., Assaf, A.: Decoupling variability management in multi-tenant saas applications. In: 2011 IEEE 6th International Symposium on Service Oriented System Engineering (SOSE), pp. 273–279. IEEE (2011)

[7] Jansen, S., Houben, G.-J., Brinkkemper, S.: Customization Realization in Multi-tenant Web Applications: Case Studies from the Library Sector. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 445–459. Springer, Heidelberg (2010)

[8] Jiang, J., Ruokonen, A., Systa, T.: Pattern-based variability management in web service development. In: Third IEEE European Conference on Web Services, ECOWS 2005, p. 12. IEEE (2005)

[9] Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., et al.: Web services business process execution language version 2.0. OASIS Standard 11 (2007)

[10] Koning, M., Sun, C., Sinnema, M., Avgeriou, P.: Vxbpel: Supporting variability for web services in bpel. Information and Software Technology 51(2), 258–269 (2009)

[11] Mietzner, R., Metzger, A., Leymann, F., Pohl, K.: Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In: Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, pp. 18–25. IEEE Computer Society (2009)

[12] Pohl, K., Bockle, G., Van Der Linden, F.: Software product line engineering: foundations, principles, and techniques. Springer-Verlag New York Inc. (2005)

[13] Sengupta, B., Roychoudhury, A.: Engineering multi-tenant software-as-a-service systems. In: Proceeding of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems, pp. 15–21. ACM (2011)
[14] Sun, C., Rossing, R., Sinnema, M., Bulanov, P., Aiello, M.: Modeling and managing the variability of web service-based systems. Journal of Systems and Software 83(3), 502–516 (2010)
[15] Svahnberg, M., Van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Software: Practice and Experience 35(8), 705–754 (2005)