# Gestion de la variabilité dans les applications SaaS multi-locataires

**Ali Ghaddar[*,**], Dalila Tamzalit[**], Ali Assaf[*]**

[*] *BITASOFT*
*60 bd. Maréchal Alphonse Juin*
*44100 Nantes*
*ali.ghaddar, ali.assaf@bitasoft.com*

[**] *Université de Nantes - Faculté des Sciences*
*LINA - CNRS UMR 6241*
*2, rue de la Houssinière*
*BP 92208 44322 Nantes cedex 03*
*Dalila.Tamzalit@univ-nantes.fr, ali.ghaddar@etu.univ-nantes.fr*

RÉSUMÉ. *La multi-location est un principe d'architecture logicielle relativement nouveau, généralement adopté lorsqu'une application est fournie sous forme d'un service : on parle alors de mode* SaaS. *Ce nouveau principe réduit considérablement les coûts de déploiement et de maintenance de l'application car tous les clients (*locataires*) partagent la même instance de cette application. Toutefois, pour toucher un grand nombre de locataires, l'application doit être personnalisable et configurable pour répondre aux exigences variables de chaque locataire. Par conséquent, les fournisseurs SaaS font face à une préoccupation supplémentaire : la gestion efficace de la variabilité au sein de l'application. Dans cet article, nous abordons cette problématique et nous essayons de réduire la complexité de sa gestion au travers de la modélisation de la variabilité et son découplage par rapport à son implémentation. La modélisation s'appuie sur OVM, une technique de modélisation de la variabilité, et le découplage vis-àvis de l'implémentation se fait au travers des couches de l'architecture supportant l'application. Notre approche est illustrée au travers d'une étude de cas de l'industrie alimentaire.*

ABSTRACT. *Multi-tenancy is a relatively new software architecture principle, generally adopted when a software application is provided as a service (SaaS). Such new principle reduce considerably the application deployment and maintenance costs, as all customers(tenants) share the same instance of such application. However, to attract a significant number of tenants, the application has to be customizable and configurable to meet the varying requirements of individual tenants. In consequence, the SaaS providers are facing an additional concern: managing*

*efficiently the application variability. In this paper, we address variability and try to reduce its management complexity by modeling it and by decoupling its implementation. The modeling is based on OVM, a variability modelling technique, and the implementation decoupling is through the layers of the architecture supporting the application. Our approach is illustrated by relying on a case study from the food industry.*

MOTS-CLÉS : *SaaS, multi-locataire, variabilité.*

KEYWORDS: *SaaS, multi-tenant, variability.*

## 1. Introduction

Ces dernières années, la tendance à externaliser certaines fonctionnalités qui ne font pas partie des compétences clés d'une entreprise, a conduit à l'apparition d'un nouveau type de fournisseurs de services. Ces derniers développent et maintiennent, sur leurs infrastructures, ces fonctionnalités qui leurs ont était confiées. Une fois celles-ci réalisées, les entreprises accèdent à ces fonctionnalités externalisées et les utilisent comme un service (souvent au travers du web). L'utilisation de ces services est facturée à l'usage et régie par un contrat de service. Ce principe d'externalisation a rencontré un vif succès auprès des petite-moyenne entreprises (PME). La raison de ce succès est purement économique. En effet, la majorité des PME estiment prendre moins de risques en confiant la réalisation de certaines fonctionnalités à un fournisseur plus expérimenté et trouvent cette option plus rentable. Le terme *Cloud Computing* résume ces efforts d'externalisation et de réalisation de services informatiques à la demande. Ce terme a émergé comme la plate-forme d'exécution pour réaliser cette approche (Sengupta *et al.,* 2011). Plus concrètement, le Cloud Computing est visualisé comme une pile contenant ces nouveaux types de services à fournir. Cela revient à fournir une infrastructure sous forme d'un service (IaaS), également à fournir une plate-forme de développement sous forme d'un service (PaaS), pour finalement fournir un logiciel sous forme d'un service (SaaS). Cet article concerne ce dernier type.

Informellement, un logiciel en mode SaaS peut être défini comme : « Un logiciel déployé sous forme d'un service hébergé et accessible sur Internet » (Chong *et al.,* 2006). Dans le modèle de ce service, le fournisseur propose une application complète, prête à être utilisée par des clients potentiels. Ces clients s'inscrivent à l'application et l'utilisent via le web au travers d'un simple navigateur. Tous les détails concernant la technologie utilisée et le serveur de déploiement de l'application sont transparents. Ce type de solution SaaS est proposé par les fournisseurs comme une alternative aux applications logicielles classiques qui nécessitent une installation chez le client, réduisant ainsi des investissements importants en termes de matériels et de ressources physiques. Pour ces raisons, le modèle SaaS a connu une croissance très importante au cours de ces dernières années, et les tendances du marché pour l'avenir continuent à être prometteuses. Selon un rapport récent d'IDC (Phil Hochmuth, 2010), le marché du SaaS a atteint 13,1 milliards de $ en revenus en 2009 et il devrait atteindre 40.5 milliards de $ en 2014, dont environ 34% d'achats de nouveaux logiciels dans le monde suivront le modèle SaaS. Cependant, du point de vue d'un fournisseur de SaaS, les véritables avantages de cette solution commencent à apparaître quand il est possible d'héberger plusieurs clients sur la même instance de l'application, au lieu d'une application dédiée à être deployée et maintenue séparement pour chaque client (voir figure 1). Cette approche est appelée *multi-locataire*, où un *locataire* désigne une organisation cliente, regroupant un certain nombre d'utilisateurs (Bezemer *et al.,* 2010b).

Toutefois, lorsque plusieurs locataires partagent la même instance de l'application, la gestion des variantes du logiciel suivant leurs exigences propres et différentes est un véritable problème. De toute évidence, cela forme une préoccupation supplémentaire

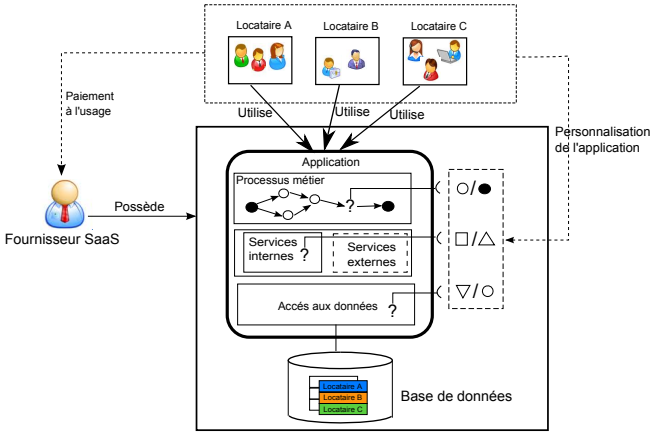


**Figure 1.** *Architecture de nos applications SaaS multi-locataire.*

qui augmente la charge du fournisseur de SaaS. Cette préoccupation est connue sous le nom de la *variabilité* (Svahnberg *et al.,* 2005), elle indique la capacité du système à être efficacement adapté et configuré pour une utilisation dans un contexte particulier. Dans cet article, nous nous focalisons sur cette préoccupation et nous essayons de réduire la complexité de sa gestion en effectuant une première étape de modélisation des différentes variations puis en découplant son implémentation à travers les différentes couches de l'application. Nous allons détailler notre approche et son contexte dans le reste de l'article. Ce dernier est organisé comme suit : la section 2 présente le contexte de travail et la problématique de la gestion de la variabilité avec une approche multi-locataire dans ce contexte ; la section 3 présente l'étude de cas et la modélisation de sa variabilité ; la section 4 présente le découplage de l'implémentation de la variabilité sur les couches de l'architecture support de l'application avant de finir sur un bref état de l'art en Section 5 et conclure notre article par la section 6.

## 2. Problématique générale : le contexte de travail et les manques

La multi-location rend le développement des applications SaaS plus complexe (Jansen *et al.,* 2010). En effet, les développeurs de l'application doivent prendre en charge la gestion et l'implémentation de la variabilité, pour répondre aux besoins spécifiques d'un locataire sans affecter les autres locataires hébergés. Cette complexité représente évidement une inquiétude supplémentaire pour les fournisseurs de SaaS, surtout que la multi-location est un principe d'architecture logicielle relativement nouveau dont peu de développeurs présentent des compétences ou des expériences suffisantes pour le gérer d'une façon efficace. À travers cet article, nous voulons précisément faire partager notre connaissance et notre expérience dans ce domaine. Une

approche pragmatique que nous considérons comme un bon point de départ pour les fournisseurs de SaaS dans cette direction, serait d'explicitement identifier la variabilité de l'application dès les premiers stades de sa conception. À partir de cette identification, la variabilité pourra être représentée et modélisée d'une manière abstraite (voir section 3), indépendamment des choix techniques et technologiques d'implémentation. D'un côté, un modèle abstrait de variabilité pourra être communiqué avec les locataires afin de les guider dans leurs choix lors de la personnalisation de l'application. De l'autre côté, un tel modèle soutient les développeurs dans les décisions à prendre lors de l'implémentation et l'intégration de la variabilité dans les artefacts de développement. Par contre, offrant une variabilité déjà planifiée dans une application SaaS, en général, limite sa flexibilité. Cependant, la gestion d'une variabilité "arbitraire" est encore une problématique avec une énorme complexité. Ainsi, un équilibre entre l'offre de la variabilité et la complexité de sa gestion est assuré dans cet article.

### 2.1. *Contexte de travail : une application multi-locataire basée sur l'architecture orientée service*

Pour implémenter la variabilité à partir de son modèle prédéfini, les caractéristiques et les artefacts du modèle de construction de l'application doivent être exploités, afin que la solution d'implémentation désirée soit élaborée en fonction. Nos applications SaaS sont construites suivant le modèle SOA (Service oriented architecture). Ce choix est lié à nos besoins industriels en termes d'interactions standardisées entre nos différents systèmes distribués et hétérogènes, ce qui peut être facilement réalisé à travers l'utilisation des services-web (la technologie d'implémentation de SOA). Nous voulons également préciser que l'architecture SOA est fréquemment utilisée pour la réalisation des applications SaaS (Guo *et al.,* 2007, Laplante *et al.,* 2008, Mietzner *et al.,* 2008b). En effet, grâce à ses caractéristiques importantes en termes d'évolution, de flexibilité et de couplage faible entre les services interagissants, ce modèle permet de réagir rapidement aux changements dans les besoins métiers, ce qui est un aspect crucial pour la réussite d'une application SaaS. Ainsi, notre travail s'inscrit dans le cadre d'architectures de l'application orientées service. Cependant, quand une application SaaS multi-locataire est construite par la composition et l'orchestration d'un ensemble de services (suivant les principes de SOA), la gestion et l'implémentation de la variabilité doivent être manipulées suivant les spécifications de ce modèle de construction. En effet, une application basée sur l'architecture SOA implique plusieurs couches d'abstraction (voir figure 3, partie inférieure), avec comme objectif de séparation des préoccupations et de flexibilité du système. Cela a pour conséquence dans de telles architectures multi-couches de répartir la variabilité sur toutes ces couches. D'après notre analyse de ce qui a été réalisé dans la communauté SOA sur la thématique de la variabilité (pas forcément dans un contexte multi-locataire), nous avons pu constaté qu'une approche qui implémente la variabilité d'une manière générale (applicable sur toutes les couches) est loin d'être réalisable. Pour cette raison, la majorité des approches proposées visent une couche spécifique, et parfois, dans un contexte spécifique. Dans ce qui suit, nous présentons les différentes approches existantes sur

les trois principales couches de cette architecture :

1) **La couche des processus métiers :** cette couche est responsable de l'orchestration et la composition des services pour réaliser les objectifs métiers de l'application. Différents types d'approches ont été proposés pour gérer la variabilité sur cette couche, leur accent étant mis sur la modification dynamique de la composition des services ainsi que la structure du processus.

- Dans (Koning *et al.,* 2009, Chang *et al.,* 2007), les auteurs essayent d'étendre le BPEL (Business process execution language (Jordan *et al.,* 2007)) afin d'intégrer les informations de la variabilité dans sa définition, ce qui se traduit par une adaptation dynamique de la composition de services au moment de l'exécution du processus.

- Le travail de (Mietzner *et al.,* 2008a) adresse les mêmes objectifs à la différence qu'il décrit la variabilité dans le processus BPEL d'une manière séparée. Un mécanisme d'adaptation est ainsi proposé pour modifier la structure du processus à base de chaque client. Ce type d'approche est plus propre que celui du premier type puisqu'il ne complique pas la définition du processus.

- Un autre type d'approches (Sun *et al.,* 2010, Nguyen *et al.,* 2011, Razavian *et al.,* 2008) consiste à intégrer les informations de la variabilité dans les méta-modèles des langages de modélisation des processus métiers (i.e. BPMN, les diagrammes d'activité d'UML, etc.). Le but de ces efforts est de modéliser la variabilité au niveau architectural, ensuite d'automatiser la génération des variantes du processus à l'aide des outils de génération du code BPEL. Le mécanisme de génération prend en entrée un modèle de processus défini par ces langages de modélisation connus. Les processus variantes qui résultent de cette étape de génération sont prêts au déploiement sur le serveur d'application contenant un moteur d'exécution du langage BPEL.

2) **La couche des interfaces de services :** cette couche encapsule les interfaces des services publiés dans un registre par les fournisseurs de services, suivant le langage WSDL (langage standard de description de service). La variabilité sur cette couche peut se produire suite à une modification de la composition de services dans la couche des processus (remplacement d'un service par un autre avec des interfaces différentes), ou bien suite à l'évolution de l'interface d'un service par son fournisseur. Les approches proposées pour gérer la variabilité à ce niveau consiste à identifier et à résoudre les conflits syntaxiques et sémantiques sur les messages d'entrées et de sorties ainsi que les signatures des méthodes de services (Kongdenfha *et al.,* 2006, Sam *et al.,* 2006, La *et al.,* 2010). Il faut noter que le remplacement d'un service par un autre indique que les deux versions de services sont partiellement ou totalement équivalente, et jouent les rôles des alternatives sur un point d'invocation variable dans le processus.

3) **La couche des composants de services :** cette couche se compose de l'ensemble des composants qui implémentent les fonctionnalités des services publiés dans la couche des interfaces. Ces composants maintiennent ainsi la qualité des services publiés (QoS). De même, plusieurs types d'approches et des méthodologies ont été proposés pour gérer et implémenter la variabilité sur cette couche.

- Dans (Khan *et al.,* 2011), les auteurs proposent des patrons de conception pour implémenter la variabilité dans le composant de service. Chaque patron peut être utilisé dans une situation de variabilité différente. Ce type d'approche est similaire au principe des patrons de conception proposés pour les langages de programmation par objets.

- Dans (Jegadeesan *et al.,* 2009), les auteurs proposent une méthode basée sur les principes de la programmation par aspects (AOP) pour implémenter la variabilité dans le composant de service. Leur approche consiste à représenter chaque variation sous forme d'un ou plusieurs aspects, séparément du noyau de composant qui encapsule le comportement commun entre les locataires. Au moment de l'exécution, un mécanisme de tissage des aspects est utilisé pour adapter le comportement ainsi que les messages d'entrées et de sorties de composant selon le locataire qui invoque le service.

- Au contraire de deux approches précédentes qui gèrent la variabilité au niveau implémentation, (Stollberg *et al.,* 2010) proposent une approche plus générale. Cette approche consiste à utiliser l'ingénierie dirigée par les modèles (MDE) pour proposer un méta-modèle de variabilité de service. Les clients peuvent personnaliser le service en résolvant son modèle de variabilité conforme au méta-modèle proposé. Plus précisément, cette technique de modélisation permet de définir des éléments fixes et variables dans le service (i.e. opérations, types de messages, propriétés etc.). Chaque client de service doit se faire attribuer les éléments fixes (fondamental pour le bon fonctionnement de service) et ensuite personnaliser la structure de service en choisissant parmi les éléments variables ceux qui correspondent à ses besoins, ainsi que d'ignorer le reste des éléments variables de sorte qu'il ne fassent plus partie de sa variante de service.

Au-delà des couches de l'architecture SOA, et étant donné que les besoins métiers spécifiques à chaque locataire se reflètent par la nécessité d'avoir une structure de données conforme aux besoins, la couche d'accès aux données est également impactée par la préoccupation de la variabilité. Pour cette raison, divers travaux (Foping *et al.,* 2009, Xuxu *et al.,* 2010, Schiller *et al.,* 2011) ont été proposés pour pouvoir étendre le schéma de la base de données partagée, afin que chaque locataire puisse stocker des données spécifiques, ainsi que de sécuriser et d'isoler les données de locataires de telle sorte qu'un locataire ne puisse jamais accéder aux données des autres.

### 2.2. *Les manques*

D'après l'analyse de toutes ces approches et techniques proposées pour implémenter la variabilité, nous pouvons remarquer qu'elles sont différentes ainsi que les circonstances dans lesquelles elles pourraient être appliquées. Cela peut conduire à une forte probabilité d'avoir, dans la même application, de nombreuses solutions de représentation de la variabilité, dont chacune est adaptée à un problème spécifique et qui plus est pour une couche spécifique. Cette diversité de solutions provoque une isolation et un manque d'uniformité, souvent inconscientes, de la modélisation de la

variabilité, ce qui complique sa réflexion et sa gestion. Nous sommes convaincus de la nécessité d'avoir un modèle global et unique de la variabilité de l'application. Ce modèle aura pour rôle d'abstraire la représentation de la variabilité pour faciliter sa réflexion et ainsi sa communication avec les locataires potentiels. En plus, un tel modèle doit maintenir un lien entre cette représentation abstraite de la variabilité et son implémentation sur les couches, afin de faciliter la traçabilité des endroits variables dans l'application, surtout avec l'intégration des nouveaux locataires et des nouvelles variations.

Un autre problème complique la gestion : selon notre expérience dans le développement des applications multi-locataire, les développeurs reçoivent habituellement les variations dans les exigences des locataires d'une manière informelle, et elles sont généralement décrites selon leurs expériences et leur propre terminologie du domaine métier. Ceci peut être différent d'un locataire à un autre et ainsi différent de ce que les développeurs comprennent en termes de concepts et des couches de l'application concernées. Le manque de formalisation et l'absence d'un modèle uniforme et global de la variabilité fait de la décision pour choisir les techniques et les couches les plus appropriées pour implémenter les variations une tache non-triviale et particulièrement risquée. Par conséquent, le risque de mauvaises décisions augmentent la charge des fournisseurs, compliquent la situation, et peuvent même conduire à l'échec du projet.

Pour surmonter ces problèmes, nous proposons une méthodologie de gestion de la variabilité basée sur deux étapes :

1) la première étape consiste à représenter les variations de l'application multi-locataire sous forme d'un modèle de variabilité abstrait et uniforme, ce qui peut être réalisé en exploitant des techniques de modélisations bien connues dans la discipline de l'ingénierie des lignes de produits logiciels (SPLE) (Bayer *et al.,* 2006, Pohl *et al.,* 2005).

2) La seconde étape consiste à choisir les techniques ainsi que les couches d'application les plus appropriées pour implémenter les variations.

## 3. Étude de cas : une application pour l'industrie alimentaire

Pour illustrer notre approche, nous allons utiliser une application pour l'industrie alimentaire (Food industry application : FIA pour faire court). FIA est une application multi-locataire basée sur l'architecture SOA. Elle permet à ses locataires (les industries alimentaires) de simuler leurs recettes avant de les fabriquer. En s'appuyant sur un service de simulation développé en interne, l'application évalue le temps et les coûts de fabrication des recettes, ainsi que leurs caractéristiques de qualité (i.e. la valeur nutritionnelle, le goût, l'odeur etc.). FIA a également un service intégré pour transporter les ingrédients de l'entrepôt du fournisseur des ingrédients aux usines des locataires. La figure 2 montre le processus métier de l'application.

Quatre acteurs principaux existent dans le processus : l'utilisateur (appartient à un locataire), l'application, le fournisseur et le transporteur des ingrédients. Dans ce qui

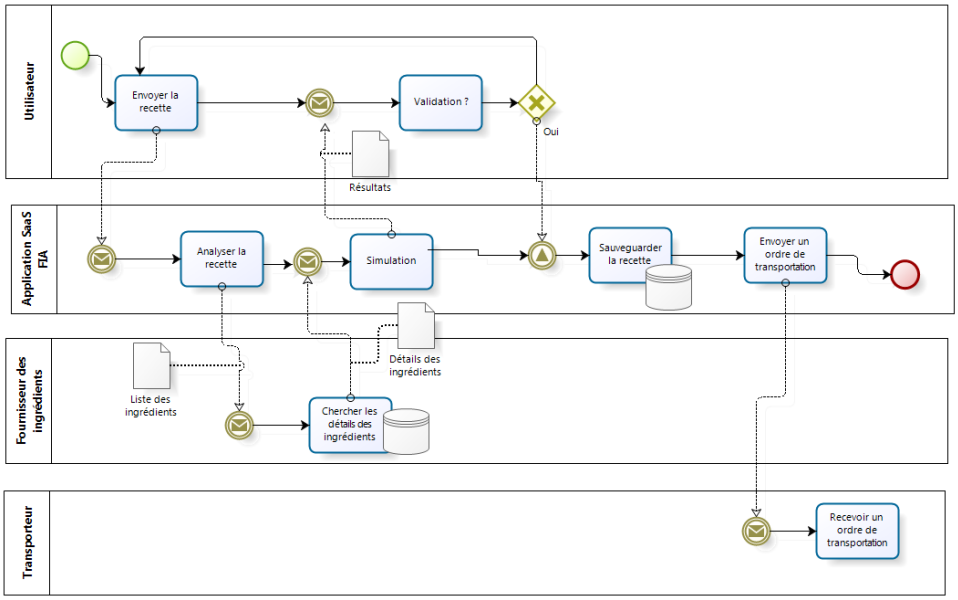


**Figure 2.** *Le processus métier de l'application.*

suit nous allons expliquer le rôle de chacun.

1) **L'utilisateur :** Après avoir accédé à l'application, l'utilisateur peut gérer ses recettes existantes ou décider de créer une nouvelle. Dans ce dernier cas, il doit spécifier les ingrédients qui la composent ainsi que leurs pourcentages respectifs dans la composition. L'utilisateur doit également décrire la procédure de cuisson de la recette (en s'appuyer sur un outil intégré dans l'application). En cliquant sur le bouton *Simuler*, la recette est envoyée et le processus de l'application est déclenché.

2) **L'application :** L'application reçoit la recette et analyse ensuite ces détails pour extraire la liste des ingrédients qui la composent. Elle invoque ensuite le service de fournisseur des ingrédients, pour récupérer les prix et les informations de qualité des ingrédients dans la liste. Ces informations sur les ingrédients ainsi que la procédure de cuisson permettront au service de simulation d'évaluer les caractéristiques de la recette. Toutefois, lorsque la simulation se termine, l'application renvoie un rapport à l'utilisateur contenant les résultats de la simulation. Une fois que l'utilisateur valide les résultats, la recette est enregistrée dans la base de données, et un ordre de transport des ingrédients est envoyé.

3) **Le fournisseur des ingrédients :** La précision des résultats de la simulation dépend de la précision des informations renvoyées par ce service en termes d'approximation de la réalité et des ingrédients réels fournis. Ces informations sont mises à jour

par le fournisseur lorsque la qualité et/ou le prix des ingrédients changent.

4) **Le transporteur des ingrédients :** Il offre un service de transport des ingrédients de l'entrepôt du fournisseur des ingrédients aux usines des locataires. Lorsque un locataire reçoit les ingrédients, il peut exécuter un véritable processus de fabrication, en étant sûr que la recette aura les caractéristiques prédites par l'application.

### 3.1. *Les variations de l'application*

Afin de détecter les variations possibles dans l'application, nous avons présenté son processus métier à un ensemble des locataires potentiels. À travers cette étape, cinq variations ont été détectées :

1) Nous avons constaté que certains locataires sont intéressés par le logiciel mais ils ont leurs propres fournisseurs d'ingrédients.

2) Certains autres locataires ont demandé pour un autre transporteur des ingrédients parce que celui proposé est coûteux. Pour cela, nous avons décidé d'ajouter un deuxième service de transport au système. Ce nouveau service est moins coûteux mais par contre il met plus de temps pour acheminer les ingrédients. Cependant, l'existence d'un fournisseur d'ingrédients propre à un locataire exclut le besoin de nos transporteurs. Généralement tous les fournisseurs d'ingrédients ont leurs propres transporteurs ou jouent les deux rôles en même temps.

3) Une autre variation a été détectée dans la façon dont le service de simulation doit se comporter. En effet, pour une simulation précise, différents facteurs d'impact liés aux propriétés des usines des locataires doivent être considérés. Des facteurs tels que la consommation d'énergie à l'usine, la perte des ingrédients sur la chaîne de production, le nombre des employés, l'effet négatif des machines sur la qualité des ingrédients, etc.

4) Nous avons également constaté que certains locataires ont des mesures de sécurités spécifiques qui leur interdisent de stocker les recettes dans une base de données partagée. Ces locataires voulaient une base de données dédiée et ils sont prêts à payer les frais supplémentaires.

5) Enfin, nous avons constaté que certains locataires appartiennent au même groupe, et ils veulent partager les informations de leurs recettes. Cependant, en choisissant une base de données dédiée, le partage des recettes ne sera plus possible.

### 3.2. *Modélisation de la variabilité*

La modélisation de la variabilité dans l'application a pour but de réduire sa complexité de gestion ainsi que de faciliter sa communication avec les locataires. Dans la figure 3 (partie supérieure), les variations dans l'application sont modélisées à l'aide d'un modèle de variabilité orthogonale (OVM) introduit dans (Pohl *et al.,* 2005). OVM fournit les concepts clés suivants pour la modélisation de la variabilité : Dans OVM, un

*point de variation (VP)* représente un ou plusieurs endroits dans le logiciel où une variation se produit, indiquant l'existence des différentes solutions, chacune d'entre elles peut entraîner un comportement de logiciel différent. Une *Variante* représente une solution possible pour un VP. Les variantes dans OVM peuvent être *obligatoires* ou *facultatives*. Une variante obligatoire doit être sélectionnée sur un point de variation, alors qu'une variante facultative peut, mais ne doit pas l'être. Le nombre de variantes qui peuvent être choisies sur un VP est limité par la cardinalité *min..max*. OVM définit également des contraintes entre les variantes, entre les VPs, et entre les variantes et les VPs : la contrainte *exclude* précise une exclusion mutuelle ; par exemple, si variant1 à VP1 exclut variant2 à VP2, le variant2 ne peut pas être choisi à VP2 si variant1 est choisi à VP1. La contrainte *requires* spécifie une implication ; c'est à dire si une variante est choisie, une autre variante doit être également choisie. En plus, OVM défini des relations de dépendances entres les éléments de la variabilité (VPs et variantes) et les artefacts de développement de l'application. La relation *represented by* indique les artefacts qui sont affectés par un VP, tandis que la relation *realized by* indique les artefacts qui implémentent les variantes. Ce dernier concept est utilisé dans notre approche pour relier le modèle de la variabilité abstrait avec son implémentation sur les couches de l'application (voir figure 3).

## 4. Découplage de l'implémentation de la variabilité

Après la modélisation de la variabilité, l'architecte ainsi que les développeurs doivent prendre la décision concernant les techniques d'implémentation et les couches les plus appropriées à considérer pour implémenter les points de variations. Dans le reste de cette section, nous décrivons comment les points de variations de l'application FIA sont résolus.

### 4.1. *Fournisseur et transporteur des ingrédients*

Ces deux points de variations sont très proches en termes de contexte et décision d'implémentation, et ils seront spécifiés ensemble. Nous allons commencer par le point de variation *fournisseur*. Selon la spécification de cette variation, certains locataires ont besoin que l'application interagisse avec les services de leurs propres fournisseurs d'ingrédients au lieu de celui proposé par le système. En effet, puisqu'il s'agit des services externes dont nous n'avons pas le contrôle, la seule manière d'implémenter cette variation est de basculer entre l'invocation de ces services au niveau du processus de l'application. Cela est réalisé par la modification dynamique de l'adresse d'invocation de service par celle qui correspond au locataire courant (les informations concernant les variantes sélectionnées par les locataires sont stockées dans des fichiers de configuration). La variation de transporteur est implémentée de la même façon. Ce mécanisme d'adaptation est fréquemment utilisé dans la composition et l'orchestration des services web (Koning *et al.,* 2009, Chang *et al.,* 2007). Par contre, le mécanisme d'adaptation sur cette couche prends en considération l'existence d'une

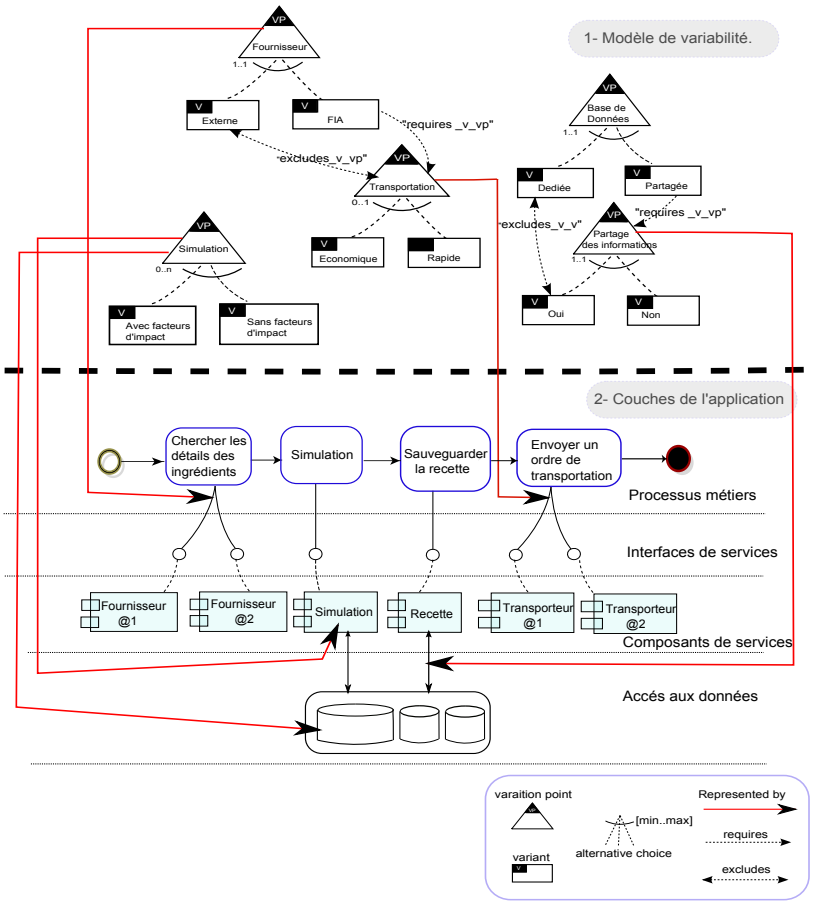


**Figure 3.** *Modélisation de la variabilité et le découplage de son implémentation.*

contrainte d'exclusion entre la variante *Externe* de VP *fournisseur* et le VP *transporteur*. Plus précisément, l'invocation de service de transport est ignorée dans le cas où la variante *Externe* est choisie.

### 4.2. *Simulation*

Afin d'assurer une simulation précise, des propriétés spécifiques à l'usine de chaque locataire doivent être considérées. Une solution rapide pour implémenter cette variation serait de créer un service de simulation par locataire. Ensuite, adapter la composition des services au niveau du processus métier, en réutilisant le mécanisme

d'adaptation existant (voire section 4.1). Par contre, avec cette solution, il deviendra vite très compliqué de maintenir le nombre croissant de services, tant que de nouveaux locataires s'abonneront à l'application. Pour éviter cette solution coûteuse, nous avons décidé d'intégrer cette variation au niveau composant de service de simulation. La solution consiste à découper l'implémentation de service en deux parties. La première partie est un composant de base qui encapsule le comportement commun entre les locataires. La deuxième partie est spécifique à chaque locataire, elle récupère de la base de données ses propriétés d'usine et applique ainsi leurs effets aux résultats de simulation. Cette deuxième partie est implémentée sous forme d'aspects qui seront injectés dynamiquement dans le composant de base (à travers un mécanisme de tissage des aspects). Cependant, cette injection est conditionnelle, elle dépend de la variante choisie par les locataires sur le VP *simulation*. Plus de détails sur cette technique d'implémentation existent dans le travail de (Jegadeesan *et al.,* 2009).

### 4.3. *Partage d'informations*

Cette variation a pour but de permettre aux locataires de partager les informations de leurs recettes (avec des autres locataires dans le même groupe), ou bien de garder ces informations isolées. Cette variation est implémentée sur la couche d'accès aux données. En effet, tous les composants de services qui sont développés en interne ( et ainsi le composant Recette) utilisent le même composant d'accès aux données. la technique adoptée pour implémenter cette variation consiste à transformer les requêtes SQL générées par ce composant. Cette transformation consiste à injecter des filtres dans les requêtes, pour assurer l'isolation ou le partage des informations. Cette manière d'implémentation est transparente aux développeurs de services, ce qui améliore leur productivité. En plus, cette méthode pourra être utilisée dans d'autres situations, où le partage des informations est nécessaire. Par exemple, dans le service de gestion des connaissances que nous prévoyons de développer, les locataires peuvent demander de partager leurs articles de recherche. Cependant, l'adoption de cette méthode nécessite l'ajout de deux nouvelles colonnes à chaque table concernée dans la base de données partagée : `LocataireID` et `GroupeID`. Par exemple, pour obtenir tous les enregistrements de la table *RECETTE*, la requête saisie par un développeur est la suivante :

```
SELECT * FROM RECETTE
```

La transformation de cette requête pour assurer l'isolation :

```
SELECT * FROM RECETTE
WHERE LocataireID='123'
```

La transformation de cette requête pour permettre le partage :

```
SELECT * FROM Recette
```

```
WHERE(LocataireID='123' OR GroupeID='3')
```

En outre, une autre raison qui nous a conduit à utiliser cette méthode est l'existence du VP *Base de données*. En effet, certains locataires ont demandé une base de données dédiée. L'implémentation de cette variation sera compliquée si les développeurs écrivent directement la requête entière, tandis que la transformation des requêtes permet une prise en charge native de cette variation. Lors de l'installation des bases de données dédiées, il suffit juste de désactiver le mécanisme de transformation.

## 5. Travaux connexes

### 5.1. *gestion de la variabilité dans SOA*

Plusieurs auteurs ont étudié la préoccupation de la variabilité dans le contexte des systèmes SOA. Dans (Chang *et al.,* 2007) les auteurs identifient quatre types de variabilités qui peuvent se produire dans les systèmes orientés services. Dans (Sun *et al.,* 2010) les auteurs présentent un cadre et des outils pour modéliser et implémenter la variabilité dans les services Web. Dans (Koning *et al.,* 2009) la variabilité est implémentée dans les processus métiers en proposant un langage VxBPEL (une extension du BPEL). Cependant, toutes ces approches se concentrent sur la variabilité dans les systèmes SOA, mais ils ne traitent pas la variabilité dans le contexte d'une application multi-locataire.

### 5.2. *SaaS et multi-location*

Dans (Jansen *et al.,* 2010) les auteurs fournissent un catalogue des techniques de personnalisation qui peuvent guider les développeurs dans le développement des applications multi-locataire. Dans (Bezemer *et al.,* 2010a) les auteurs discutent leurs expériences dans la ré-ingénierie des applications à locataire unique vers des applications multi-locataire. Dans (Mietzner *et al.,* 2009), les auteurs proposent une technique de modélisation de la variabilité dans les applications multi-locataires, ils différencient entre la variabilité interne seulement visible pour les développeurs, et la variabilité externe qui est communiquée aux locataires de l'application. Cependant, ils ne tiennent pas compte de la gestion de la variabilité à travers les différentes couches de l'application, ce que nous avons traité dans ce papier.

## 6. Conclusion et travail futur

Dans ce travail, nous avons montré l'importance et la complexité de la gestion de la variabilité dans les applications multi-locataire. Nous avons également motivé la nécessité de découpler l'implémentation de la variabilité sur les différentes couches de l'application. Nous avons commencé à partir d'un modèle de variabilité d'un cas

concret et nous avons montré différentes techniques pour implémenter la variabilité sur chaque couche du système.

Nos travaux futurs consistent à extraire les propriétés des variations qui ont influencé nos décisions d'implémentation. Une fois ces propriétés identifiées, elles peuvent être évaluées à chaque variation. Cela peut énormément aider les architectes et les développeurs à prendre les bonnes décisions et à accélérer cette tâche.

## 7. Bibliographie


Bayer J., Gerard S., Haugen O., Mansell J., Moller-Pedersen B., Oldevik J., Tessier P., Thibault J., Widen T., « Consolidated product line variability modeling », 2006.

Bezemer C., Zaidman A., « Multi-tenant SaaS applications : maintenance dream or nightmare ? », *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, ACM, p. 88-92, 2010a.

Bezemer C., Zaidman A., Platzbeecker B., Hurkmans T., t Hart A., « Enabling multi-tenancy : An industrial experience report », *Software Maintenance (ICSM), 2010 IEEE International Conference on*, IEEE, p. 1-8, 2010b.

Chang S., Kim S., « A variability modeling method for adaptable services in service-oriented computing », 2007.

Chong F., Carraro G., « Architecture strategies for catching the long tail », *MSDN Library, Microsoft Corporation*p. 9-10, 2006.

Foping F., Dokas I., Feehan J., Imran S., « A new hybrid schema-sharing technique for multitenant applications », *Digital Information Management, 2009. ICDIM 2009. Fourth International Conference on*, IEEE, p. 1-6, 2009.

Guo C., Sun W., Huang Y., Wang Z., Gao B., « A framework for native multi-tenancy application development and management », *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*, Ieee, p. 551-558, 2007.

Jansen S., Houben G., Brinkkemper S., « Customization realization in multi-tenant web applications : case studies from the library sector », *Web Engineering*p. 445-459, 2010.

Jegadeesan H., Balasubramaniam S., « A Method to Support Variability of Enterprise Services on the Cloud », *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, p. 117-124, 2009.

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*, 2007.

Khan A., Kästner C., Köppen V., Saake G., « Service variability patterns in SOC », *School of Computer Science, University of Magdeburg, Magdeburg, Germany, Tech. Rep*, 2011.

Kongdenfha W., Saint-Paul R., Benatallah B., Casati F., « An aspect-oriented framework for service adaptation », *Service-Oriented Computing–ICSOC 2006*p. 15-26, 2006.

Koning M., Sun C., Sinnema M., Avgeriou P., « VxBPEL : Supporting variability for Web services in BPEL », *Information and Software Technology*, vol. 51, n° 2, p. 258-269, 2009.

La H., Kim S., « Adapter patterns for resolving mismatches in service discovery », *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, Springer, p. 498-508, 2010.

Laplante P., Zhang J., Voas J., « What's in a Name ? Distinguishing between SaaS and SOA », *IT Professional*, vol. 10, n° 3, p. 46-50, 2008.

Mietzner R., Leymann F., « Generation of BPEL customization processes for SaaS applications from variability descriptors », *Services Computing, 2008. SCC'08. IEEE International Conference on*, vol. 2, IEEE, p. 359-366, 2008a.

Mietzner R., Leymann F., Papazoglou M., « Defining composite configurable SaaS application packages using SCA, variability descriptors and multi-tenancy patterns », *Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on*, IEEE, p. 156-161, 2008b.

Mietzner R., Metzger A., Leymann F., Pohl K., « Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications », *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*, IEEE Computer Society, p. 18-25, 2009.

Nguyen T., Colman A., Han J., « Modeling and Managing Variability in Process-Based Service Compositions », *Service-Oriented Computing*p. 404-420, 2011.

Phil Hochmuth Sally Hudson C. J. K. C. A. C. J. G., « Worldwide Software as a Service 2010-2014 Forecast : Software Will Never Be The Same. », *IDC Report Doc-223628*, 2010.

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

Razavian M., Khosravi R., « Modeling variability in business process models using uml », *Information Technology : New Generations, 2008. ITNG 2008. Fifth International Conference on*, IEEE, p. 82-87, 2008.

Sam Y., Boucelma O., Hacid M., « Web services customization : a composition-based approach », *Proceedings of the 6th international conference on Web engineering*, ACM, p. 25-31, 2006.

Schiller O., Schiller B., Brodt A., Mitschang B., « Native support of multi-tenancy in RDBMS for software as a service », *Proceedings of the 14th International Conference on Extending Database Technology*, ACM, p. 117-128, 2011.

Sengupta B., Roychoudhury A., « Engineering multi-tenant software-as-a-service systems », *Proceeding of the 3rd international workshop on Principles of engineering service-oriented systems*, ACM, p. 15-21, 2011.

Stollberg M., Muth M., « Service customization by variability modeling », *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, Springer, p. 425-434, 2010.

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*, vol. 83, n° 3, p. 502-516, 2010.

Svahnberg M., Van Gurp J., Bosch J., « A taxonomy of variability realization techniques », *Software : Practice and Experience*, vol. 35, n° 8, p. 705-754, 2005.

Xuxu Z., Qingzhong L., Lanju K., « A Data Storage Architecture Supporting Multi-level Customization for SaaS », *Web Information Systems and Applications Conference (WISA), 2010 7th*, IEEE, p. 106-109, 2010.