

## Migration d'applications à base de composants Web en services et orchestration de services Web

Chouki Tibermacine\* et Mohamed Lamine Kerdoudi †

\* LIRMM, CNRS et Université Montpellier II, France  
Chouki.Tibermacine@lirmm.fr

† Département d'informatique, Université de Biskra, Algérie  
lamine.kerdoudi@gmail.com

**Résumé.** Le développement à base de composants Web permet de construire des applications distribuées, par assemblage de composants existants obtenus à partir de bibliothèques de composants libres ou *COTS (Commercial Off-The-Shelf)*. Par ailleurs, la technologie des services Web s'est affirmée comme une des solutions les plus pertinentes pour les fournisseurs de services, comme eBay, Amazon ou FedEx, pour ouvrir leurs solutions aux développeurs tiers. Dans ce papier, nous présentons une approche pour construire des architectures orientées services Web à partir des applications à base de composants Web et les déployer sur un serveur supportant les services Web. Nous avons implémenté notre solution en utilisant une collection de technologies Java. Des composants Java EE sont les entrées de l'implémentation proposée, et un ensemble de services Web Java, des chorégraphies et des orchestrations de ces services sont fournis en sortie.

### 1 Introduction : contexte et motivations

Le développement à base de composants Web vise à découpler une application Web en modules et les rendre des entités logiciels réutilisables et personnalisables. En effet, une étape importante a été franchie vers cette modularisation dans le domaine des applications Web. Celle-ci a permis notamment de séparer la présentation (la vue), des données (le modèle) et des traitements (le contrôleur) dans l'application. Une des technologies leaders dans ce domaine est Java EE<sup>1</sup> et ses nombreux Frameworks comme Struts et JSF. Les composants Web dans ces technologies sont des entités qui peuvent être utilisés et réutilisés dans différentes applications et personnalisés en fonction des besoins des utilisateurs. Ces technologies se sont affirmées ces dernières années comme des solutions idéales pour le développement des applications complexes avec de fortes exigences en termes de maintenabilité. Une fois les composants, implémentant ces applications, sont déployés sur un serveur d'applications, il n'existe pas de moyen pour publier directement les services fournis par ces applications

---

<sup>1</sup> Java Enterprise Edition de Sun Microsystems : <http://java.sun.com/javaee/>

pour un développement par de tierces parties. Un développeur tiers ne pourra pas exploiter directement les services métier fournis par ces composants pour développer, à distance, des extensions à ceux-ci. Même, si des « souches » (*stubs*) peuvent être générées et fournies pour les applications clientes, ces souches sont dépendantes d'un langage de programmation particulier (seuls les clients Java peuvent utiliser les souches générées pour les composants EJB) et ne peuvent pas être publiés dans des bibliothèques de services.

Par ailleurs, depuis des années, la technologie des services Web s'est affirmée comme l'une des solutions les plus pertinentes pour les fournisseurs de services, comme eBay (enchères), Amazon (commerce électronique) ou FedEx (logistique), pour ouvrir leurs solutions aux développeurs tiers. Les services Web sont des fonctionnalités basées sur des standards qui sont indépendants d'un "langage de programmation" ou d'une "plate-forme d'exécution", comme WSDL (*Web Service Description Language*) ou SOAP. De nouvelles applications avec des clients légers ou lourds peuvent être construites pour accéder à ces fonctionnalités. Il suffit de formuler dans ces applications clientes des requêtes, qui intègrent des messages (SOAP) basés sur XML, et de les transmettre au fournisseur de services Web. Des messages du même type sont renvoyés à ces applications, contenant les résultats (réponse à la requête). Sur la base de ces résultats, plusieurs actions peuvent être effectuées par ces nouvelles applications pour implémenter une nouvelle logique métier. De cette façon, les fournisseurs de services Web, qui détiennent des ressources précieuses (comme les grandes bases de données de produits d'Amazon, ou des prévisions météorologiques de Météo France) offrent aux développeurs tiers la possibilité de construire des nouvelles applications par l'extension de ces services publics, et ainsi, capitaliser ces ressources.

Dans ce papier, nous présentons une approche (section 3) pour construire des architectures orientées services à partir des applications à base composants Web et de les déployer sur un serveur avec un support aux services Web. De cette façon, les développeurs des composants Web offrent aux autres développeurs la possibilité de construire des extensions de services offerts par leurs artefacts. Cette transformation passe par plusieurs étapes. Premièrement, une analyse des différents éléments, qui constituent les composants Web, est réalisée pour extraire l'ensemble de services Web potentiels. Ensuite, les messages d'entrée et de sortie liés à chaque service Web sont définis à partir des éléments analysés. Cette étape est suivie par un filtrage des opérations non pertinentes dans les services Web. Celles-ci sont éliminées sur la base d'un ensemble de contraintes de filtrage et avec l'intervention du développeur. Dans l'étape suivante, les dépendances entre les différents services Web sélectionnés sont identifiées, et utilisées pour construire des services Web composites. Finalement, l'ensemble de services Web validés (les primitifs et les composites) est déployé sur un serveur d'application, et publié sur le moteur de recherche Seekda<sup>2</sup>.

Nous avons implémenté notre solution avec une collection de technologies Java. Les composants Java EE sont les artefacts d'entrées de l'implémentation proposée, et un ensemble de services Web Java et des orchestrations/chorégraphies de ces services sont fournis en sortie. Avant de conclure et présenter les perspectives de ce travail à la fin de ce papier, nous faisons un résumé des travaux connexes.

Dans la prochaine section, nous présentons un exemple simple d'un composant Web qui est utilisé dans le reste du papier pour illustrer nos propositions.

---

<sup>2</sup> Moteur de recherché de services Web Seekda : <http://webservices.seekda.com/>

## 2 Exemple Illustratif

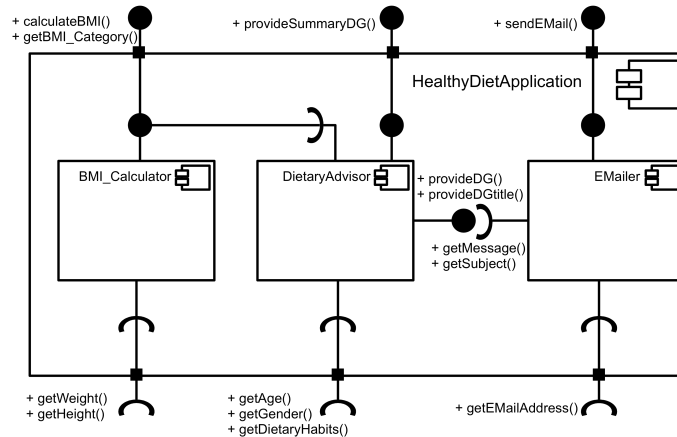


FIG. 1 – Une architecture abstraite du composant Web "HealthyDiet".

La Figure 1 montre une architecture abstraite d'un composant Web simple, qui est composé des trois sous-composants suivants :

- Le composant BMI\_Calculator calcule l'IMC<sup>3</sup> à partir des valeurs entrées par l'utilisateur via l'interface Web. Ces valeurs sont la taille et le poids.
- Le composant DietaryAdvisor demande à l'utilisateur d'entrer son âge et son titre de civilité (sexe), et de cocher certaines cases représentant les habitudes nutritionnelles et les troubles alimentaires de la personne (les faims de minuit, ...). Ce composant fournit un résumé des directives diététiques.
- Le composant EMailer demande à l'utilisateur d'entrer son adresse e-mail pour lui envoyer un e-mail avec une liste détaillée de directives diététiques.

Supposant maintenant, un développeur tiers qui souhaite étendre les services fournis par le composant présenté ci-dessus. Ce développeur souhaite apporter une solution pour vendre des produits et des recettes diététiques. Il cherche à utiliser la sortie obtenue à partir du DietaryAdvisor pour faire la publicité de ses produits avant d'envoyer l'e-mail à l'utilisateur. Pour implémenter cette solution, le tiers développeur doit soit : i) redévelopper la logique métier implémentée dans les composants Web présentés ci-dessus, ii) demander aux fournisseurs de ces composants les binaires (ou les packages source de ces composants) de l'application et les déployer sur le même espace de nom de sa solution, iii) ou demander un accès distant à ces composants. Ces scénarios rendent difficile l'accès aux composants Web pour les raisons suivantes :

**Flexibilité.** Les développeurs originaux doivent publier les composants dans un serveur de noms. Les Stubs pour donner l'accès à distance à ces services doivent aussi être générés et fournis aux développeurs tiers. Cela rend cette solution encombrante, surtout pour un déve-

<sup>3</sup> Indice de Masse Corporelle ou BMI en anglais (Body Mass Index). IMC = poids (kg) / taille<sup>2</sup> (m<sup>2</sup>)

## Migration des composants web en services et orchestrations de services web

loppement intensif par les tiers, dans le cadre d'un écosystème logiciel, par exemple. Le développement d'extensions à un système logiciel doit être ouvert aux développeurs tiers avec un minimum d'efforts.

**Portabilité.** Les développeurs tiers sont contraints d'utiliser le même langage d'implémentation que les composants originaux. Nous soutenons l'idée que le développement doit être ouvert aux développeurs tiers utilisant différentes technologies.

**Cohérence.** Supposons ici que les développeurs tiers déploient des instances de composants Web sur leurs machines. Cette solution encombrante oblige ces développeurs à installer et lancer un serveur d'application et un serveur de base de données. Cette solution peut introduire de nombreuses incohérences dans les données utilisées par ces composants. En effet, si de nouveaux standards et directives diététiques apparaissent, les différentes bases de données (utilisées par les instances originales et les nouvelles instances des composants Web) doivent être mises à jour et synchronisées.

Une solution intéressante pour ouvrir ce type de développement aux tierces parties est de transformer l'ancienne l'application en services Web qui sont décrits ci-dessous :

- *DietaryAdvising* : un service Web composé de deux opérations :
  - *calculateBMI* retourne une valeur réelle qui représente l'IMC.
  - *getBMI\_Category* retourne la catégorie de l'IMC : sous-poids, normal, surpoids ou obésité.
  - *provideSummaryDG* retourne un résumé des directives diététiques.
- *MailSending* : un service Web avec une seule opération pour l'envoi d'un e-mail.

Ces services Web sont présentés en détail à travers le processus de transformation et de génération de services Web de la section suivante. Le scénario d'extension de l'application discuté précédemment peut facilement être implémenté par l'invocation des opérations des services Web listées ci-dessus. Cette extension de l'application "HealthyDiet" peut être construite comme un processus métier (orchestration) basé sur ces services (voir les détails dans la section 3.4.2).

### 3 Approche proposée

Comme indiqué dans l'introduction, la transformation des composants Web en des services Web est un processus en cinq étapes illustré dans la Figure 2. Ce processus a été initialement introduit dans (Tibermacine et Kerdoudi, 2010). Dans ce papier, le processus est mieux détaillé et illustré par des exemples.

#### 3.1 Extraction des opérations

Premièrement, une analyse récursive des différents éléments du composant Web est effectuée pour extraire l'ensemble des services Web potentiels. Toutes les opérations dans les classes et les autres éléments de code structuré sont capturées. Par ailleurs, le code présent dans les programmes côté serveur (les pages JSP, par exemple) est formaté en une seule opération. Dans la sous-section suivante, nous expliquons comment ce formatage est réalisé.

Les opérations similaires sont réparties dans différents services Web. Cela assure un certain niveau de fiabilité et d'exactitude fonctionnelle dans les services Web publiés. D'un

côté, si un service Web n'assure pas correctement un besoin qualité (comme la disponibilité ou les performances) pour les utilisateurs d'une opération d'un service, un autre service contenant une opération similaire peut être trouvé chez le même fournisseur (fiabilité). D'un autre côté (et dans une perspective d'exactitude fonctionnelle dans la description du service Web), un service ne devrait pas fournir deux ou plusieurs opérations similaires.

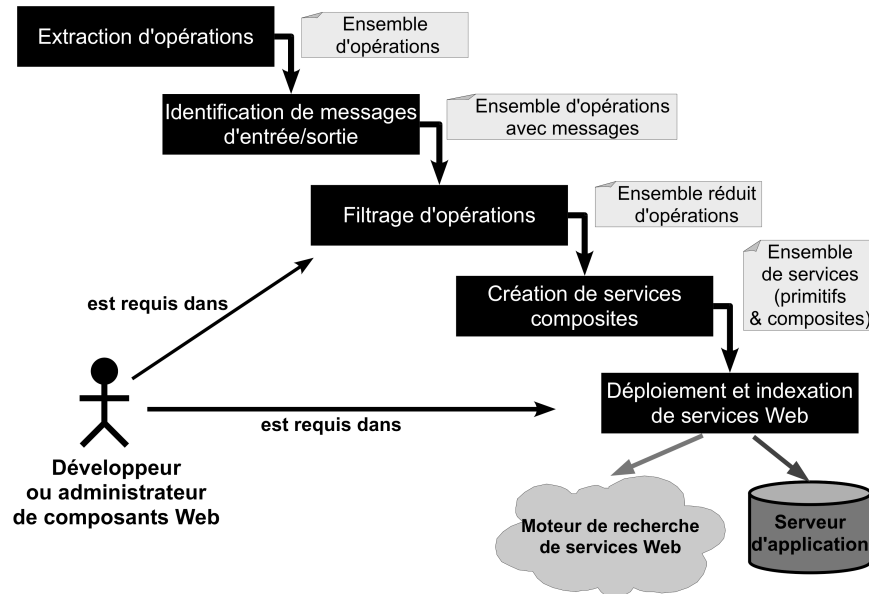


FIG. 2 – Le Processus de Transformation.

Les opérations similaires sont identifiées en utilisant certaines mesures de similarité. Dans l'implémentation actuelle, une solution basée sur les signatures des opérations a été mise en place. Elle compare le type de retour et les paramètres. Des techniques plus évoluées basées sur nos précédents travaux (Falleri et al. 2010) qui s'appuient sur l'extraction des mots clés à partir de la documentation et des signatures de services, et leurs synonymes vont être intégrées dans un futur proche.

### 3.2 Identification des messages d'entrée et de sortie

Les messages d'entrée et de sortie liés à chaque opération dans les services Web sont identifiés à partir des éléments analysés dans le composant Web. Pour les classes et d'autres éléments de code structuré, les paramètres et la valeur retournée sont formatés en messages SOAP. Le code présent dans les autres programmes (comme les pages JSP) est analysé pour extraire les valeurs d'entrée reçues dans les requêtes HTTP. Leurs types sont déduits à partir du code analysé par l'analyse statique des instructions de conversion de types.

Par exemple, dans le composant *BMI\_Calculator* de la section précédente, deux messages sont définis à partir des instructions `request.getParameter(...)` dans le fichier JSP. Ils correspondent à la taille et le poids. Au départ, ces entrées sont définies comme des variables de type chaîne de caractères. Mais, après une analyse du code JSP (Java) qui extrait ces va-

leurs, on déduit que le type final est le type float (réel). La valeur retournée est le résultat calculé par une méthode JavaBean liée à l'interface Web. Celle-ci est utilisée pour créer un message SOAP de type float (réel).

### 3.3 Filtrage des opérations

Dans cette étape, les opérations non-pertinentes des services Web sont éliminées de l'ensemble de départ selon une collection de contraintes de filtrage. Ces contraintes sont des expressions booléennes représentées par des expressions OCL<sup>4</sup> (OMG 2006), qui peuvent être ajoutées, modifiées ou supprimées par le développeur. Les expressions OCL sont limitées aux invariants. Elles naviguent dans un méta-modèle d'opérations (inspiré du méta-modèle UML2). OCL a été choisi en raison de sa simplicité (Briand et al. 2005) et de l'existence d'outils efficaces qui le supportent (OCL Toolkit (Dresden 2009) ou Eclipse MDT/OCL (Eclipse Foundation, 2009)).

Voici un exemple de contrainte de filtrage : les opérations qui utilisent une variable standard de session ne sont pas prises en compte. Celle-ci est formalisé comme suit :

```
not (self.body.usedType ->includes(t | t.name='HTTPSession'))
```

Enfin, le développeur est sollicité pour choisir parmi les opérations sélectionnées celles qui ne sont pas conceptuellement utiles en tant que parties de services Web publiés.

### 3.4 Création de services Web composites

Dans cette étape, les dépendances entre les différentes opérations sélectionnées dans les services Web sont identifiées. Il existe deux types de dépendances entre les opérations : des dépendances sous la forme d'appels d'opérations (qui donnent lieu à une chorégraphie de services Web) et des relations de navigation Web (qui permettent de générer des orchestrations de services Web). Celles-ci sont détaillées ci-dessous :

#### 3.4.1 Création de chorégraphies de services Web

Tous les appels d'opérations dans le code sont capturés. Si les opérations appelées sont publiées dans le même service Web que l'opération appelante, rien n'est fait, les appels sont laissés comme des invocations de méthode. Si les opérations appelées sont dans d'autres services Web publiés, nous vérifions si ces services sont déployés sur un serveur distant ou sur le même serveur, et s'ils s'exécutent dans la même machine virtuelle que le service appelant. Si les services sont déployés dans le même contexte d'exécution, les invocations de méthode sont laissées telles qu'elles. Sinon, ces dépendances d'opérations sont remplacées par des requêtes de services Web.

De cette façon, nous construisons des services Web composites comme des chorégraphies au niveau du code. Nous envisageons dans un futur proche de générer des spécifications de haut niveau de chorégraphies avec des langages conformes au «WS-CDL » (W3C 2005). Ces spécifications devront jouer un rôle important dans la maintenance/évolution des chorégraphies générées. Toute modification appliquée sur les collaborations de services Web devient plus simple, parce que nous aurions des spécifications à un haut niveau d'abstraction (par opposition aux chorégraphies au niveau du code).

---

<sup>4</sup> Object Constraint Language

Dans l'exemple précédent (section 2), l'opération *calculateBMI()* appelle d'autres opérations pour effectuer les calculs arithmétiques. Ces opérations (*power(n,m)*, *division(x, y)*, conversion du poids et de taille) sont extraites et assemblées dans deux différents services Web : *calculationWS* and *conversionWS*. L'opération *calculateBMI* commence premièrement par, l'invocation de l'opération de conversion du poids (*conversionWS*) pour transformer les grammes en kilogrammes. Cela est fait à travers l'appel à l'opération de division (*calculationWS*). Ensuite, les centimètres sont transformés en mètres en utilisant l'opération de conversion de taille, qui est basée sur la même opération de calcul. Enfin, les opérations *power(...)* et *division(...)* sont appelées pour obtenir la valeur finale de l'IMC. Il s'agit d'un exemple simple d'une chorégraphie créée à partir du composant Web *BMI\_Calculator*.

### 3.4.2 Création d'orchestration de services Web

Dans cette étape, les documents de navigation tels que les fichiers faces-config de composants JSF sont analysés. Cela permet l'identification des différentes relations entre les pages Web, et donc la spécification des collaborations potentielles des différents services Web extraits de ces pages. Cette tâche est implémentée selon l'algorithme suivant :

```
(01) algorithm WebNavigationParsing {
(02)   let process := ProcessFactory.createProcess();
(03)   for-each (navigationRule in WebNavigationDocument) {
(04)     let opFrom := parseSourceView(navigationRule.sourceView);
(05)     if(opFrom isNotPreviouslyInvokedInProcess process) {
(06)       let op1 := process.createInvocationTo(opFrom);
(07)       op1.setParameters(variables of process);
(08)       let returnedValue1 := op1.invoke();
(09)       process.store(opFrom,returnedValue1);
(10)     }
(11)     else {
(12)       let returnedValue1 := getStoredReturnedValueBy(opFrom);
(13)     }
(14)     let exp := navigationRule.executionCondition.expression;
(15)     let op := process.createInvocationTo(exp);
(16)     returnedValue := op.invoke();
(17)     if(returnedValue = navigationRule.executionCondition.value) {
(18)       let var := process.createVariable(returnedValue1);
(19)       let opTo := parseDestinationView(navigationRule.destinationView);
(20)       let op2 := process.createInvocationTo(opTo);
(21)       op2.setParameters(variables in process + var);
(22)       let returnedValue2 := op2.invoke();
(23)       process.store(opTo,returnedValue2);
(24)     }
(25)   }
(26) }
```

Dans cet algorithme, nous commençons par la création d'un processus, et pour chaque règle dans le document de navigation définie dans le composant Web analysé, nous identi-

fions l'opération source (ligne 04). L'opération source correspond à l'opération qui a été générée à partir de la page Web spécifiée dans la source de navigation. La même tâche est effectuée pour la page Web destination (ligne 19).

Une règle de navigation contient trois éléments : i) la vue source, qui représente la page(s) Web à partir de laquelle la navigation commence (la page qui représente le formulaire pour calculer l'IMC dans l'exemple présenté précédemment : *bmi.jsp*) ; ii) la vue destination, qui correspond à la page vers laquelle l'utilisateur sera dirigé automatiquement (la page Web qui présente l'interface du composant Web *DietaryAdvising* : *diet.jsp*); et iii) une condition d'exécution, qui contient une expression et une valeur (l'expression correspond à un appel à l'opération du JavaBean pour obtenir la catégorie de l'IMC dans le composant Web : `#{diet.getBMICategory}`), et la valeur est "obésité").

Pour chaque règle de navigation, nous testons premièrement, si cette opération est déjà appelée dans le processus durant l'analyse d'une autre règle de navigation (ligne 05). Cela assure que les invocations d'opérations ne sont pas dupliquées. Dans le cas d'une opération qui a été invoquée dans le processus, on récupère seulement la valeur retournée par cette invocation. Ce type de valeurs est stocké après chaque invocation d'opération (lignes 09 et 23). Ensuite, nous comparons la valeur obtenue par l'appel à l'opération définie dans l'expression de la condition et la valeur de cette condition (lignes 16 et 17). Si les deux valeurs sont égales, on invoque alors l'opération destination correspondante (lignes 19 à 22).

Dans l'exemple de la section 2, nous générons un processus BPEL (Business Process Execution Language). Dans la description de ce processus, nous invoquons premièrement l'opération *calculateBMI* du premier service. Ensuite, nous stockons le résultat dans une variable et invoquons la deuxième opération dans le même service pour obtenir la catégorie IMC, en utilisant le contenu de la variable comme message d'entrée. Si la valeur retournée, qui a été stockée dans la deuxième variable, est différente de "Taille normale" (c'est à dire, elle est égale aux trois autres catégories d'IMC), la troisième opération est invoquée, en utilisant la catégorie IMC stockée et d'autres informations (habitudes diététiques et troubles alimentaires) pour obtenir les directives diététiques correspondantes. Enfin, nous invoquons l'opération *SendMail* avec les données nécessaires.

A la fin de cette étape, le processus BPEL est exporté comme un service Web. Il est associé aux autres services primitifs pour le déploiement.

### 3.5 Déploiement et indexation de services Web

L'ensemble validé de services Web (les composites et primitifs) sont déployés sur un serveur d'application choisi par le développeur ou l'administrateur de l'application à base de composants Web. Si le développeur déploie les services Web sur des serveurs différents, les dépendances entre les opérations en collaboration dans les services composites sont résolues.

Pour l'indexation, la première tâche consiste à demander au développeur d'ouvrir une session ou s'inscrire pour un nouveau compte dans le moteur de recherche Seekda. Par la suite, l'intervention du développeur est requise pour entrer les URLs des services déployés sur le moteur de recherche Seekda via l'URL : [https://webservices.seekda.com/add\\_url.jsp](https://webservices.seekda.com/add_url.jsp). Une fois les services indexés, nous proposons aux développeurs un mécanisme intelligent d'extraction de mots clés, basé sur nos précédents travaux (Falleri et al. 2010). Ce mécanisme identifie des tags pour les services à partir de leur description WSDL. La liste de mots clés (basée sur les identificateurs dans les noms d'opérations de services, les noms de paramètres, ...) est proposée aux développeurs afin de tagger les services déployés.

## 4 Travaux connexes

Roger Lee et al. (2005) ont proposé une approche qui permet à un client de spécifier une requête pour rechercher une fonctionnalité donnée dans les composants déployés dans un serveur Web. En réponse, un service Web ou une composition de services Web est généré automatiquement. Comparativement à cette approche réactive, notre approche est plutôt proactive, elle ne réagit pas à une requête. Elle permet au développeur de l'application Web d'anticiper l'exportation de certaines fonctionnalités aux tierces parties.

Avec le système Wike (Han et Tokuda 2008), les développeurs peuvent définir des patrons pour extraire des informations partielles à partir de pages Web. Ces patrons sont encapsulés dans des opérations qui peuvent être exportées comme des services Web. Cette solution intéressante n'aborde pas le même problème que dans notre travail. L'analyse de pages Web basées sur un contenu statique n'est pas notre principale préoccupation. Wike est considéré comme une solution complémentaire à notre travail.

Plusieurs travaux dans la littérature ont proposé des techniques dirigées par les modèles pour générer des applications orientées services. Bauer and Müller (2004) ont définis une approche pour faire des transformations d'éléments de diagrammes de séquence UML2 (considérés comme des PIMs – Platform Independent Models) en compositions de services Web en BPEL (considérés comme des PSMs – Platform Specific Models). Dans (Yu et al, 2007) les auteurs ont proposé des règles de transformation pour convertir des modèles d'orchestrations spécifiés en CCM (Component Collaboration Architecture), qui fait partie du profil UML pour EDOC (Enterprise Distributed Object Computing) (OMG 2004), en spécifications BPEL. Une autre approche dirigée par les modèles pour la création des solutions orientées services a été proposée dans (Johnson et Brown, 2006). Dans ce travail, un profile UML a été défini pour la spécification des applications orientées services.

Tous ces travaux sont parfaitement complémentaires à notre approche. Notre transformation est effectuée de PSM à PSM. Les composants Web, qui sont des modèles spécifiques à une plateforme donnée (dans l'implémentation actuelle, JEE), sont convertis vers des services Web qui sont considérés comme d'autres modèles spécifiques à une plateforme (WSDL, Java et BPEL). Le profil UML présenté en (Johnson et Brown, 2006) peut être utilisé pour définir des modèles de haut niveau des services Web générés.

## 5 Conclusion et perspectives

Dans ce papier nous avons présenté un processus de transformation des applications à base composants Web en applications orientées services Web. Nous avons implémenté le processus de transformation dans l'outil WSGen: Web Service Generator. Les composants Web sont considérés ici comme des artefacts logiciels intégrant le code de logique métier et exportant des interfaces Web. Ce type de modules est analysé par WSGen, les différents éléments qui le composent sont extraits pour identifier les différents services Web et les collaborations possibles entre ces services. Tous les services qui en résultent sont déployés sur le Web. Nous considérons ces services, générés et déployés, comme des APIs distantes. Elles offrent la possibilité aux développeurs tiers d'étendre les fonctionnalités offertes par ces services, et d'exploiter les ressources utilisées par eux.

Au niveau conceptuel, nous envisageons d'étudier la formalisation de la transformation en un ensemble de règles déclaratives de haut niveau. Nous définissons ensuite ces

règles dans un langage déclaratif conforme au MOF/QVT (OMG, 2008) et intégrons ainsi notre solution dans un processus d'ingénierie dirigée par les modèles. Au niveau de l'outil, nous projetons de travailler sur la génération des spécifications de haut niveau des chorégraphies de services Web dans un langage de haut niveau conforme à « WS-CDL ».

## References

- B. Bauer and J. P. Müller. Mda applied: *From sequence diagrams to web service choreography*. In Proceedings of ICWE'04, pages 132–136. Springer-Verlag, 2004.
- L. C. Briand, Y. Labiche, M. Di Penta, and H. D. Yan-Bondoc. *An experimental investigation of formality in uml-based development*. IEEE TSE, 31(10), 2005.
- T. U. Dresden. Ocl compiler web site. <http://dresden-ocl.sourceforge.net/>, 2009.
- J.-R. Falleri, Z. Azmeh, M. Huchard, and C. Tibermacine. *Automatic tag identification in web service descriptions*. In Proceedings of WEBIST'10, April 2010.
- Eclipse Foundation. *Model development tools website*. <http://www.eclipse.org/modeling/mdt/?project=ocl>, 2009.
- H. Han and T. Tokuda. Wike: *A web information/knowledge extraction system for web service generation*. In Proceedings of ICWE'08, pages 354–357. IEEE CS, 2008.
- S. K. Johnson and A.W. Brown. *A model-driven development approach to creating service-oriented solutions*. In Proceedings of ICSOC'06, pages 624–636. Springer, 2006.
- R. Lee, A. Harikumar, C.-C. Chiang, H.-S. Yang, H.-K. Kim, and B. Kang. *A framework for dynamically converting components to web services*. In Proceedings of SERA'05, 2005.
- OMG. *Uml profile for enterprise distributed object computing (edoc)*. OMG Website: <http://www.omg.org/technology/documents/formal/edoc.htm>, 2004.
- OMG. *Object Constraint Language specification, version 2.0, document formal/2006-05-01*. OMG Website: <http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf>, 2006.
- OMG. *Meta object facility (mof) 2.0 query/view/ transformation specification, version 1.0, document formal/2008-04-03*. <http://www.omg.org/spec/QVT/1.0/PDF/>, April 2008.
- C. Tibermacine and M. L. Kerdoudi. *From Web Components to Web Services: Opening Development for Third Parties*. In Proceedings of ECSA'10, Springer-Verlag, 2010.
- W3C. *Web services choreography description language version 1.0, w3c candidate recommendation*. W3C Website: <http://www.w3.org/TR/ws-cdl-10/>, 2005.
- X. Yu, Y. Zhang, T. Zhang, L. Wang, J. Zhao, G. Zheng, and X. Li. *Towards a model driven approach to automatic bpel generation*. In Proc. of ECMDA, Springer-Verlag, 2007.

## Summary

Web component based development allows developing distributed applications by assembling existing COTS (Commercial Off-The-Shelf) or free components. Besides this, since many years, Web services confirmed their status of the most pertinent solution for a given service provider, like Google, Amazon or FedEx, to open its solutions for third party developers. In this paper, we present an approach to build web service-oriented architectures starting from existing web component-based applications and deploy them on a web service provider. This transformation helps server-side web application developers in transforming their "user interface"-based web components into a set of web services, choreographies and orchestrations. We implemented our solution on a collection of Java EE-related technologies.