Vers une adaptabilité dynamique des architectures orientées services

une approche basée sur la programmation par aspect et les algèbres de processus.

Mehdi Ben Hmida , Serge Haddad

Laboratoire LAMSADE (CNRS)
Université Paris-Dauphine,
Place du Maréchal de Lattre Tassigny,
Paris Cedex 16, France

RÉSUMÉ. *Actuellement, les services Web constituent la solution adéquate pour implémenter les architectures orientées service. Mais, cette technologie présente des limitations vis à vis du changement dynamique de service. D'une part, Les fournisseurs de service n'ont pas le moyen d'adapter dynamiquement un service Web existant aux changements de la logique métier. D'autre part, Les consommateurs du service n'ont pas le moyen d'adapter dynamiquement leur comportement à ce changement. Dans ce papier, nous montrons comment mettre en oeuvre une architecture orientée service auto adaptable en introduisant la programmation par aspect (PPA) et les algèbres de processus. Nous utilisons les principaux concepts de la PPA (point de jonction, coupe et conseil) dans le contexte des services Web pour changer le comportement d'un service Web (simple ou BPEL) sans toucher à son implémentation. Par la suite, nous donnons une formalisation à un service Web adaptable (service de base avec les aspects services) grâce aux algèbres de processus. Cette formalisation nous permet de générer automatiquement un client qui s'adapte dynamiquement au changement.*

ABSTRACT. *Currently, Web Services are the fitted technical solution to implement Service Oriented Architecture (SOA). However, this technology presents limitations concerning dynamic service adaptabilty. From one side, Web Services providers have no mean to dynamically adapt an existing Web Service to business requirements changes. From the other side, Web Services clients have no way to dynamically adapt themselves to the service changing in order to avoid execution failures. In this paper, we show how we achieve a dynamic adaptable SOA by introducing the Aspect Oriented Programming (AOP) paradigm and Process Algebra (PA). We apply the main AOP concepts (joinpoint, pointcut and advice) in the Web Service context to modify the behaviour of an existent Web Service without touching its implementation. Then, we pro-*

JFDLPA'07, pages 1 à 16

pose a Process Algebra formalism to specify a change-prone BPEL process (base service and aspect services) and shows how to generate automatically a client which dynamically adapt its behaviour to the service changes.

MOTS-CLÉS : *Architecture orientée service, services web, programmation orientée aspect, algèbres de processus, adaptabilité dynamique.*

KEYWORDS: *Service Oriented Architecture, Web Services, Aspect Oriented Programming, Process Algebra, Dynamic Adaptabilty.*

1. INTRODUCTION

Un service Web est un ensemble de protocoles et de normes informatique utilisés pour échanger des données entre les applications. Il représente une entité autonome qui est publiée, localisée et invoquée a travers le Web (Tidwell, 2000). Il est basé sur un ensemble de standards XML (Bray *et al.*, 2004) lui permettant d'être plus portable que les technologies précédentes (Booth *et al.*, 2002). Les services Web sont souvent composés pour implémenter la logique métier, grâce à un langage de composition qui est entrain de devenir un standard : le *Business Process Execution Language for Web Services* (Andrews *et al.*, 2003) (*BPEL4WS* ou *BPEL*). Ce langage décrit deux types de processus :

– *Les processus exécutables* spécifient les détails des processus métier et sont exécutés par un moteur BPEL.

– *les processus abstraits* spécifient l'échange public de messages entre le client et le service et constituent le protocole d'interaction.

Actuellement, les architectures orientées services présentent des limitations par rapport à leur adaptabilité dynamique aux changements de la logique métier. Ces limitations affectent les fournisseurs et les consommateurs de services.

D'un coté, les fournisseurs de services n'ont pas le moyen de changer dynamiquement une implémentation ou une composition de services existante. Ils sont obligés d'enlever le service, de le recodifier puis de le redéployer. Pendant ce temps, le service est rendu indisponible pour ses utilisateurs. D'un autre coté, comme le service est partagé par plusieurs applications clientes, si un changement affecte la description du service (WSDL : Web Service Description Language) ou le protocole d'interaction (BPEL abstrait), les consommateurs ne pourront plus interagir avec celui ci et aboutissent à des erreurs d'exécution.

Dans nos travaux antérieurs, nous avons traité les problématiques de l'adaptabilité dynamique du service ainsi que l'interaction correcte entre le client et le service. Nous avons proposé une approche basée sur la programmation par aspect (PPA) (Kiczales *et al.*, 1997) qui permet de changer le comportement d'un service Web au moment de l'exécution (Tomaz *et al.*, 2006, Hmida *et al.*, 2006). Nous avons proposé aussi une approche basée sur les algèbres de processus qui gére les interactions entre un processus BPEL et ses clients, ceci en spécifiant formellement le protocole d'interaction (BPEL abstrait) et en générant automatiquement un client qui communique correctement avec le service (Haddad *et al.*, 2006).

Dans ce papier, nous étendons ces travaux dans le but de spécifier formellement un service susceptible de changer pendant l'exécution et de générer automatiquement des clients qui adaptent dynamiquement leurs comportements aux changements du service. Nous montrons à travers une étude cas comment implémenter une architecture orientée service auto adaptable grâce à la PPA et aux algèbres de processus.

Ce papier est organisé comme suit : La section 2 présente notre étude de cas. La section 3 expose notre approche PPA qui traite de l'adaptabilité dynamique du service,

présente notre outil nommé le *tisseur des aspects services* et décrit une limitation de l'approche. La section 4 expose le formalisme basé sur les algèbres de processus qui spécifie un processus BPEL sujet au changement. Ce formalisme nous permet de générer des clients qui adaptent leurs comportements aux changements de service. La section 5 décrit les différentes phases du cycle de vie d'un service adaptable. La section 6 compare notre approche aux autres travaux de recherche existants. Nous concluons et présentons quelques nouvelles pistes de recherche dans la section 7.

2. ETUDE DE CAS

Nous prenons comme étude de cas, une compagnie qui développe des automates pour analyser le plasma sanguin. L'application de l'automate doit afficher une interface graphique spécifique à chaque profil et niveau de maturité de l'utilisateur. L'accès aux données est autorisé ou refusé selon le profil et sécurisé contre les accès non authentifiés.

La compagnie a décidé de promouvoir une architecture flexible et adaptable qui répond aux changements fréquents des besoins. La nouvelle architecture doit gérer l'évolution de la logique métier et la réutilisation des machines. Nous avons décidé d'utiliser la technologie des services Web pour implémenter cette architecture.

Considérons le scénario suivant : Nous avons un service Web qui fournit les résultats des analyses sanguines. Ce service Web a au départ une politique de sécurité qui utilise les jetons de sécurité Kerberos. La compagnie a opté par la suite pour changer cette politique de sécurité par une autre basée sur les certificats digitaux. En utilisant une approche classique, les développeurs de l'application de l'automate et ceux des applications clientes doivent réaliser manuellement les changements.

Du coté service, le développeur doit enlever le service, réécrire sa politique de sécurité et le redéployer. Cette procédure engendre une indisponibilité du service pendant une certaine période de temps et par conséquent, une indisponibilité des applications qui l'utilisent. Du coté client, si la signature de la méthode d'authentification ou bien le protocole d'interaction a changé, le développeur de l'application cliente doit réécrire manuellement toutes les invocations pour éviter les problèmes d'exécution.

3. ADAPTABILITÉ DYNAMIQUE DU SERVICE

Nous avons proposé l'architecture d'un outil qui utilise les concepts clé de la PPA (points de jonctions, coupes et conseils) dans le contexte des services Web (Tomaz *et al.*, 2006, Hmida *et al.*, 2006), pour modifier dynamiquement un service Web existant. Cet outil s'appelle le *tisseur des aspects services* ("Aspects Services Weaver (ASW)"). Le *ASW* agit de deux manières diffèrentes selon que nous voulons modifier dynamiquement un service Web élémentaire ou complexe.

1) Dans le cas d'un service Web élémentaire, le *ASW* intercepte les messages SOAP avant qu'ils n'arrivent au service et vérifie pendant l'exécution s'il y a un *aspect service* qui s'applique sur la méthode invoquée. S'il en trouve, le *ASW* dirige le message intercepté vers le *service conseil* adéquat selon son type (avant, aprés ou remplace). Les points de jonction sont définis sur l'interface du service Web (WSDL) en utilisant *XPath* (Clark *et al.*, 1999) comme langage de *coupe*. Les *services conseils* sont des services Web qui implémente le nouveau comportement. Le *ASW* joue le rôle d'intermédiaire entre le client et le service Web.

2) Dans le cas d'un service Web complexe (processus *BPEL*), le *ASW* contrôle l'exécution du document *BPEL*. Il vérifie avant l'exécution de chaque activité BPEL s'il existe des *aspects services* définis sur l'activité courante. S'il en trouve, l'*ASW* invoque les *services conseils* associés dépendant de leur type. Les points de jonction sont définis sur le document *BPEL* en utilisant *XPath*. Le *ASW* tourne coté serveur au niveau du moteur BPEL.

Considérons notre étude de cas dans le cas où nous voulons modifier des processus BPEL, le développeur de l'application de l'automate peut, par exemple, implémenter un *aspect service* appelé "analyseResults". Cet aspect indique à l'ASW qu'il faut invoquer un service d'authentification basée sur les certificats digitaux à la place du service d'authentification utilisant les jetons de sécurité Kerberos, et ceci avant d'invoquer les méthodes du service de base nécessitant une authentification. Il suffit au développeur de changer la référence du *service conseil* implémentant l'ancienne politique d'authentification par la référence du *service conseil* qui implémente la nouvelle. Par exemple, le '*fichier de description des aspects services*" dans la figure 1 indique que le *ASW* doit invoquer l'opération "digital-certificate" du *service conseil* dont la référence est "urn-wsdl-certificate-service", avant d'invoquer les méthodes qui correspondent avec l'expression XPath "//invoke[starts-with(@name,"SendResult")]" (les invocations des méthodes dont le nom commence par "SendResult").

De cette façon, pendant l'exécution du processus (étape 1 et 3 dans la figure 1), le *ASW* regarde dans le "*fichier de description des aspects services*" (étape 2 dans la Figure 1) s'il y a des *aspects services* qui s'appliquent sur l'activité courante. Il trouve par exemple, une correspondance entre l'invocation de la méthode "sendResult" et l'aspect "analyseResults". Par la suite, il invoque le nouveau comportement avant d'exécuter l'activité courante dans le cas d'un *service conseil* de type avant (étape 4 dans la figure 1). Après, il continue l'exécution normal du processus de base (étape 1 et 3 dans la figure 1).

Mais, que se passe t'il si le *service conseil* requiert de nouvelles interactions avec le client ? ces interactions ne sont pas attendues et vont provoquer des erreurs d'exécution. Nous résolvons ce problème en spécifiant formellement les interactions entre un service Web adaptable (le service de base et les aspects services) et ses clients. Nous montrons dans la section suivante comment générer automatiquement un client interagissant correctement avec un service adaptable, en étendant notre formalisme défini pour les processus BPEL abstraits.

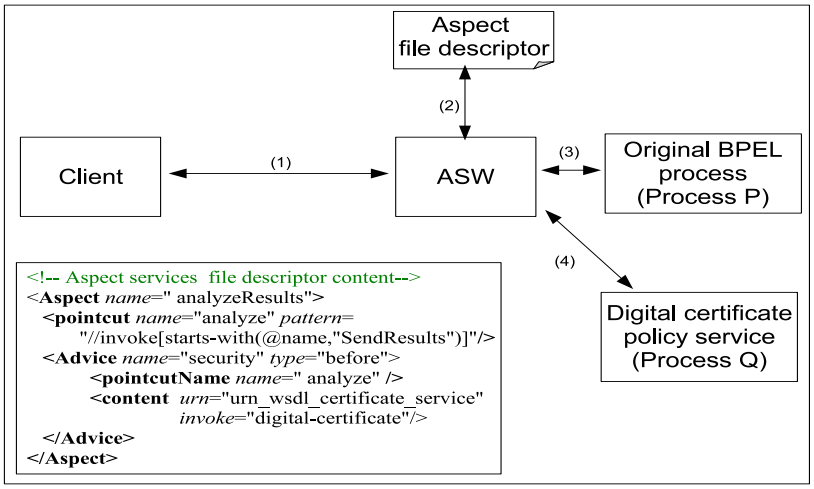

Figure 1. *Le schéma d'interaction pour l'insertion d'une politique de sécurité.*

4. ADAPTABILITE DYNAMIQUE DU CLIENT

BPEL fournit un ensemble d'opérateurs qui décrit de façon modulaire le comportement observable d'un processus abstrait. Dans l'article (Staab *et al.*, 2003), les auteurs montrent que ce type de processus est proche du paradigme des algèbres de processus illustré par exemple par CCS (Milner, 1995) ou CSP (Hoare, 1985). Par contre, les sémantiques existantes basées sur les algèbres de processus ne sont pas appropriées pour la description d'un tel processus parce que le temps est explicitement présent dans certains constructeurs BPEL.

Nous avons donc défini une nouvelle sémantique associant un *automate temporisé* (Alur *et al.*, 1994) à un processus BPEL abstrait (Haddad *et al.*, 2006). Nous utilisons cette formalisation pour prouver formellement que deux systèmes communicants (l'automate temporisé du service et l'automate de son client) interagissent correctement. Les développements théoriques suivent ces étapes :

1) L'association de règles opérationnelles à chaque opérateur BPEL abstrait.

2) La définition d'une relation d'interaction caractérisant le concept d'une interaction correcte entre deux systèmes communicants (le client et le service).

3) A partir de la relation d'interaction et de l'automate du service, nous générons automatiquement l'automate de son client. L'automate client peut ne pas exister si le service est prouvé formellement être ambigu.

4) L'automate client est interprété par un module générique présent au niveau de l'application cliente. Ce module récupère le protocole d'interaction du service, génére l'automate client associé et l'interprète. Le module affiche des interfaces de saisie permettant à l'utilisateur de saisir les paramètres requis par le processus.

Nous étendons ce formalisme pour représenter formellement un service Web qui se modifie à l'exécution. Nous pouvons ainsi générer un automate client qui s'adapte dynamiquement aux changements de l'automate du service. Dans la sous section suivante, nous présentons notre sémantique formelle pour les processus BPEL abstraits.

4.1. *Sémantique formelle des processus BPEL abstraits*

Dans le but de formaliser les processus BPEL abstraits, nous devons définir d'abord les actions (l'alphabet) de l'algèbre de processus. Les actions possibles sont la réception de message ($?m$), l'envoi de message ($!m$), les actions internes (τ) (non observables du coté client), la levée d'exception ($e \in E$), l'expiration d'un délai ($t0$) et la terminaison d'un processus ($\surd$). Nous distinguons trois types d'actions : les actions immédiates correspondant à une transition logique $(\tau, e, \surd)$, les actions asynchrones qui font écouler du temps avant l'exécution de l'action $(?m, !m)$ et l'action de synchronisation ($t0$) qui s'exécute après un délai fixé.

Maintenant, nous présentons les règles opérationnelles associées à quelques constructeurs BPEL. Pour la description de toutes les règles, le lecteur est invité à lire l'article (Haddad *et al.,* 2006).

Le processus $empty$ (représente le processus qui ne fait rien) ne peut que se terminer en exécutant l'action $\surd$.

$$empty \xrightarrow{\surd} O \qquad [1]$$

Le processus $?o[m]$ (resp. $!o[m]$) qui correspond à la réception d'un message de type m (resp. l'envoi d'un message de type m) exécute l'action $?m$ (resp. l'action $!m$) et devient le processus $empty$.

$$*o[m] \xrightarrow{*m} empty \qquad avec \ * \in \{?, !\} \qquad [2]$$

Le processus séquentiel $P; Q$ (P et Q sont des processus BPEL) qui correspond à l'exécution du processus P suivie de l'exécution du processus Q, devient le processus $P'; Q$ si le processus P exécute une action a différente de l'action de terminaison et devient P'. Si P se termine, le processus $P; Q$ exécute une action interne puis devient le processus Q.

$$\forall a \neq \surd \frac{P \xrightarrow{a} P'}{P;Q \xrightarrow{a} P';Q} \qquad [3]$$

$$\frac{P \xrightarrow{\surd}}{P;Q \xrightarrow{\tau} Q} \qquad [4]$$

En considérant notre étude de cas, le service Web qui fournit les résultats des analyses sanguines peut être formellement spécifié par le processus qui attend la réception d'un

message de type *Results request*, exécute un ensemble d'actions représentées par le processus P pour générer les résultats et renvoie le message *Results response*. La spécification du processus est la suivante :

$$La\ specification\ du\ service : \quad \textit{?o[ResReq] ; P ; !o[ResResp]}$$

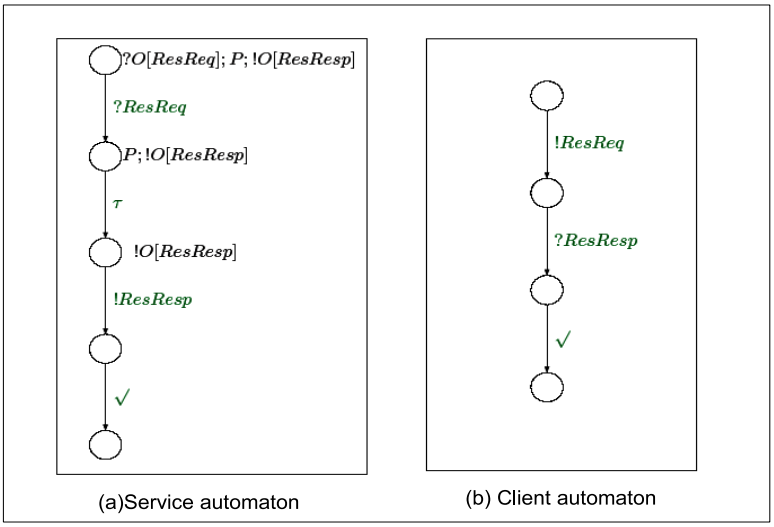

Figure 2. *Les automates service et client du processus des analyses sanguines.*

L'automate temporisé du service est représenté dans la figure 2.(a). Le service exécute l'action $?ResReq$ et devient le processus séquentiel P ; $!o[ResResp]$ (en appliquant la règle 3), puis devient le processus $!o[ResResp]$ quand P se termine (en appliquant la règle 4). Par la suite, le processus $!o[ResResp]$ exécute l'action $!ResResp$ et devient le processus $empty$ (en appliquant la règle 2) qui ne peut que se terminer (en appliquant la règle 1).

L'automate client qui communique correctement avec le service est généré automatiquement en se basant sur l'automate du service et la relation d'interaction. Dans la figure 2.(b), nous trouvons l'automate client. Cet automate exécute les actions complémentaires du service (un envoi de message du coté service correspond à une réception de message du coté client) et abstrait les actions internes τ.

4.2. *Sémantique formelle des processus BPEL adaptables*

4.2.1. *Ajout des opérateurs*

Nous étendons le formalisme précèdent avec trois nouveaux constructeurs PPA. Ces constructeurs permettent de spécifier des processus qui sont susceptibles de chan-

ger au moment de l'exécution. En considérant un processus BPEL abstrait P représentant le comportement de base, nous avons :

1) Le processus $Before(P)$ spécifie que P peut avoir un nouveau comportement qui va être introduit avant son exécution.

2) Le processus $After(P)$ spécifie que P peut avoir un nouveau comportement qui va être introduit après son exécution.

3) Le processus $Around(P)$ spécifie qu'il peut y avoir un nouveau comportement qui va remplacer P. Nous utilisons une sémantique de remplacement parce que nous considérons que le nouveau comportement peut encapsuler P.

En revenant à l'exemple précédent, si nous voulons spécifier la possibilité d'insérer au moment de l'exécution, un nouveau comportement (une politique d'authentification par exemple) avant l'exécution du sous processus P ; $!o[ResResp]$, le processus global est représenté formellement comme suit :

$$?o[ResReq]\ ;\ Before(\ P\ ;\ !o[ResResp]\ )$$

4.2.2. *Extension de l'alphabet*

Nous ajoutons trois actions dans l'alphabet de notre algèbre. Ces actions sont : $!execute(id, Q)$, $?execute(id, Q)$ (où Q représente le protocole d'interaction d'un *service conseil* identifié par id) et l'action $id.\surd$. L'action $!execute(id, Q)$ spécifie que le ASW envoie un message au client lui indiquant qu'il va commencer à exécuter le *service conseil* id. L'action $?execute(id, Q)$ spécifie que le client attend la réception d'un message, contenant l'identifiant et la description du *service conseil*. Enfin, l'action $id.\surd$ spécifie la terminaison du *service conseil* id.

Nous considérons notre étude de cas au moment de l'exécution. Avant l'exécution d'une action $!execute$, le *ASW* regarde dans le "*fichier de description des aspects services*" (étape 2 dans la figure 3) et trouve un *service conseil* d'authentification. Il génére alors l'automate correspondant (étape 4 dans la figure 3) et l'envoie au client en exécutant l'action $!execute$ (étape 5 dans la figure 3). Le client reçoit ce message en exécutant l'action complémentaire $?execute$, puis en extrait l'automate pour générer le client correspondant. De cette façon, le client peut interagir correctement avec le nouveau comportement (étape (1) et (6) dans la figure 3).

4.2.3. *La sémantique opérationnelle*

Nous présentons maintenant la sémantique opérationnelle associée à ces nouveaux opérateurs. Nous insistons sur le fait que ces opérateurs ne prennent pas en compte le temps parce que nous considérons que l'introduction d'un nouveau comportement est une action immédiate. Nous définissons l'ensemble $Advices(P)$ (où P est un processus BPEL abstrait), par l'ensemble des services conseils qui s'appliquent sur ce processus union les processus $empty$ et P. Le processus $empty$ nous sert à spécifier l'inexistence d'un *service conseil* qui s'applique à P. L'ajout du processus P à

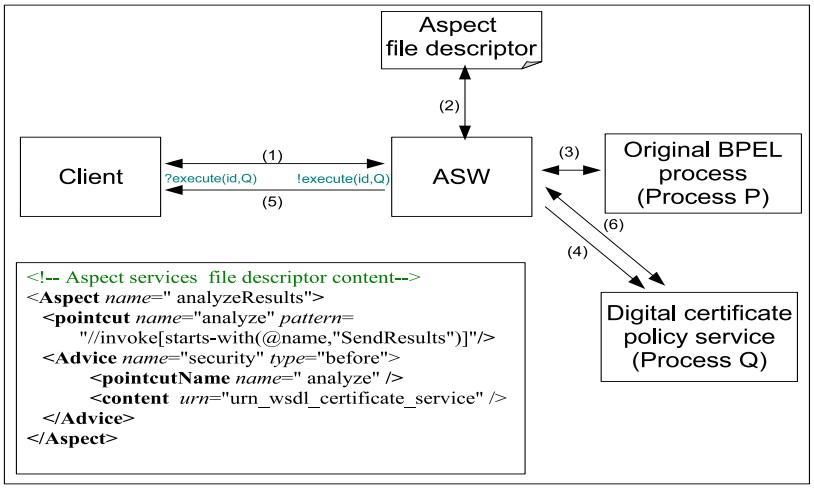

Figure 3. *Le nouveau schéma d'interaction pour l'insertion d'une politique de sécurité.*

l'ensemble $Advice(P)$ nous permet de définir la règle opérationnelle de l'opérateur $Around(P)$ (explicité dans les paragraphes suivants).

$$Advice(P) = \{sc | sc\ est\ un\ service\ conseil\ de\ P\} \cup \{empty, P\}$$

Nous notons aussi l'ajout de nouvelles règles opérationnelles pour les processus d'envoi et de réception de messages. Ces règles traitent du cas d'envoi et de réception du message $execute$. Ces règles sont énumérées ci dessous :

$$\forall m \neq execute \quad *o[m] \xrightarrow{*m} empty \qquad avec\ * \in \{?, !\} \tag{5}$$

$$*o[m] \xrightarrow{*execute(id,Q)} WaitAdvice(id) \qquad avec\ * \in \{?, !\} \tag{6}$$

$$WaitAdvice(id) \xrightarrow{id.\surd} empty \tag{7}$$

La règle 2 reste valable pour tous les messages qui sont différents de $execute$ (règle 5). Dans le cas d'envoi ou réception d'un message $execute$, l'automate évolue vers un état intermédiaire nommé $WaitAdvice(id)$ (règle 6). Cet état correspond à l'attente de la terminaison du service conseil id. Quand le service conseil id de termine, l'état $WaitAdvice(id)$ exécute l'action $id.\surd$ et devient le processus $empty$ (règle 7).

Le processus $Before(P)$ envoie par l'exécution de l'action $!execute$ un message contenant le nouveau comportement Q identifié par id et se transforme en le processus

séquentiel $WaitAdvice(id); P$. S'il n'y a pas de nouveaux comportements insérés à l'exécution, Q est le processus $empty$.

$$Before(P) \xrightarrow{!execute(id,Q)} WaitAdvice(id); P \; avec \; Q \in Advices(P) \quad [8]$$

Le processus $After(P)$ devient le processus $After(P')$ si P exécute une action a différente de l'action de terminaison $\surd$ et devient P'. Si P se termine, le processus $After(P)$ envoie par l'exécution de l'action $!execute$ un message contenant le nouveau comportement Q identifié par id et se transforme en $WaitAdvice(id)$. Q est le processus $empty$ s'il n'y a pas de nouveau comportement à insérer.

$$\forall a \neq \surd \frac{P \xrightarrow{a} P'}{After(P) \xrightarrow{a} After(P')} \quad [9]$$

$$\frac{P \xrightarrow{\surd}}{After(P) \xrightarrow{!execute(id,Q)} WaitAdvice(id)} \; avec \; Q \in Advices(P) \quad [10]$$

Finalement, le processus $Around(P)$ envoie par l'exécution de l'action $!execute$; un message contenant le nouveau comportement Q et se transforme en $WaitAdvice(id)$. S'il n'y pas de *services conseil*, Q est le processus P.

$$Around(P) \xrightarrow{!execute(id,Q)} WaitAdvice(id) \; avec \; Q \in Advices(P) \quad [11]$$

4.2.4. *Exemple d'exécution*

En considérant notre étude de cas, nous voulons insérer dynamiquement un processus d'authentification. Ce processus envoie une requête d'authentification au client demandant les informations d'authentification, reçoit ces informations et exécute un ensemble d'actions (représentées par $P1$) pour authentifier l'utilisateur. Le processus d'authentification est comme suit :

$$!o[authDataRequest] \; ; \; ?o[authDataResp] \; ; P1$$

Au cours de l'exécution du processus, quand l'automate du service adaptable est à l'état $Before(P \; ; \; !o[ResResp])$ (voir la figure 4), le service envoie la spécification du processus d'authentification en exécutant l'action $!execute(1, \; !o[authDataRequest] \; ; \; ?o[authDataResp] \; ; \; P1)$, évolue vers l'état $WaitAdvice(1)$, puis commence l'exécution du service conseil identifié par 1. Quand le service conseil se termine, le *ASW* exécute l'action $1.\surd$, puis continue l'exécution du processus de base.

Coté client, quand le client est à l'état correspondant au deuxième noeud de l'automate de la figure 5, il s'attend à recevoir un nouveau comportement. Par la suite, il reçoit ce comportement en exécutant l'action

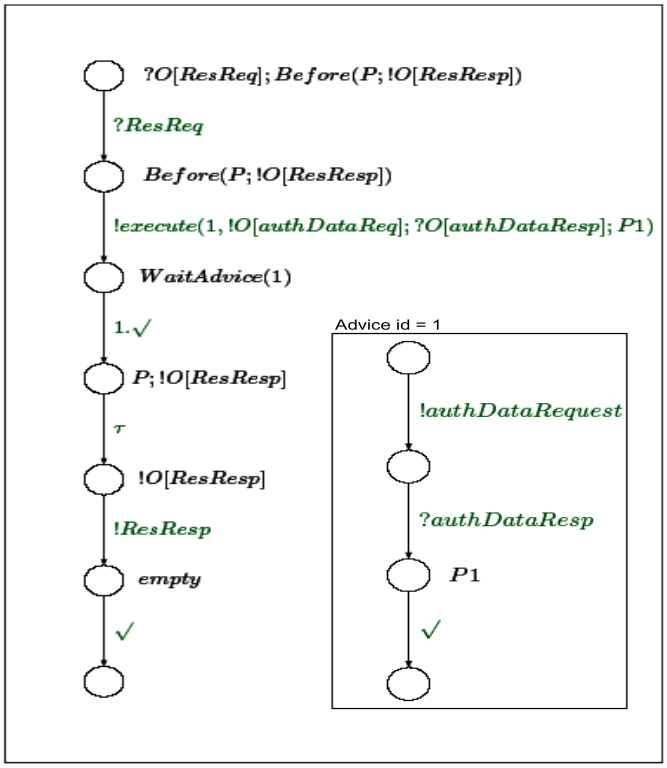

Figure 4. *L'automate du processus adaptable.*

$?execute(1, !o[authDataRequest]\ ;\ ?o[authDataResp]\ ;\ P1)$ puis évolue vers l'état $WaitAdvice(1)$. Il extrait l'automate du nouveau processus, génére l'automate client qui communique correctement avec celui ci, puis commence son exécution. Quand le nouvel automate se termine, l'automate principal exécute l'action $1.\surd$ puis continue son exécution normale.

5. CYCLE DE DÉVELOPPEMENT D'UN SERVICE WEB ADAPTABLE

Le cycle de développement d'un service Web BPEL se trouve modifier avec notre architecture. Le développeur du service Web doit de plus, prévoir les parties du processus qui sont susceptibles de changer, lors de la phase de conception du processus. Les différentes phase du cycle de vie d'un processus adaptable sont énumérées ci dessous.

Phase de conception

1) Le développeur du service crée son processus BPEL exécutable.

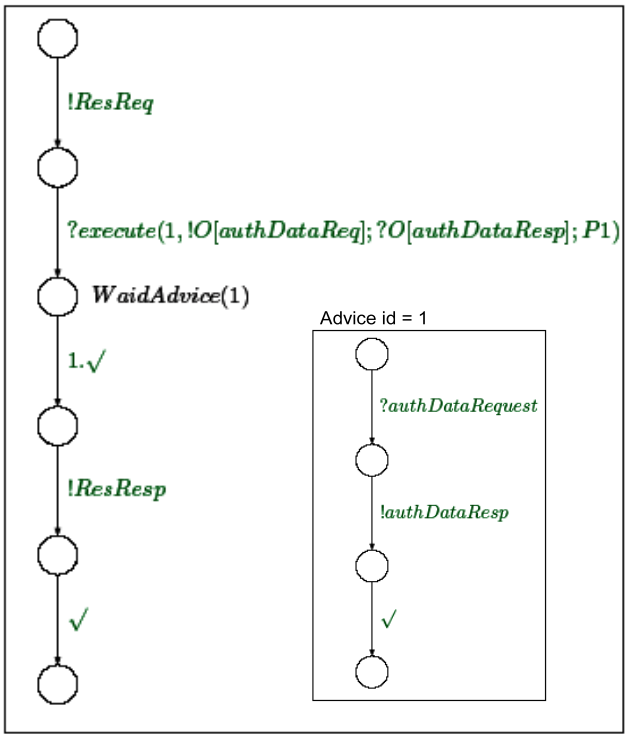

Figure 5. *L'automate du client dynamique.*

2) Il identifie par la suite, les parties du processus sujet aux changements. Il spécifie grâce à XPath, des coupes correspondant aux parties identifiées et les enregistrent dans le *fichier de description des aspects services*. Il définit aussi le type des *services conseils* associés à ces coupes (avant, aprés, remplace). Les coupes sont encapsulées dans un ensemble d'*aspects services* catégorisés par type de besoin fonctionnel ou non fonctionnel susceptible d'être modifié. Les *services conseils* associés à ces coupes ne sont pas renseignés à cette phase.

Phase de déploiement

3) A partir du processus BPEL et du *fichier de description des aspects services* (uniquement les coupes et le type des *services conseils*), le *ASW* génére l'automate adaptable du service. Cet automate est traduit automatiquement vers un processus BPEL abstrait (le protocole d'interaction) qui est publié sur un registre UDDI par exemple. Le protocole d'interaction comprend l'envoi des messages $execute$ qui doit être pris en compte lors de l'implémentation du client.

4) Le processus BPEL est déployé sur un serveur BPEL qui intègre le *ASW*. A cette phase, le développeur peut spécifier aussi les *services conseils* dans le *fichier de description des aspects services*. L'intégration du *ASW* au niveau du moteur est nécessaire si nous voulons modifier dynamiquement le processus BPEL.

Phase d'exécution

5) Pour utiliser le service adaptable, l'application cliente doit prendre en compte l'envoi des messages $execute$. Elle peut aussi intégrer notre module générique (début de la section 4).

6) Avec notre module générique, l'application cliente télécharge le protocole d'interaction à partir du registre UDDI, génére son automate du client et commence son exécution.

7) A la réception d'un message $execute$, le module en extrait le protocole d'interaction du *service conseil*, associe à cet automate l'id envoyé, et commence son exécution. Il affiche du coté client un ensemble d'interfaces graphiques lui permettant de renseigner les paramètres nécessaires au nouveau comportement.

8) A la terminaison du *service conseil*, l'automate principal du client franchit la transition $id.\surd$ et continue son exécution normale.

9) Au moment de l'exécution, le développeur du service peut redéfinir l'ensemble des *services conseils* spécifiés à la phase de déploiement.

6. DISCUSSION ET TRAVAUX RELIÉS

Dans les articles (Charfi *et al.*, 2004) et (Courbis *et al.*, 2005), les auteurs proposent deux approches basées sur la POA pour augmenter la flexibilité des services Web. Ces approches définissent des langages POA spécifiques pour ajouter dynamiquement des nouveaux comportements aux processus BPEL. Mais, aucune de ces approches ne prennent en compte la problématique de l'interaction avec le client. Elles ne décrivent pas un moyen pour spécifier des processus BPEL susceptibles de changer dynamiquement. Par conséquent, le client n'a pas la possibilité de prévoir les interactions qui peuvent être ajoutées ou modifiées pendant l'exécution du processus. Par contre, notre approche propose de spécifier formellement un tel service et permet de générer automatiquement un client qui prend en compte ses changements dynamiques.

La plateforme "Web Service Management Layer (WSML)" (Verheecke *et al.*, 2003) est une plateforme POA qui assure un couplage plus faible entre les clients et les services. WSML gère l'intégration dynamique de nouveaux services Web dans les applications clientes. WSML découvre dynamiquement ces services Web en se basant sur des critères de correspondance comme : la signature des méthodes, le protocole d'interaction ou la qualité de service (QdS). Notre approche diffère de WSML dans le sens où elle permet d'adapter un client à un service modifié et non de le remplacer par un autre. De plus, notre approche n'impose pas de correspondance qu'elle soit syntaxique ou au niveau du protocole d'interaction, mais propose un mécanisme pour gérer leurs modifications dynamiquement.

Des propositions ont émergé récemment pour spécifier formellement les services Web. La plupart de ces approches utilisent le modèle des systèmes de transition (système de transition labelisés , les réseaux de pétri, etc.) (Hamadi *et al.*, 2003, Fu *et al.*, 2004, Ferrara, 2004). Ces travaux proposent de spécifier formellement des services Web complexes afin de traiter les problématiques de vérification et de composition automatique. Mais, aucun de ces travaux ne proposent de formaliser la dynamique des architectures orientées services et de gérer le changement dynamique des interactions.

Notre approche permet donc de changer dynamiquement des services sans toucher à leur implémentation et d'adapter dynamiquement les clients aux changements du service. Par contre, l'approche ne permet pas d'adapter un client selon son contexte ou bien d'adapter l'exécution du service selon un client particulier. Par exemple, en considérant notre étude de cas, si nous voulons que les clients locaux continuent à utiliser la politique du jeton kerberos et les clients distants celle de la certification digitale, notre approche ne le permet pas. Le ASW permet de changer le comportement du service de façon globale.

Pour combler cette limitation, nous devons ajouter un fichier de contexte spécifique à chaque consommateur de service. La génération du service adaptable se fera en prenant en compte, les informations contenues dans ces fichiers.

7. CONCLUSION ET TRAVAUX FUTURS

Dans ce papier, nous avons proposé une solution basée sur la PPA et les algèbres de processus pour gérer l'adapatabilité dynamique dans les architectures orientée services. Nous avons étendu notre formalisme précédent pour spécifier des processus BPEL adaptables. Cette spécification nous permet de générer des clients qui s'adaptent au changement dynamique du service. Nous avons montré aussi, à travers une étude de cas, une mise en oeuvre de cette architecture et avons présenté le cycle de développement d'un service Web adaptable.

Nous proposons dans nos travaux futurs d'étendre l'approche pour prendre en compte le contexte du client. Nous proposons aussi de traiter la problématique de l'interaction entre les aspects (les aspects qui s'appliquent sur un même point de jonction). Finalement, nous ajoutons le nouveau formalisme à l'implémentation courante du ASW comme preuve de concepts.

8. Bibliographie

Alur R., Dill D. L., « A theory of timed automata », *Theoretical Computer Science*, vol. 126, n° 2, p. 183-235, 1994.

Andrews T. et al., *Business Process Execution Language for Web Services*, 2nd public draft release, Version 1.1. May, 2003.

Booth D., Haas H., McCabe F., al., « Web Services Architecture, W3C Working Draft », W3C Recommendation, 2002.

Bray T., Paoli J., Sperberg-McQueen C. M., Maler E., Yergeau F., « Extensible Markup Language(XML) 1.0 », 2004.

Charfi A., Mezini M., « Aspect-Oriented Web Service Composition with AO4BPEL », *Proceedings of the 2nd European Conference on Web Services (ECOWS)*, vol. 3250 of *LNCS*, Springer, p. 168-182, September, 2004.

Clark J., DeRose S., « XML Path Language (XPath) Ver. 1.0 », 1999.

Courbis C., Finkelstein A., « Weaving Aspects into Web Service Orchestrations », *ICWS '05 : Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, IEEE Computer Society, Washington, DC, USA, p. 219-226, 2005.

Ferrara A., « Web services : a process algebra approach », *ICSOC '04 : Proceedings of the 2nd international conference on Service oriented computing*, ACM Press, New York, NY, USA, p. 242-251, 2004.

Fu X., Bultan T., Su J., « Analysis of interacting BPEL web services », *WWW '04 : Proceedings of the 13th international conference on World Wide Web*, ACM Press, New York, NY, USA, p. 621-630, 2004.

Haddad S., Moreaux P., Rampacek S., « Client Synthesis for Web Services by Way of a Timed Semantics. », *ICEIS (4)*, p. 19-26, 2006.

Hamadi R., Benatallah B., « A Petri net-based model for web service composition », *ADC '03 : Proceedings of the 14th Australasian database conference*, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, p. 191-200, 2003.

Hmida M. B., Tomaz R. F., Monfort V., « Applying AOP Concepts to Increase Web Services Flexibility », *Journal of Digital Information Management (JDIM)*, vol. 4, n° 1, p. 37-43, 2006.

Hoare C. A. R., *Communicating sequential processes*, Prentice-Hall, USA, 1985.

Kiczales G., Lamping J., Maeda C., Lopes C., « Aspect-Oriented Programming », *Proceedings European Conference on Object-Oriented Programming*, vol. 1241, Springer-Verlag, Berlin, Heidelberg, and New York, p. 220-242, 1997.

Milner R., *Communication and concurrency*, Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1995.

Nicollin X., Sifakis J., « The Algebra of Timed Processes ATP : Theory and Application », *Information and Computation*, vol. 114, n° 1, p. 131-178, 1994.

Staab S., van der Aalst W., Benjamins V. R., « Web services : been there, done that ? », *IEEE Intelligent Systems [see also IEEE Intelligent Systems and Their Applications]*, vol. 18, n° 1, p. 72-85, 2003.

Tidwell D., « Web Services :The Web's Next Revolution », 2000.

Tomaz R. F., Hmida M. B., Monfort V., « Concrete Solutions for Web Services Adaptability Using Policies and Aspects », *The International Journal of Cooperative Information Systems (IJCIS)*, vol. 15, n° 3, p. 415-438, 2006.

Verheecke B., Cibrán M., Jonckers V., « AOP for Dynamic Configuration and Management of Web Services », *Proceedings of the International Conference on Web Services Europe 2003*, vol. 2853, Springer, p. 137-151, 2003.