REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDERAHMANE MIRA DE BEJAIA
FACULTE DES SCIENCES EXACTES

DEPARTEMENT D'INFORMATIQUE
ECOLE DOCTORALE RESEAUX ET SYSTEMES DISTRIBUES

Mémoire de Magistère

En vue de l'obtention du diplôme de magistère en informatique

Option : Réseaux et Systèmes Distribués

Thème

Spécification et analyse des politiques privées dans la composition de services Web

Présenté par

Mr. Tareq BOUCHAAR

Devant le jury composé de :

Président :	DAHMANI	Abdelnasser	Professeur	U. A. Mira, Bejaïa.
Rapporteur :	GODART	Claude	Professeur	U. Henri Poincaré, Nancy1.
Examinateur :	BOUKERRAM	Abdellah	M. C.	Université de Sétif.
Examinateur :	MELIT	Ali	M. C.	Université de Jijel.
Invitée :	GERMOUCHE	Nawal	Doctorante	U. Henri Poincaré, Nancy1.

Promotion 2006-2007

Dédicaces

Á ma mère

Á mon père

Á mes frères et sœurs

Á ma grande famille

Remerciements

Je tiens à remercier tous les gens qui ont collaboré, de près ou de loin, à la réalisation de ce mémoire. En premier lieu, je remercie Ma Co-promotrice **Nawal GUERMOUCHE** Doctorante à l'université de Henri Poincaré, Nancy 1 d'avoir accepté d'être mon encadreur durant cette année de magistère, et pour la confiance qu'elle ma donnée et ses précieux conseils et aides.

Je tiens à remercier vivement les responsables de l'école doctorale *ReSyD* et précisément son directeur Monsieur **TARI Abdelkamel**, docteur à l'université de Bejaia et chef de département d'informatique, ainsi que tous les enseignants qui nous ont assuré une formation de haute qualité, riche et d'actualité.

Je remercie, très particulièrement, tous **les étudiants** de l'école doctorale *ReSyD*, pour l'environnement de travail très agréable, pour leurs remarques pertinentes durant ces deux années de magistère.

Mes remerciements vont également aux membres de jury qui ont accepté de juger ce travail.

Résumé

Dans le domaine des services Web, la protection de la vie privée des utilisateurs est un enjeu majeur, plusieurs travaux de spécifications et de langages existent aujourd'hui, dont l'objectif est d'assurer la confidentialité des données privées pour les utilisateurs de services Web. Le protocole P3P (plate forme for privacy policy) est le plus connu dans ce domaine, il permet aux services Web de décrire, avec des politiques privées, leurs façons de traitement des données privées et pour les utilisateurs leurs préférences de confidentialité. Mais dans une composition automatique de services ou ces derniers bougent tout le temps (disparaissent, apparaissent ou évoluent), l'utilisateur ne peut pas assurer que ces préférences de confidentialité seront respectées par tous les services. Parce que, d'une part, il ne connait pas l'ensemble des services qui vont participer à la satisfaction de sa requête et d'autre part, ils peuvent exister des services tiers qui participent dans la composition indirectement, i.e. ils sont invoqués par des services candidats. Dans ce travail, nous allons contribuer dans ce problème et pour cela, nous allons proposer une approche de composition basée sur les politiques privées. Les services Web sont modélisés par des automates d'état fini (FSM) capables de communiquer par l'échange de messages, nous allons intégrer les politiques privées dans les transitions de ces derniers et nous allons proposer un algorithme pour la composition des services qui détecte tout genre de violation de la confidentialité des données privée.

Mots-clés: *Politiques privées, SOA, composition de services Web, analyse privée.*

Abstract

In the Web services area, the user privacy protection is a key issue. Many specifications and languages have been achieved today try to ensure the confidentiality of private data to Web services users. The P3P platform (Platform for Privacy Preferences) is the most significant proposal in this area, it allows websites to describe, with private policies, their practices on users' personal data and to communicate to users their privacy preferences. However, in an automatic composition of Web services, where they change dynamically over time (disappear, appear or change), the user can't ensure that such privacy preferences will be respected by all the services. This is because, first, the user doesn't know the set of services that will participate in the satisfaction of his request and, second, they may exist some third-party services that participate indirectly in the composition, namely those invoked by candidate services. Our work aims to solve this issue, we propose a composition approach based on private policies. Web services are modeled by finite state machines (FSM) which communicate by messages passing. We include the private policies in the transitions and we propose an algorithm for service composition, which detects any violation of the confidentiality of private data.

Keywords: *Privacy policy, SOA, Web services composition, privacy analysis.*

Sommaire

Liste des figures

Introduction générale

Aujourd'hui les entreprises travaillant dans le domaine des technologies de l'information font face à une augmentation importante de la complexité de leurs systèmes, ce qui implique de nombreux besoins. L'hétérogénéité devient inévitable et ne doit plus être un frein. Des systèmes historiquement indépendants, acquis grâce à des fusions ou l'acquisition d'autres entreprises, devraient pouvoir être intégrés facilement, et interconnectés sans générer un problème de multiplication des interfaces dues à des liaisons point à point [CHT03]. Ce type d'entreprise a acquis au fil du temps un large éventail d'applications patrimoniales qu'elle doit également pouvoir réutiliser facilement plutôt que tout réinventé. En outre, le développement de nouveaux systèmes gagnerait à être plus facile et plus rapide. Une meilleure utilisation des ressources, en évitant les applications trop complexes et en privilégiant la modularité et la réutilisation, permettrait également un plus grand retour sur investissement.

Une solution de plus en plus adoptée est l'approche orientée service (*Service Oriented Architecture*). Ce type d'architecture favorise l'utilisation de patrons architecturaux [STAL06] pour faciliter l'intégration d'applications.

Par ailleurs, l'architecture orientée services est souvent assimilée aux services Web, et les deux termes sont utilisés de manière interchangeable. Bien que l'adoption de protocoles et de standards fondés sur des services Web permette de simplifier et de généraliser l'architecture orientée services, il s'agit bien de deux choses distinctes. L'architecture orientée services est une méthode de conception de systèmes permettant de déterminer comment des ressources informatiques seront intégrées et quels services seront accessibles aux utilisateurs. En revanche, les services Web constituent une méthodologie de mise en œuvre qui utilise des protocoles et des standards spécifiques fonctionnant dans le cadre d'une solution SOA.

Les services Web sont devenus une des principales technologies pour construire des applications de business complexes. Ils permettent aux applications d'intéropérer en se fondant sur des standards tels que XML, SOAP, WSDL, UDDI, etc. Quoique ces protocoles permettent de bâtir des applications et de les mettre en production, de nombreuses évolutions restent à apporter pour offrir la prise en compte de critères de qualité de service tels que la sécurité.

La sécurisation d'une infrastructure fondée sur une architecture SOA s'avère beaucoup plus complexe que celle des environnements traditionnels. L'agrégation (la composition) de services de provenances diverses implique la nécessité de pouvoir proposer un modèle d'abstraction qui assure la sécurité des données échangées entre les différents fournisseurs de services. Beaucoup de travaux visent à assurer la sécurité au sein d'une infrastructure basée sur les services Web, aujourd'hui, il existe plusieurs standards, spécification et langages tel que : ws-security, ws-policy, P3P, etc. qui ont

comme objectif d'assurer la sécurité des données privées entre les services Web et aussi entre le client, qui est l'origine des ces données privées, et les services disponibles pour faire le traitement désiré par ce client.

Parce que les services Web visent les consommateurs qui accompagnent toujours l'augmentation rapide de l'Internet car ils sont disponibles pour les banques, les achats en ligne, e-Learning, soins médicaux, et les gouvernements en ligne, etc. Donc, chacun de ces services exige l'information personnelle du consommateur dans un formulaire ou un autre. Cela mène à des inquiétudes sur la confidentialité de ces informations. Donc, pour les services Web, l'enjeu majeur est d'assurer la confidentialité pour les données privées.

L'objectif de ce travail consiste à proposer une méthode de composition de ces services Web, basée sur les politiques privées, ces services sont totalement indépendants et peuvent être fournis par plusieurs fournisseurs de services. Cette méthode de composition doit préserver la confidentialité des données privées, fournies par un utilisateur, par tous les services candidats. Pour cela, nous allons utiliser les politiques privées dont le rôle est double. Pour un utilisateur, elles permettent de définir des préférences de confidentialité, qui désignent comment ce dernier veut que ses données privées soient traitées. Et pour un service Web, elles permettent de définir la manière avec laquelle, ce service consomme les données privées des utilisateurs.

Ce mémoire est organisé comme suit : dans un premier chapitre, nous allons définir l'architecture orienté service (SOA), en expliquant son rôle dans l'intégration des systèmes hétérogènes et leurs interopérabilités, ainsi que les avantages qu'elle apporte pour les entreprises qui travaillent dans le domaine du B2B. Dans un deuxième chapitre, nous allons introduire la notion des services Web, en détaillant l'architecture de ces derniers, leur fonctionnement et les différentes technologies permettant la communication, la découverte et la description des services Web. Le troisième chapitre est un résumé pour des technologies (standards, spécifications et langages) qui permettent l'implémentation, la composition et la sécurité des services Web. Notre contribution vient dans le quatrième chapitre, dans lequel, nous allons expliquer les modèles de politiques privées et de services Web utilisés dans notre approche, ainsi que l'algorithme de composition proposé. Le dernier chapitre est consacré à l'implémentation de l'approche proposée et aux différents tests effectués pour valider l'efficacité de l'algorithme et sa capacité à calculer les compositions valides et à détecter les violations des données privées. Nous terminons ce mémoire par une conclusion générale et les perspectives futures de notre travail.

Chapitre 1. Architecture orientée service (SOA).

1.1. Introduction

L'orientation services permet d'organiser des ressources informatiques distribuées dans une solution intégrée, en éliminant les informations isolées, et conférant davantage de souplesse à l'entreprise. Elle organise les ressources informatiques en modules, en créant des processus métier faiblement couplés qui intègrent des informations provenant de différents systèmes. L'un des facteurs de réussite d'une architecture orientée services est la création de processus métier peu soumis aux contraintes de l'infrastructure informatique sous-jacente, afin de fournir à l'entreprise toute la latitude dont elle a besoin.

L'architecture orientée services permet de créer toute une nouvelle génération d'applications dynamiques (parfois nommées applications composites). Ces applications offrent aux utilisateurs des informations plus précises et plus complètes ainsi qu'une meilleur connaissance des processus, mais aussi la possibilité d'accéder à ces informations en utilisant la forme et la présentation les mieux adaptées, que ce soit par le biais du Web, d'un client riche ou d'un dispositif mobile.

1.2. Définition de l'architecture orientée services (SOA)

L'orientation services est un moyen d'intégrer des informations provenant de différents systèmes. Chaque ressource informatique, qu'il s'agisse d'une application, d'un système ou d'un partenaire commercial, est accessible en tant que service. Ces fonctionnalités sont accessibles via des interfaces ; cependant des problèmes peuvent survenir lorsque les fournisseurs de services n'utilisent pas les mêmes systèmes d'exploitation ou protocoles de communication, les services pouvant alors devenir inopérants.

L'orientation services utilise des protocoles standards et des interfaces conventionnelles, généralement des services Web, pour faciliter l'accès au logique métier et aux informations entre les divers services. Plus spécifiquement, l'architecture orientée services permet aux fonctionnalités et interfaces services d'être organisées en processus. Chaque processus est lui-même un service, qui offre désormais une nouvelle fonctionnalité complète. Dans la mesure où le nouveau processus est accessible via une interface standardisée, la mise en œuvre sous-jacente des fournisseurs de services individuels peut être modifiée sans que cela n'ait d'impact sur la façon dont le service est exploité.

1.3. Les avantages de l'architecture orienté service

L'architecture orientée services est avant tout un moyen de conférer à l'entreprise une plus grande souplesse à partir des investissements informatiques existants. Les solutions SOA connectent des systèmes et permettent ainsi d'automatiser des processus

de transfert d'informations auparavant manuels. Parallèlement, les solutions SOA créent les principaux services requis pour que les utilisateurs appropriés accèdent aux ressources adéquates.

Les avantages que génère l'architecture orientée services se situent à deux niveaux : au niveau de l'organisation informatique et au niveau de l'utilisateur. Au final, les avantages se cumulent pour engendrer une augmentation considérable de la flexibilité et de l'efficacité.

Au niveau du service informatique, l'intégration SOA simplifie la gestion de ressources distribuées entre plusieurs plateformes, exige moins de matériel, plus fiable, repose sur des standards et plus rentable.

Au niveau de l'entreprise, l'architecture orientée services permet le développement d'une nouvelle génération d'applications dynamiques répondant à des besoins précis de l'entreprise, indispensables à sa croissance et à sa compétitivité. Les solutions SOA offrent :

- **Une meilleure collaboration avec les clients et les fournisseurs :** En rendant des applications dynamiques et des services d'entreprise accessibles à des clients et à des fournisseurs externes, les solutions SOA permettent non seulement une meilleure collaboration, mais contribuent également à accroître la satisfaction des clients et des partenaires.
- **Une meilleure prise de décision :** En regroupant l'accès aux services et aux informations de l'entreprise dans un ensemble d'applications métier composites et dynamiques, les solutions SOA permettent aux décisionnaires de bénéficier d'informations plus précises et plus complètes. Ils ont également la possibilité d'accéder à ces informations en utilisant le format et la présentation (Web, client riche, dispositif mobile) qui répondent le mieux à leurs besoins.
- **Une plus grande efficacité des employés :** L'accès rationalisé aux systèmes et aux informations ainsi que l'amélioration des processus métier contribuent à accroître l'efficacité des employés. Les employés peuvent en effet se consacrer aux processus à valeur ajoutée importants et aux activités de collaboration, plutôt que d'avoir à se conformer aux limitations et restrictions des systèmes informatiques sous-jacents.

1.4. Les concepts de bases de l'architecture orienté service

Les principaux concepts intervenant dans l'architecture des services Web sont [MEL04]:

- **Le fournisseur du service** : d'un point de vue conceptuel, il désigne la personne ou l'organisation responsable juridiquement du service. D'un point de vue

opérationnel, les fournisseurs de services peuvent également désigner le serveur qui héberge les services déployés.

- **Le client du service** : comme pour le fournisseur, il représente une personne ou une organisation, comme il désigne également, d'un point de vue opérationnel, l'application cliente qui invoque le service.
- **Le registre des services** : il représente la personne ou l'organisation responsable de l'hébergement du dépôt de publication des services. D'un point de vue fonctionnel, il représente l'entité logicielle qui joue le rôle de l'intermédiaire entre les clients et les fournisseurs de services. Le concept registre ou dépôt de service est essentiel dans l'architecture services Web. Il joue un rôle central dans le processus de localisation des besoins et dans l'interopérabilité, car il est supposé fournir aux clients les informations techniques et sémantiques sur le fonctionnement du service et ceci dans des langages formels (interprétables par les machines).
- **Le service** : le service est une notion abstraite pour désigner les fonctionnalités d'un agent logiciel qui implémente une fonctionnalité du service. Quand on parle de service, il est essentiel de faire la distinction entre le service vu d'un angle sémantique et sa réalisation qui est l'entité logicielle. Nous allons détailler ce concept par la suite.
- **La description du service** : c'est la spécification du service exprimée dans un langage de description interprétable par les machines. Il existe deux niveaux de description de services. Une description technique dans laquelle le service est vu en termes de messages, de formats, de types, de protocoles de transport et d'une adresse physique. Cette description joue le rôle d'un contrat d'interaction entre le client et le service. La deuxième description concerne la sémantique du service. Cette dernière peut avoir une existence formelle comme elle peut prendre la forme d'un accord entre le fournisseur et le client.
- **Les messages** : c'est la plus petite unité d'échange de données entre les clients et les services. Le concept message ne reflète pas la sémantique de son contenu, seuls la structure et les mécanismes de transport sont considérés dans l'architecture. La structure des messages qui permettent l'invocation des différentes unités fonctionnelles qui composent le service doit figurer dans la description du service. Ce concept est détaillé aussi par la suite.
- **La ressource** : le concept ressource désigne l'identifiant du service. Il constitue un point important dans l'architecture des services Web. Chaque service doit avoir une identification en termes d'adresse.

1.5. Les acteurs classiques d'une SOA

Le SOA peut être à la fois une architecture au sens propre du terme et un modèle de programmation. De plus, comme le domaine de métier d'une entreprise est en constante évolution, les systèmes de cette entreprise doivent suivre cette évolution pour répondre à de nouveaux problèmes. Une architecture orientée service permettrait d'assembler des composants et des services pour construire et fournir dynamiquement des solutions à ces problèmes.

La figure 1 illustre une vision simpliste d'une architecture orientée service et des interactions entre les différents acteurs. C'est auprès d'un composant appelé **registre de services** ou **courtier de service** que les **fournisseurs de service** exposent leurs services grâce à leurs descripteurs de service. L'**usager** interroge le registre pour découvrir et sélectionner le service qui répond à ses exigences. La liaison entre l'usager et le fournisseur d'un service est réalisée une fois que des accords concernant l'utilisation du service sont établis. Un quatrième acteur peut apparaître : le **certifieur de service**, chargé de surveiller le respect de ces accords.

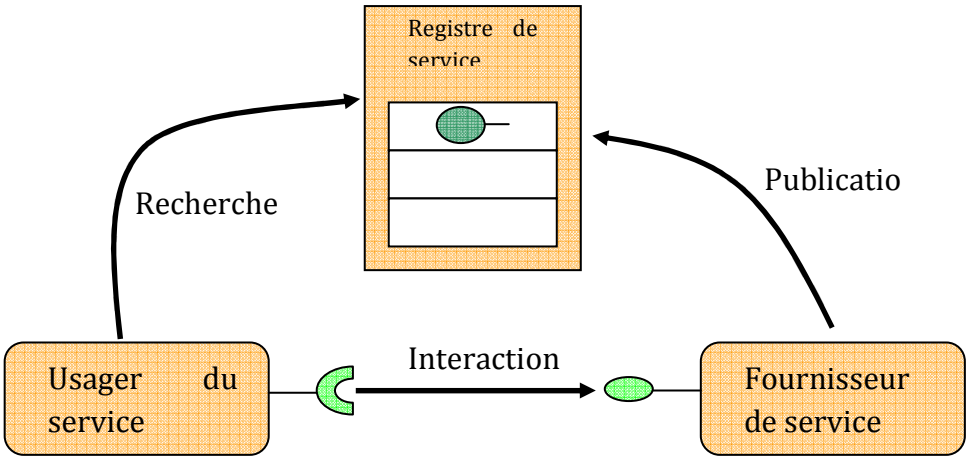


Figure 1. Les acteurs d'une architecture orientée service [TOU05].

Une architecture orientée service doit donc mettre à disposition des mécanismes de publication, de recherche (aussi appelée courtage) et d'invocation de service. L'invocation de service doit suivre un contrat défini entre le demandeur et le fournisseur. Enfin retenons que l'architecture orientée services, en tant qu'architecture, est indépendante du domaine d'application. Cependant des instances de cette architecture, spécifiques à un domaine, sont nécessaires pour la construction de systèmes concrets [LAK06].

1.6. Les principes d'une architecture orienté service

L'architecture orientée service se base sur les principes suivants [RAY07]:

- **Alignement métier** : Construire et organiser le système à partir des réalités métiers, qui doivent se retrouver dans ses constituants.
- **Neutralité technologique** : Assurer une indépendance totale entre les interfaces et les implémentations. L'élément qui utilise un service ne doit pas être contraint ni par la technologie d'implémentation, ni par sa localisation (potentiellement distribué).
- **Mutualisation** : Favoriser la réutilisation de services métiers par plusieurs lignes métiers ou applications. Permettre la construction de services de haut niveau par combinaison de services existants.
- **Automatisation des processus métier** : Isoler la logique des processus métiers sur des composants dédiés qui prennent en charge les enchaînements et les échanges de flux d'information.
- **Echanges orientés Document** : Les informations échangées par les services possèdent une structure propre, guidée par les besoins métiers.

1.7. Le cycle de vie d'une architecture orienté service

Les principaux actifs informatiques d'une entreprise sont ses données, ses systèmes existants, ses applications métier, ses progiciels et ses partenaires commerciaux. Chacune de ces ressources constitue un fournisseur de services chargé de produire plusieurs résultats très spécifiques, tels que des inventaires ou des données client.

L'orientation services relie ces sources d'informations hétérogènes et autonomes, créant ainsi un pont entre une gamme étendue de systèmes d'exploitation, de technologies et de protocoles de communication. Pour cela, l'orientation services utilise un processus itératif qui consiste à créer (**phase d'exposition**) de nouveaux services, à regrouper (**phase de composition**) ces services en applications composites plus globales et à rendre les résultats accessibles aux utilisateurs.

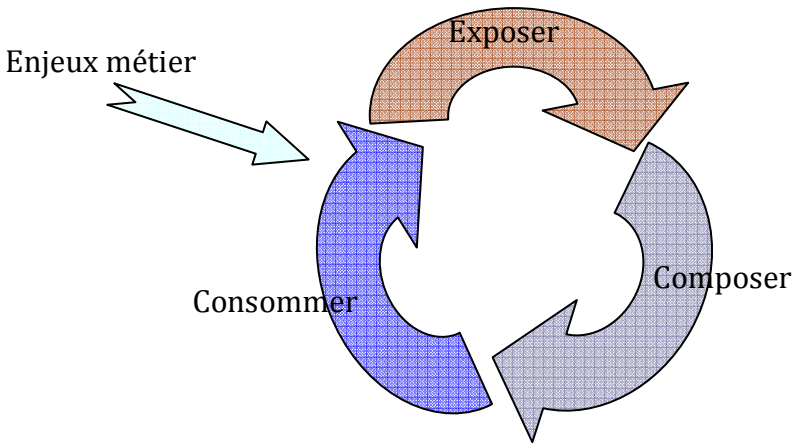


Figure 2. Le cycle de vie d'une SOA [MIC07].

1.7.1. Exposition

La phase d'exposition de l'approche SOA est consacrée aux services devant être créés à partir des applications et données sous-jacentes. La création de services peut être effectuée de manière précise (un seul service est associé à un seul processus métier) ou globale (plusieurs services effectuent un ensemble de fonctions connexes).

La phase d'exposition détermine également comment les services doivent être mis en œuvre.

1.7.2. Composition

Une fois créés, les services peuvent être combinés en services, applications ou processus métier plus complexes. Dans la mesure où les services existent indépendamment les uns des autres et de l'infrastructure informatique sous-jacente, ils peuvent être combinés et réutilisés avec une grande flexibilité. Et à mesure que les processus métier évoluent, l'entreprise peut ajuster ses règles et pratiques sans être contrainte par les limitations des applications sous-jacentes.

1.7.3. Consommation

Lorsqu'une application ou un processus métier a été créé, cette fonctionnalité doit être accessible (consommation) aux autres systèmes informatiques ou aux utilisateurs. Les utilisateurs peuvent consommer le service composé via différents accès, notamment via des portails Web, des clients riches, etc.

1.8. Exemple d'une SOA

SOA propose un modèle d'architecture informatique basé sur l'émergence d'une couche de services. Ces services offrent une vue « logique » des traitements et données existant

déjà ou à développer. Chaque service encapsule ces traitement et données et masque ainsi l'hétérogénéité du système d'information. Un exemple d'un SOA peut être comme le suivant [FGPR06] :

Un service « gestion des clients » offre par exemple une « vision client » unifiée. Pour cela il agrège les informations éparses dans le SI : il récupère la fiche d'identité du client et les noms des contrats. Etc. tout ce travail d'agrégation est masqué aux utilisateurs de ce service.

Le service « gestion du catalogue produit » met à disposition des informations sur les produits de l'entreprise. Ce service masque la récupération des tarifs, la récupération de certaines caractéristiques techniques et l'accès au catalogue Marketing (promotion en cours).

Le service « gestion des stocks » met à disposition les informations récupérées.

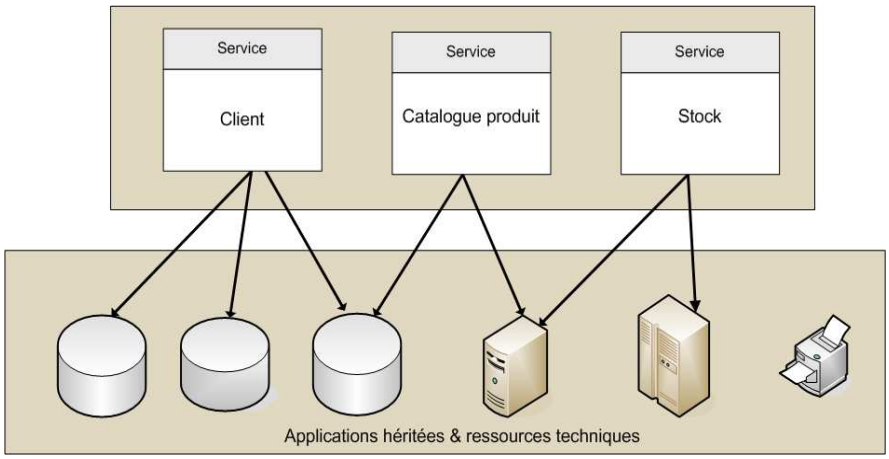


Figure 3. Encapsuler l'existant par des services.

1.8.1. Construire de nouveaux services

L'exemple précédent est incomplet. Pour le compléter, on peut par exemple introduire la gestion de prise de commande. Lors de prise d'une commande, il faut d'abord vérifier que le client existe et que son contrat est valide, puis vérifier que les produits existent dans le catalogue. Etc. avant de lancer sa livraison.

La solution au sens SOA est d'ajouter un service « gestion de commande », l'exemple suivant qui utilise les services déjà en place pour exécuter le traitement décrit ci-dessus :

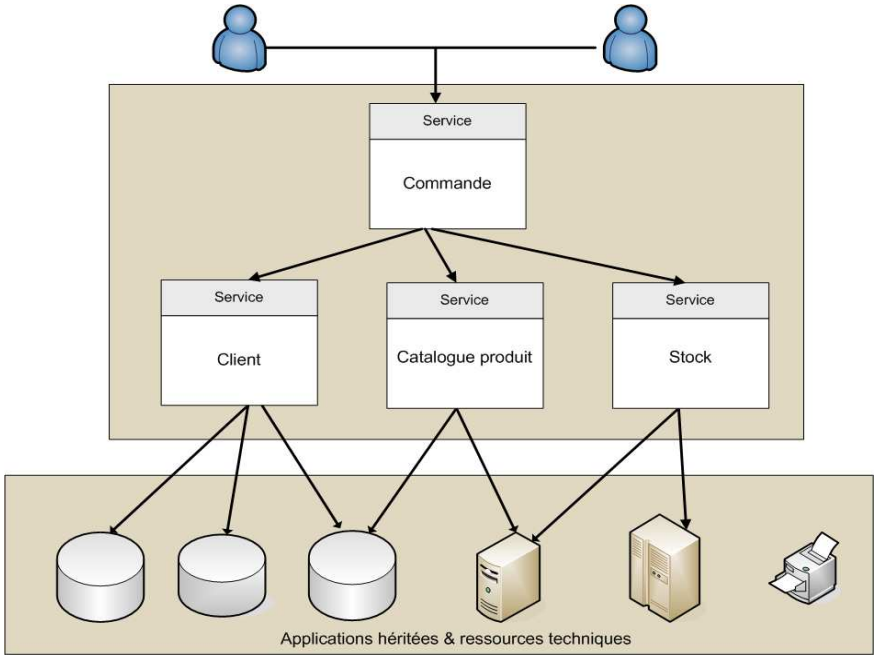


Figure 4. Création d'un nouveau service par composition.

Cet exemple mit en lumière un principe très important de l'approche SOA : l'approche SOA favorise la construction de nouveaux services par composition de service existants [FGPR06].

1.9. Le concept de service

Le mot-clef dans Architecture orienté service est « service ». Un service est une fonction encapsulée dans un composant que l'on peut interroger à l'aide d'une requête composée d'un ou plusieurs paramètres et fournissant une ou plusieurs résultats.

Un service correspond à une fonctionnalité dont le comportement est défini par un contrat d'utilisation et est fourni par une entité nommée fournisseur de service. Les fournisseurs de service sont découverts à l'exécution à l'aide d'un intermédiaire [occ06].

Un service est une unité d'activité métier qui apporte de la valeur au consommateur, qui communique par message et qui peut utiliser d'autres services. C'est-à-dire, est un composant logiciel de signification fonctionnelle distinctive qui encapsule typiquement un concept à niveau élevé d'affaires [LAK06].

1.9.1. Les propriétés d'un service

Couplage faible : un service est en couplage faible quand il n'est pas autorisé à appeler directement un autre service. Il délègue cette responsabilité à un traitement spécialisé que l'on nomme fonction d'orchestration. D'autre manière, on associe la propriété de couplage faible à l'indépendance d'activation du service vis-à-vis des technologies d'implémentation, c'est-à-dire un service est activable indépendamment de sa

technologie d'implémentation. Pour ce faire, l'activation se réalise par l'envoi (et la réception) d'un message XML.

- **Interopérabilité** : l'appel au service fonctionne quel que soit le langage et le système d'exploitation du consommateur.
- **Contrat d'utilisation** : accord réciproque entre une application fournissant d'un service et une application utilisant ce service, basé sur la description de l'offre faite par le fournisseur. Un service est décrit par un contrat d'utilisation qui précise ses conditions d'usage et les devoirs que le consommateur doit respecter. Un service expose un contrat d'utilisation décrit en deux parties. Une partie abstraite qui déclare les messages d'entrée et de réponse du traitement offert et une partie concrète qui décrit les standards et les protocoles techniques utilisés pour l'activation du service.
- **Réutilisabilité** : permettre la réutilisation d'une fonctionnalité par différents consommateurs d'un même service.
- **Abstraction** : toute référence à l'implémentation du service doit être retirée de l'interface.
- **Composabilité** : des collections de services peuvent être coordonnées et assemblées pour former des services composites.
- **Autonomie** : un service a sa propre implémentation, son propre déploiement. Il peut être modifié sans que cela affecte les autres services ou partenaires avec lesquels il est en relation. Un service fonctionne de manière autonome c'est-à-dire qu'il ne bloque pas le consommateur pendant que ce dernier s'exécute.
- Les services sont conçus de sorte qu'ils puissent être trouvés et évalués par l'intermédiaire des mécanismes disponibles de découverte.
- **Forte granularité** (*coarse-grained*) : la granularité des services se rapporte à l'éventail de fonctionnement exposé par un service. Le niveau de granularité dépend généralement de l'objectif de l'entité logicielle, quand il s'agit d'un service, ce dernier a tendance à être plus fort que le niveau de granularité des objets ou des composants parce que un service peut être implémenté en regroupant des objets, des composants et des services à granularité fine, et en les exposant en tant qu'unité unique via l'utilisation de façades ou interfaces.

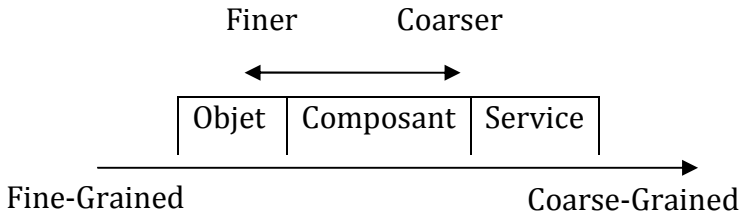


Figure 5. Le niveau de granularité [LAK06].

– **Un fonctionnement sans état** (*stateless*) : *Stateless* (sans état) et *Stateful* (avec état) sont les termes utilisés dans l'informatique pour indiquer qu'un composant possède une mémoire ou non. Un composant à état acquiert un contexte lié au composant solliciteur. Il ne peut pas être partagé par plusieurs composants en même temps. Un composant sans état libère les ressources dès que son exécution est terminée, n'est pas lié à un client spécifique, ce qui augmente la disponibilité.

1.9.2. Structure d'un service

Le service est la brique de base de l'architecture. Il se décompose en deux parties : la vue externe (ou spécification de service), qui expose la facette service, et la vue interne, qui décrit le contenu du service.

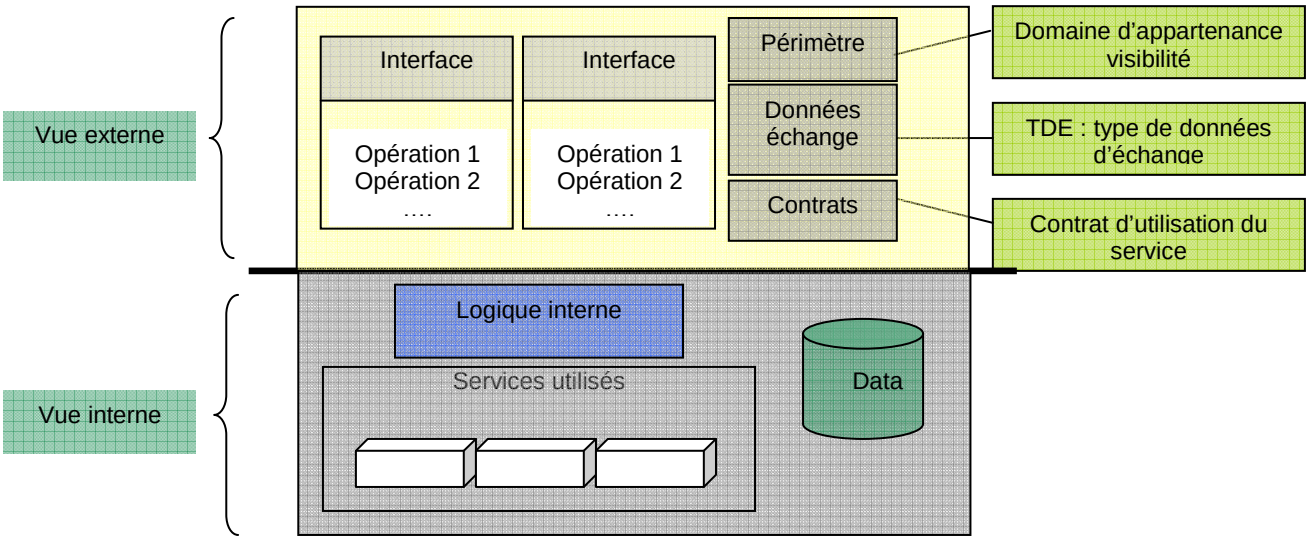


Figure 6. Structure du service [RAY07].

1.9.2.1. La vue externe

La vue externe est constituée par un ensemble d'opérations de service regroupées en interfaces, et de l'appareillage pour les utiliser (types des données échangées, contrat de service, propriétés, etc. Cette spécification est décrite en général par un fichier.

1.9.2.2. La vue interne

La vue interne contient des informations relatives à la logique interne comme le détail des traitements, les algorithmes, les référentiels ou les bases de données utilisées. On y trouve également les références vers les autres services utilisés par le service. Cette vue est masquée aux consommateurs du service.

1.9.3. Contrat de service

Le contrat de service joue un rôle majeur : il détaille les conditions d'utilisation du service sous forme de pré et post conditions, protocoles, et contraintes non fonctionnelles.

Pour demander un service, le consommateur émet une requête vers le pourvoyeur. Celui-ci renvoie (en générale une réponse).

A chaque couples (requête, réponse) correspond une opération effectuée par le service : l'opération est exécutée par le service sur la réception de la requête associée. Le contrat spécifie aussi la liste des opérations offertes par le service et pour chaque opération, il spécifie la requête et la réponse.

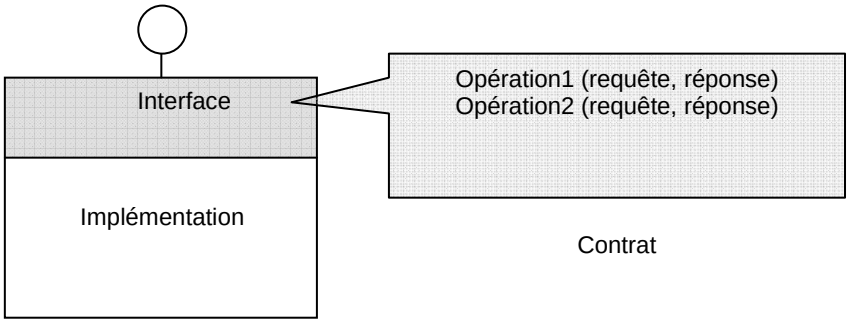


Figure 7. Notion de contrat [FGPR06].

En effet, la simple spécification fonctionnelle des opérations de services ne suffit pas à garantir le fonctionnement correct du système déployé. Les contraintes non fonctionnelles (Tableau 1), comme le temps de réponse ou le nombre d'invocations par seconde permettent de fixer les termes du contrat opérationnel entre consommateur et fournisseur de service.

Type de contraintes	Exemple
Disponibilité	Taux d'indisponibilité par une plage d'horaire.
Performance	Temps de réponse moyen.
Fiabilité	Taux d'erreur.
Sécurité	Politique de droits d'accès, non répudiation.
Administration	Journalisation, statistiques.

Tableau 1. Contraintes non fonctionnelles [RAY07].

L'interface ne détaille pas comment les opérations sont implémentées, et avec quelle technologie. Le contrat peut être formalisé dans n'importe quel langage respectant un standard, interne à l'entreprise ou partagé en respectant un standard basé sur XML. Sans

ce contrat, l'émetteur et le receveur de requêtes d'invocations devraient pré-supposer qu'ils utilisent, par exemple, le même langage d'information ou du moins des technologies compatibles, et qu'ils sont dans des conditions préétablies de bonne communication pour échanger.

1.9.4. Les messages

Lorsqu'un consommateur souhaite utiliser une opération d'un service, il lui transmet un message transportant sa requête. Le message est transporté sur le réseau via un protocole de communication formalisant et organisant les échanges sur le réseau. En retour, et lorsque cela sera nécessaire, le client recevra un autre message, qui sera également transporté via le même protocole ou un autre protocole de communication.

Cela traduit une volonté de faible couplage entre le pourvoyeur de service et le consommateur de service. Le pourvoyeur est libre d'améliorer ou de changer à volonté l'implémentation du service pourvu qu'il respecte son contrat d'engagements initiaux avec chaque consommateur.

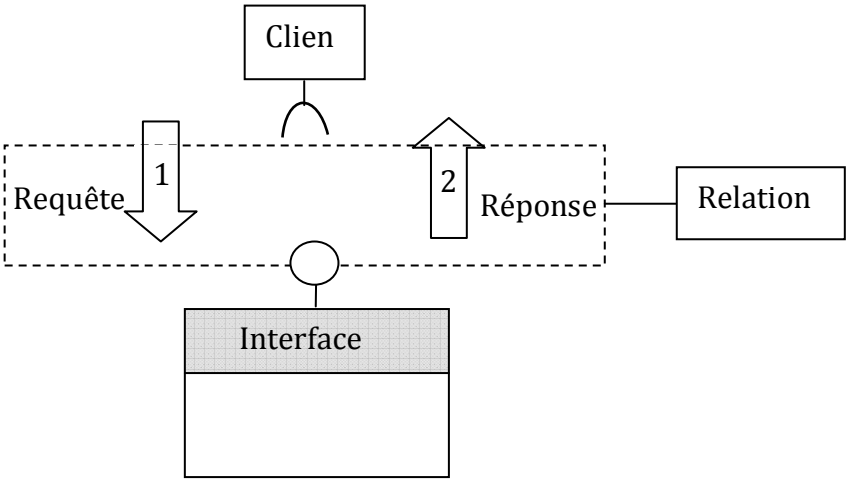


Figure 8. L'échange de messages entre consommateur et pourvoyeur [FGPR06].

Le faible couplage permet donc au **pourvoyeur** de maîtriser [FGPR06]:

- Une plus grande diversité de consommateurs ;
- La garantie de la qualité de son service ;
- Les modifications de l'implémentation de son service.
- Le faible couplage permet par ailleurs au **consommateur** :
- Une facilité de développement (pas besoin de connaître l'implémentation) ;
- Une fiabilité de transaction (le contrat est explicite, il n'y pas donc de surprise) ;
- Un changement possible de fournisseur (un autre fournisseur qui accepte de respecter le même contrat).

1.10. Le concept d'application composite

Pour pouvoir consommer des services, il est nécessaire d'introduire un nouveau type d'applications, qualifiées de « **composite** » dans une approche SOA, c'est-à-dire construite en composant des appels de service. Remarquons bien que l'approche orienté service ne peut apporter sa valeur ajoutée aux utilisateurs finaux que via ces applications composites [FGPR06]. Ces applications peuvent s'exposer à leur tour sous forme de service contractualisé pour leurs consommateurs (une composition est un service).

Une architecture de référence se dessine donc, peu à peu pour, toute architecture orientée service.

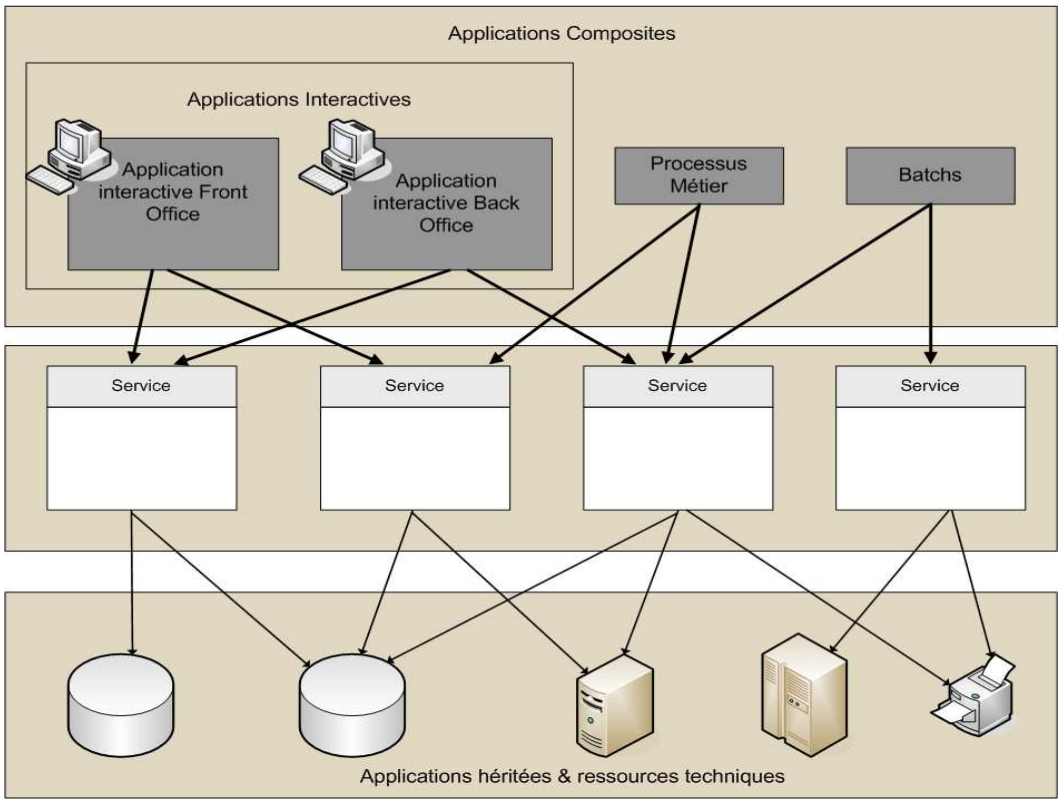


Figure 9. L'architecture de référence SOA [FGPR06].

Un service peut donc être consommé soit par :

- Un utilisateur, via une application composite interactive,
- Un système traditionnel (non SOA),
- Un processus métier,
- Un autre service SOA.

Lorsque le service est une partie prenante d'une composition, il doit être sollicité en cohérence avec les autres services composés. Ceci implique de partager un contexte

d'échange entre les services. Ce contexte permet le passage d'information entre plusieurs services, que ce soit des informations métier ou des informations technique.

Un service ou une composition de service doivent expliquer leur mode de prise en compte du contexte [FGPR06] :

- Service sans contexte : l'échange contient l'intégralité des informations requise par le pourvoyeur.
- Service avec contexte : le contexte est passé par valeur ou par variable durant l'échange entre le consommateur et le pourvoyeur.

L'approche SOA implique de gérer le concept de contexte, il est important de noter que le contexte est fourni au service par son consommateur : le service n'a pas à mémoriser ce contexte. Tout service doit être sans état.

1.11. Le concept d'orchestration de services

Un processus métier au sens SOA est vu comme un enchaînement plus ou moins automatisé de services effectué par un unique intermédiaire, le « chef d'orchestre ». D'où le nom d'**orchestration de service** pour ce type de composition. La logique de cet enchaînement (séquentiel, parallèle, etc.) est décrite, comme dans le Workflow classique, par un modèle d'orchestration directement exécuté par le chef d'orchestre, appelé moteur d'automatisation de processus (BPM[1]). L'orchestration décrit également la logique de contrôle et le flux de données interne d'un processus métier.

La différence entre un Workflow traditionnel et un processus métier SOA est qu'un processus SOA exclut l'homme dans la boucle. Il s'agit donc de processus de type e-business, recevant et traitant automatiquement les événements métier [FGPR06].

1.12. Conclusion

En conclusion, SOA n'est pas une technologie, elle est un ensemble de concepts constituant un modèle cohérent d'architecture pour faciliter la flexibilité du SI via l'émergence de services intégrant/réutilisant des applications existants et, par ailleurs SOA est une démarche progressive d'application.

SOA est utilisable qu'il s'agisse de technologies JEE, .Net, ou autres, via des implémentations propriétaires ou Open source, etc. Elle ne signifie pas Web service. Car à l'intérieur d'un SI, une SOA peut se réaliser sans Web services, sans même d'ailleurs utiliser le Web ou les technologies du Web.

[1] BMP : Business Process Management

L'objectif de ce chapitre est donc de clarifier les concepts utilisés dans une SOA, en premier lieu la notion de service et d'architecture de services puis celle de composition de services. Ainsi que les fondements du concept d'orientation service et notamment la notion de contrat et de messages.

Le prochain chapitre sera consacré pour un autre domaine très lié à l'architecture orienté services qui est les Web services, nous allons voire pourquoi les Web services sont utiles.

Chapitre 2. Les services Web.

2.1. Introduction

Le progrès du Web a permis la naissance d'une nouvelle génération de systèmes caractérisés par une meilleure intégration d'application hétérogènes et une meilleure communication entre ses différents composants. Cela a surgi le besoin de permettre qu'une application client invoque un service d'une application serveur en utilisant Internet. Ce besoin a été l'origine de ce qui se connaît comme services web.

Les services Web regroupent tout un ensemble de technologies bâties sur des standards (SOAP, WSDL, UDDI). Ils permettent de créer des composants logiciels distribués, de décrire leurs interface et de les utiliser indépendamment, ils permettent aussi de connecter des applications différentes, l'utilité de cette technologie devient évidente et importante. Pour cette raison les activités de recherche et développement autour du sujet des services Web ont un dynamisme très haut.

Les services Web peuvent être composés pour donner d'autres services appelés services composés ou services composites. Un service composite est un service dont l'exécution fait intervenir d'autres services. Les services intervenant dans cette exécution s'appellent des services composants. La composition de services Web vient pour apporter une réponse aux problèmes d'intégration et d'interopérabilité des systèmes d'information.

Dans ce chapitre nous allons, décrire en détaille les différentes technologies les plus liées aux services Web. Afin d'expliquer comment ces services Web peuvent faire communiquer des applications utilisant des langages différents, et qui sont implémentées sur des plates formes différentes.

2.2. Définition des services Web

Le consortium W3C[2] définit un service Web par :

« A Web service is a software application identified by a URI[3], whose interfaces and binding are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols ».

Un service Web est une application logicielle identifiée par un URI dont les interfaces et les liaisons sont définies, décrites et découvertes en XML et supporte une interaction directe avec les autres applications logicielles en utilisant des messages XML via un protocole Internet. Sa définition peut être découverte par d'autres Services Web, il peut

[2] W3C: World Wide Web Consortium. Organisme de normalisation des standards du Web. *http://www.w3.org*

[3] URI : Uniform Ressource Identifier.

interagir directement avec d'autres Services Web à travers le langage XML et en utilisant des protocoles Internet. L'objectif ultime de l'approche Services Web est de transformer le Web en un dispositif distribué de calcul où les programmes (services) peuvent interagir de manière intelligent en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes [KT03].

IBM donne dans un tutoriel la définition suivante des web services [WEB01]:

« Les web services sont la nouvelle vague des applications web. Ce sont des applications modulaires, auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le web. Les web services effectuent des actions allant de simples requêtes à des processus métiers complexes. Une fois qu'un web service est déployé, d'autres applications (y compris des web services) peuvent le découvrir et l'invoquer ».

2.2.1. Les bénéfices des services Web

Pour l'entreprise, disposer des services Web c'est avant tout ouvrir son système d'information à autres usages, d'autres besoins et d'autres clients extérieurs à l'entreprise. La mise en œuvre des services Web dans l'entreprise donne les bénéfices suivants [KM03] :

- Les services Web permettent d'automatiser facilement les processus métiers : les services Web permettent d'intégrer, gérer et automatiser rapidement les processus métier intra et interentreprises en échangeant des informations au format XML. L'assemblage des différents services peut être orchestré par un moteur de Workflows qui contrôle à la volée l'exécution de l'application en fonction du contexte d'exécution et les règles de déroulement du processus métier.
- Les services Web facilitent l'interopérabilité entre systèmes et plates formes hétérogènes : la mise en œuvre des services Web facilite le dialogue entre environnement hétérogène. Comme les services Web peuvent être implémentés sur différents plates formes et avec des langages variés, ils deviennent un moyen technique intéressant pour interconnecter des modules s'exécutant sur des plates formes hétérogènes. Grâce au support étendu des standards publics et des spécifications telles que le XML[4] et HTTP[5], les services Web assurent le plus haut niveau d'interfonctionnement entre les applications de l'entreprise et celles de partenaires.

[4] XML : eXtensible Markup Language.
[5] HTTP : HyperText Transfer Protocol.

2.2.2. Propriétés d'un service Web

Le consortium W3C définit un service Web comme étant une application ou un composant logiciel vérifiant les propriétés suivantes :

- il est identifié par une URI (Uniform Ressource Identifier) ;
- ses interfaces et ses liens peuvent être décrits en XML ;
- sa définition peut être découverte par d'autres services Web ;
- il peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant des protocoles Internet.

2.3. Architecture des services Web

L'architecture des services Web est une instance d'architecture orientée service (*Service Oriented Architecture*) [MEL04]. Elle propose une perspective globale sur le développement, la gestion et le fonctionnement des services Web. La définition de l'architecture services Web consiste à mettre en évidence les services, les relations entre ces services ainsi qu'un ensemble de contraintes qui assurent l'objectif premier des services Web à savoir l'interopérabilité. Elle est implémentée à l'aide des diverses technologies organisées en quatre couches comme illustre la figure suivante [KM03]. Chaque couche de la pile des services Web répond à des préoccupations fonctionnelles différentes telle que la sécurité, la messagerie, les transactions, le routage, le Workflow etc. comme représenter dans la figure les différentes couches de la pile des services Web s'interfacent avec des standards. Les fonctions de couche supérieures reposent sur celles de couches inférieures. La partie gauche représente les standards appliqués à chaque couche.

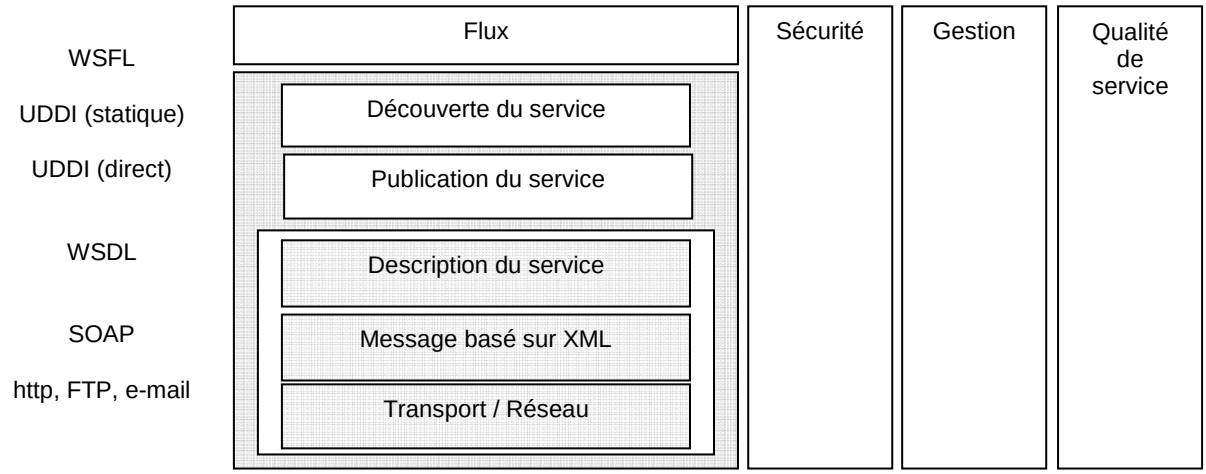


Figure 10. Cadre architecturel des services Web.

Au niveau du transport et pour assurer la connectivité, plusieurs protocoles peuvent supporter les services Web : HTTP, SMTP[6], FTP.

Le niveau suivant est le niveau message basé sur le standard SOAP qui assure le transport des massages encapsulés au format XML.

Le niveau description de service est dédié à la description de document en WSDL. WSDL définit l'interface et les mécanismes d'interaction de service. Il est nécessaire de spécifier le contexte métier, la qualité de service et les relations de service à service.

2.3.1. L'architecture de référence

Une architecture de référence est nécessaire afin de préserver les objectifs initiaux visés par les services Web lors des évolutions technologiques successives. Le W3C a déjà mis en place une architecture de référence des Services Web qui s'articule autour des trois rôles suivants [W3C02]:

- **Le fournisseur de services**: correspond au propriétaire du service. D'un point de vue technique, il est constitué par la plate-forme d'accueil du service.
- **Le client**: correspond au demandeur de service. D'un point de vue technique, il est constitué par l'application qui va rechercher et invoquer un service. L'application cliente peut être elle-même un Web service.
- **L'annuaire des services**: correspond à un registre de description de services offrant des facilités de publication de services à l'intention des fournisseurs ainsi que des facilités de recherche de services à l'intention des clients.

Les interactions de base entre ces trois rôles incluent les opérations de publication, de recherche et de liens d'opération.

2.3.2. Le cycle de vie d'un service Web

Le cycle de vie d'un service Web se déroule de la façon suivante : une fois créé, le service est déployé sur le réseau (local ou Internet). Puis un utilisateur ayant des besoins spécifiques va rechercher un service correspondant à ses besoins à l'aide d'un annuaire spécialisé. Enfin, une fois le service trouvé, l'utilisateur va invoquer le service : une communication va être mise en place entre l'utilisateur et le service Web.

[6] SMTP : Simple Mail Transfer Protocol (protocole simple de transfert de courrier).

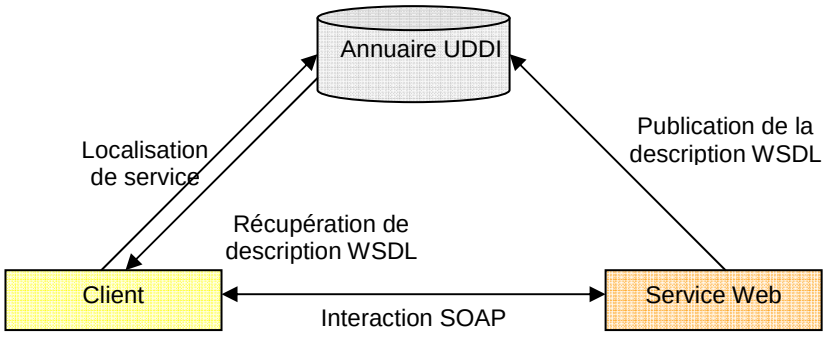


Figure 11. Déploiement, découverte et invocation de services Web [MEL04].

Ce cycle de vie, représenté par la figure précédente, fait appel aux trois protocoles: SOAP[7], WSDL[8] et UDDI[9] qui seront expliqués en détaille dans la suite de ce chapitre.

2.4. Apports XML

XML « eXtensible Markup Language » est un format texte simple, très flexible tiré du SGML[10] (l'ISO 8879). À l'origine, il est conçu pour la publication électronique à grande échelle, XML joue aussi un rôle de plus en plus important dans l'échange d'une large variété de données sur le Web et ailleurs.

W3C recommande depuis 1998 XML en tant que standard de description de données. XML est un méta langage permettant d'identifier la structure d'un document. Un document est composé d'une définition de sa structure et d'un contenu. La structure d'un document XML est souvent représentée graphiquement comme un arbre. La racine du document constitue le sujet du document, et les feuilles sont les éléments de ce sujet. De ce fait, XML est alors flexible et extensible, et est devenu rapidement le standard d'échange de données sur le web [W3C].

XML reprend les qualités de SGML (et utilise sa syntaxe):

- un balisage logique hiérarchique.
- une description facultative du document ; elle peut être extérieure ou intégrée au document.
- une portabilité quasi universelle.

Il permet de créer des pages Internet sophistiquées, de structurer un document, de décrire des données. Il est aussi utilisé pour les échanges entre machines et/ou programmes, mêmes étrangers entre eux.

[7] SOAP : Simple Object Access Protocol
[8] WSDL : Web Service Description Langage
[9] UDDI : Universal Description, Discovery and Integration
[10] SGML: Standard Generalized Markup Langage (language normalisé de balisage généralisé)

2.4.1. XML et les services Web

XML est le choix des protocoles du Web comme support de communication entre les composants il est aujourd'hui un standard qui permet de décrire des documents structurés transportables sur les protocoles communs d'Internet.

Pour les services Web, XML est la langue maternelle de ces derniers [CHA02], il est systématiquement utilisé avec les *namespaces* et la spécification *XML Shema*, tous les deux sont indispensables pour exprimer les structures de données habituellement complexes figurants dans les messages échangés. Dans le mécanisme de transport entre services Web, les requêtes, leurs résultats et les erreurs éventuelles résultants de leurs invocations, sont tous écrits en XML.

Donc XML constitue la technologie de base des architectures services web, il est un facteur important pour contourner les barrières techniques. En effet, il apporte à l'architecture l'extensibilité et la neutralité vis à vis des plates-formes et des langages de développement. De plus, grâce à la structuration, XML permet la distinction entre les données des applications et les données des protocoles permettant ainsi une correspondance facile entre les différents protocoles.

L'interopérabilité entre les systèmes hétérogènes demande des mécanismes puissants de correspondance et de gestion des types de données des messages entre les fournisseurs et les clients. C'est une tâche où les schémas de type de données XML s'avèrent bien adaptés [MEL04].

XML facilite l'utilisation des services Web :

- La séparation du contenu d'un document, de sa structure et de sa représentation facilite l'échange d'informations entre les différents partenaires.
- Permet la composition de services plus complexes à travers la définition des enchaînements entre les services grâce à BPEL4WS[11].
- Favorise l'émergence de standards d'industrie dans la mesure où les structures de données sont réutilisées et réutilisables.

2.5. Protocole SOAP

Le protocole SOAP (Simple Object Access Protocol) est un standard du consortium W3C définissant un protocole qui assure des appels de procédures à distance (RPC) s'appuyant principalement sur le protocole HTTP et sur XML. SOAP permet aux services Web d'échanger des informations dans un environnement décentralisé et distribué tout

[11] BPEL4WS: Business Process Execution Language for Web Services.

en s'affranchissant des plates-formes et des langages de programmation utilisés. Il définit un ensemble de règles pour structurer les messages envoyés.

Le principal objectif du protocole SOAP est de permettre la normalisation des échanges de données. Il définit un moyen uniforme d'échange de donnée encodées XML sous forme d'appel de procédure à distance RPC (*Remot Procedure Call*) en utilisant HTTP comme protocole de communication, mais aussi les protocoles SMTP et POP[12].

SOAP assure l'interopérabilité entre composants tout en restant indépendant des plates formes et des langages de programmation (Java/RMI, CRBA, COM/DCOM, etc.). Les paquets de données qui circulent sous forme de texte structurés au format XML. Tout type de données peut ainsi être transporté même si les données binaires devraient faire l'objet d'un encodage spécifique. Ce sont les choix technologiques basés sur HTTP, SOAP et WSDL qui permettent de garantir cette interopérabilité.

Le protocole SOAP permet de traverser les firewalls car il ne passe pas par des ports précis mais il utilise le protocole HTTP qui lui permet de traverser les proxys et les firewalls, contrairement à d'autres modes de communication telles que les communications via les sockets, Java, RMI[13] et etc.

2.5.1. Place de SOAP dans la pile de communication entre les services Web

Dans un système d'information distribuée, les applications s'interagissent les unes avec les autres souvent à l'aide du RPC (*Remote Procedure Call*). SOAP comporte trois parties principales : un Framework de messagerie, un standard d'encodage et le mécanisme RPC [KM03].

SOAP, une forme de RPC utilisé dans les services Web, est un langage basé sur XML qui permet de décrire et de déclencher des procédures distances au-dessus du protocole HTTP. Pour cela, le client transmet un message XML au serveur qui contient le nom de la procédure à exécuter et la valeur des paramètres de cette procédure.

[12] POP : Post Office Protocol.
[13] RMI : Remote Methode Invocation.

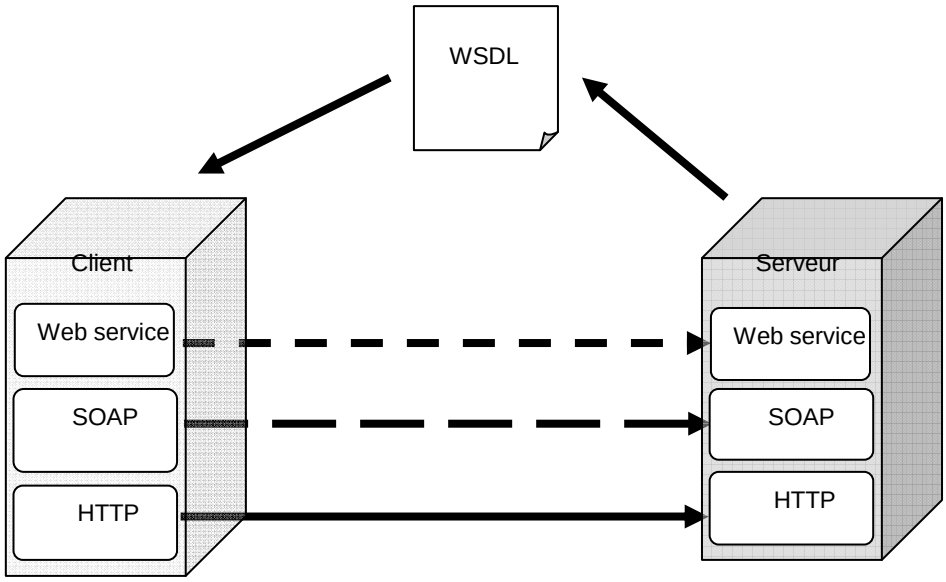


Figure 12. Place de SOAP dans la pile de communication entre service Web [KM03].

La figure 12 positionne le protocole SOAP parmi les autres composants de communication de l'environnement d'exécution des services Web. En réponse à l'appel SOAP effectué par le client, le serveur exécute le traitement et retourne la valeur du résultat sous forme d'un document XML indiquant si l'exécution a réussi et en précisant la (ou les) valeur (s) de retour. Dans ce cas, XML, n'est pas utilisé pour transporter des documents mais pour véhiculer des appels de procédure et leur résultat.

2.5.2. Le Framework de messagerie SOAP

Le Framework de messagerie SOAP requiert que les massages SOAP soient composés d'une enveloppe qui contient un *header* et un *body* l'exemple suivant représente le squelette d'un message SOAP contenant un SOAP header block et un SOAP body [GHM07] :

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

L'enveloppe est l'élément englobant et structurant du message, les deux autres balises de l'exemple concernant le SOAP *header* et le SOAP *body*. La partie *header* du message

véhicule les métadonnées du message tandis que le *body* contient le corps du message lui-même.

SOAP fait une utilisation intensive de *namespaces* XML. Par l'exemple, le *namespace ENV* dans l'exemple ci-dessus caractérise les éléments relatifs à SOAP : *envelope, header* et *body.* Les éléments *header* dans un document SOAP peuvent être signalés *mustUnderstand*, signifiant que le serveur qui traite le message SOAP doit, d'une manière exclusive, soit comprendre ce type d'élément *header* soit rejeter le message. Les messages sont rejetés en générant les SOAP *faults*. Ces derniers (*faults*) apparaissent lors qu'une erreur survient lors du traitement d'un message. Les erreurs peuvent avoir plusieurs causes, mais le plus souvent, elles concernent le champ *header* non reconnu, un message qui ne peut être authentifié ou les erreurs qui apparaissent lorsqu'on invoque une méthode de traitement du message. Les messages SOAP sont échangés entre l'utilisateur de service et le fournisseur du service dan des enveloppes SOAP contenant une requête pour réaliser une action et le résultat de cette action. Les enveloppe SOAP sont formatés XML et assez facile à décoder.

2.5.2.1. Déclaration d'un namespace

Pour déclarer un namespace, on associe un préfixe avec URI. Cette association est définie comme un attribut avec un élément *xmlns : prefixe*. Dans cet élément, *xmlns* est le mot clé namespace tandis que le préfixe est défini par l'utilisateur [GHM07].

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Body>
  <m:GetTradePrice>
    env: encodingStyle = http://www.w3.org/2001/106/soap-envelope
     xmlns:m=http://example.org/2001/06/quotes>
   <symbol>DIS</symbol>
  </m: GetTradePrice >
 </env:Body>
</env:Envelope>
```

2.5.2.2. Structure du message SOAP

Il existe deux formats de message SOAP correspondants respectivement aux messages sans pièces jointes et ceux qui peuvent en disposer. La figure suivante représente le format d'un message sans pièces jointes [ML07].

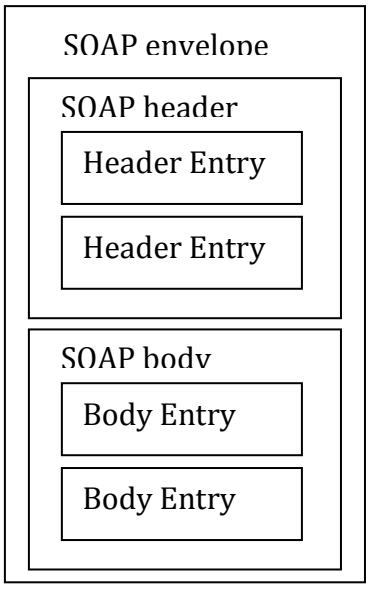


Figure 13. Le format d'un message SOAP sans pièces jointes.

Un message SOAP sans pièces jointes comporte trois parties :

- L'enveloppe (obligatoire) : contient le nom du message suivi d'un domaine de nom (namespace) qui indique la version du schéma XML-SOAP supporté.
- L'entête (optionnel) : est utile quand le message doit être traité par plusieurs intermédiaires.
- Le corps (obligatoire) : renferme les méthodes et les paramètres qui seront exécutés par le destinataire final.

Dans le cas d'un message avec pièces jointes. Le message peut comporter une ou plusieurs parties d'attachements ajoutées au message SOAP de base. La partie SOAP ne peut contenir seulement que le contenu XML, comme résultat, si un contenu du message n'est pas au format XML, il doit apparaître dans la partie attachement. Ainsi, par exemple, si votre message contient un fichier image, vous devez avoir une partie attachement pour les véhiculer dans un message SOAP. Il faut noter que toute partie attachement peut contenir tout type de contenu, il peut donc aussi des données au format XML. La figure 14 présente le format d'un message SOAP avec pièces jointes :

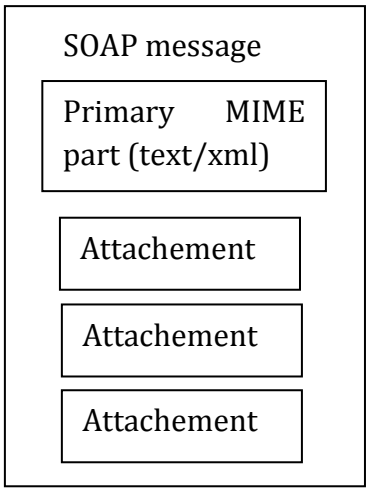


Figure 14. Le format d'un message SOAP avec pièces jointes.

2.5.3. Encodage/sérialisation pour les objets

La spécification SOAP d'encodage/sérialisation indique comment les objets sont encodés ou sérialisés quand ils envoyés à l'aide du protocole SOAP. Disposer d'un standard d'encodage pour les messages SOAP signifie que les objets peuvent être encodés dans des messages SOAP d'une manière standard et décodés aussi par le destinataire d'une manière standard.

Par exemple, un tableau d'entiers peut être encodé en SOAP comme suit :

```
<soap-enc : array soap-enc : arrayType='xsd :int[3]'>
      <soap-enc :int>8</soap-enc :int>
       <soap-enc :int>9</soap-enc :int>
      <soap-enc :int>5</soap-enc :int>
</soap-enc : array>
```

Le *run time* SOAP implanté côté client et côté serveur effectue l'encodage/décodage. Les mécanismes SOAP d'encodages/sérialisation sont essentiellement utilisés avec le RPC.

2.5.4. Invocation de services d'objets distants via SOAP RPC

La spécification SOAP RPC décrit la possibilité d'utiliser le protocole SOAP pour invoquer les procédures sur réception de la fin du message SOAP. Elle permet d'invoquer un objet distant en communiquant les informations nécessaires à l'appel : nom de la méthode et sa signature dans un message au format XML et via HTTP. La réponse à la requête est aussi renvoyée encapsulée au format XML. Après la mise en forme de la requête SOAP, elle est envoyée au serveur Web. Le parseur XML du serveur vérifie la cohérence du message avant de le transmettre sur HTTP. De même, le parseur XML, du serveur destinataire vérifie la cohérence du message reçu et le rejette s'il n'est pas valide.

Pour invoquer une procédure distance il suffit de connaître le nom de la procédure et les arguments de cette méthode. Dès que vous avez invoqué une méthode en utilisant SOAP, la librairie SOAP utilisé encode ou sérialise les arguments en se basant sur le standard d'encodage SOAP.

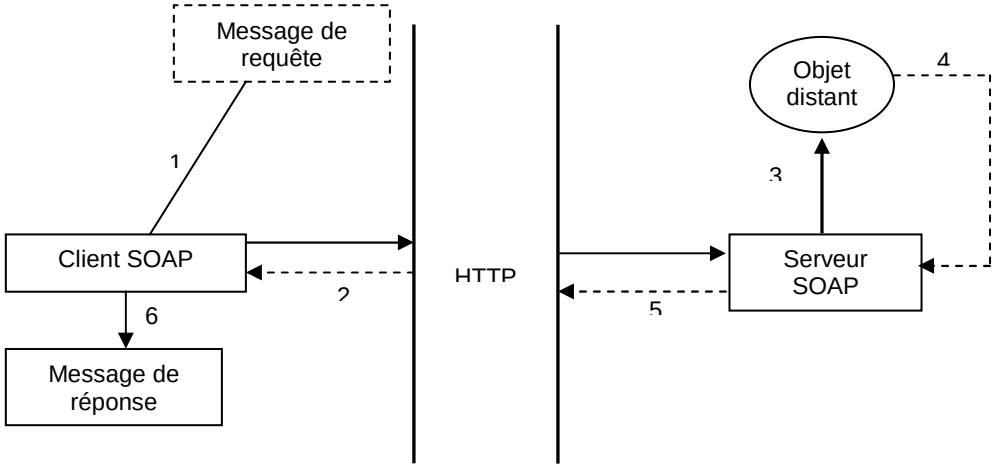


Figure 15 . Etapes d'invocation d'objets distants avec SOAP.

Comme illustre cette figure, les différentes étapes d'invocation d'objets distants avec SOAP sont les suivantes [KM03]:

- Le programme client SOAP crée un document XML contenant les informations nécessaires pour invoquer les services d'un objet distant. Ce document est incorporé dans une enveloppe SOAP avant d'être transmis sous forme d'une requête POST HTTP.
- Le message est transmis via une connexion http standard vers le serveur.
- Le serveur SOAP reçoit le message, analyse sa cohérence et l'aiguille vers l'objet concerné.
- L'objet effectue le traitement demandé et renvoie les résultats au serveur SOA.
- Le résultat est renvoyé au client à travers un document SOAP encapsulé dans un en-tête de réponse http.
- Le SOAP client reçoit les résultats, ouvre l'enveloppe SOAP et envoie le résultat au demandeur initial.

Pour faire une invocation d'une méthode distante, les informations suivantes sont nécessaires :

- URI de l'objet cible ;
- Nom de la méthode ;
- Signature de la méthode (facultative) ;
- Paramètres de la méthode ;

- Données de l'en-tête (facultatif).

Les appels et les réponses de méthodes RPC sont transmis encapsulés dans l'élément *body* de SOAP.

2.6. Description des services Web avec WSDL

Le *Web Service Description Language* (WSDL) est un format XML qui décrit les services Web offerts par un composant applicatif. Dans la pratique, on utilise WSDL pour créer un fichier qui identifie les services fournis par un composant applicatif et l'ensemble d'opération constituant ces services. Les services sont représentés en WSDL comme un maillage de terminaisons agissant sur des massages contenant des informations sur des documents ou des procédures indépendamment des protocoles de transport des messages utilisés. WSDL repose sur SOAP/HTTP/MIME[14] comme mécanisme d'invocations distantes et sur la spécification de schémas XML comme système de type.

2.6.1. Structure d'un document WSDL

Un document WSDL est composé de deux parties, une partie abstraite (ou interface) et une partie concrète (ou implémentation). Les opérations et les messages sont décrits d'une manière abstraite et liés à un protocole réseau concret et un format de message pour définir une destination [THO04].

- Partie abstraite : décrit les messages que le service envoie et reçoit et l'opération associe au modèle d'échange de ces messages.
- Partie concrète : indique :
- Le transport et le format pour un ou plusieurs interfaces.
- associés un port (un point final) et une adresse de réseau avec lequel est lié.
- Le service qui groupe un ensemble de points finaux qui implémentent une interface commune.

[14] MIME : Mutipurpose Internet Mail Extention.

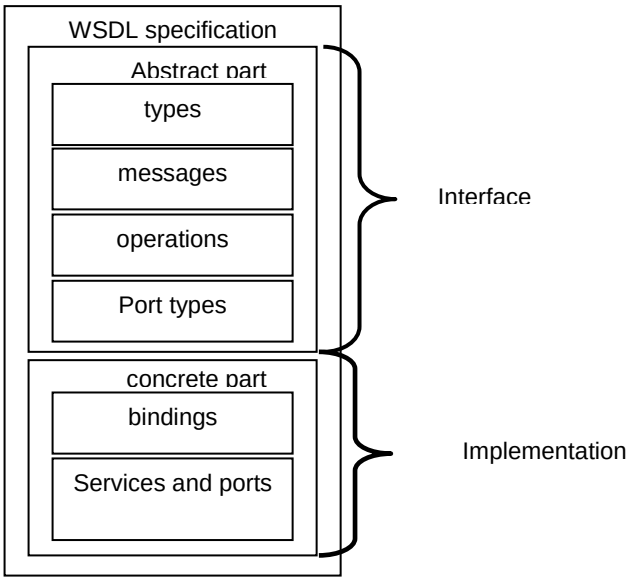


Figure 16. Structure d'un document WSDL [THO04].

L'interface est une définition abstraite ou réutilisable de service qui peut être instanciée est référencée par différentes définitions d'implémentation (IDL, Java) de service.

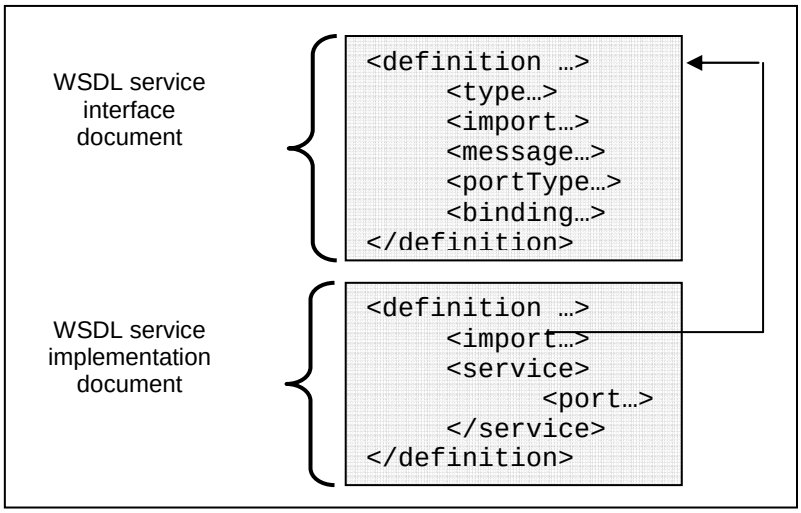


Figure 17. Relation entre la partie interface et implémentation [KM03].

L'interface contient des éléments WSDL qui constituent la partie réutilisable de la description du service :

- type : pour décrire les types de données complexes ;
- *message* : pour décrire les types de données XML et les paramètres dans le message ;
- *portType* : pour définir les opérations (signatures) du service Web ;
- *binding* : pour décrire le protocole, le format de données, les attributs de sécurité et d'autres attributs ;

Figure 18. Le schéma XML d'un document WSDL.

La définition de l'implémentation de service est un document WSDL qui contient les éléments *import* et *service*. Ce document implémentation de service contient une

description de service qui implémente une interface de service. Au moins un des éléments *import* va contenir une référence au document interface de service WSDL. Les opérations dans le fichier WSDL peuvent être orientées document ou orientées RPC comme défini par l'attribut style de l'élément <*soap :binding*> dans le fichier.

Un document WSDL est apparaît tout simplement comme une série de définitions. Il y a un élément définition à la racine qui peut encapsuler d'autres. La grammaire d'un document WSDL se présente sous la forme du schéma XML (figure 18) [THO04].

2.6.2. Les éléments de WSDL

Dans un document WSDL, l'élément *documentation* permet d'insérer des commentaires destinés à documenter ce document. Cet élément peut être utilisé à l'intérieur de n'importe quel élément WSDL. Les principaux éléments dans un document WSDL sont [CMR07]:

2.6.2.1. Les types (type)

L'élément type contient les définitions des types de données appliqués aux messages échangés. Afin de garantir une interopérabilité maximale ainsi qu'une grande indépendance au niveau des plates formes, WSDL utilise XSD en tant que système de type.

```
<definition …/>
   <type>
        <xsd:schema …. >
  </type>
<definition …/>
```

2.6.2.2. Les messages (message)

Ils sont décomposés en une ou plusieurs parties. Chaque partie est associée à un type à l'aide de l'attribut *type message*. L'attribut *name* d'un message fournit un nom unique à chacun des messages définis à l'intérieur du document WSDL. Afin de décrire le contenu d'un message, l'attribut *name* de l'élément *part* fournit un nom unique à chacune des parties d'un message.

```
<message name="OrderMsg">
  <part name="productName" type="xs:string"/>
  <part name="quantity" type="xs:integer"/>
</message>
```

2.6.2.3. Les types de port (portType)

L'élément *portType* combine des éléments de messages pour former une opération à sens unique ou aller-retour complète. Par exemple, un *portType* peut combiner une demande et un message de réponse dans une opération simple de requête/réponse. Un

portType peut définir plusieurs opérations, chaque opération est identifiée dans le document WSDL par l'élément *operation* dans le quel, on déclare le nom de la méthode et les données passées en paramètres.

```
<portType name="procurementPortType">
    <operation name="orderGoods">
       <input message = "OrderMsg"/>
   </operation>
</portType>
```

2.6.2.4. Les liaisons (binding)

L'élément *binding* définit les protocoles de communication utilisés lors des appels du service Web. Cette définition permet de faire la liaison entre le document WSDL et le protocole de communication SOAP et entre les messages et les méthodes. Dans l'exemple suivant, l'attribut *name* définit un nom unique pour chaque liaison dans le document WSDL. L'attribut *type* permet à la liaison de faire référence au *portType* qu'elle relie. Les éléments *input, output, fault* et *operation* sont en relation avec les éléments du *portType* correspondant.

```
<binding name="ProcurementSoapBinding" type="tns:procurementPortType">
<soap:binding style="document"
              transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="orderGoods">
 <soap:operation soapAction="http://example.com/orderGoods"/>
  <input>
    <soap:body use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
  </output>
</operation>
</binding>
```

2.6.2.5. Les ports (port)

Les éléments *port* sont les points de communication qui spécifient l'adresse d'une liaison. Ils sont constitués d'un *binding* et d'une URI. Un *port* ne doit pas indiquer plus d'une adresse.

2.6.2.6. Les services (service)

Les *services* sont des collections des *ports*. L'attribut *name* fournit un nom unique parmi tous les services définis dans le document WSDL. Aucun *port* ne peut communiquer avec un autre *port*, par exemple la sortie d'un *port* ne peut correspondre avec l'entrée d'un autre.

```
<service name="ProcurementService">
  <port name="ProcurementPort" binding="tns:ProcurementSoapBinding">
    <soap:address location="http://example.com/procurement"/>
  </port>
</service>
```

En plus de ces éléments principaux, le fichier WSDL définit également l'élément *Import* qui est employé pour importer d'autres fichiers WSDL. Par exemple, deux fichiers WSDL peuvent importer les mêmes éléments de base mais inclure leurs propres éléments de service pour rendre le même service disponible à deux adresses physiques [BOU06].

2.7. La spécification UDDI

Dans un environnement ouvert comme Internet, la description d'un service Web n'est d'aucune utilité s'il n'existe pas de moyen de localiser aussi bien les services Web que leurs descriptions WSDL : c'est le rôle des référentiels UDDI (*Universal Description, Discovery and Integration*). La spécification UDDI constitue une norme pour les annuaires de services Web. Les fournisseurs disposent d'un schéma de description permettant de publier des données concernant leurs activités, la liste des services qu'ils offrent et les détails techniques de chaque service. La spécification offre également une API (*Application Programming Interface*) aux applications clientes, pour consulter et extraire des données concernant un service et/ou son fournisseur.

UDDI est un standard, né de l'initiative d'un certain nombre d'entreprises, dont Microsoft, IBM, Sun, Oracle, Compaq, Hewlett Packard, Intel, SAP et bien d'autres qui sont réunies pour développer une spécification basé sur des technologies standard afin de faciliter la collaboration entre partenaires dans le cadre d'échanges commerciaux.

L'annuaire UDDI permet de publier et de découvrir des informations sur une entreprise et ses services Web. Il contient des informations sur les entreprises et les services qu'ils publient. L'inscription d'une société à l'annuaire UDDI lui permet de se présenter et présenter ses services. L'adoption de cet annuaire par les entreprises permettra d'accélérer les échanges commerciaux de type B2B.

2.7.1. Mécanismes d'accès aux services fournis par un nœud UDDI

Grâce à un jeu d'API XML basé sur SOAP, on peut interagir avec UDDI au moment de la conception et de l'exécution des services Web pour découvrir des données techniques et administratives sur les services et les entreprises qui les publient, de manière à ce que ces services puissent être invoqués et utilisés. UDDI repose sur le protocole de transport SOAP et assure que les requêtes et les réponses sont des objets UDDI envoyés sous forme des messages SOAP.

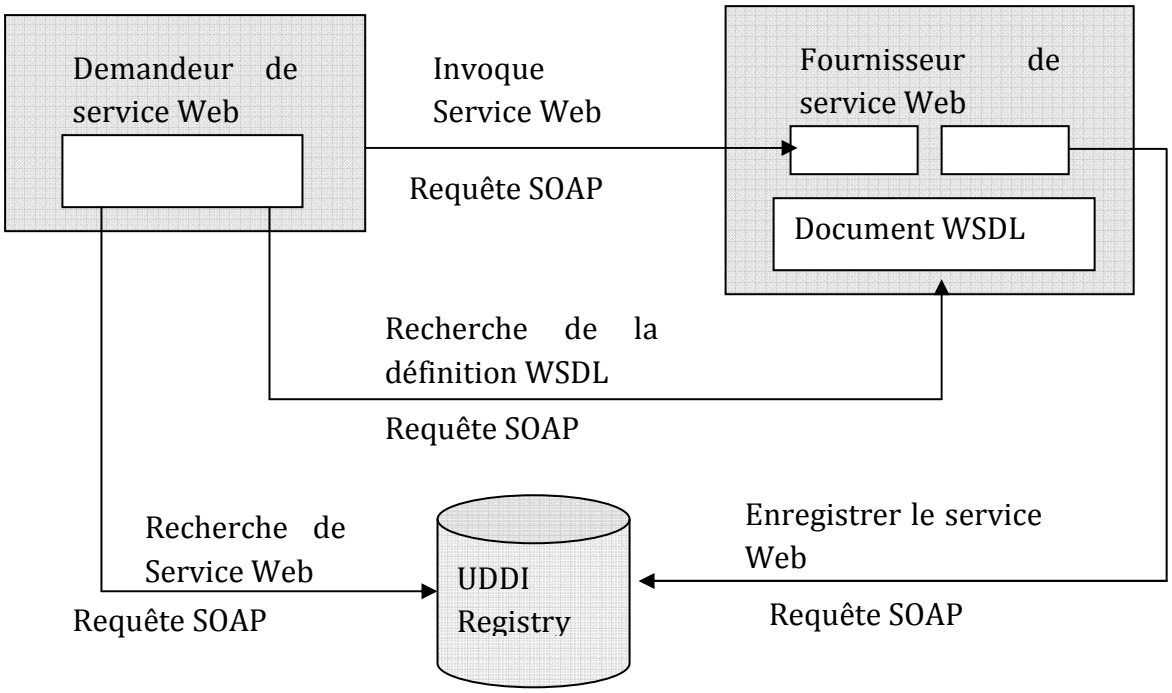


Figure 19. Mécanismes d'accès aux services Web fournis par un nœud UDDI Registry [KM03].

L'annuaire UDDI est consultable sous plusieurs facettes :

- Les pages blanches recensent les entreprises ; elles comportent les informations sur les entreprises et contiennent les informations telle que le nom de l'entreprise, ses coordonnées et des descriptions de l'entreprise.
- Les pages jaunes comprennent la description, au format WSDL, des services Web déployés par les entreprises. Elles répertorient les services Web par catégorie. Les pages jaunes sont composées d'informations permettant de classifier l'entreprise. Ces informations s'appuient sur des standards de classification normalisés. Une entreprise peut disposer de plusieurs entrées dans l'annuaire des divers services et produits qu'elle propose.
- Les pages vertes fournissent des informations techniques détaillées sur les services fournis. Ces pages contiennent des informations sur les processus métier, des descriptions de services et des informations de liaison sur les services.

UDDI fournit aux clients des mécanismes de localisation dynamique d'autres services Web. A travers l'interface UDDI une entreprise peut dynamiquement connecter ses processus métier à ceux externes fournis par d'autres entreprises.

2.7.2. Modèle d'information UDDI

Pour que les services Web soient significatifs ils ont besoin de fournir des informations au delà des caractéristiques techniques du service elle-même.

Pour le commerce électronique ou pour des buts alternatifs, les entreprises et les fournisseurs peuvent employer UDDI pour représenter des informations sur des services Web d'une manière standard. Quelques questions peuvent alors être posées à un enregistrement d'UDDI [CHR04] :

- Trouver les réalisations de services Web qui sont basées sur une définition d'interface abstraite commune.
- Trouver les fournisseurs de services Web qui sont classifiés selon un arrangement de classification ou un système connu de marque (normalisé).
- Déterminer la sécurité et transporter les protocoles soutenus par un service Web donné.
- Publier une recherche des services basés sur un mot-clé général.

Cacher les informations techniques sur un service Web et puis mettre à jour cette information au temps d'exécution.

Ces scénarios et d'autres sont permis par la combinaison du modèle de l'information d'UDDI et de l'ensemble d'API d'UDDI. Puisque le modèle de l'information est extrêmement normalisé, il peut adapter des différents types des modèles, des scénarios et de technologies. Les spécifications ont été écrites pour être flexibles de sorte qu'elles puissent absorber un ensemble divers de services et ne pas être attachées à n'importe quelle technologie particulière. Quand un nœud UDDI expose son information pour un service Web sous forme de XML, il ne limite pas les technologies des services dans ce qui concerne l'information qu'il stocke ou bien les manières dans lesquelles cette information est décorée du méta-data.

Le modèle d'information UDDI comporte cinq structures de données principales, définies dans la spécification sous forme de schéma XML [CHR04]:

- ***BusinessEntity*** : Une structure de données supérieure dans UDDI est la structure de *businessEntity*, employée pour représenter des entreprises et des fournisseurs dans UDDI. Elle contient des informations descriptives sur le fournisseur et au sujet des services qu'il offre. Ceci inclurait l'information telle que des noms et des descriptions dans des langues multiples, l'information de contact et l'information de classification. Les descriptions de service et l'information technique sont exprimées dans un *businessEntity* par les structures contenues de *businessService* et de *bindingTemplate*.
- ***businessService*** : Chaque structure de *businessService* représente la logique de groupage des services Web. Au niveau de service, il ne reste aucune information technique fournie pour ces services ; plutôt, cette structure permet la possibilité d'assembler un ensemble de services sous une rubrique commune. Chaque *businessService* est le fils logique d'un *businessEntity*. Chaque *businessService* contient l'information descriptive (encore, des noms, des descriptions et

l'information de classification) décrivant le but des différents services Web qu'il contient.

- ***bindingTemplate*** : Chaque structure de *bindingTemplate* représente un service Web individuel. Contrairement aux structures de *businessService* et de *businessEntity*, qui sont orientées vers des informations auxiliaires sur des fournisseurs et des services, un *bindingTemplate* fournit les informations techniques requises par des applications pour lier et agir l'un sur l'autre avec le service Web décrit. Il doit contenir le point d'accès pour un service donné ou un mécanisme d'adressage indirect qui mènera un au point d'accès. Chaque *bindingTemplate* est le fils d'un *businessService*. Un *bindingTemplate* peuvent être décorés du méta-data qui permet la découverte de ce bindingTemplate, selon un ensemble de paramètres et de critères.
- ***tModel (Technical Model)*** : les modèles techniques, ou les *tModels* sont employés dans UDDI pour représenter des concepts ou des constructions. Ils fournissent une structure qui permet la réutilisation. Le modèle de l'information d'UDDI est basé sur cette notion des caractéristiques partagées et emploie des *tModels* pour engendrer ce comportement. Pour cette raison, les *tModels* existent en dehors des Relationship retenue père-fils entre le *businessEntity*, le *businessService* et les structures de *bindingTemplate*. Pour décrire un service Web qui se conforme à un ensemble particulier de caractéristiques, transports et protocoles, des références aux *tModels* qui représentent ces concepts sont placés dans le *bindingTemplate*. D'une telle manière, des *tModels* peuvent être réutilisés par les *bindingTemplate* multiples.
- ***PublisherAssertion*** : assertions contractuelles entre partenaires dans le cadre des échanges d'exécution d'un service.

2.7.2.1. Structure de BusinessService

La structure de *businessService* représente un service logique et contient l'information descriptive en termes d'affaires. Une entreprise peut enregistrer plusieurs services. L'élément *BusinessEntity* décrit précédemment peut contenir un ou plusieurs éléments *BusinessService* pour chacune de ces familles de services. Le type d'informations contenues dans l'élément *BusinessService* correspond aux informations pages jaunes d'une entreprise.

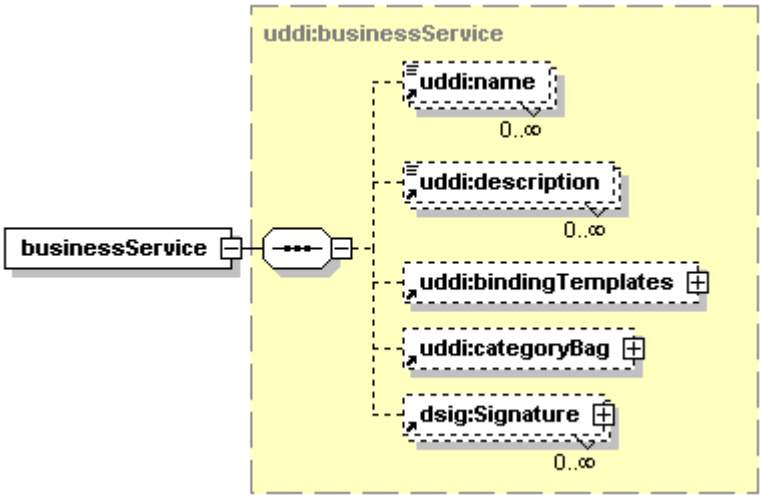


Figure 20. Diagramme de structure d'un *BusinessService* [CHR04].

Les éléments de cette structure sont :

- L'élément *name* est obligatoire. Il contient le nom des personnes qui ont enregistré ce *BusinessService*.
- L'élément *description* contient la description du *BusinessService* dans une ou plusieurs langues.
- L'élément *BindingTemplate* contient les modèles de liaisons donnant la description du service technique.

L'élément *CategoryBag* contient une liste de couples nom/valeur. Ces informations peuvent être des codes produits, des codes services ou des codes géographiques.

2.7.2.2. Structure de BindingTemplate

Des descriptions techniques des services Web sont fournies par des entités de *bindingTemplate*. Chaque *bindingTemplate* décrit un exemple d'un service Web offert à une adresse de réseau particulière, typiquement donnée sous forme d'URL. Le *bindingTemplate* décrit également le type de service Web offert en utilisant des références aux *tModels*, aux paramètres spécifiques à l'application, et aux arrangements. Chaque *bindingTemplate* est contenu dans un *businessService*.

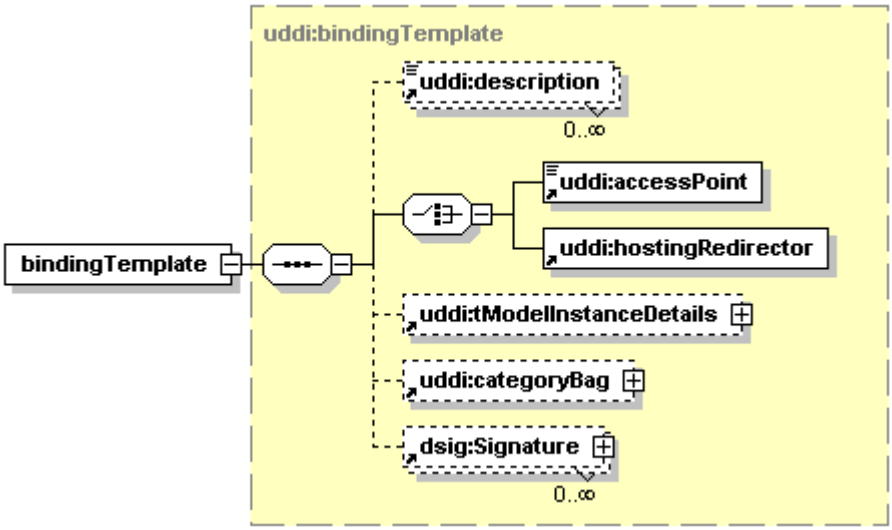


Figure 21. Diagramme de structure d'un *BindingTemplate* [CHR04].

Les attributs et les éléments contenus dans cette structure sont les suivants [CHR04]:

- L'élément *description* contient les descriptions du modèle de liaison.
- L'élément *accesspoint* est le point d'accès à l'information. Il contient le point d'entrée du service. Les valeurs de point d'accès valides comprennent l'URL, l'adresse émail ou même le numéro de téléphone.
- L'élément *hostingRedirector* est un élément pointant vers un autre modèle de liaison et utilisé lorsque le point d'accès n'est pas indiqué.

2.7.2.3. Structure de tModel

La structure *tModel* à pour rôle de décrire les spécifications des services Web à enregistrer. Elle comporte les informations permettant de connaître les normes aux quelles le service est conforme afin d'avoir une description suffisamment précise du service. Le nom doit se présenter suivant le format URI (*Uniform Ressouce Identifier*) et doit, pointer vers l'emplacement du fichier correspond, généralement au format WSDL. La définition de structure pour un *tModel* est présentée par le diagramme suivant :

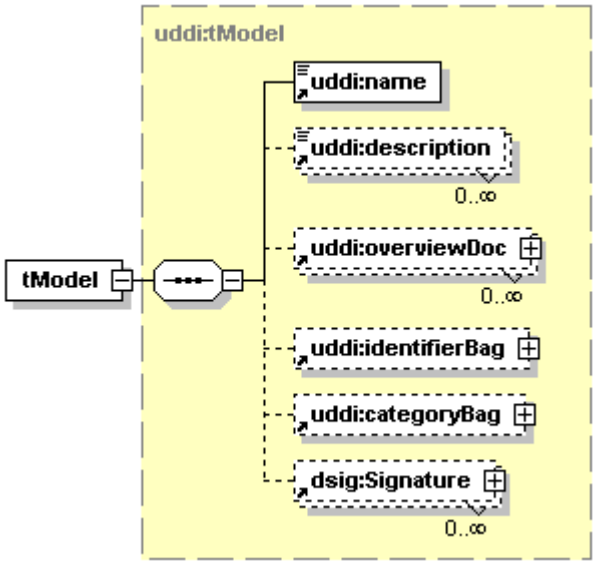


Figure 22. Diagramme de structure d'un tModel [CHR04].

Un *tModel* est constitué des éléments suivants :

- L'élément *name* est la liste de personnes qui ont enregistré cette structure.
- L'élément *description* donne une description de *tModel*.
- L'élément *overviewDoc* permet de désigner les références liées au *tModel*.
- L'élément *identifierBag* est constitué d'une liste de couples nom/valeur utilisé pour enregistrer les numéros d'identification du *tModel*.
- L'élément *categoryBag* est constitué d'une liste de couples nom/valeur utilisé pour enregistrer les informations de classification du *tModel*.

2.7.2.4. Structure de publisherAssertion

Cette structure comporte les assertions contractuelles entre organismes pour les services publiés. Ces assertions représentent un ensemble de règles contractuelle d'invocations de services présentées sous la forme de protocole entre deux partenaires est défini à travers ces assertions. La définition de cette structure se présente dans le diagramme suivant :

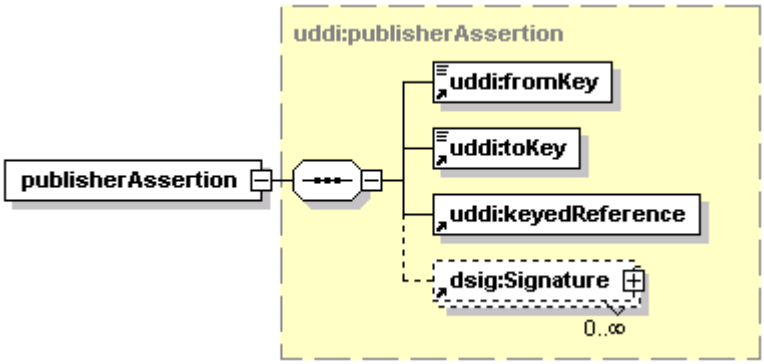


Figure 23. Diagramme de structure d'un *publisherAssertion* [CHR04].

Les éléments de cette structure sont :

- formKey qui désigne la première entité métier pour laquelle l'assertion est créée.
- *toKey* qui désigne la seconde entité métier pour laquelle l'assertion est créée.
- *keyReference* qui désigne le type de relation.

L'élément *keyReference* permet de décrire les catégories et les identifications de l'entreprise. La plupart des entreprises peuvent être classifié en fonction de leur activité.

2.7.3. Vue d'ensemble de structure de données UDDI

Lors de la modélisation de données de description de services dans l'annuaire, une entité métier *BusinessEntity* est une information liée au métier qui regroupe un ensemble d'éléments de services *BusinessService* qui vont être publiés. Chaque élément de service *BusinessService* contient les informations techniques et des descriptives sur un service Web. Un *BusinessService* contient une collection d'éléments *BindingTemplate* qui décrivent l'accès à l'information et la manière dont le *BusinessService* utilise les différentes spécifications techniques. Une spécification technique est modélisée sous forme d'un *tModel* pouvant lui-même modéliser des concepts différents : un type de service ou une taxinomie.

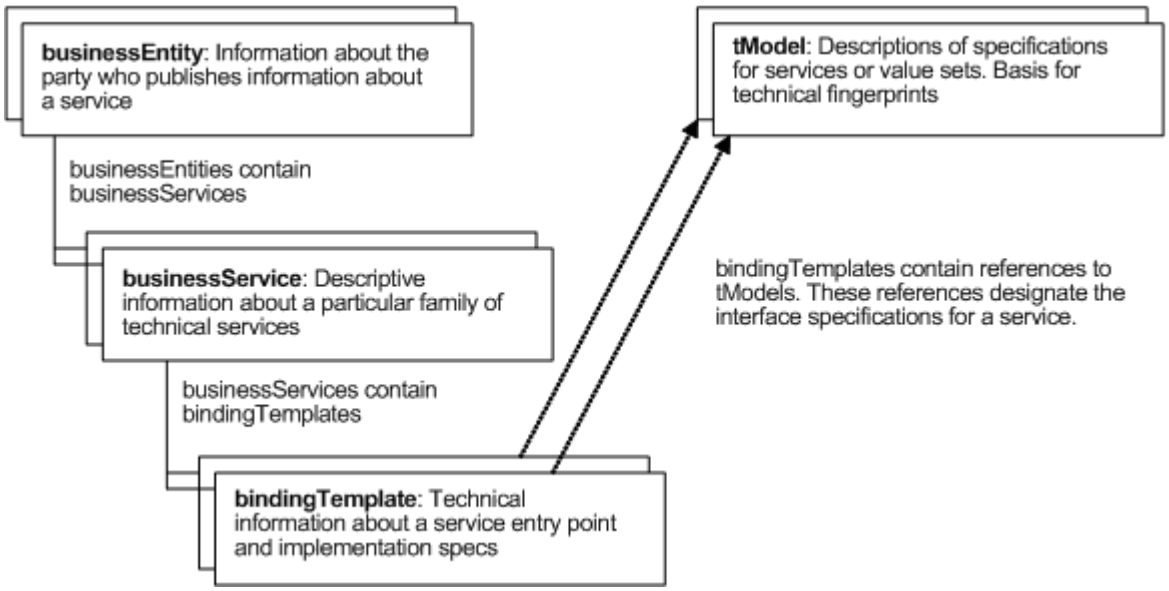


Figure 24. Structures de données de noyau d'UDDI [CHR04].

2.7.4. Relation entre éléments WSDL et structures UDDI

UDDI et WSDL sont complémentaires dans l'implémentation des services Web. UDDI est une méthode de publication et de recherche de description de service. Les structures de données de UDDI fournissent un support pour définir aussi bien les informations métier et celle de service. L'information de description de service définit dans WSDL est complémentaire aux informations trouvées dans un nœud UDDI qui fournit le support à plusieurs types de description de service. Une implémentation de service est publiée dans un nœud UDDI sous forme d'un *businessService* avec un ou plusieurs *bindingTemplate*. La publication des implémentations de service WSDL provoque successivement dans le nœud UDDI la mise à jour d'UDDI *businessService* et d'UDDI *bindingTemplate*.

Toutes les interfaces de service WSDL sont publiées dans un nœud UDDI sous forme d'un *tModel*. Chacun de ces *tModel* est catégorisé pour identifier une description de service WSDL.

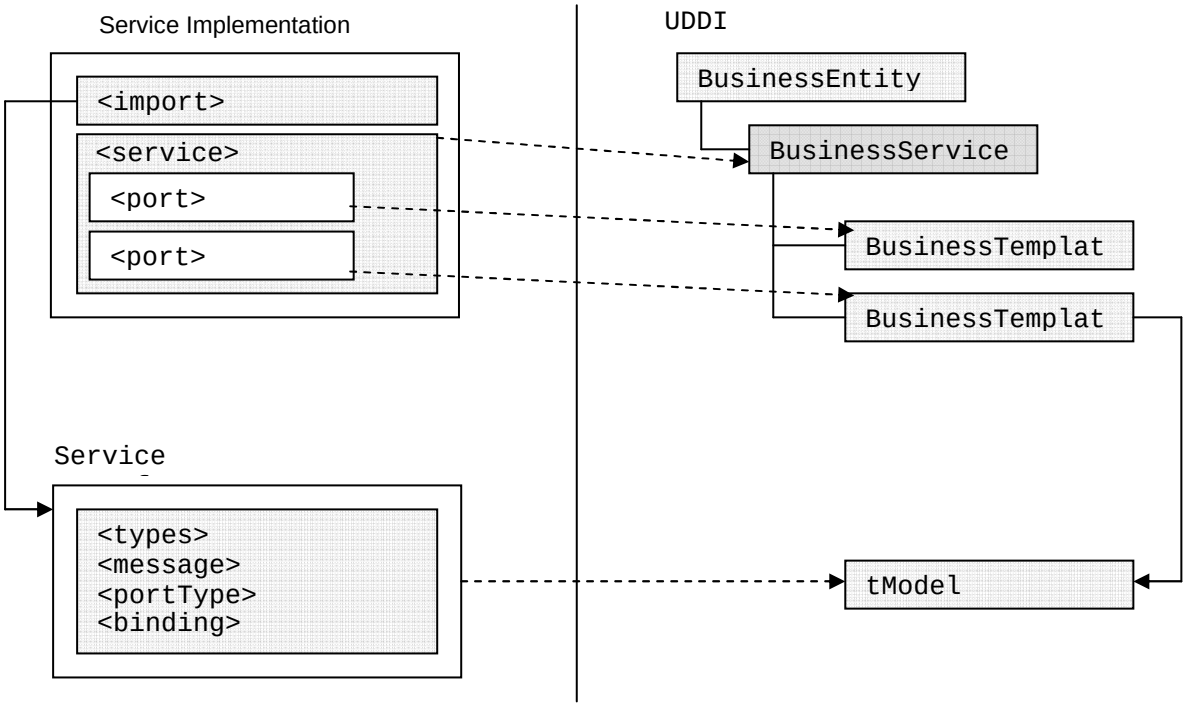


Figure 25. Relation entre éléments WSDL et structures UDDI [KM03].

2.8. Conclusion

Les services Web assurent, à travers l'Internet, l'interaction entre les applications, les ordinateurs et les processus métier en permettant d'accéder à partir d'un seul site Web à plusieurs services distants. Ce nouveau modèle de programmation et de déploiement d'applications assure l'interconnexion de services Web en se basant sur les standards suivants : HTTP, SOAP, WSDL, UDDI.

Dans ce chapitre, nous avons détaillé comment les services Web permettent à des applications de dialoguer via Internet par échange de messages, et ceci indépendamment des plates formes et les langages sur les quelles sont implémentées. Nous avons également exploré les différentes technologies, qui permettent la mise en œuvre des services Web.

Nous continuerons avec les services Web dans le chapitre suivant, et nous allons présenter en détail deux concepts importants qui sont : la composition des services Web (ou les services Web composites) et la sécurité des services Web (les politiques privées), en fournissant la description, le fonctionnement et les différents types et les langages (spécifications et standards) de définition de ces concepts.

Chapitre 3. Composition et sécurité des services Web.

3.1. Introduction

La composition de services Web vient apporter une réponse aux problèmes d'intégration et d'interopérabilité des systèmes d'information. Elle s'appuie sur des techniques permettant d'assembler des services Web pour réaliser des processus métier par des primitives de contrôles (boucles, tests, traitements d'exception ...) et d'échanges (envoi et réception de messages) [KON06]. Elle permet de définir les activités participant à un processus métier, une activité pouvant être un service élémentaire ou un service composé.

Pour les entreprises, composer des services, permet de faire communiquer leurs applications avec celles d'autres entreprises quelles que soient leur localisation géographique et les plates-formes de développement afin de réduire les coûts et de faciliter leur interopérabilité.

La sécurisation d'une infrastructure fondée sur une architecture SOA s'avère beaucoup plus complexe que celle des environnements traditionnels. L'agrégation de services de provenances diverses implique la nécessité de pouvoir proposer un modèle d'abstraction qui découple la sécurité de celle des modèles liés aux différents fournisseurs de services.

Dans ce chapitre, nous allons expliquer les différents spécifications et standards permettant la composition des services Web et les différents spécifications et standards définis pour assurer la sécurité et la vie privée des utilisateurs au sein d'une infrastructure basée sur les services Web.

3.2. Composition des services Web

Un service web est dit *composé* ou *composite* lorsque son exécution implique des interactions avec d'autres services web afin de faire appel à leurs fonctionnalités. La composition de services web spécifie quels services ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'exception.

3.2.1. Description et fonctionnement

La composition consiste à combiner les fonctionnalités de plusieurs services Web au sein d'un même processus métier (ou business process) dans le but de répondre à des demandes complexes qu'un seul service ne pourrait pas satisfaire. Le processus métier est une représentation concrète des tâches à accomplir dans une composition. Le déroulement d'une composition nécessite la réalisation des étapes suivantes [MRI07]:

- **La découverte des services Web** pouvant répondre aux besoins de la composition se fait généralement de manière manuelle par envoi de requêtes aux annuaires UDDI. De nombreux travaux visent à automatiser cette étape qui joue

un rôle très important dans la composition dynamique des services web, elle est liée essentiellement à la représentation d'une manière uniforme des services Web et à la découverte des services en fonction des contraintes et des besoins. Il y a deux approches pour la découverte des services Web [HAL05]: la première approche est basée sur la description syntaxique des services Web (WSDL), la deuxième approche est basée sur les ontologies, plusieurs langages comme DAML-S (*Darpa Agent Markup Language*) et OWL (*Ontology Web Language*) qui permettent de décrire la sémantique des services Web en se basant sur des ontologies.

- **L'organisation des interactions** entre les services Web composés est distinguée par les termes chorégraphie et orchestration. Une composition est associée à une spécification pour gérer les échanges de messages et mettre en place les structures de contrôle nécessaires (boucles itératives, opérateurs conditionnels et gestion des exceptions). Plusieurs langages de spécification ont été proposés dans la littérature : WSCI, WSFL, XLANG, etc.
- **L'exécution de la composition** est l'étape d'invocation effective des services Web participants à une composition. Une spécification de composition est hébergée par un serveur et son exécution est prise en charge par un moteur de composition. La surveillance de l'exécution par le moteur de composition permet de gérer certaines erreurs dynamiquement.

Nous pouvons classer les différentes techniques de composition des services Web en deux grandes familles :

- Les techniques de composition statiques, c'est-à-dire, qui sont définies à l'aide de processus métier (orchestration et chorégraphie). Il s'agit d'un type de composition dans lequel les services Web à composer sont pré-calculés avant qu'un client ne fasse une requête du service composite. Ce type de composition peut être appliqué dans des environnements « stable » ou les services Web participants sont toujours disponibles et/ou le comportement du service composite est le même pour tous les clients [HAL05].
- Les techniques de composition dynamiques, c'est-à-dire, l'agrégation de services Web permettant de résoudre un objectif précis soumis par un utilisateur en prenant en compte ses préférences. Cette composition peut se faire avant ou pendant l'exécution des services Web. La composition dynamique apparaît la plus intéressante car, d'une part, elle promet d'être capable de faire face à un environnement très dynamique dans lequel, des services apparaissent et disparaissent rapidement, et d'autre part, elle permet de mieux satisfaire les besoins de chaque client. Parmi les approches utilisées dans les techniques de composition dynamique on trouve : l'approche intelligence artificielle, l'approche multi-agent, etc.

L'étude du problème de composition des services Web a permet de développer plusieurs techniques jusqu'à présent dont l'objectif est de rendre le problème de composition (semi) automatique. Ces techniques se basent sur deux approches : une première approche qui est syntaxique qui visent à définir des compositions en fonction de la syntaxe des services Web. La deuxième se base sur une description sémantique des services Web via des ontologies.

3.2.2. La définition de l'Orchestration et de la Chorégraphie

Les termes orchestration et la chorégraphie décrivent deux aspects de créer des processus métiers de services Web composés (composites). L'orchestration se réfère à un processus exécutable. La chorégraphie trace la séquence de messages pouvant impliquer plusieurs parties et plusieurs sources. Le chevauchement des deux termes et petit, mais la figure 26 illustre leur relation l'un de l'autre à un haut niveau.

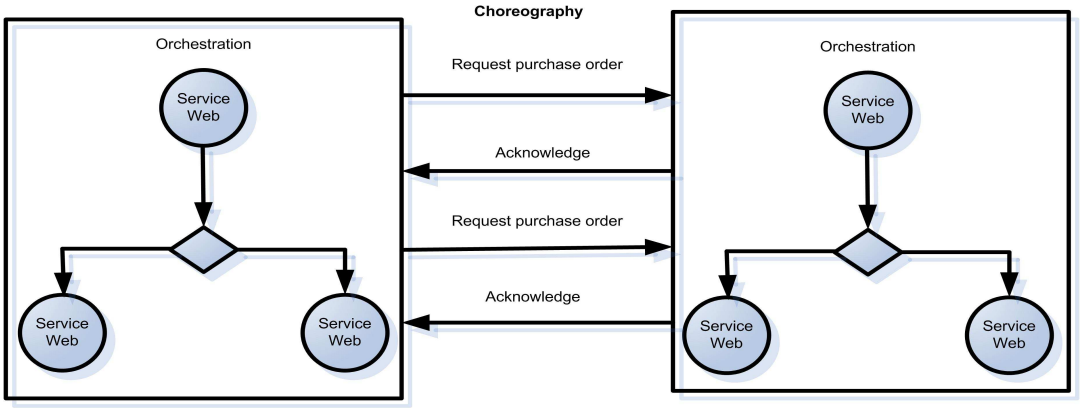


Figure 26. Orchestration contre chorégraphie [PEL03].

3.2.2.1. L'orchestration de services Web

L'orchestration de services permet de définir l'enchaînement des services selon un canevas prédéfini, et de les exécuter à travers des "scripts d'orchestration". Ces scripts sont souvent représentés par des procédés métier ou des WorkFlow inter ou intra-entreprise. Ils décrivent les interactions entre applications en identifiant les messages, et en branchant la logique et les séquences d'invocation. L'orchestration décrit la manière par laquelle les services web peuvent interagir ensemble au niveau des messages, incluant le logique métier et l'ordre d'exécution des interactions. Ces interactions peuvent couvrir des applications et/ou des organisations, et le résultat peut être un modèle de procédé de longue durée, transactionnel, et multi-étapes [PEL03].

La figure 27 montre le WorkFlow dans l'orchestration des services web. Un coordinateur prend le control de tous les services web impliqués et coordonne

l'exécution des différentes opérations des services web qui participent dans le processus.

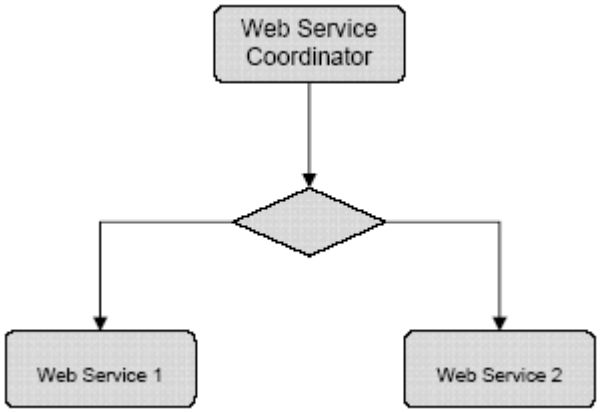


Figure 27. Orchestration de services Web.

3.2.2.2. La chorégraphie

La chorégraphie trace la séquence de messages pouvant impliquer plusieurs parties et plusieurs sources, incluant les clients, les fournisseurs, et les partenaires. La chorégraphie est typiquement associée à l'échange de messages publics entre les services web, plutôt qu'à un procédé métier spécifique, exécuté par un seul partenaire [PEL03].

La collaboration dans la chorégraphie des services web peut être représentée par la figure suivante :

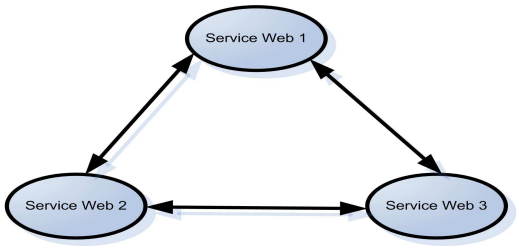


Figure 28. La chorégraphie de services Web.

Il y a une différence importante entre l'orchestration et la chorégraphie de services web. L'orchestration se base sur un procédé métier exécutable pouvant interagir avec les services web internes ou externes. L'orchestration offre une vision centralisée, le procédé est toujours contrôlé du point de vue d'un des partenaires métier. La chorégraphie est de nature plus collaborative, chaque participant impliqué dans le procédé décrit le rôle qu'il joue dans l'interaction. Beaucoup de standards se sont intéressés initialement soit à l'orchestration soit à la chorégraphie.

3.2.3. Spécifications et standards de composition de services Web

L'orchestration et la chorégraphie sont des compositions statiques de services web permettant de créer un processus métier (WorkFlow). Les principaux langages d'orchestration et/ou de chorégraphie de services web proposés sont représentés sur la figure suivante dans l'ordre chronologique d'apparition sur le marché [JAM05]:

- XLANG - XML Business Process Language (Microsoft);
- BPML - Business Process Modeling Language (BPMI) ;
- WSFL - Web Service Flow Langage (IBM) ;
- WSCL - Web Service Conversation Language (Hewlett-Packard);
- WSCI - Web Service Choregraphy Interface (SUN);
- BPEL4WS - Business Process Execution Language for Web services.

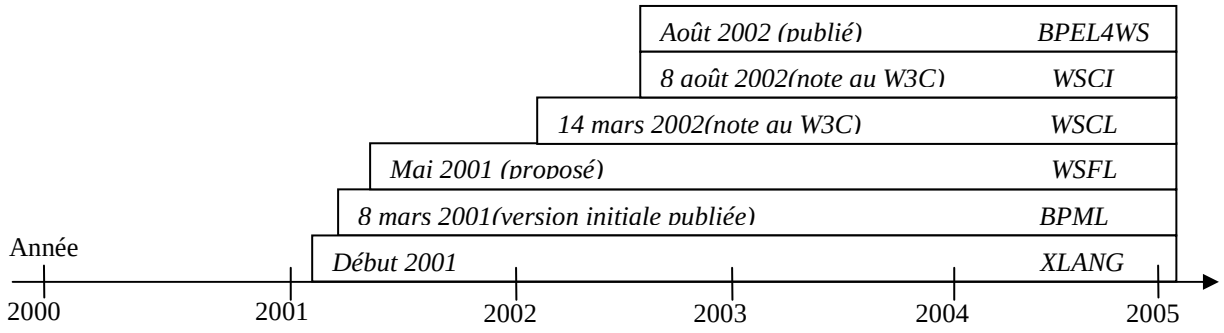


Figure 29. Ordre chronologique d'apparitions des langages d'orchestration et/ou de chorégraphie.

Ils existent plusieurs langages pour la composition de services web. Dans cette section nous allons présenter brièvement quelques parmi ceux qui sont cités au dessus.

3.2.3.1. XLANG

Créé par Microsoft, le XLANG est une extension de WSDL. Il fournit en même temps un modèle pour une orchestration des services et des contrats de collaboration entre celles-ci. XLANG a été conçus avec une base explicite de théorie de calcul. Les actions sont les constituants de base d'une définition de processus de XLANG. Les quatre types d'opérations de WSDL (requête/réponse, sollicitation de la réponse, le sens unique, et la notification) peuvent être employés comme actions de XLANG. Le fichier de description de service XLANG contient donc la description WSDL, et y ajoute les éléments suivants :

- **Le comportement (*behavior*) :** Le comportement spécifie l'enchaînement des activités par des balises propres à XLANG. La partie comportement contient :

- Les actions XLANG : *operation*, *delayFor*, *delayUntil*, et *raise* ;
- Le contrôle XLANG : qui définit l'enchaînement de l'exécution des actions, et qui peut être l'une des commandes : *empty*, *sequence*, *switch*, *while*, *all*, et *pick*.
- **Le contexte (*context*) :** Le contexte spécifie les transactions éventuelles du processus et les références utilisées dans le document. Il permet de définir :
- Des variables locales, constituées de corrélations et de références aux ports.
- Les corrélations sont des propriétés simples ou complexes, permettant de corréler les messages entre eux.
- L'ensemble des corrélations constitue une sorte d'index de messages ;
- **Le contrat (*contract*) :** Le contrat spécifie les relations entre services XLANG. Le contrat est spécifié dans un fichier à part. Ce fichier importe les fichiers XLANG des services participant à ce contrat, et il précise la manière dont les ports de ces services sont liés.

Microsoft a exploité le langage XLANG dans son environnement BizTalk [CHA02]. BizTalk contient un environnement de développement : BizTalk Orchestration Designer, permettant de générer les « *Schedule* » XLANG (les fichiers XLANG sont appelés *Schedule*). Le serveur BizTalk utilise ensuite ces *Schedule* XLANG pour instancier les processus à exécuter.

3.2.3.2. WSFL - Web Services Flow Language

Le langage WSFL est une proposition effectuée par IBM, concernant la composition de services web. WSFL est un langage XML, pour décrire la composition de services web. Il offre deux styles de composition [CHA02]:

- la description explicite de la succession des étapes et de l'enchaînement des appels aux opérations des services Web, appelée processus métier ;
- Un modèle d'interaction de service Web pris deux à deux. C'est une description abstraite, basée sur les deux fichiers WSDL, en laissant les services web afin de les implémenter plus tard.

Dans le langage WSFL, six éléments spécifient une composition de services web :

- *flowSource* et *flowSink* indiquent les données en entrée et en sortie de la composition;
- *serviceProvider* indique les services web participant à la composition ;
- *activity* décrit une opération ou une étape du dialogue ;
- *controlLink* et *dataLink* spécifient respectivement comment se succèdent les étapes et les données qui sont passées d'une étape à la suivante.

Le processus WSFL est donc une succession d'opérations dont le flot de contrôle et le flot de données sont explicités séparément.

L'implémentation de l'activité est soit déléguée à un service externe (par la balise *Export*), soit exécutée à partir d'autres activités du processus WSFL (indiqué par la balise *Internal*). Dans le cas de la délégation, le service externe est à son tour soit explicitement nommé, soit implicitement indiqué par un élément `"Locator"` identifiant le fournisseur de service.

L'activité comporte des attributs facultatifs :

- L'attribut *exitCondition* : condition de sortie.
- L'attribut *join* : condition jointe.
- Ainsi qu'une condition de transition facultative, qui peut être ajoutée dans les liens de contrôle *controlLink*.

WSFL définit également des opérations de "contrôle du cycle de vie". Ces opérations lancent et contrôlent les activités :

- Spawn crée une instance de l'activité ou de la composition, et la démarre.
- *Call* exécute une instance d'activité ou de composition.
- *Enquire* renvoie l'état d'une activité ou d'une composition.
- *Terminate* arrête une instance d'activité ou de composition.
- *Suspend* suspend l'exécution en cours de l'instance.
- *Resume* reprend l'exécution d'une instance suspendue.

Le logiciel "MQ Series WorkFlow" de IBM, connu aujourd'hui sous le nom de "WebSphere Process Manager", a pris en charge la spécification WSFL, afin d'automatiser les flots de procédés métier.

3.2.3.3. BPEL4WS - Business Process Execution Language for Web Services

BPEL4WS est un langage pour les processus métier basé sur XML conçu pour permettre charger/partager les données distribuées, même à travers des organismes multiples, en employant une combinaison de services web. Il décrit l'interaction des processus métiers basés sur les services Web, à la fois au sein des entreprises et entre elles. BPEL4WS rend possible l'interopérabilité entre des activités commerciales basées sur des technologies différentes.

Le modèle de procédé BPEL4WS forme une couche au-dessus du WSDL. Il définit la coordination des interactions entre l'instance de procédé et ses partenaires. BPEL4WS

contient les caractéristiques d'un langage structuré en blocs (*block structured language*) du XLANG, ainsi que les caractéristiques d'un graphe direct de WSFL.

BPEL4WS permet de modéliser deux types de procédé :

- Le *procédé abstrait* : spécifie les échanges de messages entre les différentes parties, sans spécifier le comportement interne de chacun d'eux ;
- Le *procédé exécutable* : spécifie l'ordre d'exécution des activités constituant le procédé, des partenaires impliqués dans le procédé, des messages échangés entre ces partenaires, et le traitement de fautes et d'exceptions spécifiant le comportement dans les cas d'erreurs ou d'exceptions.

Les partenaires

La relation entre le procédé et un partenaire est une relation poste à poste (*peer-to-peer*). Le partenaire est en même temps le consommateur d'un service que le procédé produit, et le producteur d'un service que le procédé consomme. Le lien de partenariat (*partner link*) définit le rôle que joue chacun des deux partenaires dans une conversation. Chaque lien de partenariat a un type (*partnerLinkType*).

Les messages

BPEL4WS ajoute des propriétés aux messages échangés entre partenaires. Un exemple de ces propriétés est la corrélation, qui permet de faire le lien entre les messages appartenant à la même conversation.

Les activités

Le procédé dans BPEL4WS est constitué d'activités, liés par un flot de contrôle. Ces activités peuvent être *basiques* ou *structurées*. Les activités basiques sont : *invoke* pour invoquer une opération dans un service web ; *receive* pour attendre un message d'une source externe ; *reply* pour répondre à une source externe ; *wait* pour attendre un certain temps ; *assign* pour copier les données d'une place à l'autre ; *throw* pour lancer une erreur d'exécution ; *terminate* pour terminer l'instance du service web ; et *empty* qui ne fait rien (utile pour la synchronisation des activités concurrentes).

Les activités structurées sont composées d'autres activités basiques et structurées. Les types d'activités structurées sont : *sequence* pour définir un ordre d'exécution ; *switch* pour l'acheminement conditionnel ; *while* pour les boucles ; *pick* pour attendre l'arrivée d'un événement ; *flow* pour l'acheminement parallèle ; *scope* pour regrouper les activités afin qu'elles soient traitées par le même gestionnaire d'erreur et *compensate* pour invoquer les activités de compensation par le gestionnaire d'erreur, pour défaire l'exécution déjà complétée d'un regroupement d'activité.

Les données

Le procédé dans BPEL4WS a un état, cet état est maintenu par des variables contenant des données. Ces données sont combinées afin de contrôler le comportement du procédé, pour cela il utilise les expressions. Les expressions permettent d'ajouter des conditions de transition ou de jointure au flot de contrôle. L'affectation (*assignment*) permet de mettre à jour l'état du procédé, en copiant les données d'une variable à une autre, ou en introduisant de nouvelles données en utilisant les expressions.

Dans BPEL4WS il n'y a pas de flot de données, il se sert des variables pour passer une donnée d'une activité à une autre, à l'aide de l'affectation.

IBM, Microsoft et BEA ont réalisé ensemble la spécification BPEL4WS, qui étend et remplace les précédentes spécifications XLANG de Microsoft, et WSFL d'IBM. Ce qui explique le peu d'implémentation de la spécification WSFL. BPEL4WS, en revanche, a eu beaucoup de succès, et beaucoup d'implémentations industrielles ont été créées pour prendre en charge cette spécification. IBM a même remplacé, dans le logiciel WebSphere, la spécification WSFL qu'il a pris en charge au début, par la spécification BPEL4WS, pour gérer les procédés métier.

3.2.3.4. BPML - Business Process Modelling Language

BPML est un Meta langage de modélisation des processus métiers dont les premières spécifications sont apparues en 2001. Il permet de définir un modèle abstrait d'interaction entre collaborateurs participant à une activité de l'entreprise, voire entre une organisation et ses partenaires. Les processus métiers sont représentés par un flux de données, un flux d'événements sur lesquels on peut influer en définissant des règles métier, des règles de sécurité, des règles de transactions. On peut ensuite lancer l'exécution du modèle (pour les spécialistes il s'agit d'un automate à états finis) et vérifier le fonctionnement théorique des différents processus [JDN].

Un processus métier BPML est un enchaînement d'activités simples, complexes, et de processus incluant une interaction entre participants dans le but de réaliser un objectif métier. Les éléments de définition d'un processus sont [CHA02] :

Les messages

Le message est l'unité d'interaction entre processus métier. Le processus métier BPML orchestre les échanges entre divers participants par la coordination de l'envoi et de la réception de messages selon un jeu de règles explicite.

Les participants

Les participants BPML sont les parties prenantes à un processus métier. Ce sont toutes les ressources disponibles de l'entreprise (humaines, matérielles et logicielles). Il existe deux types de participants dans BPML : les participants statiques (où l'identité et le

comportement sont connus à l'avance), et les participants dynamiques (ajoutés au moment de l'exécution).

Les activités

L'activité est l'unité d'exécution du processus. Les participants et les processus sont représentés en BPML comme des activités productrices et consommatrices de messages. Il existe deux classes d'activités dans BPML :

- les tâche simples, soit une communication (synchrone ou asynchrone) avec un participant telle que la consommation ou production d'un message (opération ou exception).
- les tâches complexes composées à partir de tâches simples dont les messages sont visibles par tous les participants auxquels ils s'adressent.

Les transactions

L'activité peut posséder un attribut supplémentaire afin de spécifier si elle est exécutée dans un contexte transactionnel. Il est également possible de définir des compensations pour les transactions longues partielles. BPML définit deux modèles de transactions : coordonnées (pour les délais courts), et étendues (pour les délais longs).

Les exceptions

BPML définit un système de gestion d'exceptions, en ajoutant un dispositif de récupération d'erreur au moteur de processus (balise *ocFault*).

Les règles métier

Les règles du processus gouvernent le choix des tâches, la gestion de la consommation ou de la production de messages, et la réaction aux erreurs. Elles sont du style :

```
Si {conditions} alors {actions}
```

Les conditions portent sur les variables globales du processus ou sur les données échangées entre participants. Les actions déclenchent d'autres tâches BPML.

3.2.3.5. WSCL - Web Services Conversation Language

Le langage WSCL est une soumission de Hewlett-Packard au World Wide Web Consortium, sous forme d'une note, le 14 mars 2002. L'objectif consiste à décrire la séquence d'interactions possibles avec un service web particulier [ARI02].

WSCL propose de décrire à l'aide de document XML les services Web et mettant l'accent sur les conversations de ceux-ci, c'est-à-dire sur les messages échangés et leurs ordres. Il a été conçu pour s'employer conjointement avec WSDL. Les définitions WSDL peuvent être utilisées par WSCL pour décrire les opérations possibles ainsi que leur chorégraphie.

Les éléments d'une spécification WSCL sont les suivants [HAL05]:

- Les schémas des documents XML échangés au cours d'une conversation. Ces schémas ne font pas partie du document de spécification WSCL, ce sont des documents séparés que l'on référence par un URL dans la spécification de la conversation ;
- Les *interactions* modélisant les actions de la conversation sous forme des échanges de documents. Il existe cinq types d'interactions dans WSCL [ARI02]:
- *Send* : le service envoie un document,
- *Receive* : le service reçoit un document,
- *SendReceive* : le service envoie un document et en attend un en retour,
- *ReceiveSend* : le service reçoit un document et en renvoie un en retour,
- *Empty* : ne contient pas de message à échanger, le rôle de ce type est de modéliser le début et la fin d'une conversation.
- Les *transitions* spécifiant la relation d'ordre entre les opérations. Pour cela, une transition possède une interaction source et une interaction de destination. Il est également possible de spécifier un type de document pour la transition source ainsi qu'une condition sur la transition ;
- La *conversation* donne la liste de toutes les interactions et les transitions qui forment une conversation. Elles définissent aussi d'autres propriétés tel que le nom de la conversation ainsi que les interactions de début et de fin.

WSDL se charge de décrire les services web, et WSCL se charge de décrire les conversations entre deux services web. Ainsi, ces deux descriptions sont séparées, afin de permettre la réutilisation (une même conversation WSCL peut avoir lieu entre différentes paires de services web décrites en WSDL, et un service web décrit en WSDL peut participer à plusieurs conversations WSCL). Cette approche différencie WSCL des autres approches, tels que XLANG et WSFL.

Un fournisseur de service peut soit fournir la définition de conversation WSCL directement à l'utilisateur du service, soit l'enregistrer en tant que des "*tModel*" dans un annuaire UDDI, pour qu'elle soit à disposition de tous ceux qui ont accès à cet annuaire [JAM05].

3.2.3.6. WSCI - Web Services Choreography Interface

Le langage WSCI est une Initiative proposée par Sun, SAP, BEA, et Intalio. L'objectif consiste à prendre en compte la collaboration d'application à application. Cette initiative s'est concrétisée par une note au W3C, le 8 août 2002.

WSCI est un langage XML, permettant de décrire la chorégraphie entre les services web. Il propose de se focaliser sur la représentation de services Web en tant qu'interfaces décrivant le flot de messages échangés. Il propose aussi de décrire le comportement externe observable du service Web [HAL05]. Le langage WSCI définit les concepts suivants [ASS02]:

L'interface

WSCI a le but de décrire le comportement extérieur observable d'un service Web dans un échange des messages. Le comportement décrit est en termes de dépendances temporelles et logiques entre les messages que ce service web échange avec d'autres services web dans le contexte d'un scénario donné. Ce comportement est décrit dans un ou plusieurs processus contenus dans l'interface. Un service web peut avoir plusieurs interfaces lui permettant de jouer différents scénarios.

Les activités et leur chorégraphie

WSCI décrit le comportement d'un service web en termes d'activités chorégraphies. La chorégraphie décrit les dépendances temporelles et logiques entre les activités. Les activités peuvent être atomiques (activité d'envoie et/ou de réception d'un message, ou activité d'attente d'une durée définie) ou complexes (composées d'autres activités). L'activité complexe définit la chorégraphie des activités dont elle est composée.

WSCI supporte plusieurs types de chorégraphies:

- l'exécution séquentielle : les activités doivent être exécutées dans un ordre séquentiel ;
- l'exécution parallèle : Toutes les activités doivent être exécutées, mais ils peuvent être exécutés dans tout ordre.
- la boucle : les activités sont exécutées par une boucle basée sur l'évaluation d'une condition ou d'une expression.
- l'exécution conditionnelle : un ensemble ou plusieurs ensembles d'activité est exécuté selon l'évaluation d'une condition (*switch*) ou bien selon l'occurrence d'un évènement (*choice*).

Les processus

Un processus est une portion de comportement qui est libellé par un nom. Le comportement est décrit par un processus qui peut être réutilisé en référençant ses noms, le processus est l'unité réutilisable de WSCI. Plusieurs méthode dans les quelles un processus est utilisé (ou réutilisé) selon la propriété qui crée ce dernier :

- Réception d'un ou plusieurs messages qui sont définis comme déclencheurs de la chorégraphie décrit par le processus.

- Appel au processus. Les appels respectifs sont décrits explicitement dans l'interface.
- Créer le processus à l'intérieur de l'implémentation du service. Dans ce cas, l'instanciation réelle du processus n'est pas montrée dans l'interface.

Le contexte

Le contexte décrit l'environnement dans lequel s'exécute un ensemble d'activités. Chaque activité est définie dans un et un seul contexte. Le contexte est un concept WSCI, il décrit l'environnement d'exécution en terme : des déclarations (propriétés locales ou définitions locales de processus) disponibles pour les activités, des événements d'exception qui peuvent arriver, les propriétés transactionnelles associées à l'exécution des activités. Les contextes peuvent être emboîtés.

Chaque définition du contexte a un rapport à un ensemble particulier d'activités, et chaque activité est définie dans une définition du contexte. La définition du contexte pour les processus de haut niveau est supposée pour être vide. La définition du contexte décrit l'environnement pour un l'ensemble d'activités selon :

- L'ensemble de déclarations qui sont disponibles aux activités;
- L'ensemble d'événements exceptionnels qui peuvent se produire pendant l'exécution des activités, et le comportement exceptionnel a déclenché par ces événements;
- les propriétés transactionnelles associées à l'exécution des activités.

Les transactions

WSCI permet d'associer une transaction dans le contexte. Ainsi, les activités contenues dans cette transaction seront exécutées d'une manière atomique. La transaction peut contenir un ensemble d'activités de compensation, exécutées s'il y a besoin de défaire la transaction une fois qu'elle est terminée. Les transactions sont soit atomiques, soit emboîtées (composées d'autres transactions).

La corrélation de messages

Dans WSCI, une conversation représente un échange de messages entre deux ou plusieurs services web, participant à un scénario. Un service web peut être engagé dans plusieurs conversations en même temps, avec le même service ou avec des services différents. La corrélation est le mécanisme par lequel un message, reçu par le service, est associé à une conversation particulière. Ainsi, les différentes conversations sont distinguées par les instances de corrélation (une instance de corrélation est un ensemble de valeurs de propriétés).

Les exceptions

WSCI permet de déclarer d'un comportement exceptionnel qui est exposé par un service Web à un point donné dans une chorégraphie. La déclaration de comportement exceptionnel fait partie de la définition du contexte et associe des exceptions à un ensemble d'activités que le service Web exécutera en réponse à ces exceptions. Il est possible de déclarer une occurrence des types suivants d'exceptions :

- Réception d'un message particulier qui est considéré comme exceptionnel dans ce contexte.
- L'occurrence d'une faute; cette faute peut correspondre à la réception d'un message de faute WSDL, ou à la génération d'une faute par le service lui-même.
- L'occurrence d'un timeout.

Ainsi, lorsqu'une exception se produit, cela n'arrête pas la chorégraphie en entier, mais seulement le contexte où l'exception s'est produite après avoir exécuté les activités spécifiques à cette exception.

3.3. Technologies de confidentialité dans les services Web

Récemment il y a des demandes croissantes et des discussions au sujet des technologies de la confidentialité des services de Web dans l'industrie et la communauté de la recherche. En général, les politiques privées décrivent les pratiques des données d'une organisation : quelle information ils collectent de la part des consommateurs et qu'est ce qu'ils font (buts) avec cette information. Pour permettre une protection confidentielle pour les consommateurs de service Web à travers des domaines multiples, le consortium World Wide Web (W3C) a publié en 2002 [ABG02] un document appelé « *Web services architecture (WSA) requirements*» qui définit quelques besoins de la confidentialité spécifiques pour les services Web comme un futur sujet de la recherche [HFB04]. À ce moment, il existe des technologies de confidentialité des services Web standardisée. Nous présentons brièvement, par la suite, quelques technologies de la confidentialité des services Web.

3.3.1. Les exigences de l'architecture des services Web (WSA Requirements)

Le document [ABG02] décrit un ensemble d'exigences pour une architecture de référence standard des services Web développée par le W3C. Ces exigences sont projetées pour guider le développement de l'architecture de référence et pour fournir un ensemble de contraintes mesurables sur la mise en œuvre (l'implémentation) des services Web, avec lesquelles, la confidentialité dans les services Web peut être déterminée.

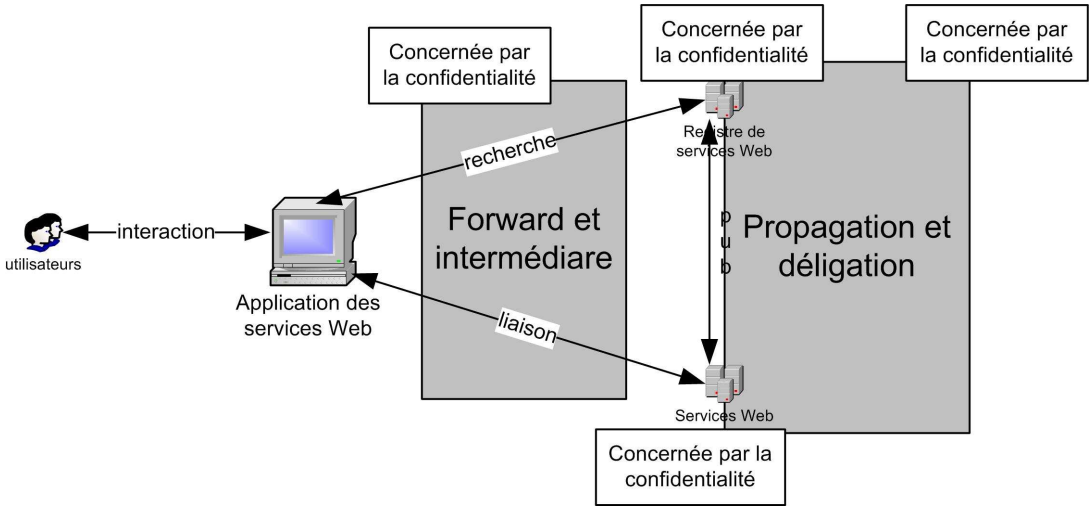


Figure 30. Les concernés par la confidentialité dans WSA [HFB 04].

La figure 30 représente les différentes parties concernées par la confidentialité dans le contexte d'une WSA (Architecture des Services Web). Sur le côté gauche, l'interaction des utilisateurs avec les services Web se fait via l'échange de l'information. Les informations échangées entre ces derniers peuvent contenir des données sensibles, donc, l'échange doit respecter la confidentialité des informations. Se reporter au modèle des services Web Publier/Découvrir/Invoquer, le fournisseur publient leurs services dans des registres (UDDI) pour que le publique accède. Les services Web sont décrits dans des documents WSDL. Alors, les utilisateurs (demandeurs des services Web) découvrent les services Web appropriés dans des annuaires UDDI. Dans beaucoup des cas, on peut avoir un médiateur (c.-à-d., un locateur des services), cela aide pour trouver des services Web appropriés pour les demandeurs. Une fois les services Web sont trouvés, l'application essaie d'invoquer chaque service Web en utilisant des messages SOAP.

Dans un point de vue utilisateur, les concernés par la confidentialité sont les registres UDDI et les services Web. Par exemple, les utilisateurs peuvent vouloir que les registres protègent leurs informations privées comme l'identité et les informations recherchées dans ces registres. De plus, les utilisateurs peuvent vouloir aussi validez les politiques de confidentialité de l'application et celles des services Web utilisés. Cela veut dire, que l'application des services Web doit invoquer seulement les services tels que leurs politiques de confidentialité sont satisfaites. D'un autre point de vue, des politiques de confidentialité sont définies dans des UDDI pour des entités de businesse spécifiques et les politiques de confidentialité définies dans le WSDL des services Web doivent consister avec.

3.3.2. La spécification WS-Security

WS-Security est un exemple du consensus industriel sur l'évolution des services Web. Cette spécification est aujourd'hui un standard de l'OASIS. Cette famille de spécifications

est critique pour les services Web inter organisations. Son intérêt a su fédérer les principaux acteurs et organisations (Microsoft, IBM, Sun, Oracle…). Plutôt que de proposer un nouveau Framework de sécurité, WS-Security s'inspire des mécanismes existants en utilisant XMLEncryption, XML Signature, ainsi que des jetons Kerberos ou des certificats X509v3 dans les en-têtes des messages SOAP pour offrir les trois mécanismes de protection des services Web : la transmission de jeton de sécurité, la signature numérique et le chiffrement de ces messages.

WS-Security offre un grand niveau d'abstraction permettant d'intégrer de nombreux systèmes de sécurité (Kerberos, PKI, XrML, SAML, SSL/TLS…) sur de multiples plates-formes. En outre, la flexibilité du modèle proposé par WS-Security, lui permettra d'évoluer pour supporter de nouveaux standards de sécurité.

3.3.2.1. Une sécurité fondée sur les messages

En se fondant sur une sécurité de niveau messages, WS-Security permet de sécuriser une solution mettant en œuvre une multiplicité de plates-formes, sans nécessité d'avoir la maîtrise de la configuration des différents nœuds extrémités ou intermédiaires intervenant dans l'échange et en offrant des compléments en termes de chiffrement et de non répudiation. Dans ce modèle de sécurité, les messages XML contiennent les informations d'identification, les signatures numériques qui peuvent être chiffrées. La figure 31 illustre le fonctionnement d'une solution mettant en œuvre cette spécification.

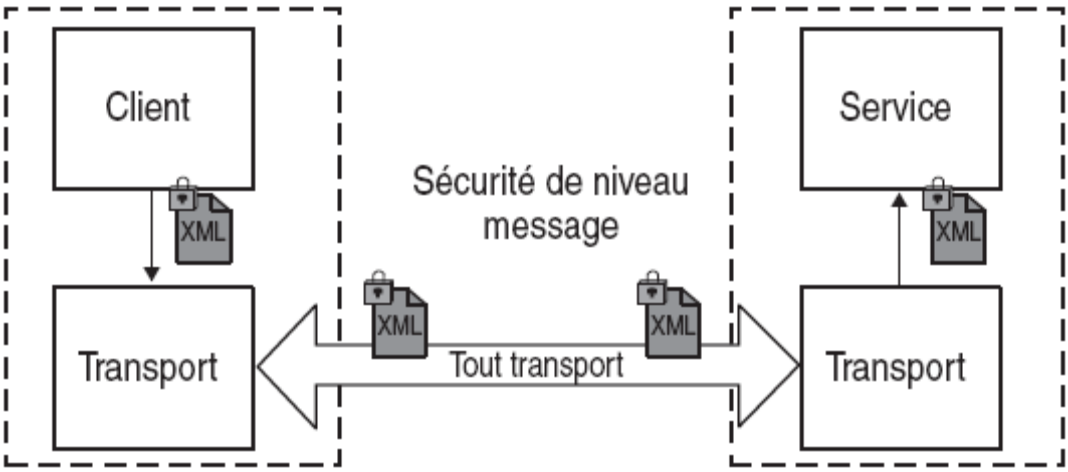


Figure 31. Sécurité au niveau message [MG04].

La spécification WS-Security définit un ensemble de « SOAP extensions » standards qui implémentent la sécurité (intégrité et confidentialité). Plutôt que de proposer un nouveau Framework de sécurité, WS-Security s'inspire des mécanismes existants permettant la mise en place d'une infrastructure sécurisée et propose trois mécanismes constituant un support flexible de protection des services Web :

– La transmission de jeton de sécurité par des mécanismes standards d'encodage et d'inclusion de jetons binaires de sécurité au sein du message SOAP et fondés sur

un modèle auto descriptif et extensible. En effet, les caractéristiques de ces jetons y sont présentes et de multiples formats peuvent être utilisés : couple utilisateur/mot de passe, certificats X.509 v3, tickets Kerberos, jetons XML : SAML, XrML.

– La signature numérique des messages pour assurer leur intégrité en vérifiant par cryptographie que le message SOAP n'a pas été altéré depuis qu'il a été signé en se fondant sur le standard *XML Signature*, défini par le W3C. Pour signer le message SOAP, l'émetteur a besoin de sa clé privée et pour que le récepteur puisse vérifier la signature du message SOAP, il a besoin de la clé publique de l'émetteur.

– Le chiffrement de ces messages afin de garantir la confidentialité des échanges en se fondant sur le standard *XML Encryption* défini par le W3C. Le chiffrement du message SOAP crée un secret cryptographique qui est partagé avec le destinataire du message. Pour chiffrer le message SOAP, l'émetteur a besoin de la clé publique du destinataire, et le récepteur a besoin de sa clé privée pour déchiffrer le message.

Chacune de ces fonctions peut être utilisée séparément ou s'associer pour adresser de nombreux scénarios d'architecture de services sécurisés. WS-Security ne définit pas les méthodes de chiffrement (*XML Encryption*) ou de signature des messages (*XML Signature*), mais comment incorporer les informations spécifiques à ces standards dans l'en-tête d'un message SOAP. En outre, cette spécification permet de définir les mécanismes de transmission de multiples jetons de sécurité, à travers de multiples domaines de confiance, en utilisant de multiples formats de signature et de chiffrement.

Pour véhiculer les informations de sécurité, WS-Security définit donc un entête générique pour le message SOAP indépendant du mécanisme de sécurité mis en œuvre. Cet en-tête doit pouvoir héberger plusieurs jetons de sécurité permettant d'identifier l'appelant, des informations sur la manière dont le message a été signé ou chiffré et des informations pour localiser la clé (contenue ou pas dans le message). Le schéma de cet en-tête est le suivant :

```
<xs:element name="Security">
   <xs:complexType>
     <xs:sequence>
       <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded">
       </xs:any>
     </xs:sequence>
    <xs:anyAttribute processContents="lax"/>
   </xs:complexType>
</xs:element>
```

L'utilisation de WS-Security et des protocoles associés n'exclut pas l'utilisation d'un protocole de transport sécurisé. Il est tout à fait pertinent de cumuler les deux approches.

3.3.3. La specification P3P (Platform for Privacy Preferences)

P3P (*Platform for Privacy Preferences*) est un standard du consortium W3C (en 16 avril 2002 P3P1.0 devient une recommandation du W3C) [CLM 02b], permet aux sites Web d'exprimer leurs politiques de confidentialité dans un format normalisé en matière de protection des données personnelles.

3.3.3.1. Le fonctionnement de P3P

La spécification P3P1.0 définit la syntaxe et la sémantique des politiques de confidentialité et les mécanismes permettant d'associer ces politiques aux ressources Web. Les politiques P3P consistent en déclarations utilisant le vocabulaire P3P afin d'exprimer des pratiques touchant à la confidentialité des données. Les politiques P3P appellent également des éléments du schéma de données de base, un jeu standard d'éléments de données que tout agent utilisateur P3P devrait reconnaître.

La spécification P3P définit [CLM 02b]:

- Un schéma standard des données qu'un site Web est susceptible de collecter, appelé schéma de données de base P3P ;
- Un jeu standard d'usages, de destinataires, de catégories de données et d'autres divulgations concernant la politique de confidentialité ;
- Un format XML pour exprimer une politique de confidentialité ;
- Un moyen permettant d'associer des politiques de confidentialité aux pages Web, ou aux sites Web, et aux cookies ;
- Un mécanisme de transport des politiques P3P via le protocole HTTP.

Le but de P3P version 1.0 est double. Premièrement, permettre aux sites Web d'annoncer leurs pratiques de collecte de données de manière normalisée, lisible par une machine et facilement disponible. Deuxièmement, permettre aux utilisateurs du Web de savoir quelles données seront collectées par les sites visités, comment ces données seront utilisées, et quels usages de ces données ces utilisateurs accepteront ou bien refuseront.

Dans la suite, nous examinons brièvement le noyau qui caractérise le langage des politiques P3P et aussi bien d'APPEL. Voyez [CLM 02b] et [CLM 02a] pour compléter les caractéristiques du langage des politiques P3P et APPEL respectivement.

3.3.3.2. Langage de politique P3P

Les politiques P3P sont décrites dans un format XML comme une séquence de déclaration d'éléments qui incluent les sous éléments suivants [AKR 03]:

- **Purpose** (Objectif) : décrive l'objectif pour lequel l'information est collectée. Plusieurs objectifs peuvent être listés dans une déclaration d'élément, si les objectifs ont les mêmes valeurs pour le destinataire, les données de groupes et rétention, ils sont définis dans une déclaration d'élément différente.
- **Recipient** (Destinataire) : décrit les utilisateurs destinés pour les informations rassemblées. Des destinataires multiples peuvent être spécifiés dans une déclaration.
- **Retention** (Rétention): définie la durée pour laquelle l'information collectée sera sauvegardée.
- **Data-group** (Donnée de groupe) : fournie la liste des données individuelles (qui sont spécifié en utilisant la balise DATA) qui sont collectées pour l'objectif dans l'élément.
- **Consequence** (Conséquence) : La déclaration d'un éléments peut, en option, contenir un élément <CONSEQUENCE> dont la présentation permettra à un utilisateur humain d'obtenir plus informations concernant les pratiques d'un site.

P3P à des valeurs prédéfinit, pour le **purpose** (12 choix), **recipient** (6 choix) et **retention** (5 choix) [CLM 02b].

L'élément purpose doit contenir l'une ou plusieurs des intentions suivantes :

- Current : l'achèvement et le support d'activité pour laquelle on a fourni des données.
- Admin : Administration du site Web et du système. C'est le cas du traitement des informations concernant les comptes hébergés et des informations utilisées pour la sécurisation et la maintenance du site et pour la vérification de l'activité du site Web par le site ou par ses agents.
- Pseudo-analysis : déterminer les habitudes, les intérêts, et d'autres caractéristiques de l'individus, mais ne pas identifier l'individus spécifiques.
- Contact : contactant les visiteurs pour vendre des services ou produits à travers une voie de communication autre que la voix téléphonique.

P3P a aussi prédéfini des types d'éléments de données. C'est aussi possible d'assigner des catégories aux éléments de données. Une politique peut fournir opt-in ou opt-out valeurs pour l'attribut requis des éléments purpose et recipient. La valeur opt-in dit que l'utilisateur doit fournir le consentement explicite au purpose/recipient énoncé. La valeur opt-out donne à l'utilisateur la souplesse pour rejeter le purpose/recipient spécifié.

Exemple d'une politique

Volga est une libraire qui a besoin d'obtenir certaine information personnelle minimum pour compléter une transaction d'achat. Cette information inclut le nom, l'adresse d'expédition, et le numéro de la carte de crédit. Volga utilise aussi l'achat historique des clients pour offrir des recommandations des livres pour lesquelles il a besoin de l'adresse émail des clients. La figure 32 représente comment la politique de Volga peut être définie dans le langage de politique P3P.

```
<POLICY>
  ... ...
 <STATEMENT>
    <PURPOSE><current/></PURPOSE>
    <RECIPIENT><ours/><same/></RECIPIENT>
    <RETENTION><stated-purpose/></RETENTION>
    <DATA-GROUP>
      <DATA ref="#user.name"/>
      <DATA ref="#user.home-info.postal/>
      <DATA ref="#dynamic.miscdata">
        <CATEGORIES><purchase/></CATEGORIES>
      </DATA>
    </DATA-GROUP>
 </STATEMENT>
 <STATEMENT>
   <PURPOSE>
   <individual-decision required="opt-in"/>
   <contact required="opt-in"/>
   </PURPOSE>
   <RECIPIENT><ours/></RECIPIENT>
   <RETENTION><business-practices/></RETENTION>
   <DATA-GROUP>
     <DATA ref="#user.home-info.online.email/>
     <DATA ref="#dynamic.miscdata">
       <CATEGORIES><purchase/></CATEGORIES>
     </DATA>
   </DATA-GROUP>
 </STATEMENT>
</POLICY>
```

Figure 32. Exemple d'une politique privée de dans P3P.

Le premier statement dit que le nom, l'adresse postale et les diverses données de l'achat (i.e., titres du livre, numéro de la carte de crédit, etc.) seront utilisés pour compléter la transaction de l'achat courante. Le deuxième statement permet au Volga d'utiliser les diverses données d'achat pour créer des recommandations personnalisées et leurs émails pour les clients. Cependant, la valeur opt-in de l'attribut *required* des objectifs "individual-decision" et "contact" impliquent que le consentement du client explicite est nécessaire [AKR03].

3.3.3.3. Langage de préférences APPEL

APPEL est un dialecte XML qui sert à décrire les préférences confidentialité de l'utilisateur. Il complète P3P. En fait, utilisant les namespaces XML dans un document APPEL, les balises définies par P3P sont entrelacées lourdement avec ceux définis par APPEL. Les préférences de la confidentialité sont exprimées dans APPEL comme une liste de règles. Ces règles sont opposées à une politique dans l'ordre dans qui ils paraissent. Une règle consiste en deux parties:

- Rule *behavior (Rule head)* : Spécifie l'action à être pris si la règle est satisfaite. Le comportement peut être *request* : impliquant que la politique se conforme aux préférences indiquées dans le corps de la règle. Il peut être *block* : impliquant que la politique ne respecte pas les préférences d'utilisateur.
- *Rule body* : Fournit le modèle qui est correspondu contre une politique. Le format d'un modèle suit la structure de XML utilisée dans la spécification des politiques décrite précédemment.

Une règle d'APPEL est satisfaite avec la correspondance de ses expressions et (récursivement) leurs sous expressions. Chaque expression APPEL a un attribut *connective* qui définit les opérateurs logiques entre ses sous expressions. L'attribut *connective* peut être : *or*, *and*, *non-ou*, *non-et*, *or-exact* et *and-exact*. La valeur par défaut de l'attribut *connective* est *and*.

Deux connectives inhabituels sont and-exact et or-exact, dont la sémantique est comme suit :

- *and-exact* : la correspondance est réussie si (a) toutes les expressions contenues peuvent être trouvées dans la politique et (b) la politique contient seulement des éléments inscrits dans la règle. Pour le connectif *and*, seulement la partie (a) doit être satisfaite, pas la partie (b).
- *or-exact* : la correspondance est réussie si (a) une où plus des expressions contenues peuvent être trouvées dans la politique et (b) la politique contient seulement des éléments inscrits dans la règle. Pour le connectif *or*, seulement la partie (a) doit être satisfaite, pas la partie (b).

Exemple d'une préférence APPEL

Jane est un consommateur conscient de la confidentialité qui veut des détailles sur l'utilisation de ses informations personnelles seulement pour compléter ses transactions d'achat. En même temps, Jane veut des recommandations individualisées envoyées par courrier électronique et n'empêche pas que son historique d'achat soit utilisé à cette fin. Cependant, elle veut la possibilité d'opt-in ou d'opt-out et ne veut pas sortir ses informations à un service qui n'offre pas ce choix.

```
<appel:RULESET>
  <appel:RULE behavior="block">
    <POLICY>
      <STATEMENT>
        <PURPOSE appel:connective="or">
    <admin/><develop/><tailoring/>
    <pseudo-analysis/><pseudo-decision/>
          <individual-analysis/>
          <individual-decision required="always"/>
          <contact required="always"/>
          <historical/><telemarketing/>
          <other-purpose/><extension/>
        </PURPOSE>
      </STATEMENT>
    </POLICY>
  </appel:RULE>
<appel:RULE behavior="block">
   <POLICY>
     <STATEMENT>
       <RECIPIENT appel:connective="or">
         <delivery/><other-recipient/>
         <unrelated/><public/><extension/>
       </RECIPIENT>
     </STATEMENT>
   </POLICY>
 </appel:RULE>
  <appel:RULE behavior="request"/>
  <appel:OTHERWISE/>
  </appel:RULE>
</appel:RULESET>
```

Figure 33. Exemple d'une préférence de confidentialité dans APPEL.

La figure 33 montre les préférences de Jane, elle inclut de deux règles. La première règle bloque tous les objectifs d'autres que le courant, mais laisse la possibilité d'opt-in ou d'opt-out des objectifs *individual-decision* et *contact*. La deuxième règle assure que les seuls destinataires possibles des données de Jane sont le service ou ses agents suivant les mêmes pratiques de confidentialité. La troisième règle permet aux données d'être sorti si les deux premières règles ne sont pas satisfaites. On observe, que la politique de Volga conforme aux préférences de Jane. La première règle dans les préférences de Jane n'est pas satisfaite parce que l'ensemble de buts dans la politique de Volga ne croise pas avec l'ensemble correspondant dans la règle de Jane, à l'exception de *contact* et *individuel-décision*. De la même façon, la deuxième règle dans les préférences de Jane n'est pas satisfaite parce qu'aucun des destinataires, dans la règle de Jane, est présent dans la politique de Volga.

Bien que le protocole P3P fournisse le moyen technique permettant aux utilisateurs d'être informés des politiques de confidentialité avant de confier des renseignements personnels, il n'offre aucun mécanisme technique qui garantisse le comportement des sites conformément à leurs politiques. Les produits qui mettent en œuvre cette spécification peuvent fournir une aide dans ce sens, mais cela n'est pas traité par cette

spécification. En outre, le protocole P3P ne comprend aucun mécanisme de transport des données ou de sécurisation des données personnelles en transit ou stockées.

3.4. Conclusion

La composition de services permet de combiner des services Web élémentaires afin d'obtenir des services plus élaborés. Elle décrit un ensemble d'interactions ou processus métier, faisant intervenir différents services Web. La composition est décrite indépendamment de sa implémentation future i.e. elle indique uniquement les types de services Web nécessaires mais ne précise pas nominativement les services Web qui seront utilisés. Par ailleurs, la composition peut comporter plusieurs niveaux en permettant à des services Web élaborés d'être à leur tour combinés pour construire de nouveaux services.

Pour les utilisateurs, fournir des données privées à des services Web en ligne n'est pas facile vue la vulnérabilité d'Internet. Plusieurs travaux (standards et spécifications) sont mis en place pour assurer la sécurité au sein d'une architecture basée sur les services Web qui permettent aussi aux utilisateurs d'exprimer leurs préférences de confidentialités. Dans ce chapitre nous avons détaillé quelques langages et spécifications qui permettent la composition des services et la confidentialité des données privées des utilisateurs.

Un utilisateur définit ses préférences privées qui désignent la manière ou ses données privées doivent être exploitées par les services Web, et un service Web définit sa politique privée qui est la façon, avec laquelle, les données privées des utilisateurs seront utilisées. Mais dans un environnement dynamique ou les services Web évoluent et se changent tout le temps, pour une requête de l'utilisateur, la combinaison qui satisfait cette requête doit être calculée dynamiquement, i.e. on doit chercher une composition de services parmi ceux qui existent qui satisfait cette requête qui respecte dans toute l'exécution du processus ses préférences. Dans telle composition, il y a des services qui participent directement dans cette composition, i.e. sont impliqués par la requête de l'utilisateur, et d'autres servies tiers qui sont invoqués par les services participants.

Dans le chapitre suivant, nous allons proposer une approche permettant la composition des services Web pour satisfaire les requêtes des utilisateurs. La composition doit garantir la confidentialité des données privée, définie par des préférences de confidentialités au niveau de la requête, par tous les services candidats.

Chapitre 4. Une approche basée coordination, médiation et politiques privées pour la composition des services Web.

4.1. Introduction

Dans ce travail, l'approche de composition est basée sur des services Web capables de communiquer via l'échange des messages, ces services sont modélisés par des automates d'état fini où les transitions sont définies par un envoie d'un message, réception d'un message ou exécution d'une opération. Une requête est exécutée grâce à la coordination des services, si la coordination ne satisfait pas la composition attendue (la requête), nous synthétisons un nouveau service appelé médiateur. Ce dernier a comme rôle de générer l'ensemble des messages manquants qui sont nécessaires pour compléter la composition.

Dans la suite de ce chapitre, nous allons essayer de répondre aux questions suivante : (1) où et comment intégrer les politiques privées dans les services Web ? (2) comment combiner les services pour satisfaire une requête (*goal service*), en se basant sur les politiques privées des services ? (3) dans quel cas et comment générer le médiateur si un inter-blocage est survenu ?

4.2. Les politiques privées

Une méthode effective et flexible pour protéger la confidentialité des données privées est d'utiliser les politiques privées. Le consommateur définie ses préférences de confidentialité sous forme d'une préférence et le fournisseur définie aussi ses pratiques sur les données privées sous forme d'une politique, lors de l'interaction du consommateur avec le fournisseur leurs politiques seront utilisées pour assurer la sécurité des données privées. Pour cela, nous allons adapter un modèle des politiques privées et préconiser une négociation pour permettre une exécution sensible aux préférences de confidentialité du consommateur.

4.2.1. Le modèle des politiques privés

Le modèle des politiques privées proposé dans [GBC07] qui est une extension des catégories des règles définies dans la plate forme des préférences de confidentialité P3P, permet une interaction entre le client et le fournisseur. Le client spécifie à travers des règles, appelées les préférences de confidentialité (*privacy preferences*), la façon avec laquelle les données privées peuvent être utilisées par le fournisseur. Et le fournisseur spécifie à travers des règles, appelées politiques de confidentialité (*privacy policy*), comment les données privées seront utilisées.

Dans ce modèle, une politique privée est une règle spécifiée par le fournisseur, elle décrive la façon d'utilisation des données privées du client. Comme représenté dans la figure 34, une politique privée est définie comme un ensemble fini du terme d'utilisation de donnée (*terms of data usage*) noté par *DTU*, une *tdu* consiste aux données privées et aux objectifs pour lesquels ces données doivent être collectées.

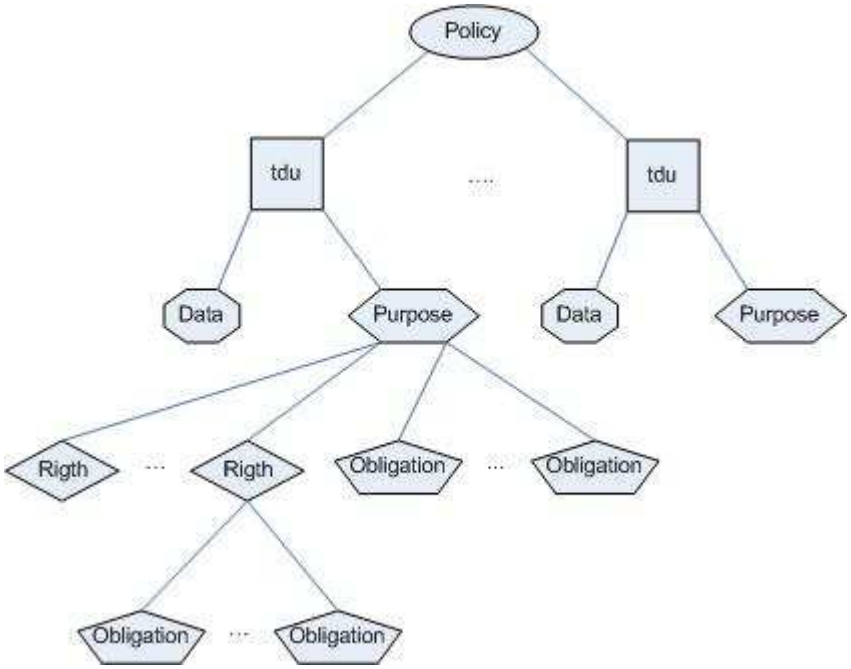


Figure 34. Le modèle des politiques privées [GBC07].

- Un objectif (*purpose*) est une action représente le besoin d'un client, il est exécuté par une entité donnée. L'entité utilise les données du client pour satisfaire la requête (*purpose*) dans un délai donnée. Un purpose implique deux types d'actions : les *rigths* (droits) et les *obligation*s (obligations).
- *Rigths* : un *rigth* est une action où le fournisseur est autorisé à l'exécuter. Pour chaque *rigth*, on spécifie : l'entité autorisée à l'exécuter, le délai durant lequel l'entité possède ce *rigth* et le délai pendant lequel le *rigth* doit être accompli une fois activé. Aussi, un *rigth* peut contenir un ensemble d'*obligations*.
- *Obligation*s : une *obligation* est une action où le fournisseur doit l'accomplir après la collection des données privées pour assurer leurs sécurités. Une *obligation* est spécifiée comme un *rigth*, mais elle n'implique pas des actions.

Pour établir une conversation entre le client et le fournisseur, les préférences du client doivent consister avec la politique du fournisseur. Une préférence consiste avec une politique, si la politique est plus restrictive que la préférence. Nous définissons les ensembles suivants qui permettent la comparaison entre les politiques :

U : l'ensemble des entités ; D : l'ensemble des données ; A : l'ensemble des actions ; P : l'ensemble des objectifs (*Purpose*) ; O : l'ensemble des obligations ; R : l'ensemble des droits (*Rigths*) ; I : l'ensemble des intervalles. U, D, A sont représentés comme une hiérarchie avec une ontologie utilisé pour comparer le niveau de restriction de deux politiques. A partir de ces ensembles, nous pouvons écrire un *tdu, purpose*, *rigth*, et *obligation* sous forme des tuples comme suit :

- une *politique* privée est un ensemble des termes d'utilisation de données *tdu*, ou *tdu* est un élément de D X P.
- un *purpose p* est définit avec le tuple $(a, u, \mu, S^R, S^O) \in A \times U \times I \times 2^R \times 2^O$, où *a* est l'action qui identifie le *purpose*, *u* est l'entité qui va exécuter le *purpose*, μ est l'intervalle dans lequel *u* doit exécuter *a*, S^R est l'ensemble de *rigths* et S^O est l'ensemble des *obligations* associées avec le *purpose*.
- un *rigth r* est définit avec un tuple $(a', u', v', \mu', S^O) \in A \times U \times I \times I \times 2^O$ où *a'* est l'action qui identifie le *rigth r*, *u'* est l'entité autorisé à exécuter l'action *a'*, *v'* est le délai dans lequel l'entité *u'* est autorisé à exécuter l'action *a'*, μ' est le délai dans lequel l'action *a'* doit être accomplit une foi activé.
- une *obligation o* est définit par le tuple $(a'', u'', v'', \mu'') \in A \times U \times I \times I$ tel que *a''* est une action qui identifie l'*obligation o*, *u''* est l'entité qui va exécuter *o*, *v''* est l'intervalle du temps de validité pour l'action *a''*, μ'' est l'intervalle de temps dans lequel l'action *a''* doit être exécutée une fois activé.

Puisque le fournisseur peut divulguer les données collectées à un tiers, nous distinguons entre deux types d'entité :

- *ours* : pour spécifier les entités du service qui collecte les données privées ;
- *others* : pour spécifier les entités du service tiers pour les quelles le service peut divulguer les données privées collectées.

4.2.2. Les préférences

Les préférences sont définies de deux façons : des préférences privées locales que le client spécifie par rapport à un fournisseur pour désigner comment il souhaite que ce dernier utilise ses données privées. En outre, le fournisseur, vu comme un client, peut envoyer les données privées collectées à un service tiers, donc, il doit spécifier à travers des règles appelées les préférences externes comment le tiers doit utiliser les données privées (en respectant les préférences locales du fournisseur et aussi les préférences du client qui est l'origine de données privées). Les préférences locales et externes constituent les préférences du client.

Par ce que le service fournisseur, dans son exécution, peut aussi invoquer un autre service Web (il devient un client), donc, il peut aussi faire suivre les données collectées du client pour un service tiers. Comme représenté sur le schéma suivant, nous pouvons avoir des invocations imbriquées pour la même requête du client.

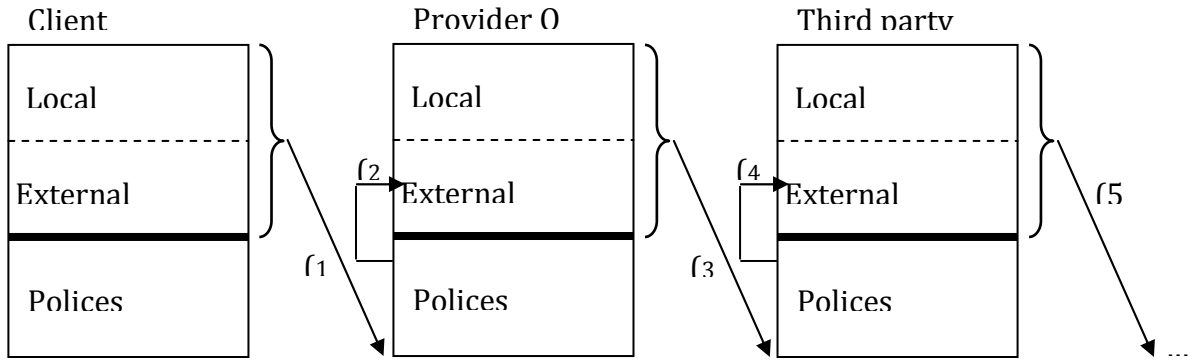


Figure 35. Extraction des préférences externes à partir des politiques [GBC07].

L'interaction, entre le service Q et le tiers, est basée sur les préférences du service Q et les politiques du tiers. Pour informer un tiers sur les restrictions du client à travers le service Q, nous ajoutons les restrictions du service Q dans ses préférences externes (2) comme représenté sur le schéma. Donc, nous propageons l'ensemble de *tdu* de la politique du service Q sur les préférences externes (2). Par conséquent, nous ajoutons chacune de *tdu* de la politique à l'ensemble *TDU* des préférences quand l'entité responsable pour accomplir l'objectif (droit, obligation respectivement) est une entité externe (*others)*.

4.2.3. Exemple d'une politique privée

Nous considérons dans cet exemple un service de réservation d'hôtel, le service suggère une liste d'hôtels pour chaque client pour spécifier une destination désirable (la ville). Une fois le client a fait le choix, il est demandé de fournir un numéro de carte de crédit CCN pour compléter la réservation. Puisque ces données sont très sensibles, le client pourrait demander une sorte de questions concernant l'utilisation confidentielle de ses informations, donc, il doit spécifier sous forme d'une préférence ses restrictions sur le CCN, pour pouvoir comparer sa préférence à la façon avec laquelle le service de réservation traite les données privées qui est exprimée sous forme d'une politique.

Les restrictions, par rapport au service de réservation, sur le CCN sont les suivantes : le service technique collecte le numéro de la carte de crédit CCN pour payer la réservation de l'hôtel correspondant (p1) dans 20 minutes après l'obtention du CCN. La banque qui est une entité externe possède le droit de vérifier la validité du CCN (r1) dans 15 minutes après la réception du CCN. Si le droit (r1) est déclenché, l'action correspondante doit être exécutée dans 5 minutes. Le service financier doit détruire le CCN (o1) dans 10 minutes après l'accomplissement de l'objectif. De plus, la banque doit crypter le CCN (o2) dans 60 minutes après sa réception. La destruction (o1) et le codage (o2) doivent être exécutés immédiatement après leur déclenchement, d'où, le délai est spécifié par un intervalle du temps vide [0,0].

Ainsi, la politique correspondante pour CCN peut être représentée comme suit :

$plcy = \{(CCN, p1)\}$ ou

$p1 = (Pay\,\mathrm{Re}\,servation, Ours : FinancialService, \mu : [0,20\,\mathrm{min}], \{r1\}, \{o1\})$

$r1 = (ValidityVerification, Others : Bank, v : [0,15\,\mathrm{min}], \mu : [0,5], \{o2\})$

$o1 = (Destruction, Ours : FinancialService, v : [t(p1), t(p1) + 10\,\mathrm{min}], \mu : [0,0])$ tel que :

$t(p1) \in [0,20\,\mathrm{min}]$ est l'intervalle du temps pour satisfaire le purpose.

$o2 = (Encrypt, Others : Bank, v : [0,60\,\mathrm{min}], \mu[0,0])$

4.2.4. La comparaison du niveau de restriction entre deux politiques

Pour comparer deux politiques, nous allons se baser sur leurs niveaux de restriction. Pour cela, plusieurs paramètres peuvent être impliqués, la figure suivante représente le premier paramètre qui est la partie des hiérarchies disponibles pour les *purposes,* les *utilisateurs* et les *données* : *hiérarchie des purposes, hiérarchie des utilisateurs* et *hiérarchie des données* qui sont utilisées pour comparer les données, les actions et les utilisateurs reliés aux objectifs (*rights* et *obligations* respectivement) comme expliqué dans l'exemple des politiques suivant.

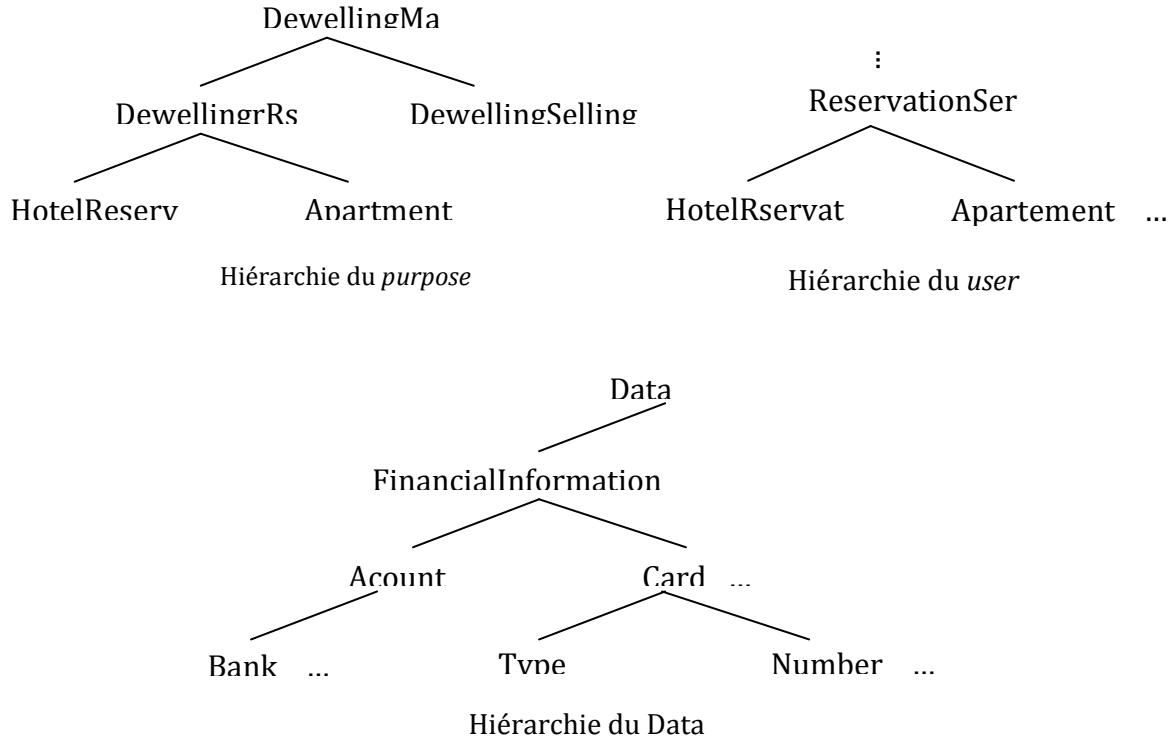


Figure 36. Exemple des hiérarchies.

Nous considérons les deux politiques *plcy1* et *plcy2* suivantes tel que :

$plcy1 = \{(FinancialInformation,\ p1)\} ou$:
$p1 = (\text{Re}\, servationDewelling, Ours : \text{Re}\, servationService, \mu : [0,30\ \text{min}], \{r1\}, \{o1\})$
$r1 = (ValidityVerification, Others : FinancialPartner, v : [0,15\ \text{min}], \mu : [0,5], \{o2\})$
$o1 = (Destruction, Ours : \text{Re}\, servationService, v : [t(p1), t(p1) + 10\ \text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\ \text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.
$o2 = (Encrypt, Others : FinancialPartner, v : [0,60\ \text{min}], \mu[0,0])$

$plcy2 = \{(NumberCard,\ p2)\}$ ou :
$p2 = (\text{Re}\, servationHotel, Ours : HotelService, \mu : [0,20\ \text{min}], \{r'1\}, \{o'1\})$
$r'1 = (ValidityVerification, Others : Bank, v : [0,15\ \text{min}], \mu : [0,5], \{o'2\})$
$o'1 = (Destruction, Ours : FinancialService, v : [t(p1), t(p1) + 10\ \text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\ \text{min}]$ est l'intervalle du temps pour satisfaire le purpose p2.
$o'2 = (Encrypt, Others : Bank, v : [0,60\ \text{min}], \mu[0,0])$

Pour l'instant, nous pouvons remarquer que la politique *plcy2* est plus restrictive que la politique *polcy1*, puisque les données privées considérées dans *plcy1* concernent les informations financières sont moins restrictives que les données considérées dans *plcy2*, comme illustrées dans la figure précédente (hiérarchie des data), les informations financières rassemblent le compte et le numéro de carte de crédit. Donc, les données spécifiées dans *plcy2* (numéro de carte de crédit) sont plus restrictives que les données spécifiées dans *plcy1* (les informations financières). De plus, le *purpose* spécifié dans la politique *plcy1* est *ReserveDwelling* qui rassemble la réservation de l'hôtel et de l'appartement (hiérarchie des purpose), il est moins restrictif que le *purpose p2* de la politique *plcy2*, puisque la réservation de l'hôtel est un sous objectif de *ReserveDwelling*. Dans la politique *plcy1,* le service de réservation doit exécuter *ReserveDwelling* dans 30 minutes après la réception des informations financières. L'intervalle spécifié dans la politique *plcy2* est inclus dans celui spécifié dans la politique *plcy1*, donc, l'intervalle du *plcy2* est plus restrictif que l'intervalle du *plcy1*.

La comparaison du niveau de restriction entre les politiques implique les autres éléments qui composent la politique qui sont : *les rights, les obligations et les purposes*.

4.2.4.1. La comparaison des obligations $\leq_O$

Une *obligation* $o2 = (a''_2, u''_2, v''_2, \mu''_2)$ est plus restrictive que l'*obligation* $o1 = (a''_1, u''_1, v''_1, \mu''_1)$ notée $o1 \leq_o o2$ si et seulement si : $a''_2 \subseteq a''_1, u''_2 \subseteq u''_1$, $v''_2 \subseteq v''_1$, $\mu''_2 \subseteq \mu''_1$.

Parce que un *rigth* implique un ensemble d'*obligations*, la comparaison de ces derniers permettre la comparaison entre les *rigths*.

4.2.4.2. La comparaison des Rights $\leq_R$

Un *rigth* $r2=(a'_2, u'_2, v'_2, \mu'_2, S_2^{O'})$ est plus restrictive qu'un *rigth* $r1=(a'_1, u'_1, v'_1, \mu_1, S_1^{O'})$ noté par $r1 \leq_R r2$ si et seulement si : $a'_2 \subseteq a'_1$, $u'_2 \subseteq u'_1$, $v'_2 \subseteq v'_1$, $\mu'_2 \subseteq \mu'_1$, $\forall o'_1 \in S_1^{O'}, \exists o'_2 \in S_2^{O'}$ tel que $o'_1 \leq_O o'_2$.

Comme un *purpose* peut contenir des ensembles de *rights* et d'*obligations*, les comparaisons au-dessus permettent la comparaison entre des *purpose*.

4.2.4.3. La comparaison des purposes $\leq_P$

Un *purpose* $p2=(a_2, u_2, v_2, S_2^R, S_2^O)$ est plus restrictive que le *purpose* $p1=(a_1, u_1, v_1, S_1^R, S_1^O)$ noté par $p1 \leq_P p2$ si et seulement si : $a_2 \subseteq a_1$, $u_2 \subseteq u_1$, $v_2 \subseteq v_1$, $\forall r_2 \in S_2^R, \exists r_1 \in S_1^R$ tel que $r_1 \leq_R r_2$, $\forall o_1 \in S_1^O, \exists o_2 \in S_2^O$ tel que $o_1 \leq_O o_2$.

Un terme d'utilisation des données (*tdu*) définis un objectif pour lequel les données privées sont collectées, la comparaison précédente des *purposes* permette la comparaison suivante des *tdu*.

4.2.4.4. La comparaison des termes d'utilisation des données (tdu) $\leq_{TDU}$

Un terme d'utilisation de donnée $tdu_2=(d_2, p_2)$ est plus restrictif que le terme d'utilisation de donnée $tdu_1=(d_1, p_1)$ noté par $tdu_1 \leq_{TDU} tdu_2$ si et seulement si : $d_2 \subseteq d_1 \wedge p_1 \leq_P p_2$.

Une politique privée (respectivement une préférence) est définie par un ensemble fini de *tdu*, les comparaisons précédentes des *tdu* permettent la comparaison entre les politiques privées.

4.2.4.5. La comparaison des politiques $\leq_{\text{Private}}$

Une politique *plcy2* est plus restrictive qu'une politique *plcy1* noté par : $plcy2 \leq_{\text{Private}} plcy1$ si et seulement si $\forall tdu_1=(d1, p1) \in plcy1, \exists tdu_2=(d2, p2) \in plcy2$ tel que $tdu_1 \leq tdu_2$.

Une préférence *pref* consiste avec une politique *plcy* si et seulement si *plcy* est plus restrictive que la préférence *pref* ($pref \leq_{\text{Private}} plcy$). Puisque une préférence est définit, comme une politique, avec un ensemble fini des *tdu*, nous pouvons utiliser la comparaison entre les politiques, pour comparer entre *pref* et *plcy*.

4.2.4.6. Exemple de comparaison entre deux politiques

Prenant l'exemple défini en (4.2.3), nous redéfinissons les politiques privées *plcy1* et *plcy2* pour les deux services : R*eservationService* et *HotelService* comme suit :

$plcy1 = \{(FinancialInfo,\ p1)\}$ ou :
$p1 = (\text{Re}\, servationDewelling, Ours : \text{Re}\, servationService, \mu : [0,30\ \text{min}], \{r1\}, \{o1\})$
$r1 = (ValidityVerification, Others : FinancialPartner, v : [0,15\ \text{min}], \mu : [0,5], \{o2\})$
$o1 = (Destruction, Ours : \text{Re}\, servationService, v : [t(p1), t(p1) + 10\ \text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,30\ \text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.
$o2 = (Encrypt, Others : FinancialPartner, v : [0,60\ \text{min}], \mu[0,0])$

$plcy2 = \{(NumberCard,\ p'1), (NumberCard,\ p'2)\}$ ou :
$p'1 = (\text{Re}\, servationHotel, Ours : HotelService, \mu : [0,20\ \text{min}], \{r'1\}, \{o'1\})$
$r'1 = (ValidityVerification, Others : Bank, v : [0,15\ \text{min}], \mu : [0,5], \{o'2\})$
$o'1 = (Destruction, Ours : FinancialService, v : [t(p1), t(p1) + 10\ \text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\ \text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.
$o'2 = (Encrypt, Others : Bank, v : [0,60\ \text{min}], \mu[0,0])$

$p'2 = (Car\ \text{Re}\, servation, Ours : TransportServices, \mu : [0,20\ \text{min}], \{r'2\}, \{o'3\})$
$r'2 = (ValidityVerification, Others : Bank, v : [0,15\ \text{min}], \mu : [0,5], \{o'4\})$
$o'3 = (Destruction, Ours : TransportServices, v : [t(p2), t(p2) + 10\ \text{min}], \mu : [0,0])$ tel que :
$t(p2) \in [0,20\ \text{min}]$ est l'intervalle du temps pour satisfaire le purpose p2.
$o'4 = (Encrypt, Others : Bank, v : [0,60\ \text{min}], \mu[0,0])$

Nous remarquons dans l'exemple que pour le deuxième service (*plcy2)*, le numéro de carte de crédit (CCN) peut être utilisé pour réserver une voiture, c-à-dire, la réservation d'un hôtel oblige une réservation d'une voiture, ce qui n'est pas autorisé dans la politique *plcy1*. Donc la politique *plcy2* viole la politique *plcy1*.

4.3. Une composition basée sur les politiques privées des services Web

Maintenant, nous faisons face au problème de la composition automatique des services Web. Puisque chaque service Web à des politiques et des préférences, la composition (combinaison) doit préserver la confidentialité des données privées provenant des différents services qui participent à cette composition. Pour cela, en se basant, bien sûrs, sur les préférences et les politiques de chaque service. La composition des services Web a pour objectif de satisfaire une requête complexe qui implique l'exécution de plusieurs services Web. Une requête est une composition attendue et le problème consiste à trouver (calculer) une composition (collaboration) qui satisfait cette requête. Donc, à partir de l'ensemble des services des partenaires et la description abstraite de

l'application attendue (requête du client), nous visons à fournir une composition exécutable qui implémente l'application abstraite en respectant les données privées du client ainsi que les préférences de tous les services participants.

4.3.1. Une approche basée coordination, médiation et politiques pour la composition des services Web

Nous allons intégrer le modèle des politiques privées détaillé au début de ce chapitre pour arriver à une compositions automatique des services Web qui prend en compte la confidentialité des données en utilisant les politiques privées.

L'objectif de cette approche de composition est de satisfaire une description d'une application abstraite appelée *goal service*. Lors de la construction d'une combinaison, conforme avec ce *goal service*, à l'aide des services disponibles, on peut tomber sur une trace de composition bloquée, i.e. une trace ou un message est attendu mais il n'est jamais envoyé (inter-blocage). Pour prévenir ce cas la, on va essayer de générer le médiateur qui produit les messages manquants, donc, le rôle du médiateur est de maître possible la conversation entre les services dans le cas d'un inetr-blocage.

En résumé, cette approche est basée **coordination** : les services s'échangent des messages entre eux pour qu'ils puissent communiquer (chorégraphie), **médiation** : lorsque un service ne peut pas communiquer avec un autre (message perdu ou incompatibilité des signatures), le médiateur intervient pour compléter les messages manquants et continuer la composition (orchestration). Et **politiques privées** : lors de la composition, nous vérifions que le service à invoquer respecte la préférence de confidentialité du client ainsi que la préférence du service invoquant.

4.3.2. Le modèle des services Web

Le langage OWL-S décrive un modèle de processus pour les services Web en spécifiant les séquences du processus atomiques qui peuvent être exécutés. L'ontologie OWL-S est définie sous trois sous-ontologies qui décrivent respectivement le profil (ce que le service fait), le modèle de processus (comment le service fait) et l'accès (comment accéder au service). Ce dernier définit la structure des messages et les liaisons physiques sur WSDL, par contre, le modèle du processus décrive le processus atomique que le service peut exécuter et dans lequel il change ses états. Nous définissons un automate d'état fini (*finite state machine*) pour chaque service, qui peut être extrait à partir du modèle de processus de l'ontologie OWL-S.

```
<AtomicProcess ID=checkSSN>
   <hasInput>
     <Input ID=ssn>
        <parameterType resource=#SecurityNumber>
     </Input>
   </hasInput>
   <hasOutput>
      <Output ID=adr>
        <parameterType resource=#Adress>
      </Output>
   </hasOutput>
   <hasOutput>
      <Output ID=res>
        <parameterType resource=#Validity>
      </Output>
   < /hasOutput>
     ...
```

Figure 37. La partie de la spécification OWL-S de l'opération atomique *checkSSN*.

Pour cela, un processus atomique est correspond à une opération dans le WSDL, il est définie par les entrées, les pré-conditions et les résultats. La figure 37 illustre une partie d'une description OWL-S d'un processus atomique *CheckSSN*, qui est spécifié par une entrée (*SSN*) et les résultats (*adr,res*).

Un service Web est représenté par un automate d'état fini dont l'alphabet est un ensemble d'actions et de messages. Un message peut être un message envoyé noté par *!m*, ou bien, un message reçu noté par *?m*, il est constitué d'une liste de paramètres notée par $m(d_1,...,d_n)$ où $m(\bar{d})$. Une action *a* représente une opération, elle contient une liste de paramètres en entrée $d_1,...,d_m$ et une liste de paramètres en sortie $d'_1,...,d'_m$ est notée par $a(d_1,...,d_m;d'_1,...,d'_m)$.

D'où, un service Web Q est un tuple (S,s_0,F,M,A,C,T) où S est l'ensemble des états, s_0 est l'état initial, F est l'ensemble des états finaux ($F \subseteq S$), M est l'ensemble des messages, A est l'ensemble des actions, C est l'ensemble des contraintes sur le service et T est l'ensemble des transitions tel que : $T \subseteq S \times (M^{?!} \cup A) \times C \times S$, où $M^{?!} = \{?m | m \in M\} \cup \{!m | m \in M\}$. Une transition (s,a,s') est notée par $s \xrightarrow{a} s'$.

Une trace est une séquence de plusieurs transitions, définie comme suit : $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 ... \xrightarrow{\alpha_{n-1}} s_n$. Nous définissons trois type de transitions, i.e. une action *a* peut être un envoi d'un message (*!m*), une réception d'un message (*?m*) ou bien une opération ($operation\ (d_1,...,d_m;d'_1,...,d'_m)$).

4.3.2.1. Exemple de modélisation d'un service Web par un FSM

Considérant le service de la sécurité sociale (*SS service*) dont le rôle est de vérifier la validité du numéro de la sécurité sociale d'une personne. Le *SS service* commence par

recevoir le message *checkSSN(ssn)* qui implique, comme donnée, le numéro de la sécurité sociale *ssn*. Puis, *SS service* vérifie la validité du *ssn* via l'opération *checkSSN(ssn ; adr,res)*, qui a comme entrée le *ssn* et comme sorties l'adresse (*adr*) et le résultat de la validité (*res*). Ensuite, le service envoie le message *checkSSN(ssn,adr,res)* qui représente le résultat du traitement du *SS Service*. L'automate d'état fini correspond au *SS service* est le suivant :

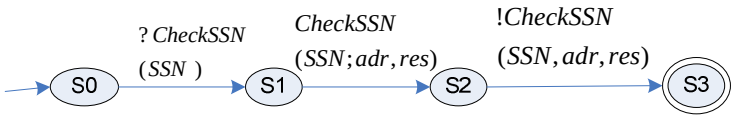


Figure 38. Le FSM du service SS service.

4.3.3. Une conversation basée sur les politiques privées entre les services Web

La conversation entre les services est nécessaire pour satisfaire les business protocoles qui sont devenus une partie importante du Web, car une application de business ne peut pas être satisfait par un seul service Web, mais par la composition de plusieurs services, telles applications impliquent une conversation entre les services en échangeant des données privées et confidentielles qui ne doivent pas être vulnérable pour n'importe quel service.

La conversation entre les services Web, que ça soit une chorégraphie (*peer to peer*), ou bien une orchestration (par médiateur), se fait par échange des messages. Un service Web invoque un autre service avec un envoi d'un message et récupère le résultat du traitement avec la réception d'un autre message généré par le service invoqué. Puisque, un service Web est représenté par un automate d'état fini, la seule action à effectuer par le service pour passer à l'état suivant à partir de son état initial est la réception d'un message (l'invocation d'un service se fait par l'envoi d'un message).

Pour préserver la confidentialité des données lors d'une conversation entre les services, nous allons utiliser les politiques privées, i.e., nous allons intégrer les politiques dans les messages échangés comme suit : chaque service, qui utilise des politiques et des préférences, doit mentionner ses politiques au niveau des réceptions des messages et ses préférences au niveau des envois des messages. Dans ce cas, si un service veut envoyer un message avec la préférence *pref* qui spécifie les préférences de confidentialité sur la donnée contenue dans le message, il peut assurer que sa préférence *pref* sera respectée et sa donnée ne sera pas violée par le service destinataire. En effet, un service client peut choisir, parmi un ensemble des services, celui qui va être invoqué (qui va recevoir son message) selon la politique privée de ce service. C'est-à-dire, le client peut récupérer les politiques privées appliquées par les services et les comparer avec sa préférence, donc, le service qui va être invoqué est celui qui a une politique plus restrictive que *pref*.

4.3.3.1. Exemple de conversation basée sur les politiques entre les services Web

Prenant le service de la sécurité social (*SS service*) présenté dans l'exemple précédent. Ce service doit spécifier sa politique au niveau de la réception du *SSN* et sa préférence au niveau de l'envoi du résultat. Donc, le *SS service* devient comme suit :

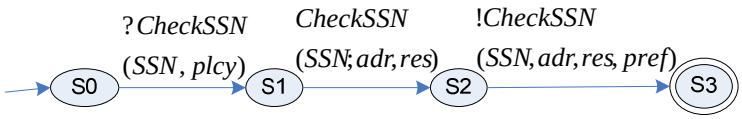


Figure 39. Le FSM avec politiques privées du service SS service.

Tel que *plcy* est :

$plcy = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (ValidityVerification, Ours : CheckSSN, \mu : [0,20\,\min], \{o1\})$
$o1 = (Destruction, Ours : CheckSSN, v : [t(p1), t(p1) + 10\,\min], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\min]$ est l'intervalle du temps pour satisfaire le purpose p1.

Et *pref* est la préférence du *SS service* sur les données qu'il envoie. Dans *pref,* on trouve deux parties : les préférences externes et les préférences locales.

Un service client qui veut vérifier la validité de son *SSN* peut utiliser le service *SS service* dans le cas ou sa préférence de confidentialité sur le *SSN* est moins restrictive que *plcy* qui est définie au niveau de la réception de *SSN* par la transition $S_0 \xrightarrow{?CheckSSN(SSN, plcy)} S_1$.

Dans le cas ou le *SS service* doit envoyer le résultat de son traitement à un autre service *SS service'* pour compléter la requête du client (la composition du *SS service* et *SS service'*), il doit vérifier la politique de *SS service'* avec sa préférence. Puisque, le *SSN* est une donnée privée du client (externe), donc, les préférences du service client doivent être aussi vérifiées. Pour cela, on ajoute la politique *plcy* du *SS service* à ses préférences externes. C-à-dire, à chaque envoi d'un message (d,$pref_d$) dans lequel on spécifie des préférences $pref_d$, si la donnée à envoyer d est externe (elle a été reçue par un message (d,$plcy_d$), on ajoute la politique $plcy_d$ aux préférences externes dans $pref_d$.

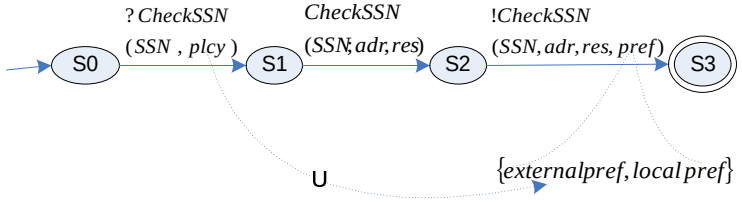


Figure 40. Extraction des préférences externes à partir des politiques.

4.4. Un algorithme pour la composition des services Web basée sur les politiques privées

Maintenant, nous introduisons les étapes principales de l'algorithme de composition des services Web basé sur les politiques privées. Notre point de départ est le *goal service* qui sera satisfait par la combinaison d'un ensemble de services. En se basant sur le *goal service*, nous construisons un cluster éligible qui rassemble tous les services pertinents (qui peuvent participer à la composition), une fois le cluster éligible est construit, nous procédons à la composition. Dans certains cas, la composition échoue parce qu'il existe des problèmes pour la conversation des services qui est basée sur l'échange de messages. Il y a deux raisons qui peuvent bloquer la conversation : (1) un service qui attend un message qui n'a pas été envoyé auparavant par un autre service du cluster, ou bien (2) le message attendu a été envoyé mais sa signature est incompatible, i.e., un service attends un message avec un type de données différent que le type de données contenus dans le message envoyé. Pour remédier à ces problèmes, nous générons le médiateur dont le rôle est de compléter les messages qui sont nécessaires pour connecter la composition des services.

4.4.1. Exemple de motivation

Nous allons considérer un exemple de collaboration en soins médicaux pour le quelle trois partenaires doivent collaborer ensemble : (1) l'hôpital, (2) la sécurité social et (3) la pharmacie. Les trois partenaires sont représentés respectivement par les services suivants : (1) le service de gestion de l'hôpital (*HM service*), (2) le service de sécurité social (*SS service*) et (3) le service de gestion de la pharmacie (*PM service*). Chaque service de chaque partenaire est décrit par son automate d'état fini. La requête pour laquelle on veut composer ces services est : le docteur peut demander l'antécédent du patient et/ou enregistrer les nouveaux diagnostiques. Si le docteur définit des nouveaux diagnostiques et prescrire des drogues pour le patient qui est identifié par son numéro de sécurité social *SSN*, le *PM service* doit être invoqué pour fournir les drogues au patient, dans ce cas, le patient peut s'inquiéter pour son *SSN*, puisque la vulnérabilité de ce dernier touche directement sa vie privée, alors il peut spécifier des contraintes (des préférences de confidentialité) qui seront respectées par les services qui vont recevoir son *SSN*. *HM service* et *PM service* exécutent leurs tâches si le numéro de la sécurité social est valide, pour cela, ils utilisent *SS service* pour vérifier la validité de *SSN*.

Vous pouvez remarquer que la satisfaction de telle application nécessite la combinaison du trois services. D'où le problème à résoudre est : comment combiner les services pour satisfaire un tel business application, en respectant les préférences de confidentialité imposées par patient sur son *SSN*.

Les trois services Web *HM service*, *SS* service, *PM service* peuvent être représentés par leurs automates d'état fini comme suit :

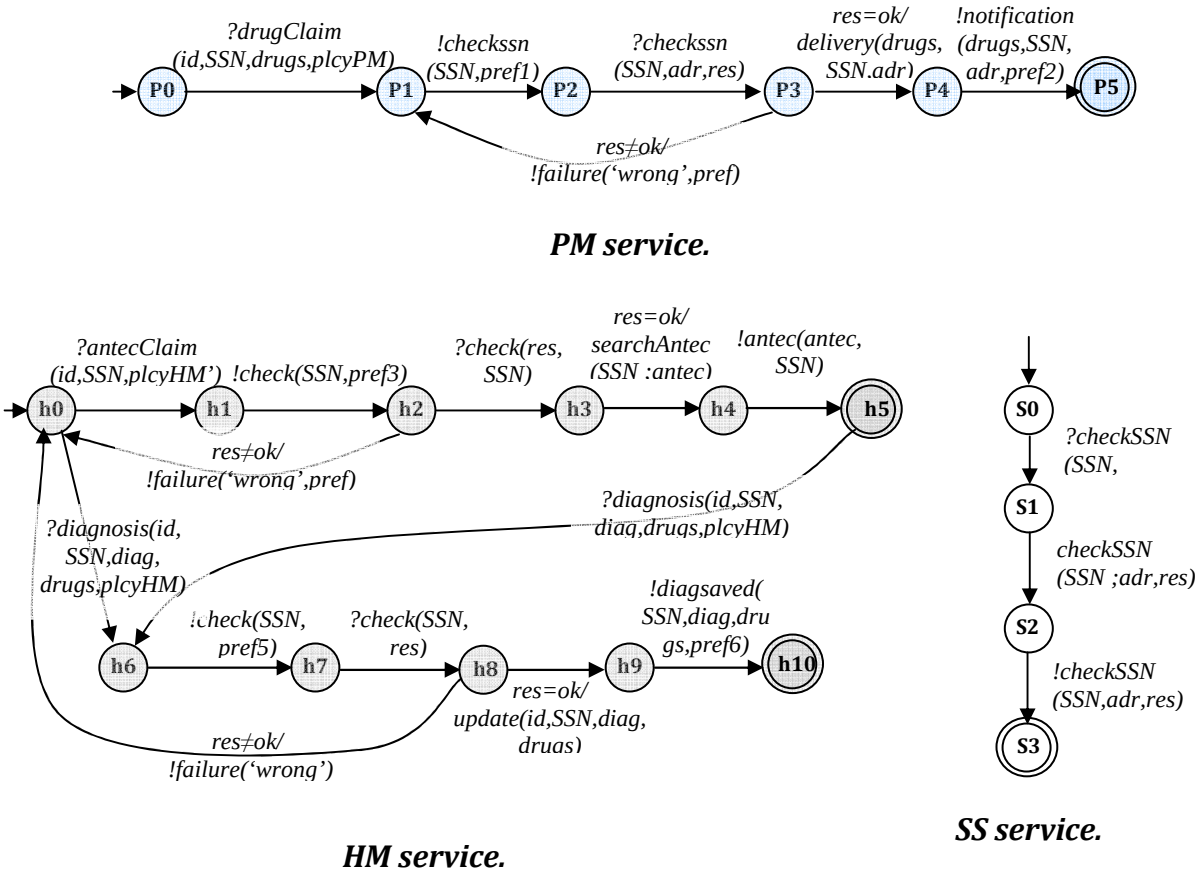


Figure 41. Les FSM des trois services SS service, HM service et PM service.

Tel que les politiques appliquées par le *HM Service* sur le *SSN* sont :

$plcy_{HM}' = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (SearchAntecedent, Ours : searchAntec, \mu : [0,20\ \min], \{r1\})$
$r1 = (ValidityVerification, Others : CheckSSN, \mu : [0,20\ \min], \{o1\})$
$o1 = (Destruction, Others : CheckSSN, v : [t(p1), t(p1) + 10\ \min], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\ \min]$ est l'intervalle du temps pour satisfaire le purpose p1.)

$plcy_{HM} = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (Upatediagnosis, Ours : Update, \mu : [0,20\ \min], \{r1\})$
$r1 = (ValidityVerification, Others : CheckSSN, \mu : [0,20\ \min], \{o1\})$
$o1 = (Destruction, Others : CheckSSN, v : [t(p2), t(p2) + 10\ \min], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\ \min]$ est l'intervalle du temps pour satisfaire le purpose p1. *)*

La politique de *SS Service* sur le *SSN* est :

$plcy_{SS} = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (ValidityVerification, Ours : CheckSSN, \mu : [0,20\,\min], \{o1\})$
$o1 = (Destruction, Ours : CheckSSN, v : [t(p1), t(p1) + 10\,\min], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\min]$ est l'intervalle du temps pour satisfaire le purpose p1.

La politique du *PM Service* sur le *SSN* est :

$plcy_{PM} = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (Delivery, Ours : delivery, \mu : [0,60\,\min], \{r1\})$
$r1 = (ValidityVerification, Others : CheckSSN, \mu : [0,20\,\min], \{o1\})$
$o1 = (Destruction, Ours : delivery, v : [t(p1), t(p1) + 10\,\min], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\min]$ est l'intervalle du temps pour satisfaire le purpose p1.

Les préférences qui sont intégrées aux envois de messages dans les trois services sont des préférences externes qui sont extraites à partir des politiques $plcy_{PM}$, $plcy_{HM}$ et $plcy_{SS}$. Par exemple, le *HM service* doit vérifier la validité du *SSN* qui est reçu avec la politique $plcy_{HM}$ en envoyant le message *!check(SSN, pref5)* au service de validation *SS service*, donc *pref5* est le *right r1* dans $plcy_{HM}$. I.e., *pref5* est :

$pref\,5 = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (ValidityVerification, CheckSSN, \mu : [0,20\,\min], \{o1\})$
$o1 = (Destruction, CheckSSN, v : [t(p2), t(p2) + 10\,\min], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\min]$ est l'intervalle du temps pour satisfaire le purpose p1. *)*

4.4.2. Le goal service

Le *goal service* représente la requête du client. Est une application abstraite qui définie une composition attendue des services. Il est représenté, comme un service, sous forme d'un automate d'état fini, dans lequel on trouve des envois des messages, des réceptions des messages et des opérations que le client demande. Les envois de messages dans le *goal service* représentent les messages que le client reçoit et les réceptions des messages représentent les messages que le client envoie. Contrairement aux services de la communauté, les politiques privées sont associés aux messages du *goal service* comme suit : associer les préférences aux messages entrants, car c'est le client qui va envoyer ces messages et associer les politiques aux messages sortants, car c'est le client qui va les recevoir.

4.4.2.1. Exemple du goal service

Continuant avec l'exemple de motivation, qui contient les trois services *SS service*, *HM service* et *PM service*. L'application abstraite (*goal service*), décrite pour l'exemple de motivation, que nous voulons satisfaire par la composition de ces trois services est définie par son automate d'état fini comme suit :

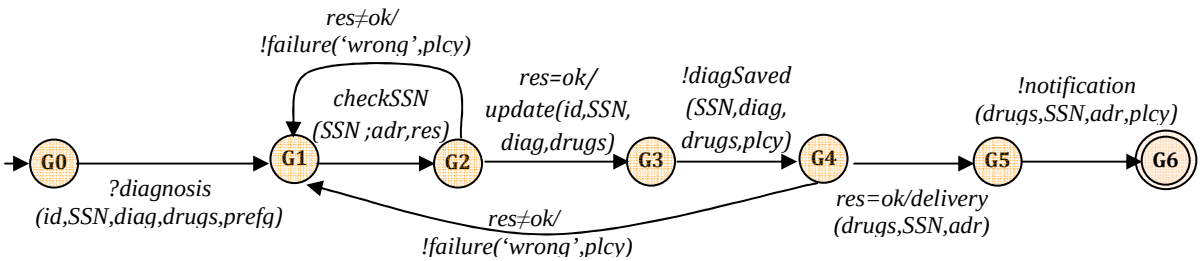


Figure 42. Exemple du goal service.

Tel que la préférence imposée sur le *SSN* dans cette requête est :

$pref_g = \{(SSN, p1), (SSN, p2), (SSN, p3)\}$ ou :
$p1 = (ValidityVerification, CheckSSN, \mu : [0,20\,\text{min}], \{o1\})$
$o1 = (Destruction, CheckSSN, v : [t(p1), t(p1) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.)
$p2 = (Upatediagnosis, Update, \mu : [0,20\,\text{min}], \{r2\})$
$r2 = (ValidityVerification, CheckSSN, \mu : [0,20\,\text{min}], \{o1\})$
$o1 = (Destruction, CheckSSN, v : [t(p2), t(p2) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p2) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p2.)
$p3 = (Delivery, delivery, \mu : [0,60\,\text{min}], \{r1\})$
$r1 = (ValidityVerification, Others : CheckSSN, \mu : [0,20\,\text{min}], \{o1\})$
$o1 = (Destruction, delivery, v : [t(p3), t(p3) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p3) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p3.

4.4.3. La construction du cluster éligible

Plusieurs services Web peuvent être impliqués dans une composition. A partir du *goal service,* nous pouvons définir un sous ensemble des services (cluster éligible) pour optimiser la coordination. Le cluster éligible regroupe les services qui sont capables d'exécuter ou moins une opération du *goal service.*

Pour le *goal service* $Q_g = (S_g, s_{0g}, F_g, M_g, A_g, C_g, T_g)$, un service Q appartient au cluster éligible tel que $Q = (S, s_0, F, M, A, C, T)$, si et seulement si : $A \cap A_g \neq \phi$.

4.4.4. La coordination des services

Une fois le cluster éligible est construit, on procède à la coordination des services dont le résultat peut être considéré comme un automate globale. Pour construire cet automate global, on introduit le concept de *configuration* qui représente l'état de l'automate global à un temps donnée. Une *configuration* définit l'évolution des états des services lors de l'interaction. Dans la configuration initiale tous les services sont dans leurs états initiaux, à partir d'une configuration donnée, l'automate global porte une nouvelle configuration quand il existe un service qui change son état par le déclenchement d'une transition. Pour construire les configurations qui satisfont le *goal service*, on introduit le concept de *transition acceptable*. Donc, pour une transition du *goal service*, une transition de l'automate global est dit acceptable si elle satisfait la transition du *goal service* (i.e. exécute la même action ou le même message avec des politiques privées compatibles), ou bien la transition est un envoi de message, ou bien la transition est une réception de message. Pour qu'une transition de réception d'un message soit acceptable, le message doit être envoyé par une acceptable transition, i.e., il existe une transition prédécesseur qui permet l'envoi de ce message, ou bien, le médiateur est capable de le produire.

Pendant cette étape (la coordination des services), les services du cluster éligible échangent des messages entre eux pour construire l'automate globale, (au fur et à mesure, on accepte des transitions d'envoi et de réception de messages). Puisque les messages échangés peuvent contenir des données privées, alors, on doit préserver leurs confidentialités en respectant les préférences spécifiées avec ces données. Par exemple : pour accepter une transition de réception d'un message, à partir d'une configuration donnée, on doit vérifier que les préférences de l'acceptable transition qui a envoyé le message seront respectées par les politiques de la transition de la réception.

Pour respecter les politiques privées des services et du *goal service* lors de la composition, nous procédons comme suit :

- Pour les services de la communauté (i.e. les services du cluster éligible), on associe les préférences aux messages sortants et les politiques aux messages entrants ;
- Pour le *goal service*, on associe les préférences aux messages entrants, car les messages entrants dans le *goal service* représentent les messages que le client envoie. Et on associe des politiques aux messages sortants, car les messages sortants dans le *goal service* représentent les messages que le client reçoit.

Donc, pour satisfaire une transition du *goal service* :

- S'il s'agit d'un envoie de message (un message sortant dans le *goal service* pour lequel on associe les politiques du client, car c'est le client qui va recevoir le message): on cherche une transition dans la communauté qui envoie les mêmes données tel que ses préférences sont moins restrictives que les politiques du client.
- S'il s'agit d'une réception de message (un message entrant dans le *goal* pour lequel on associe les préférences du client, car c'est le client qui va envoyer le message): on cherche une transition dans la communauté qui reçoit les mêmes données, tel que ses politiques sont plus restrictives que les préférences du client.
- Dans le cas d'une opération, on cherche une transition dans la communauté des services qui exécute la même opération.

4.4.5. Le calcul des transitions acceptables

A chaque fois qu'un service du cluster éligible change son état, nous avons des nouvelles transitions à calculer (acceptable au non). Pour calculer une transition acceptable nous procédons comme suit :

Soient le *goal service* Q_g, le médiateur Q_{med} et l'ensemble des services du cluster éligible $\{Q_i\}_{i=1,\ldots,n}$. Pour une transition $(S_g, a_g, S_g^{'})$ dans Q_g, s'il existe une transition $(S_i, a, S_i^{'})$ dans les Q_i, alors la transition combinée $((S_1 \ldots S_i \ldots S_n), a, (S_1 \ldots S_i^{'} \ldots S_n))$ est dite acceptable, si et seulement si, une des conditions suivantes est vérifiée :

- $a_g = a$: il s'agit de satisfaire une transition du *goal service*, donc 3 cas possibles :
  - Si c'est un envoi de message, i.e., $a_g = !m(\bar{d}, plcy)$ et $a = !m(\bar{d}, pref)$ alors : $pref \leq_{private} plcy$;
  - Si c'est une réception de message, i.e., $a_g = ?m(\bar{d}, pref)$ et $a = ?m(\bar{d}, plcy)$ alors : $pref \leq_{private} plcy$;
  - Si c'est une opération, i.e., $a_g = oper_g(\bar{d}; \bar{d^{'}})$ et $a = oper_a(\bar{d}; \bar{d^{'}})$ alors : $oper_g = oper_a$.
- $a = !m(\bar{d}, pref)$;
- $a = ?m(\bar{d}, plcy)$ et :
  - Il existe une transition acceptable qui a envoyé $!m(\bar{d}, pref)$ dans un chemin de $(S_{01} \ldots S_{0i} \ldots S_{0n})$ à $(S_1 \ldots S_i \ldots S_n)$; ou bien, il existe une transition $S_i \xrightarrow{!m(\bar{d}, pref)} S_i^{'}$ dans le Q_{med} ; et
  - $pref \leq_{private} plcy$.

En résumé, à partir du *goal* service et le cluster éligible, les étapes principales pour calculer une composition qui satisfait le *goal service* sont :

1. Initialement, la configuration courante du *goal service* est son état initial, et la configuration courante des services du cluster est définie par l'état initial de chaque service.

2. Pour chaque transition du *goal service* qui peut être déclenchée à partir de son état courant, et

3. Pour chaque transition qui peut être déclenchée à partir de l'état courant de chaque service du cluster éligible :

4. On vérifie si la transition exécute la même opération que le *goal service*, ou bien le même message avec des politiques compatibles, i.e., dans le cas d'un envoi, la politique du *goal* doit respecter la préférence du service et dans le cas d'une réception, la politique du service doit respecter la préférence du *goal*. Alors,

5. On sauvegarde la transition calculée comme une transition acceptable.

6. Pour prévenir les cycles infinis, on vérifie s'il y a un cycle dans le *goal service* et dans le service qui déclenche la transition. De plus, on vérifie s'il y a un état final dans le *goal* qui correspond à un état final dans le service.

7. Si la transition acceptable calculée exécute la même opération (resp. le même message avec des politiques compatibles) que la transition du *goal service*, alors, pour l'état cible de la transition satisfait du *goal service* et pour l'état cible de la configuration de la transition acceptable calculée, on calcule récursivement des transitions acceptables.

8. Sinon, si l'acceptable transition exécute un envoi de message, alors pour la configuration courante du *goal service* et pour l'état cible de la l'acceptable transition calculée, on calcule récursivement des transitions acceptables.

9. Sinon, si la transition calculée exécute une réception de message avec la politique *plcy*, alors :

10. On vérifie si le message peut être envoyé, i.e., on a calculé une transition acceptable qui déclenche l'envoi du message et que les préférences de cette transition sont respectées par *plcy*. Sinon, les données impliquées dans la réception du message sont disponibles[15] et la politique de la réception respecte les préférences de ces données. Alors,

11. On génère le médiateur pour compléter la composition, i.e., générer la transition qui envoie le message manquant avec les préférences des données.

[15] Les données ont été échangées dans d'autres messages.

12. Autrement, si la politique de la transition calculée ne respecte pas les préférences de l'acceptable transition qui a envoyé le message, ou bien les données ne sont pas disponibles, ou bien les données sont disponibles mais la politique de réception ne respecte pas leurs préférences. D'où le médiateur ne peut pas générer le message manquant, alors la transition calculée est non acceptable.

13. Si l'état cible de transition satisfait du *goal service* est un état final, alors fin de l'algorithme avec succès.

14. Pour l'état cible de la configuration courante du *goal* et pour l'état cible de la dernière transition acceptable calculée, si on ne peut pas calculer des transitions acceptables, alors échec de l'algorithme.

Comme mentionné dans les étapes 7 et 8, on exécute récursivement ces étapes pour chaque nouvelle configuration d'une acceptable transition.

Le pseudo algorithme qui prend en entrée le *goal service* et le *cluster eligible* et vérifie si la coordination des services du cluster peut satisfaire le goal service en appliquant les étapes précédentes, et qui a en sortie le médiateur et la composition des services si elle existe est le suivant :

```
Input : Le goal service Qg = (Sg, s0g, Fg, Mg, Ag, Cg, Tg), les services du cluster
éligible
        Q1 = (S1, s01, F1, M1, A1, C1, T1), …, Qn = (Sn, s0n, Fn, Mn, An, Cn, Tn), l'état
courant du
        goal service est sg = s0g, l'état courrant des services s =
(s01, s02, …, s0n)
Output: La coordination Q = (S, s0, F, M, A, C, T), le médiateur
        Qmed=(Smed, s0med, Fmed, Mmed, Cmed, Tmed)

Coordination(sg, S )
Debut
 /* Arrêt du test */
 si((pour sg,il n'existe pas des transitions) ou (pour sg, toutes les
transition
     sont visitées))
 et (touts les services sont dans leurs états initiaux)) alors
     L'état courant de coordination satisfait le goal service.
 sinon
     si pour la transition (sg, a, s'g) ∈ Tg de goal service, il n'existe aucune
        transition (si, a', s'i) ∈ Ti du service Qi tel que a'= a = ou a'∈ Mi
alors
        la trace courante de la coordination ne peut pas satisfaire la
        trace courante de goal service.
     finsi
 finsi
```

pour chaque transition $(s_g, a, s_g') \in T_g$ de goal service **faire**

pour chaque $i \in \{1,...,n\}$ **et chaque** transition $(s_i, a', s_i') \in T_i$ **faire**

$\overline{S}' := (s_1,...,s_i',...,s_n)$

Si $(a = a')$ **alors** /* satisfaire une transition du goal service : 3 cas */

Si ($a_g = !m(\overline{d}, plcy)$ **et** $a = !m(\overline{d}, pref)$ **et** $pref \leq_{private} plcy$) **ou**

($a_g = ?m(\overline{d}, pref)$ **et** $a = ?m(\overline{d}, plcy)$ **et** $pref \leq_{private} plcy$)**ou**

($a_g = oper_g(\overline{d}, \overline{d}')$ **et** $a = oper_a(\overline{d}, \overline{d}')$) **alors**

$s := s \cup \overline{s}$

$T := T \cup (\overline{s}, a', \overline{s}')$

Si $\overline{s}' \in F1 \times ... \times Fn$ **alors** $F := F \cup \overline{s}'$

Si $a' \in M_i$ **alors** $M := M \cup a'$

Si $(s_g' \in F_g)$ **alors**

Fin de l'algorithme avec succès

Sinon

Coordination(s_g' **,** $\overline{S}'$ **)**

finsi

finsi

sinon /* satisfaire une transition d'un service du cluster éligible */

si

($a = !m(\overline{d}, pref)$) **ou**

($a = ?m(\overline{d}, plcy)$ **et** $MediatorConstruction(s, ?m(\overline{d}, plcy), \overline{s}')$) **alors**

$s := s \cup \overline{s}$

$T := T \cup (\overline{s}, a', \overline{s}')$

Si $\overline{s}' \in F1 \times ... \times Fn$ **alors** $F := F \cup \overline{s}'$

Si $a' \in M_i$ **alors** $M := M \cup a'$

Coordination(s_g **,** $\overline{S}'$ **)**

sinon

c'est une opération qui n'est pas spécifiée dans le goal service, donc , la coordination échoue.

finsi

finsi

Figure 43. Algorithme de composition des services Web basé sur les politiques privées.

4.4.6. Génération du médiateur

Le but du médiateur est de générer les messages manquants pour établir la conversation entre les services Web dans le cas d'un inter-blocage. Pour générer ces messages, on vérifie d'abord si la donnée du message attendu est disponible, i.e., si un service Q_j est bloqué dans un état donné s_j, s'il existe une transition

$((S_1...S_j...S_n),?m(\bar{d},plcy),(S_1...S_j^{'}...S_n))$ tel que, il n'existe pas une transition précédente qui permet l'envoi du message $?m(\bar{d},pref)$. Les étapes suivantes sont utilisées pour générer le médiateur :

```
Bool mediatorConstruction(s̄ ,?m(d̄ ,plcy), s̄' )
debut
    si il existe une transition (-,!m(d̄ ,pref),-) apparente dans le
    chemin de s̄0 à s̄ tel que pref <= plcy alors :
            Smed := Smed ∪ s̄'
            Tmed := Tmed ∪ (s,ε, s̄' )
            si s̄' ∈ F alors  Fmed := Fmed ∪ s̄'  finsi
            return true
    Sinon
     /* toutes les données d̄ sont disponible dans un autre message */
            Si ∃ m'(d̄' ,plcy')∈ Mcoord et d̄ ∈ d̄' et plcy' <= plcy alors
                pref := extract_pref(plcy')
                Smed := Smed ∪ s̄'
                Tmed := Tmed ∪ (s,!m(d̄ ,pref), s̄' )
                Mmed := Mmed ∪ !m
                si s̄' ∈ F alors Fmed := Fmed ∪ s̄'  finsi
                return true
            finsi
            return false
    finsi
fin
```

Figure 44. Génération des messages manquants par le médiateur.

4.4.7. Application à l'exemple de motivation

Pour illustrer l'approche proposée, et faire apparaître le comportement de l'algorithme, nous allons appliquer ce dernier sur l'exemple de motivation. Pour cela, nous considérons les trois services : *HM service*, *PM service*, *SS service* représentés dans la figure 8 et le *goal service* associé représenté dans la figure 9. Donc, le processus qui permet de générer la combinaison (figure 43) et le médiateur (figure 44) peut être illustré comme suit :

La configuration initiale de coordination est (*h0p0s0*) (i.e., *HM*, *PM*, *SS* dans leurs états initiales) et le *goal service* est dans l'état initiale g0. La première transition du *goal service* est (*g0;?diagnosis(id,ssn,diag,drugs,pref);g1*). L'algorithme de coordination commence par le calcule de la première configuration (*h6p0s0*) qui peut être atteinte

lorsque *HM service* déclenche la transition (*h0;?diagnosis(id, ssn,diag,drugs,plcy) ; h6*). Comme cette transition satisfait la transition du *goal service* (la même action et *pref* $\leq_{Private}$ *plcy*), donc, le module de la coordination construit la transition acceptable ((*h0p0s0*)*;?diagnosis(id,ssn,diag,drugs,pref);* (*h6p0s0*)) et mis à jour la description de la coordination. Par conséquent, le médiateur construit une transition vide (*(h0p0s0);ε; (h6p0s0)*) car les deux transitions reçoivent le même message et mis à jour la description du médiateur. Donc, la configuration courante du *goal service* devient *g1* et la configuration de la coordination devient (*h6p0s0*). Récursivement, pour cette configuration et pour la configuration *g1* du *goal service* le module de la coordination commence par le calcule de la première configuration possible (*h7p0s0*). La configuration (*h7p0s0*) peut être atteinte lorsque le *HM service* déclenche la transition (*h6;!check(ssn,pref);h7*). Comme cette transition ne satisfait celle du *goal service* (*g1; checkSSN(ssn; adr; res); g2*) mais elle tire un output message. Donc, le module de la coordination construit la transition acceptable (*(h6p0s0 ;!check(ssn,pref);(h7p0s0)*). En parallèle, le module médiateur construit une transition vide (*(h6p0s0);ε;(h7p0s0)*). Comme la transition acceptable calculée ne satisfait pas la configuration courante du *goal*, donc, la configuration courante du *goal* reste *g1* et la configuration courante de la coordination devient (*h7p0s0*). Pour ces deux configurations le module de la coordination calcule la configuration (*h7p0s1*) qui peut être atteinte lorsque le *SS service* déclenche la transition (*s0;?checkSSN(ssn,plcy);s1*). Cette transition ne satisfait pas celle du *goal* (*g1; checkSSN(ssn; adr; res); g2*), mais elle permet un input message. Dans ce cas, le module médiateur vérifie si le message est envoyé en vérifiant la description de la coordination. Comme ce message n'a été envoyé précédemment, le module vérifie, en utilisant les informations locales des services, si la donnée *ssn* est disponible, i.e., le *ssn* a été calculé par une opération ou bien il a été reçu ou envoyé par un autre message. Nous pouvons remarquer que, dans le *goal* service, le client envoie le *ssn* dans le message *diagnosis(id,ssn,diag,drugs,pref)* et, dans la description de la coordination, le *HM service* envoie le message *!check(ssn,pref)*. Donc, le *ssn* doit être disponible dans les informations locales (les informations système) du *HM service*, ensuite, le module médiateur va considérer les préférences de l'acceptable transition (*(h6p0s0 ;!check(ssn,pref);(h7p0s0)*) (car, c'est la plus récente) pour les vérifier avec la politique de la transition à calculer (*s0;?checkSSN(ssn,plcy);s1*), si *pref* $\leq_{Private}$ *plcy*, alors le module médiateur va générer le message *!check(ssn,pref)* pour atteindre la configuration *(h7p0s1)* à partir de la configuration *(h7p0s0)*. Ainsi, le module médiateur génère la transition (*(h7p0s0);!checkSSN(ssn); (h7p0s1)*) et mis à jour la description du médiateur.

En utilisant les mêmes étapes, pour chaque transition acceptable et selon le *goal service*, on calcule récursivement des transitions acceptables. L'automate d'état fini généré, qui correspond à la combinaison des trois services qui simule l'exécution de l'application abstraite (le *goal service*), est représenté par la figure 45:

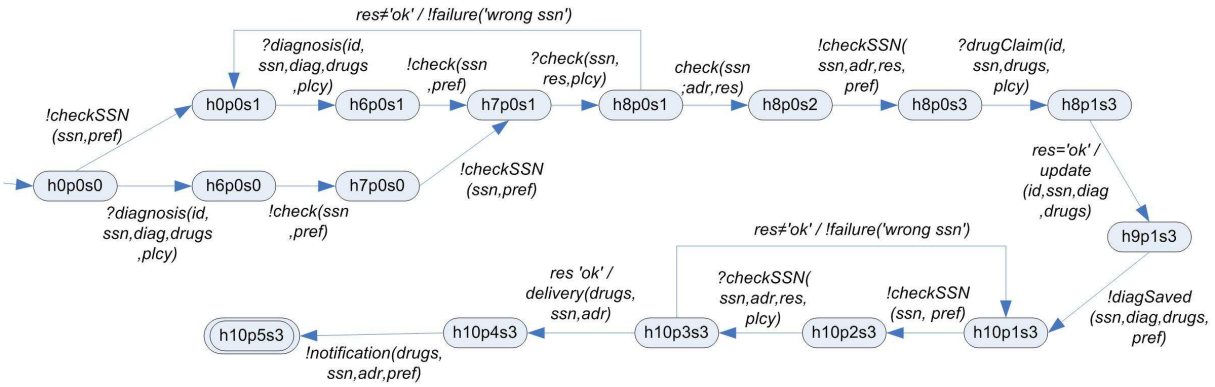


Figure 45. La combinaison des trois services simulant le goal service.

En principe, la stratégie de la composition consiste à composer les services sans la participation du médiateur. Cependant, s'il est nécessaire (comme dans notre exemple : le *HM service* envoie le *!check(ssn,pref)* par contre le *SS service* reçoit *?checkSSN(ssn,plcy)*), on génère le médiateur dont le rôle est de produire les messages manquants mais qui sont nécessaire pour compléter la composition. L'automate d'état fini généré pour simuler le rôle du médiateur pour cette composition est représenté par la figure 46 :

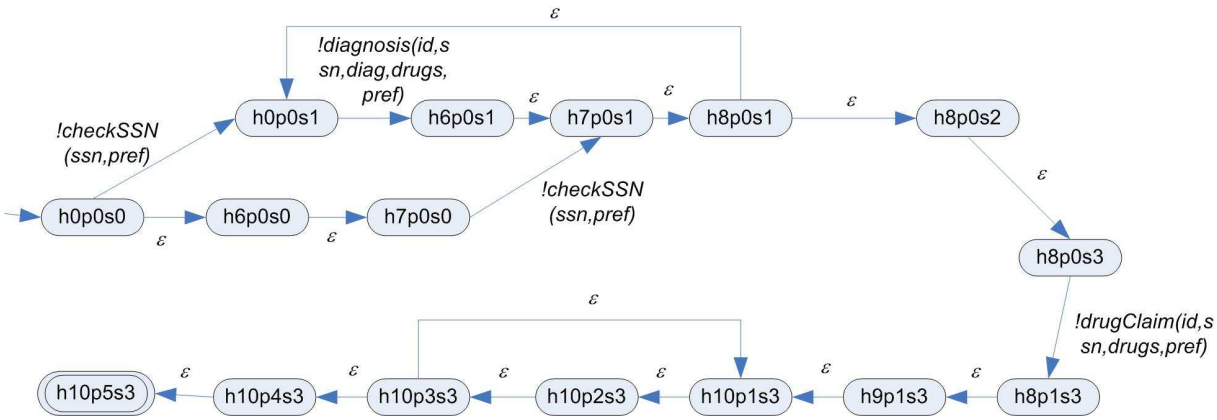


Figure 46. La description du médiateur.

4.5. Conclusion

Dans ce chapitre, nous avons proposé une méthode de composition de services Web. Cette méthode est basée sur : la coordination des services, c'est-à-dire, la communication entre les services Web se fait avec envois et réceptions de messages, la médiation, i.e. dans le cas ou la communication est aboutie à un inter-blocage, le médiateur intervient pour construire le message manquant au bon déroulement de la composition, et politiques privées : la composition prend en compte les préférences de confidentialité du client définies dans sa requête qui seront respectées par tous les services participant, ainsi que toutes les préférences de ces derniers.

Dans notre approche, un service Web est représenté par un automate d'état fini, dont les transitions sont : un envoi d'un message, réception d'un message ou une opération. L'état initial de l'automate ne débute que des transitions de type réception de message, ces messages permettant l'invocation de ce service. Une transition qui ramène à un état final est forcément un envoi de message qui doit contenir le résultat de l'exécution d'une opération ou la totalité du service. L'algorithme de composition proposé prend en entrée la requête du client sous forme d'un service qui s'appel le *goal service*, ce dernier est une application abstraite représente l'enchainement attendu des opérations que le client veut effectuer par rapport à ses données privées, et un ensemble de services candidats.

Dans le chapitre suivant, nous allons implémenter cette approche avec une plate-forme basée sur l'algorithme de composition proposé.

Chapitre 5. Implémentation et analyse de l'approche proposée.

5.1. Introduction

Pour valider l'approche proposée, nous avons implémenté une plate-forme de composition des services Web basée sur les politiques privées et l'algorithme de coordination proposé, qui permet aussi la création et le sauvegarde des services Web ainsi que les requêtes des utilisateurs et leurs exécutions. C'est-à-dire, dans notre application, une requête exécutée avec succès devient un service disponible dans l'UDDI de l'application qui peut être réutilisé par autres utilisateur et qui peut aussi participer à l'exécution des autres requêtes plus complexes.

Pour être analysée, dans l'implémentation de notre approche, nous avons utilisé les fichiers XML pour représenter les services Web et les politiques privées. Dans un fichier XML, nous avons met toutes les informations descriptives ainsi que toutes les entités communicantes (les envoies et les réceptions des messages). Un service Web est représenté par un seul fichier XML et pour chacune de ses politiques privées est représentée par un seul fichier XML. A partir de la spécification XML, le programme récupère les différentes informations dans des structures de données à fin de procéder à l'analyse et la composition.

Dans ce chapitre, nous détaillerons l'architecture de la plate-forme de composition des services Web avec les politiques privées basée sur l'algorithme proposé, et nous expliquons tout les cas d'utilisations implémentés dans cette application. Dans un premier point, nous détaillerons l'architecture de l'application en expliquant le rôle de chaque tiers. Dans un deuxième point, nous donnerons la structure que doit avoir le fichier XML d'un service Web et une politique privée. Et dans un dernier point, nous montrons la capacité de notre algorithme à calculer la composition satisfaisante et à détecter une violation de politiques privées avec des traces d'exécution pour chaque cas de test.

5.2. Architecture globale de la plate-forme de composition (WSCBPP)

La plate-forme de composition des services Web WSCBPP est une application Web qui peut être déployée en ligne et qui s'exécute sur le navigateur. Elle est développée avec la plate-forme J2EE en utilisant le Framework GWT (*Google Web ToolKit*), l'architecture de la plate-forme est de trois tiers (les utilisateurs, le serveur d'application et le serveur UDDI). Elle permet aux utilisateurs de créer des services de les sauvegarder et d'exécuter des requêtes de composition en se basant sur l'algorithme de composition proposé. L'architecture globale de WSCBPP représentée par la figure 47:

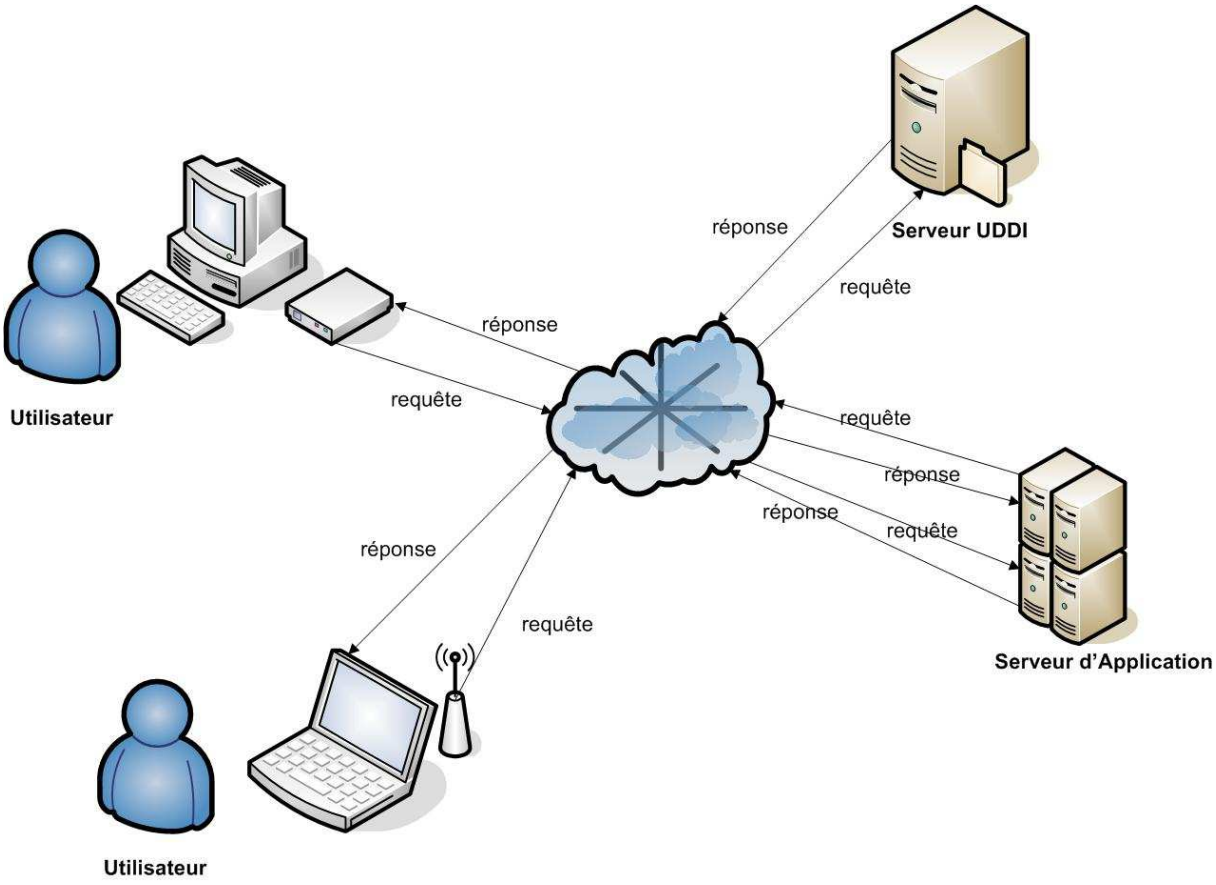


Figure 47. Architecture globale de WSCBPP.

Les utilisateurs envoient des requêtes au serveur d'application, ces requêtes peuvent être : création d'un nouveau service, consultation d'un service qui existe déjà ou une demande d'exécution d'une requête de composition.

Le serveur d'application est un ensemble de services Web développés avec la technologie GWT qui permettent l'exécution de différentes requêtes des utilisateurs, qui sont des invocations à ces services. Donc, le serveur d'application reçoit les requêtes des utilisateurs sous forme d'appels RPC vers ses services. Le service Web invoqué fait le traitement (exécute l'algorithme de composition dans le cas d'une requête de composition), ensuite il envoie la réponse. Pour cela, il peut solliciter le serveur UDDI pour sauvegarder un nouveau service créé, récupérer un service ou un ensemble de services pour exécuter l'algorithme de composition.

Le serveur UDDI est l'ensemble de fichiers XML qui représentent tout les services avec leurs politiques qui existent dans le système, l'accès à ces fichiers se fait à l'aide d'une couche qui contient tout les méthodes nécessaires pour sauvegarder, rechercher ou récupérer un ensemble de services comme des fichier XML ou bien sous forme des structures de données qui sont implémentées pour faire le traitement au niveau de serveur d'application.

L'écran principal de l'application est représenté par l'imprime d'écran suivant. Dans la partie gauche de l'écran, il y a tout les services Web qui existent dans le système et aussi toutes les requêtes déjà exécutées par les utilisateurs. Et dans la partie à droite, on trouve deux onglets :

- Un onglet de configuration pour construire une requête. Dans cet onglet, on doit mentionner un libellé pour la requête, sélectionner les services qui participe à l'exécution de cette requête (par défaut c'est tout les services) et on peut aussi sélectionner un *goal service* ou bien crée un autre.
- Un onglet de visualisation qui permet l'affichage du fichier XML correspond au service sélectionné à partir de la partie gauche, ainsi les fichiers XML des politiques privées de celui-ci.

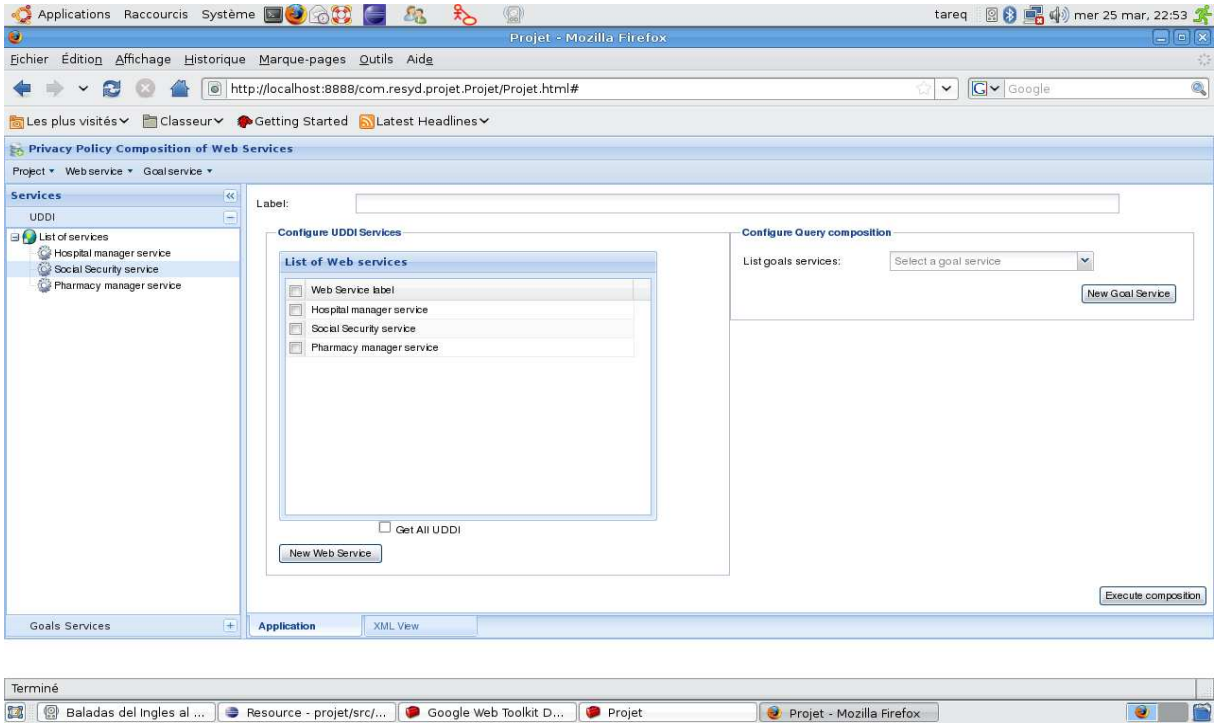


Figure 48. Ecran principal de WSCBPP.

Pour afficher un service Web, on sélectionne ce service dans la partie gauche (la liste des services) puis on clique sur l'onglet *XML view* pour qu'il devient visible, et pour afficher une politique privée ou une préférence de ce service, on clique sur ce dernier avec le bouton droit de la souri, il y a un menu qui s'affiche dont les items sont les politiques et préférences du services, on clique sur un item, le fichier XML de la politique (respectivement la préférence) sélectionnée et afficher dans l'onglet *XML view*.

L'imprime d'écran suivant représente est un exemple d'affichage du fichier XML d'un service Web dans l'onglet *XML view*.

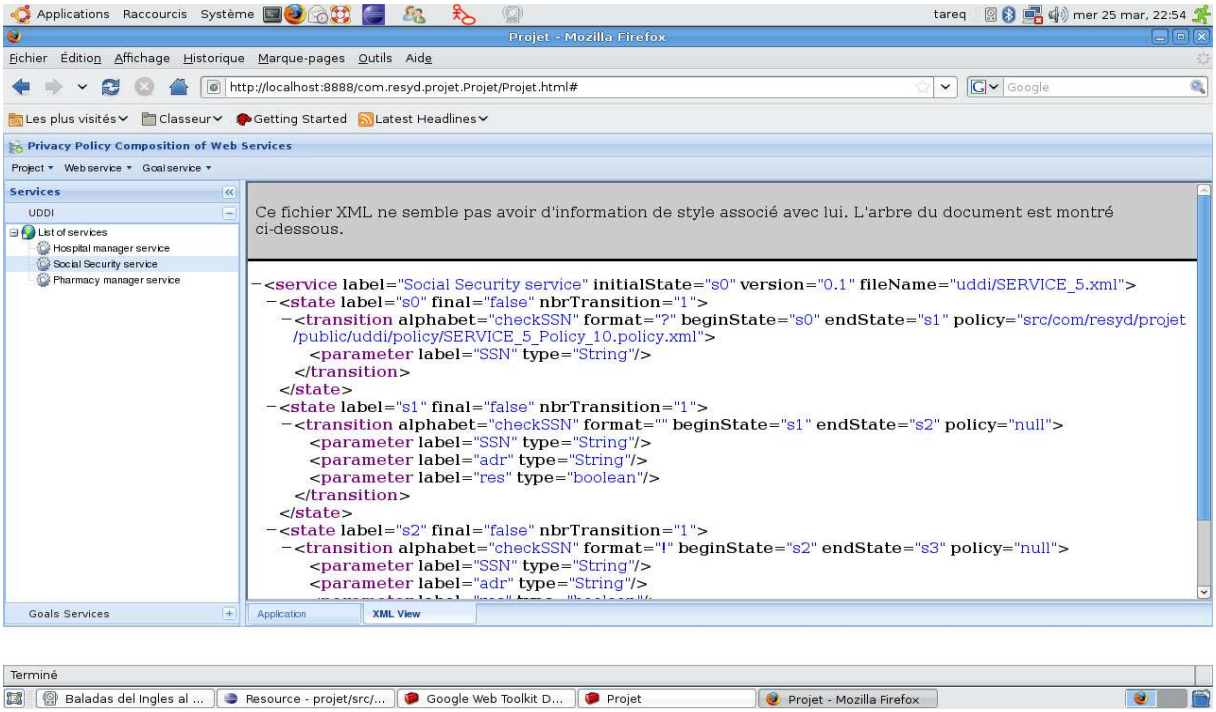


Figure 49. Exemple d'affichage d'un service Web.

5.3. La spécification XML

Les fichiers XML utilisés pour représenter un service Web et une politique privée, doit respecter un format XML préalablement définit. Les schémas XML, que doit respecter ces fichiers qui sont utilisés dans l'application, sont donnés dans les figures 50 et 51:

```
<xsd :schema xmlns='http://www.w3.org/2001/XMLSchema' targetNamespace='http://www.resyd.org'
xmlns='http:/www.resyd.org' elementFormDefault='qualified'>
<xsd:element name=service >
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref='state' minOccur='1'/>
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref='transition' minOccur='0' />
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element ref='parameter' minOccur='1' />
                </xsd:sequence>
              </xsd:complexType>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name='service' type='xsd:string'>
<xsd:element name='state' type='xsd:string'>
<xsd:element name='transition' type='xsd:string'>
<xsd:element name='parameter' type='xsd:string'>
</xsd:schema>
```

Figure 50. Schéma XML d'un service Web.

Un service Web est une séquence d'états, chaque état est une séquence de transitions et chaque transition est une séquence de paramètres.

```
<xsd :schema xmlns='http://www.w3.org/2001/XMLSchema' targetNamespace='http://www.resyd.org'
xmlns='http:/www.resyd.org' elementFormDefault='qualified'>
<xsd:element name=policy >
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name=tdu >
            <xsd:complexType>
               <xsd:element ref='entity' minOccur='1'/>
               <xsd:element ref='executeInterval' minOccur='1'/>
               <xsd:sequence>
                <xsd:element ref='right' minOccur='1'/>
                   <xsd:complexType>
                      <xsd:element ref='entity' minOccur='1'/>
                      <xsd:element ref='executeInterval' minOccur='1'/>
                      <xsd:element ref='validityInterval' minOccur='1'/>
                      <xsd:sequence>
                         <xsd:element ref='obligation' minOccur='0' />
                           <xsd:complexType>
                              <xsd:element ref='entity' minOccur='1'/>
                              <xsd:element ref='executeInterval' minOccur='1'/>
                              <xsd:element ref='validityInterval' minOccur='1'/>
                           </xsd:complexType>
                      </xsd:sequence>
                   </xsd:complexType>
                </xsd:element>
               </xsd:sequence>
               <xsd:sequence>
                  <xsd:element ref='obligation' minOccur='0' />
                     <xsd:complexType>
                        <xsd:element ref='entity' minOccur='1'/>
                        <xsd:element ref='executeInterval' minOccur='1'/>
                        <xsd:element ref='validityInterval' minOccur='1'/>
                     </xsd:complexType>
                </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

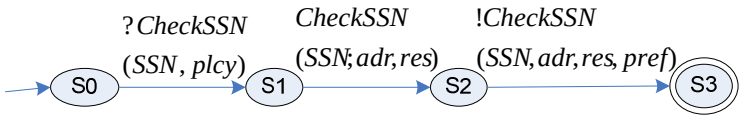
Figure 51. Schéma XML d'une politique privée.

Une politique privée est une séquence de termes d'utilisation de données *tdu*, chaque *tdu* est composé d'une entité, un intervalle d'exécution, une séquence de droits (*right*) et une séquence d'obligations (*obligation*). Un *right* est composé d'une entité, un intervalle d'exécution, un intervalle de validation et une séquence d'obligations. En fin, une obligation est composée d'une entité, un intervalle d'exécution et un intervalle de validation.

5.3.1. Exemple de fichiers XML d'un service Web

Prenant l'exemple de motivation du chapitre précédent, les fichiers XML qui représentent le service de la sécurité social (*SS service*) défini par son automate d'état fini suivant et la politique *plcy,* tel que :

L'automate d'état fini est :



Et *plcy* est :

$plcy_{SS} = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (ValidityVerification, Ours : CheckSSN, \mu : [0,20\,\min], \{o1\})$
$o1 = (Destruction, Ours : CheckSSN, v : [t(p1), t(p1) + 10\,\min], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\min]$ est l'intervalle du temps pour satisfaire le purpose p1.

Sont :

```
<?xml version="1.0" encoding="UTF-8"?>
<service label="Social Security service" initialState="s0" version="0.1"><state label="s0" final="false" nbrTransition="1">
<transition alphabet="checkSSN" format="?" beginState="s0" endState="s1" policy="src/com/resyd/projet/public/uddi/
        policy/Social Security service0.policy.xml">
                <parameter label="SSN" type="String" />
        </transition>
</state>
<state label="s1" final="false" nbrTransition="1">
        <transition alphabet="checkSSN" format="" beginState="s1" endState="s2" policy="null">
                <parameter label="SSN" type="String" />
                <parameter label="adr" type="String" />
                <parameter label="res" type="boolean" />
        </transition>
</state>
<state label="s2" final="false" nbrTransition="1">
        <transition alphabet="checkSSN" format="!" beginState="s2" endState="s3" policy="null">
                <parameter label="SSN" type="String" />
                <parameter label="adr" type="String" />
                <parameter label="res" type="boolean" />
        </transition>
</state>
<state label="s3" final="true" nbrTransition="0" />
</service>
```

Figure 52. Spécification XML de *SS Service.*

```
<?xml version="1.0" encoding="UTF-8" ?>
 <policy>
      <tdu data="SSN" purpose="ValidityVerification">
         <entity label="CheckSSN" type="ours" />
         <executeInterval begin="0" end="20" />
         <obligation action="Destruction" afterPurpose="ValidityVerification">
            <entity label="CheckSSN" type="ours" />
            <validationInterval begin="0" end="10" />
            <executeInterval begin="0" end="0" />
         </obligation>
      </tdu>
 </policy>
```

Figure 53. Spécification XML de la politique privée *plcy*.

5.4. Tests et analyse

Dans ce qui suit, nous testons la capacité de notre algorithme à détecter les différents types de violation de politiques privées, et cela à travers des exemples choisis de façon à ce que chaque exemple contient un type d'erreur. Nous avons supposé dans l'implémentation de l'algorithme les points suivants :

- La préférence d'un envoi de message (output message) qui ramène à un état final est vide ;
- La préférence d'un message d'erreur (ou autres messages qui ne contiennent pas des données privées comme les messages de confirmation ou les messages d'alerte) est vide ;
- Dans un service Web, tout les inputs messages ont la même politique privée qui celle du service Web ;
- Pendant l'exécution d'une requête (composition des services) un service Web qui arrive a son état final, il reste à son état final jusqu'à la fin de l'exécution de cette requête. C-à-dire, dans l'exécution d'une requête, on n'exécute pas un service plus d'une fois.
- Le médiateur peut générer un message manquant si les paramètres du message existent dans une autres transition acceptable déjà calculée avec le même ordre et le même nombre et cette transition acceptable est un envoie de message.

5.4.1. Scénario d'une exécution valide

Dans ce scénario, nous allons expliquer le rôle du médiateur pour résoudre un problème d'inter-blocage dû au non compatibilité (au niveau des signatures) entre un envoi de message et sa réception. Pour cela, nous considérons les trois services de l'exemple de motivation : *SS Service*, *HP Service* et *PM Service* qui sont représentés par leurs automates d'état fini et leurs politiques comme suit :

Les politiques privées $polcy_{HM}$, $polcy_{PM}$, $polcy_{SS}$ des services sont :

$plcy_{HM} = \{(SecuritySocialNumber, p1), (SecuritySocialNumber, p2)\}$ ou :
$p1 = (SearchAntecedent, Ours : searchAntec, \mu : [0,20\,\text{min}], \{r1\})$
$r1 = (ValidityVerification, Others : CheckSSN, \mu : [0,20\,\text{min}], \{o1\})$
$o1 = (Destruction, Others : CheckSSN, v : [t(p1), t(p1) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.)
$p2 = (Upatediagnosis, Ours : Update, \mu : [0,20\,\text{min}], \{r1\})$
$r1 = (ValidityVerification, Others : CheckSSN, \mu : [0,20\,\text{min}], \{o1\})$
$o1 = (Destruction, Others : CheckSSN, v : [t(p2), t(p2) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1. *)*

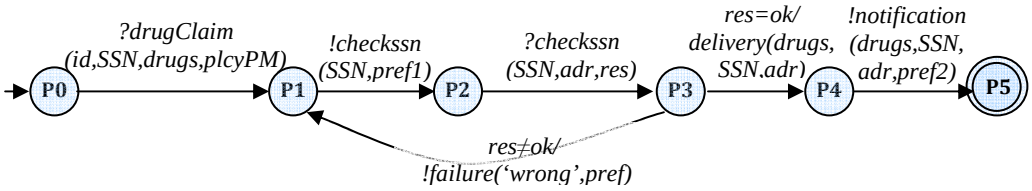
La politique de *SS Service* sur le *SSN* est :

$plcy_{SS} = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (ValidityVerification, Ours : CheckSSN, \mu : [0,20\,\text{min}], \{o1\})$
$o1 = (Destruction, Ours : CheckSSN, v : [t(p1), t(p1) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.

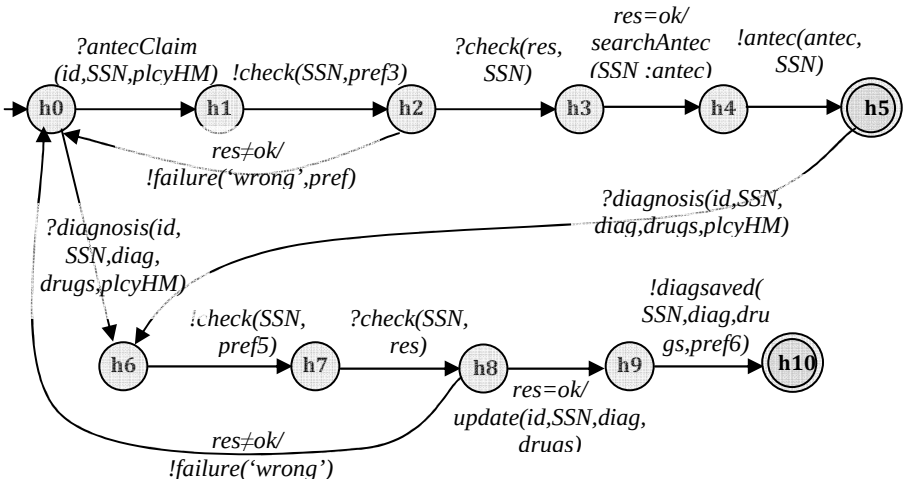
La politique du *PM Service* sur le *SSN* est :

$plcy_{PM} = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (Delivery, Ours : delivery, \mu : [0,60\,\text{min}], \{r1\})$
$r1 = (ValidityVerification, Others : CheckSSN, \mu : [0,20\,\text{min}], \{o1\})$
$o1 = (Destruction, Ours : delivery, v : [t(p1), t(p1) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.

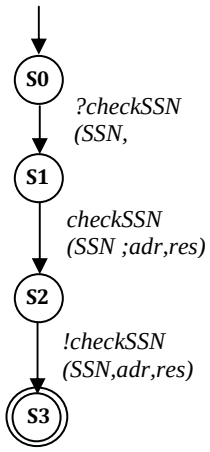
Les automates d'état fini des services sont :



PM service.



HM service.



SS service.

Le service *HM Service*, dans son exécution, doit invoquer le *SS Service*, cette invocation se fait dans les transitions (H1, H2) et (H6, H7) avec les envois de messages : *!check(SSN,pref3)* et *!check(SSN,pref5)*. De son côté de *SS Service* s'invoque avec la réception d'un message de la forme : *?checkSSN(SSN,plcy$_{SS}$)*, dans ce cas, l'algorithme fait appel au médiateur pour qu'il génère les messages manquants (*!checkSSN(SSN,pref3)* et *!checkSSN(SSN,pref5)*. Pour cela, le médiateur vérifie dans les envois de message déjà calculés (acceptés) s'il y a un envoi avec les mêmes paramètres (la même signature, le même type et le même ordre), s'il trouve un message envoyé qui correspond a ces conditions, il vérifie ensuite si la préférence d'envoi est respectée par la politique de réception *polcy$_{SS}$*. Dans notre exemple le médiateur génère le message manquant si *polcy$_{SS}$<=pref3*.

La requête du client sur laquelle nous allons exécuter ce scénario, qui nécessite la composition des trois services, est celle de l'exemple de motivation (*le goal service*), qui est représentée par un automate d'état fini et une préférence de confidentialité.

La trace d'exécution qui simule ce cas de test est la suivante :

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~ DEBUT DE L'ALGORITHME ~~~~~~~~~~~~~~~~~~~~~~~~~~~~

* * * Exécution du scénario 1 (exécution valide)

* Cluster éligible :
      - 1 - Hospital manager service
      - 2 - Pharmacy manager service
      - 3 - Social Security service

* Goal service : Goal service

* * * Transition Goal Ã  satisfaire : ?diagnosis g0-g1
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?antecClaim h0-h1
      -1- vérification du paramÃ¨tres : [id String, SSN String] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
      -1- vérification du paramÃ¨tres : [SSN String] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [?diagnosis h0-h6]
  * ACCEPTEE : ?diagnosis h0-h6

* * * Transition Goal Ã  satisfaire : checkSSN g1-g2
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (SuccÃ¨s)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
      -1- vérification du paramÃ¨tres : [SSN String] (SuccÃ¨s)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
* [!check h6-h7]
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?check h7-h8
      -1- vérification du paramÃ¨tres : [SSN String, adr String, res boolean] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
```

```
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (SuccÃ¨s)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
      -1- vérification du paramÃ¨tres : [SSN String] (SuccÃ¨s)
      -2- vérification du politiques :  (SuccÃ¨s)
    ~~~~~~~~~~ médiateur (SuccÃ¨) ~~~~~~~~~~
  * [?checkSSN s0-s1]
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?check h7-h8
      -1- vérification du paramÃ¨tres : [SSN String, adr String, res boolean] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (SuccÃ¨s)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [checkSSN s1-s2]
  * ACCEPTEE : checkSSN s1-s2

* * * Transition Goal Ã  satisfaire : !res!=ok/failure g2-g1
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?check h7-h8
      -1- vérification du paramÃ¨tres : [SSN String, adr String, res boolean] (SuccÃ¨s)
      -2- vérification du politiques :  (SuccÃ¨s)
    ~~~~~~~~~~ médiateur (SuccÃ¨) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (SuccÃ¨s)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [?check h7-h8, !checkSSN s2-s3]
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (SuccÃ¨s)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [!res!=ok/failure h8-h0]
  * ACCEPTEE : !res!=ok/failure h8-h0

* * * Transition Goal Ã  satisfaire : res=ok/update g2-g3
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (SuccÃ¨s)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [!res!=ok/failure h8-h0, res=ok/update h8-h9]
  * ACCEPTEE : res=ok/update h8-h9

* * * Transition Goal Ã  satisfaire : !diagSaved g3-g4
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (SuccÃ¨s)
      -2- vérification du politiques :  (SuccÃ¨s)
    ~~~~~~~~~~ médiateur (SuccÃ¨) ~~~~~~~~~~
  * [!diagSaved h9-h10, ?drugClaim p0-p1]
  * ACCEPTEE : !diagSaved h9-h10

* * * Transition Goal Ã  satisfaire : !res!=ok/failure g4-g1
  * [!checkSSN p1-p2]
  * [?checkSSN p2-p3]
  * [!res!=ok/failure p3-p1]
  * ACCEPTEE : !res!=ok/failure p3-p1

* * * Transition Goal Ã  satisfaire : res=ok/delivery g4-g5
  * [!res!=ok/failure p3-p1, res=ok/delivery p3-p4]
  * ACCEPTEE : res=ok/delivery p3-p4

* * * Transition Goal Ã  satisfaire : !notification g5-g6
  * [!notification p4-p5]
  * ACCEPTEE : !notification p4-p5
```

```
* * * Affichage de la composition :

 * Exécution du scénario 1 (exécution valide), Etat initial : h0p0s0

 ~ Etat : h0p0s0 final : false
    - Transitions : ?diagnosis h0p0s0-h6p0s0
 ~ Etat : h6p0s0 final : false
    - Transitions : !check h6p0s0-h7p0s0
 ~ Etat : h7p0s0 final : false
    - Transitions : ?checkSSN h7p0s0-h7p0s1
 ~ Etat : h7p0s1 final : false
    - Transitions : checkSSN h7p0s1-h7p0s2
 ~ Etat : h7p0s2 final : false
    - Transitions : ?check h7p0s2-h8p0s2
 ~ Etat : h8p0s2 final : false
    - Transitions : !checkSSN h8p0s2-h8p0s3
 ~ Etat : h8p0s3 final : false
    - Transitions : !res!=ok/failure h8p0s3-h0p0s0
    - Transitions : !res!=ok/failure h8p0s3-h0p0s0
    - Transitions : res=ok/update h8p0s3-h9p0s3
 ~ Etat : h9p0s3 final : false
    - Transitions : !diagSaved h9p0s3-h10p0s3
 ~ Etat : h10p0s3 final : false
    - Transitions : ?drugClaim h10p0s3-h10p1s3
 ~ Etat : h10p1s3 final : false
    - Transitions : !checkSSN h10p1s3-h10p2s3
 ~ Etat : h10p2s3 final : false
    - Transitions : ?checkSSN h10p2s3-h10p3s3
 ~ Etat : h10p3s3 final : false
    - Transitions : !res!=ok/failure h10p3s3-h10p1s3
    - Transitions : !res!=ok/failure h10p3s3-h10p1s3
    - Transitions : res=ok/delivery h10p3s3-h10p4s3
 ~ Etat : h10p4s3 final : false
    - Transitions : !notification h10p4s3-h10p5s3
 ~ Etat : h10p5s3 final : true
```

5.4.2. Scénario d'une exécution non valide

Une exécution non valide est causée par la non satisfaction d'une opération du goal (coordination impossible, i.e. il n'existe pas un service qui réalise l'opération), ou bien par la non compatibilité entre les politiques privées des services. Dans ce scénario, nous nous intéressons à la deuxième cause qui est le non compatibilité entre les politiques privées des services Web. Pour cela, nous définissons deux cas possibles : la politique privée du service est moins restrictive que la préférence du client, ou bien, la politique du service viole la préférence du client, c-à-dire, dans la politique du service, on trouve des objectifs qui ne sont pas mentionnés dans la préférence du client.

5.4.2.1. Cas 1 : la politique du service est moins restrictive que la préférence du client

Pour simuler ce cas, nous allons considérer le service *SS Service* avec sa politique actuelle, et nous allons modifier dans la préférence du *HM Service* qui invoque le *SS Service* dans la deuxième transition avec le message *!check(SSN,pref5)*. La préférence *pref5* est plus restrictive que la politique du *SS Service* car elle exige pour l'exécution de l'objectif *ValidityVerification* un intervalle de 10 min, alors que, le service *SS Service* réalise la vérification dans un intervalle de 20 min.

$pref5 = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (ValidityVerification, Ours : CheckSSN, \mu : [0,10\min], \{o1\})$
$o1 = (Destruction, Ours : CheckSSN, v : [t(p1), t(p1) + 10\min], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\min]$ est l'intervalle du temps pour satisfaire le purpose p1.

La trace d'exécution de cas est la suivante :

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ DEBUT DE L'ALGORITHME ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

* * * Exécution du scénario 1 (exécution valide)

* Cluster éligible :
      - 1 - Hospital manager service
      - 2 - Pharmacy manager service
      - 3 - Social Security service

* Goal service : Goal service

* * * Transition Goal Ã  satisfaire : ?diagnosis g0-g1
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
     ~ vérifier si le médiateur peut calculer : ?antecClaim h0-h1
       -1- vérification du paramères : [id String, SSN String] (Echec)
     ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
     ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
       -1- vérification du paramètres : [id String, SSN String, drugs String] (Echec)
     ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
     ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
       -1- vérification du paramètres : [SSN String] (Echec)
     ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [?diagnosis h0-h6]
  * ACCEPTEE : ?diagnosis h0-h6

* * * Transition Goal Ã  satisfaire : checkSSN g1-g2
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
     ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
       -1- vérification du paramètres : [id String, SSN String, drugs String] (SuccÃ¨s)
       -2- vérification du politiques : (Echec)
     ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
     ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
       -1- vérification du paramètres : [SSN String] (Succès)
       -2- vérification du politiques : (Echec)
     ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [!check h6-h7]
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
     ~ vérifier si le médiateur peut calculer : ?check h7-h8
       -1- vérification du paramÃ¨tres : [SSN String, adr String, res boolean] (Echec)
     ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
     ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
       -1- vérification du paramÃ¨tres : [id String, SSN String, drugs String] (SuccÃ¨s)
       -2- vérification du politiques : (Echec)
     ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
     ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
     ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
       -1- vérification du paramÃ¨tres : [SSN String] (SuccÃ¨s)
       -2- vérification du politiques : (Echec)
     ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ FIN DE L'ALGORITHME (Echec) ~~~~~~~~~~~~~~~~~~~~~~~~~~
```

5.4.2.2. Cas 2 : la politique du service viole la préférence du client

Dans ce cas, la politique du service peut faire une action (*purpose, right* ou *obligation*) qui n'est pas mentionnée dans la préférence du client ou bien le contraire, c-à-dire, elle ne fait pas une action (*right* ou *obligation*) qui existe dans la préférence du client.

Pour ce scénario, nous allons considérer la préférence du *HM Service* dans la transition (*h6, h7*), dans laquelle, il invoque le *SS Service* par le message : *!Check(SSN, pref5)* tel que *pref5* est :

$pref\,5 = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (ValidityVerification, Ours : CheckSSN, \mu : [0,20\,\text{min}], \{o1\})$
$o1 = (Destruction, Ours : CheckSSN, v : [t(p1), t(p1) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.

Et la politique du *SS Service* est :

$polcy_{SS} = \{(SecuritySocialNumber, p1)\}$ ou :
$p1 = (ValidityVerification, Ours : CheckSSN, \mu : [0,20\,\text{min}], \{r1\})$
$r1 = (Save, Ours : CheckSSN, v : [t(p1), t(p1) + 10\,\text{min}], \mu : [0,0])$ tel que :
$t(p1) \in [0,20\,\text{min}]$ est l'intervalle du temps pour satisfaire le purpose p1.

Nous remarquons que $polcy_{SS}$ n'a pas l'obligation de destruction du *SSN* exigée par la préférence *pref5*, en plus, $plcy_{SS}$ a le droit, de le sauvegarder après la vérification de validation. La trace d'exécution de cas est la suivante :

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~ DEBUT DE L'ALGORITHME ~~~~~~~~~~~~~~~~~~~~~~~~~~~~

* * * Exécution du scénario 1 (exécution valide)

* Cluster éligible :
      - 1 - Hospital manager service
      - 2 - Pharmacy manager service
      - 3 - Social Security service

* Goal service : Goal service

* * * Transition Goal Ã  satisfaire : ?diagnosis g0-g1
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?antecClaim h0-h1
      -1- vérification du paramètres : [id String, SSN String] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramètres : [id String, SSN String, drugs String] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
      -1- vérification du paramètres : [SSN String] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [?diagnosis h0-h6]
  * ACCEPTEE : ?diagnosis h0-h6

* * * Transition Goal Ã  satisfaire : checkSSN g1-g2
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramètres : [id String, SSN String, drugs String] (Succès)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
      -1- vérification du paramètres : [SSN String] (Succès)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
  * [!check h6-h7]
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?check h7-h8
      -1- vérification du paramètres : [SSN String, adr String, res boolean] (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?drugClaim p0-p1
      -1- vérification du paramètres : [id String, SSN String, drugs String] (Succès)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~
    ~~~~~~~~~~ Dans le médiateur ~~~~~~~~~~
    ~ vérifier si le médiateur peut calculer : ?checkSSN s0-s1
      -1- vérification du paramètres : [SSN String] (Succès)
      -2- vérification du politiques : (Echec)
    ~~~~~~~~~~ médiateur (Echec) ~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~ FIN DE L'ALGORITHME (Echec) ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

5.5. Conclusion

Dans ce chapitre, nous avons implémenté notre approche pour la composition de services Web en utilisant les politiques privées, à travers une plate-forme WSCBPP, conçue avec la technologie GWT, qui utilise l'algorithme de composition proposé pour l'exécution des requêtes complexes. Ces requêtes nécessitent la coordination de plusieurs services et un médiateur. WSCBPP contient un UDDI dans le quel on sauvegarde touts les services créés par les utilisateurs avec leurs politiques de confidentialités ainsi que touts les services complexes qui se génèrent à travers des

exécutions valides des requêtes complexes. Elle contient aussi, un serveur d'application qui fait le traitement des requêtes complexes, en appliquant l'algorithme de coordination et de médiation proposé et qui permet la création, le sauvegarde et la recherche des services Web dans l'UDDI.

Nous avons appliqué notre algorithme sur plusieurs scénarios de tests qui montrent les différents cas particuliers ou la coordination ne doit pas réussir et d'autres cas ou la coordination doit réussir après l'intervention du médiateur qui résout l'interblocage.

- Dans un premier scénario, l'algorithme fait appel au médiateur pour résoudre un interblocage dû au non compatibilité entre un message envoyé et sa réception à cause de la différence de signatures. Dans ce cas, le médiateur génère le message nécessaire au bon déroulement de la coordination, en vérifiant la disponibilité des données ainsi que les politiques de confidentialités de ces données.
- Le deuxième scénario est une exécution non valide, d'une requête d'un client, causée par la différence de niveau de restriction entre la préférence du client et politique du service.
- Le dernier scénario illustre une autre exécution non valide d'une requête causée par la violation des politiques de confidentialités.

Conclusion générale et perspectives

Dans l'approche de composition proposée qui est basée sur la coordination, la médiation et les politiques privées, nous avons modélisé les services Web par des automates d'état fini. Cela peut être justifié par le fait qu'un automate d'état fini décrit facilement le comportement d'un service Web et sa conversation avec les autres services. Ainsi, lors d'une conversation entre des services Web qui sont représentés par des automates d'état fini, il est très simple de détecter les inter-blocages et les cycles.

Le modèle, des politiques privées utilisé, est une extension des catégories des règles définies dans la plate forme des préférences de confidentialité *P3P*. Ce modèle permet de représenter une politique privée par un ensemble de *tdu* (terme d'utilisation de données). Un *tdu* contient les données privées et un objectif pour le quel ces données privées sont collectées ou fournies, un objectif est un ensemble de droits et d'obligations et un droit peut contenir des obligations. Donc, le modèle utilisé permet aux utilisateurs et aux services Web d'exprimer d'une façon précise leurs préférences et politiques privées, il facilite la comparaison entre les politiques (le niveau de restriction et la violation), il est extensible, et il peut être implémenté facilement.

Nous avons intégré les politiques privées dans les transitions des services Web et dans la requête de l'utilisateur. Une requête est un service Web qui représente la composition attendue, elle contient les préférences et les politiques du client. En se basant sur cette idée, nous avons construit un algorithme qui calcule une composition valide par rapport à une requête (une composition valide doit satisfaire la requête du client et garantir la confidentialité des données privées du client et des services). Pour cela, l'algorithme vérifie, lors de chaque conversation entre le client et un service Web ou entre deux services Web, le niveau de restriction et le non violation entre les politiques concernées.

Le travail réalisé peut être amélioré, pour cela, nous avons met comme perspectives futures les points suivants :

- Enrichir le modèle des politiques privées par autres concepts comme les politiques négatives (les prohibitions) ;
- Introduire dans la modélisation des services Web la notion du temps et le time out ;
- Améliorer l'algorithme de composition, pour qu'il prend en compte la notion de substitution des services (remplacer un service par un autre service équivalent) ;

Bibliographie

[ABG02] Daniel Austin, Abbie Barbir, Sharad Garg. *Web Services Architecture Requirements*. W3C Working Draft 29 April 2002. http://www.w3.org/TR/2002/WD-wsa-reqs-20020429

[AKR 03] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu. *An XPath-based Preference Language for P3P*. IBM Almaden Research Center. 24 Mai 2003, Budapest, Hungary.

[ARI02] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma Scott Williams. *Web Services Conversation Language (WSCL) 1.0*. W3C Note 14 March 2002. http://www.w3.org/TR/2002/NOTE-wscl10-20020314/

[ASS02] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, Sinisa Zimek. *Web Service Choreography Interface (WSCI) 1.0*. W3C Note 8 August 2002. http://www.w3.org/TR/2002/NOTE-wsci-20020808

[BCT04] B. Benatallah, F. Casati, and F. Toumani. Analysis and management of web service protocols. *23rd International Conference on Conceptual Modeling*, November 2004.

[BHL06] Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin. Namespaces in XML 1.0 (Second Edition). W3C Recommendation 16 out 2006. http://www.w3.org/TR/REC-xml-names/

[BOU06] Abdelmalek Boudries. Annotations sémantiques et services Web. Mémoire de magistère, école doctorale ReSyD, Béjaia. Promotion 2005-2006.

[CHA02] Jean-Marie Chauvet. *Service Web avec SOAP, WSDL, UDDI, ebXML…*. Edition Eyrolles 2002.

[CHR04] Luc Clement, Andrew Hately, Claus von Riegen, Tony Rogers. UDDI Version 3.0.2 Spec Technical Committee Draft. OASIS 2004/10/19. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[CHT03] Kishore Channabasavaiah, Kerrie Holley, Edward Tuggle, *Migrating to a service-oriented architecture*. IBM Software Group, 16 Décembre 2003.

[CHO01] Jérôme CHOCHON. Méta représentation EXPRESS des schémas XML pour la validation de documents XML. Laboratoire d'Informatique Scientifique et Industrielle Ecole Nationale Supérieure de Mécanique et d'Aérotechnique.

Cedex 13/12/2001.

[CLM 02a] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori. *A P3P Preference Exchange Language 1.0 (APPEL1.0)*. W3C Working Draft 15 April 2002 http://www.w3.org/TR/2002/WD-P3P-preferences-20020415

[CLM 02b] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, Joseph Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*, W3C Recommendation 16 April 2002. http://www.w3.org/TR/2002/REC-P3P-20020416/

[CMR07] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation 26 June 2007. http://www.w3.org/TR/2007/REC-wsdl20-20070626

[FGPR06] Xavier Fournier-Morel, Pascal Grojean, Guillaume Plouin, Cyril Rognon, *SOA le guide de l'architecture*. DUNOD, Paris 2006. Page 34-37,40-50.

[GBC07] Nawal Guermouche, Salima Benbernou, Emmanuel Coquery and Mohand-Said Hacid. *Privacy-Aware Web Service Protocol Replaceability*. ICWS 2007. IEEE International Conference 9-13 July 2007.

[GPR08] Nawal Guermouche, Olivier Perrin, and Christophe Ringeissen. *A Mediator Based Approach For Services Composition*. International Conference on Software Engineering Research, Management and Applications (SERA'08). Prague, Czech Republic, August 20-22, 2008.

[GHM07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition. W3C Recommendation 27 April 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[HAL05] Halfoune Nadia. *Vers une composition dynamique de services Web*. Mémoire de magistère en informatique, école doctorale ReSyD. Université de Béjaia 2005.

[HFB 04] Patrick C. K. Hung Elena Ferrari and Barbara Carminati. *Towards Standardized Web Services Privacy Technologies*. IEEE International Conference on 6-9 July 2004

[HS04] Richard Hull, Jianwen Su. Tools *for Composite Web Services: A Short Overview*. Bell Labs Research Lucent. Technologies Department of Computer Science UC Santa Barbara

[HUL04] Richard Hull. *Web Services Composition: A Story of Models, Automata and Logics*. Bell Labs, Lucent Technologies Sardinia, Italy September 9-10, 2004.

[JAM05] Sonia JAMAL. *Environnement de procédé extensible pour l'orchestration Application aux services web*. Thèse doctorat de l'Université Joseph Fourier de Grenoble 1. Le 13 décembre 2005.

[JDN] JDN: le journal du net. http://www.journaldunet.com/encyclopedie/definition/433/34/20/business_process_modelling_language.shtml

[KM03] Hubert Kadima, Valérie Monfort. Les services Web : Techniques, démarche et outils : XML, WSDL, SOAP, UDDI, Rosetta, UML. DUNOD, Paris 2003. Page 3-14.

[KON06] Jacqueline Konaté. *Le traitement des erreurs dans l'exécution des services web composés*. Projet de recherche master 2 spécialités système d'information. L'université joseph fourier grenoble cedex 9. juin 2006.

[KT03] Patrick Kellert et Farouk Toumani, Les services Web sémantiques, Rapport de Recherche, LIMOS/RR 03-15, Juillet 2003.

[LAK06] Kachna Lakhdar. Infrastructure pour la création de dépôts de protocoles de service Web. Thèse de magistère ReSyD Béjaia 2006.

[MEL04] Tarek MELLITI. *Interopérabilité des services Web complexes. Application aux systèmes multi-agents*. Thèse doctorat, université Paris IX Dauphine, école doctorale EDDIMO, décembre 2004.

[MEL04] Tarek MELLITI. Interopérabilité des services Web complexes. Application aux systèmes multi-agents. Thèse Doctorat de l'Université Paris IX Dauphine soutenue le 8 décembre 2004.

[MG04] V. Monfort et S. Goudeau. *Web services et interopérabilité des SI*. DUNOD, Paris 2004.

[MIC07] Découvrez SOA, l'architecture orientée services, http://www.microsoft.com/france/biztalk/solutions/soa/decouvrez/presentation.mspx Paru le 20 février 2007.

[ML07] Nilo Mitra, Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation.27 April 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[MRI07] Michael Mrissa. Médiation Sémantique Orientée Contexte pour la

Composition *de Services Web*. Thèse doctorat de l'université Claude Bernard Lyon I (spécialité informatique). 15 November 2007.

[occ06] Audrey Occello. Capitalisation de la sûreté de fonctionnement des applications soumises à des adaptations dynamiques : le modèle exécutable satin. PhD thesis, université de Nice-Sophia, 2006.

[PEL03] Chris Peltz. *Web Services Orchestration and Choreography*. Technical report Hewlett-Packard Company 2003.

[RAY07] Gilbert RAYMOND, SOA : Architecture Logique Principes, structures et bonnes pratiques. SOFTEAM 2007.

[Stal06] Michael Stal - Using Architectural Patterns and Blueprints for *Service-Oriented Architecture*, IEEE Software, Volume 23, 2006.

[THO04] Wanasanan Thongsongkrit. Web Services Description Language (WSDL).2004

[TOU05] Lionel TOUSEAU. *Accords de niveau de service dans les plateformes dynamiques à services*. L'équipe ADELE du laboratoire LSR-IMAG de Grenble. 21 juin 2005.

[WEB01] Venu Vasudevan . 04 avril 2001. http://webservices.xml.com/pub/a/2001/04/04/webservices/index.html

[W3C02] http://www.w3.org/2002/ws/

[W3C] World Wide Web Consortium. Extensible Markup Language (XML). http://www.w3.org/XML/