Ecole Doctorale Galilée

Thèse présentée pour l'obtention le grade de
Docteur de l'Université Paris 13 Nord

Spécialité : Sciences de l'Ingénieur
Mention Réseaux et Technologies de l'information

Tuo ZHANG

Vers une médiation de composition dynamique de Services Web dans des environnements ubiquitaires

Soutenue publiquement le 08/12/2014 devant le jury composé de

Rapporteurs :

Xiuzhen CHEN, *Associate Professor*, *Shanghai Jiaotong University*

Ahmed SERHROUCHNI, Professeur, Télécom ParisTech

Examinateurs :

Saadi BOUDJIT, Maître de conférences, Université Paris 13

Noëmie SIMONI, Professeur, Télécom ParisTech

Directeur de thèse :

Ken CHEN, Professeur, Université Paris 13

Remerciements

Je tiens à exprimer mes plus vifs remerciements à Monsieur Ken CHEN qui fut pour moi un directeur de thèse attentif et disponible malgré ses nombreuses charges. Votre compétence, votre rigueur scientifique et votre clairvoyance m'ont beaucoup appris. Je vous remercie de tout mon cœur non seulement sur ma recherche scientifique mais aussi l'énorme aide sur ma vie personnelle.

J'exprime tous mes remerciements à l'ensemble des membres de mon jury : Ahmed SERHROUCHNI, Xiu zhen CHEN, Saadi BOUDJIT, Noëmie SIMONI et Ken CHEN.

Je remercie tous les membres et ex-membres de l'équipe-projet ANR/UBIS pour le climat sympathique dans lequel ils m'ont permis de travailler. Surtout au responsable de ce projet, Madame Noëmie SIMONI, qui m'a mis sur la route de recherche depuis mon stage de Master 2 et d'avoir permis le financement de mes deux premières années de thèse.

J'adresse toute ma gratitude à tous mes ami(e)s et à toutes les personnes qui m'ont aidé dans la réalisation de ce travail. Je remercie Monsieur Azeddine BEGHDADI le directeur du labo pour m'avoir accueilli dans L2TI et de m'avoir permis de travailler dans d'aussi bonnes conditions. Je souhaiterais remercier tous les membres de L2TI ainsi que Michaël ROI et Eric BUSVELLE de l'Université de Bourgogne. Ils ont été mes collèges pendant mon ATER durant mes deux dernières années de thèse. J'ai beaucoup appris au niveau de l'enseignement sous leur direction.

Mes derniers remerciements iront évidemment à ma famille. Je pense tout d'abord à mes parents sans qui l'enfant que j'étais ne serait pas devenu l'homme que je suis. Ensuite je dois remercier ma femme qui partage ma vie et mon fils qui m'apporte beaucoup de joie durant ma vie de doctorant. Sans eux, j'aurai peut-être finalisé la thèse avec deux ans de moins, mais sans eux, je ne sais pas ce que serait la responsabilité d'un homme.

à mon fils ZHANG Jiaxin dans le paradis,
et à tous ceux que je ne nomme pas, mais qui se reconnaîtront
Paris, le 03/11/2014
ZHANG Tuo

Résumé

De nos jours, l'ouverture à la concurrence stimule les fournisseurs de services à être de plus en plus compétitifs et à attirer de plus en plus d'abonnés afin de faire face aux fortes pressions du marché. L'approche *user-centric,* qui consiste à fournir le plus rapidement possible des services adaptés aux besoins de l'usager, attire de plus en plus d'attention suite à l'émergence de l'environnement ubiquitaire. L'interopérabilité, aussi bien celle entre utilisateur et service que celle entre les services, est favorisée par l'adoption de SOA (architecture orientée service) comme modèle de développement, ainsi que les services Web qui combinent les avantages de ce modèle avec les langages et technologies développés pour Internet. Notamment, la composition dynamique de service Web est considérée comme un atout majeur, qui permet de répondre à des requêtes complexes en combinant les fonctionnalités de plusieurs services au sein d'une session personnalisée.

Les services sont divers et variés et des services similaires pourraient être fournis depuis des plates-formes hétérogènes. Dans un environnement ubiquitaire, les utilisateurs sont mobiles, cette mobilité se manifeste aussi bien à travers les réseaux empruntés que les terminaux utilisés. Ceci entraine une possible mobilité de services (aussi bien en termes de serveur effectif qu'en termes de services équivalents). C'est dans ce contexte dynamique et ubiquitaire, qui est celui choisi par le projet ANR/UBIS (dont cette thèse est partiellement issue), que nous avons mené nos recherches sur le thème particulier de la médiation de composition dynamique de services web. Plus précisément, nous proposons une approche de médiation qui consiste à recenser puis organiser divers services concrets (aussi bien SOAP que RESTFul) pour constituer une panoplie des services abstraits d'une part, puis, à partir desquels, à offrir aux usagers les possibilités de réaliser des services personnalisés selon leur besoins (logique métier) par composition dynamiquement de ces services abstraits et de leur association avec le service concret le plus approprié. Nous avons considéré les trois types de composition de services (SOAP/SOAP, SOAP/RESTful, RESTful/RESTful) dans notre médiation. Selon le souhait de l'usager, cette composition (*Mashup* du coté de médiateur) peut lui être retournée et que l'usager peut invoquer de manière autonome, ou bien le médiateur peut assurer la réalisation du service composé et fournir seulement le résultat final à l'usager. Dans ce dernier cas, les différentes mobilités peuvent être prises en compte par le médiateur, qui s'appuie sur les mécanismes de la communauté virtuelle préconisés par le projet UBIS pour activer les services concrets les plus appropriés correspondant à un service

abstrait. C'est ainsi que nous arrivons à maintenir la continuité de services tout en respectant la QoS demandée, dans le contexte général de l'architecture UBIS.

Mots Clés

User-centric, Ubiquité, Hétérogénéité, Mobilité, Service Web, Composition dynamique de service, SOA, Mashup, Médiation, Session personnalisée

Abstract

Nowadays, high market pressure stimulates service providers to be more competitive in order to attract more subscribers. The *user-centric* approach, which aims to provide adapted services to user's needs, is attracting a great attention thanks to the emergence of ubiquitous environment. The interoperability, either that between users and services or that among services, is favored by the adoption of SOA (Service Oriented Architecture), as a development model as well as the Web services that combine the advantages of this model with the language and development technologies devoted to Internet-based applications. In particular, the dynamic Web service composition is currently the main practice which allows achieving enhanced services, as an answer to increasing complex requests by users for various types of services, by combining functionalities of multiple services within a single and personalized service session.

Indeed, already available services are numerous and of various natures, similar services can be provided by heterogeneous platforms. In a ubiquitous environment, users are mobile, either by changing the access network or by changing the terminal, or even both of them. This leads in turn to potential needs on mobility of services, both in terms of the (physical) server and in terms of the (equivalent) services. It is in this dynamic and ubiquitous context that we have conducted our research. In particular, we focused on the particular topic of mediation of dynamic composition of web services. We proposed a mediation approach which consists in identifying and organizing various concrete services (both SOAP and RESTful) to form a set of abstract services, and, from this knowledge base, to provide users the possibility to realize personalized service session according to their needs through dynamic composition of some of the abstract services and their mapping to best suited concrete services. We considered three types of service composition (SOAP/SOAP, SOAP/RESTful, RESTful/RESTful) in our mediation. Depending on the user's will, this composition (Mashup on the side of the mediator) can be returned to him/her, so that he/she can invoke it autonomously; or the mediator can ensure the realization of the composed services and provide only the final result to the user. In the latter case, the mediator can handle the aforementioned different mobility. This feature is achieved by exploring the mechanism of the *virtual community* to select the most appropriate concrete service corresponding to the abstract service and maintain the continuity of service while respecting its requested QoS. The *virtual community* has been developed within the ANR/UBIS project (to which part of this thesis is related).

Keywords

Ubiquitous, User-centric, Heterogeneity, Mobility, Web service, Dynamic web service composition, SOA, Mashup, Mediation, Personalized session

Tableau des matières

Chapitre 1

Introduction

1.1 Contexte d'étude

Les dernières décennies ont été marquées par le développement rapide des systèmes d'information distribués, et tout particulièrement par la démocratisation de l'accès à Internet. Internet a fourni une étonnante plate-forme de succès pour le développement du réseau et service, en soutenant une variété de technologies incroyables, y compris la publication (le Web), la communication (email, IM, VoIP), le partage des données (P2P, Usenet), le contrôle (les appareils en réseau, SSH, *remote desktop*), etc. Cette évolution du monde informatique a entrainé les nouveaux paradigmes d'interaction entre services.

Parmi tous les événements, l'émergence du Web a eu la plus grande influence. Le Web pourrait être le premier vrai succès du système distribué parmi toutes les tentatives de développement par rapport au langage de programmation, la plate-forme, et les fournisseurs indépendants à grande échelle. Le Web a une influence sur le développement de l'informatique distribuée depuis 1990. Le Web a montré plus d'avantages comme une plate-forme de calcul distribué que d'autres. Il a été redécouvert par *Simple Object Access Protocol* (SOAP) et *Representational State Transfer* (REST) avec la marée de Web 2.0 et *Service-oriented Architecture* (SOA).

L'avènement du web 2.0 apporte un impact significatif sur la facilité d'utilisation de service et l'approvisionnement en encourageant la participation du client final (ou bien l'utilisateur, sachant qu'ici il n'y a pas une grande différence). Un utilisateur est non seulement considéré comme un consommateur de service, mais il est aussi encouragé à être un producteur/contributeur de service. Web 2.0 a révolutionné les méthodes de génie logiciel ainsi que la manière de l'interaction avec les fonctionnalités du logiciel. Ces fonctionnalités ne sont plus propres à une application indépendante, elles sont décomposées en services Web génériques et publiées sur un registre commun pour la promotion du réseau partagé tran-organisationnel, la collaboration, la réutilisation et l'intégration. Pour le web 3.0, c'est du côté

des anonymes qu'il faut regarder. Ils ouvrent la voie à une nouvelle ère d'Internet. Alors que le Web 2.0 désigne un web participatif, intelligent, et social, le Web 3.0 désignera un Web libre, anonyme et activiste.

Le SOA, Architecture orientée services (en anglais, *Service-oriented architecture*), vise à fournir une intégration des services en « couplage lâche » résidant dans un environnement hétérogène. De telles architectures sont dites à « couplage lâche », car les constituants de l'architecture dialoguent au moyen de message. L'adoption de SOA dans le monde informatique a été renforcée par plusieurs implémentations en utilisant les technologies, par exemple SOAP, RPC, REST, Java RMI et WCF etc. Vu populairement comme les éléments constitutifs de SOA, le Service Web (en anglais, *Web Service-WS*) est un genre d'application auto-descriptif et indépendant de la plate-forme qui peut être invoqué sur le Web.

En plus du Web, le *Cloud Computing*, l'*Internet of things* et l'informatique mobile ont également apporté des nouvelles technologies et visions pour l'informatique distribuée et l'internet du futur, et surtout des nouvelles générations de réseaux et de services (NGN/NGS) . Les applications, les services et les objets distribués dans NGN/NGS, sont de plus en plus présents dans l'*Entreprise Application Integration* (EAI), *Business to Business* (B2B), *Business to Consumer* (B2C) et *Peer to Peer* (P2P) ainsi que dans les dispositifs de calculs étendus de serveurs dédiés aux ordinateurs personnels et les appareils mobiles. De plus en plus des services et applications distribués sont utilisés sur Internet au-delà des réseaux hétérogènes (LAN, WAN et les réseaux mobiles etc.). La maîtrise de cette NGN/NGS passe par la capacité des fournisseurs et des opérateurs à rendre leurs services de plus en plus composables, et de les structurer ensuite dans une architecture convergente, autonome, distribuée et trans-organisationnelle.

Dans le cadre de cette thèse, nous nous intéressons tout d'abord à l'interopérabilité entre les services. Les services sont implantés par les fournisseurs, qui mettent à disposition les descriptions de services sous forme de fichier. Les services Web fournissent diverses fonctionnalités qui incluent le paiement en ligne, les prévisions météorologiques, réservations de vol, ou tout simplement la récupération des données. En plus, la vision des services en tant que composants logiciels indépendants met en exergue les possibilités de coordination ou de composition de plusieurs services pour fournir des fonctionnalités avancées. Afin de faciliter l'utilisation et la composition des services, les applications doivent réaliser des interactions de « couplage lâche », c'est-à-dire qu'elles doivent être capables de fournir et d'utiliser les services tout en restant indépendantes les unes des autres, et sans divulguer leur

fonctionnement interne. Cette exigence est satisfaite par l'adoption de protocoles et de langages standardisés qui fournissent un accès uniforme aux services et à leurs descriptions.

Le protocole standardisé SOAP vise à résoudre le problème de l'interopérabilité en RPC orienté objet par la normalisation. Avant SOAP, il était presque impossible de programmer le côté client et le côté serveur d'un programme RPC dans deux langages différents. Le WS a réalisé un grand progrès par rapport à ses prédécesseurs tels que RPC et XML-RPC. Cependant, la question de l'interopérabilité est restée pour les services web développés dans les langages et les plates-formes différentes. Avec le développent du web, pas mal de grandes entreprises/fournisseurs d'internet comme Google, Yahoo, Microsoft, Amazon, Facebook et Twitter ont commencés à utiliser un style non-normalisé, qui est REST pour étiqueter leurs produits. Même s'il y a des débats sur les SOAP versus REST, l'internet de futur devient de plus en plus hétérogène sur de nombreuses dimensions, y compris les services, les données, les réseaux et les objets physiques. Ce sont des défis significatifs pour la vison de l'internet du futur. En particulier, les technologies syntaxiques et sémantiques adaptatives, les normes partagées et la médiation sont exigées pour assurer l'interopérabilité des entités hétérogènes comme les services, les réseaux et les objets. Dans le cadre de cette thèse, les facteurs techniques qui nous intéressent se manifestent dans un contexte technologie où convergent les problématiques des besoins *user-centric*, hétérogénéité, et mobilité.

1.1.1 user-centric

Au cours des années, le contexte technologique a évolué vers une approche *user-centric* touts en passant, comme le montre la Fig 1.1, par différentes approches : la première, appelée *system-centric*, implique que le hardware soit au centre de l'architecture : la deuxième, appelée *application-centric*, considère l'application comme le point focal qui conditionne les demandes de l'utilisateur ; et la troisième, appelée *network-centric*, se concentre sur les réseaux et sur les supports de connexions. Contrairement aux autres approches, le contexte *user-centric* met l'utilisateur au centre de l'architecture, et impose ainsi aux fournisseurs et aux opérateurs d'adapter leurs services, leurs produits hardware et leurs réseaux de façon à être toujours « au service » de cet utilisateur. Ce dernier n'est donc plus obligé de se plier aux différentes contraintes de traitement et de connexion. Il réclame l'accès à n'importe quel service, n'importe où, n'importe quand et par n'importe quel moyen sans aucune restriction technique et/ ou économique.

Dans le cadre de cette approche *user-centric*, tout utilisateur désire établir dynamiquement une session unique et personnalisée. La session est vue comme une mise en relation

temporelle entre l'utilisateur et l'ensemble du système (depuis son terminal jusqu'à la plate-forme de services, tout en passant par les composants des réseaux d'accès et du réseau cœur). Elle est personnalisée, ce qui permet à tout utilisateur de composer sa propre session de services, à l'aide de différent type de service (Telco, Web, et IT (*Information Technology*)), et selon une logique adéquate à son contexte ambiant, ses préférences fonctionnelles, ainsi que ses préférences non-fonctionnelles (la QoS demandée). Dans ma thèse, nous nous concentrerons sur le service Web avec SOAP et RESTful. La session est unique, ce qui nécessite sa maintenance en temps réel, d'une manière dynamique et transparente vis-a-vis de l'utilisateur. Ceci permet ainsi d'avoir une session par utilisateur tout au long de son déplacement spatial et temporel, et malgré les modifications qui sont dues à la personnalisation de services.

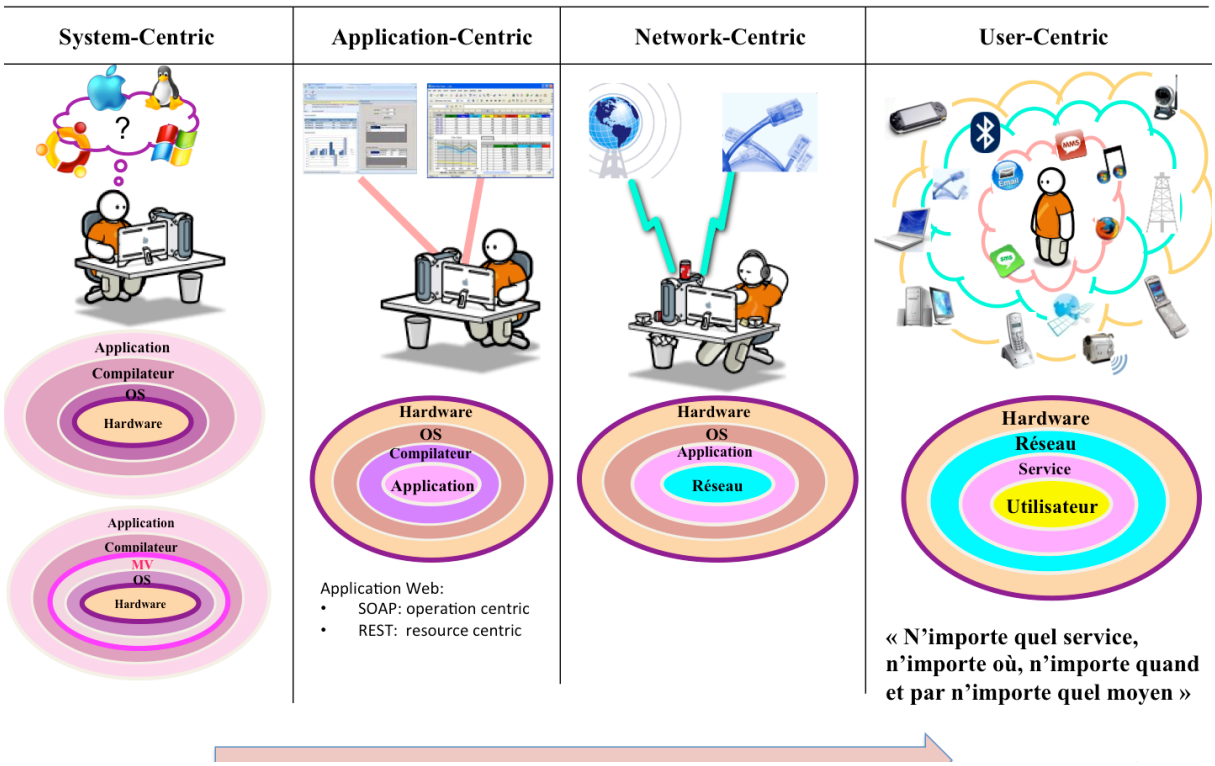


Fig. 1.1 Evolution du contexte technologique vers l'approche *user-centric*

1.1.2 Hétérogénéité

L'hétérogénéité est une caractéristique de noyau pour l'internet de futur qui vise à englober tous les types de ressources et les présenter comme services. Ces services peuvent être hébergés sur les plates-formes et couvrir toutes les ressources web des machines fixes aux équipements dispositifs sans fils. Cette hétérogénéité peut être donc divisée en deux dimensions:

1) Hétérogénéité des plates-formes Middleware : elle peut émerger en raison de ressources différentes. Il faut avoir un mécanisme intégré dans la plate-forme Middleware qui peut nous fournir l'interopérabilité nécessaire.

2) Hétérogénéité de service : en fait, les services sont développés en utilisant les modèles conceptuels et technologies différentes. Il faut réaliser la composition de service d'une manière indépendante du langage de programmation, du fournisseur, des systèmes d'exploitation et du modèle de donnée etc.

1.1.3 Mobilité

En effet, un enjeu essentiel pour l'internet du futur réside dans la conception explicite de protocole pour l'équipement sans fils, étant donné que la majorité des entités connexes sont désormais mobile. La mobilité est une notion plus concrète que l'adaptabilité. Dans le cadre de l'approche « *Every Service for Everyone, Anywhere, Anytime and Anyhow* », tout utilisateur désire conserver sa session de bout-en-bout. Elle lui permet d'accéder d'une manière continue à ses services, quel que soit terminal, le réseau d'accès, le réseau cœur, et le fournisseur de services choisi. De plus, il réclame une session continue qui prend bien en considération ses préférences fonctionnelles et sa QoS de bout-en-bout demandée. Dans ce nouveau contexte NGN et *user-centric*, les exigences au niveau mobilité deviennent des défis importants qui augmentent la complexité de la gestion des ressources. Afin de dépasser ce problème de continuité de session, nous devons préciser les différents évènements qui peuvent surgir durant la session de l'utilisateur et peuvent impacter sa continuité. Dans cette thèse nous considérons deux types de mobilité:

1) Mobilité de l'utilisateur : elle représente le passage d'un utilisateur d'un terminal à un autre. En effet, de nos jours, tout utilisateur se trouve dans un environnement hétérogène qui contient des terminaux très diversifiés. Par conséquent, l'utilisateur est devenu capable d'accéder à ses services tout en passant d'un terminal à un autre. Pour cela, ce type de mobilité spatiale pose un défi vis-à-vis de la continuité de session.

2) Mobilité du service : elle est dû essentiellement au principe d'ubiquité de services qui consiste à avoir des services ubiquitaires. C.à.d. des services ayant la même fonctionnalité et une QoS équivalente. Par conséquent, le principe de mobilité de services représente le remplacement d'un service demandé par l'utilisateur par un autre service qui lui soit ubiquitaire. D'après l'approche Trans-organisationnelle que nous supportons, le service remplaçant peut appartenir à un autre fournisseur de services. En effet, ce type de mobilité devient un défi de plus en plus important à

gérer, surtout avec le *booming* du marché des services et l'augmentation de la concurrence qui incite les fournisseurs à déployer des services ubiquitaires pour mieux répondre aux besoins de leurs utilisateurs.

Dans ce cas-là, l'architecture pour concevoir l'adaptation dynamiquement des compositions doit être conçue pour être plus efficace. Les stratégies existantes utilisées pour la composition dynamique et adaptée devraient être revues et étudiées dans ce contexte. Dans un scénario général, où les nœuds peuvent être mobiles, de nouvelles approches sont nécessaires pour coordonner la composition de service. La coordination doit être adaptable aux changements de topologie de l'environnement mobile et gérer la mobilité, ainsi que les déconnexions du réseau et d'autres types de défaillances de nœuds ou de service. La coordination doit examiner les modes de mobilité, la durée de vie de la batterie de la plate-forme, la tolérance aux fautes et la fiabilité. Messages de diffusion, par exemple, devraient être évités car ils imposent généralement une charge lourde au réseau. Les appareils mobiles sont souvent incapables de traiter les demandes à distance en raison de la surcharge du système ou des déconnexions au niveau du réseau. La composition et les protocoles de coordination devraient être tolérants à de telles défaillances et conduire à une indisponibilité des ressources.

1.2 Motivations

Les avantages de service web, tels que leur architecture « couplage lâche » et l'interopérabilité standardisée attirent de plus en plus des intérêts dans la recherche d'académique et de l'industrielle. Dans le service web, les fonctionnalités implémentées par les processus métiers internes sont déployés et exposés comme le service qui peut être découvert et accessible via le web. Un processus métier est une transformation qui produit une valeur ajoutée tangible à partir d'une sollicitation initiale. Il est divisé en activités (ou tâches), réalisées par des acteurs (humains ou automatiques) avec l'aide de moyens adaptés, qui contribuent chacune à l'obtention du résultat escompté. Par exemple, l'interaction entre le client et le service relie généralement sur le message *binding* de SOAP/HTTP [Berners-Lee *et al.* 1996, Box *et al.* 2000, Dryden and Session 1998], le client, une application du métier logique (par exemple, *e-Commerce workflow*) invoque le service web en envoyant un message SOAP [Curbera *et al.* 2002, Curbera *et al.* 2005] avec la requête de service attachée. Le service web reçoit et analyse le message SOAP. La communication entre eux est garantie par l'interopérabilité normalisée, et aucun tiers de Service Broker ou un coordonnateur n'est nécessaire. Par conséquence, l'intégration des services autonomes et indépendants est obtenue

à un faible cout par rapport aux technologies conventionnelles [Christensen *et al.* 2001, Ferguson *et al.* 2003].

Néanmoins, même avec les avantages mentionnés ci-dessus, un service web isolé n'est pas une solution magique à tous les problèmes des applications distribuées, surtout avec le développement de Web 2.0/3.0, c'est de plus en plus difficile de répondre aux divers besoins de l'utilisateur seulement par certain service spécifique ou une composition statique de service. Donc, la composition dynamique de service est l'une des questions les plus cruciales à régler.

On peut comprendre la composition dynamique de service par d'un scénario normé « mon voyage ». Les services dans « mon voyage » sont présents dans la figure 1.1 ci-dessous.

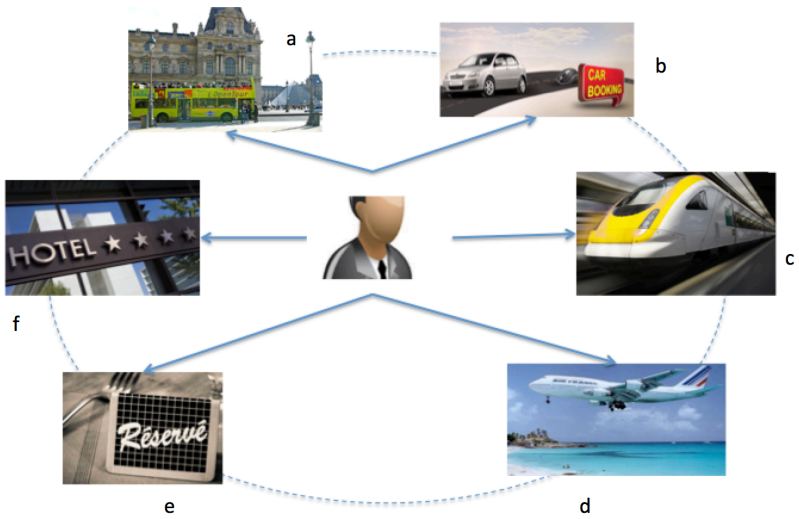


Fig. 1.2 : Scénario « mon voyage »

Considérant le scénario « mon voyage » dans la figure 1.2, nous supposons que les services en ligne sont *(a)* réservation le bus, *(b)* la voiture, *(c)* le train, *(d)* l'avion, *(e)* le restaurant et *(f)* l'hôtel, l'utilisateur peut utiliser les services un par un pour atteindre son objectif de planification de son voyage. Par exemple, si cette personne veut voyager de Rome à Paris, il peut d'abord réserver le billet d'avion en utilisant le système de réservation des vols. Ensuite, il peut réserver un billet de bus/voiture/train en fonction de ses exigences. Il peut également réserver une chambre d'hôtel en utilisant la réservation de l'auberge. Mais c'est moins efficace. L'utilisateur a besoin donc d'un service qui contient tous les services mentionnés ainsi que les relations et les dépendances implicites entre eux (par exemple, on doit réserver les transports avant la réservation de l'hôtel). La situation la plus efficace c'est que

l'utilisateur puisse profiter de l'un des services ou bien des multiples fonctionnalités des services en même temps. Si une personne veut planifier un voyage du point de départ à la destination, alors elle peut bénéficier de toutes les réservations nécessaires, au même endroit, par l'accès à un service de package, puisque ce que l'on attend de ce service, ce sont toutes les fonctionnalités des différents services indépendants et de leurs relations implicites. Par exemple, les compositions dynamiques de service peuvent être *(d,c)→(a,b)→(f,e)*, c'est-à-dire que l'on doit normalement réserver le train *(c)* et l'avion *(d)* avant la réservation du restaurant *(e)* et de l'hôtel *(f)*, et que cette dernière doit être faite avant la réservation du bus *(a)* et de la voiture *(b)* d'après les relations implicites, la séquence de *(c,d),(a,b)* et *(e,f)* dépend du cas spécifique et des exigences de l'utilisateur. Le processus métier sera donc plus simple et efficace en utilisant ce service de package illustré dans la figure 1.3 qui est une composition dynamique de service mentionnée plus tôt que l'utilisation de plusieurs services séparés comme illustrée dans la figure 1.2.

Plus généralement, les technologies de Web 2.0/3.0 adoptent la composition de service comme un mécanisme de prestation de service de base qui permet la création et l'exécution de services complexes en intégrant les services distribués et autonomes.

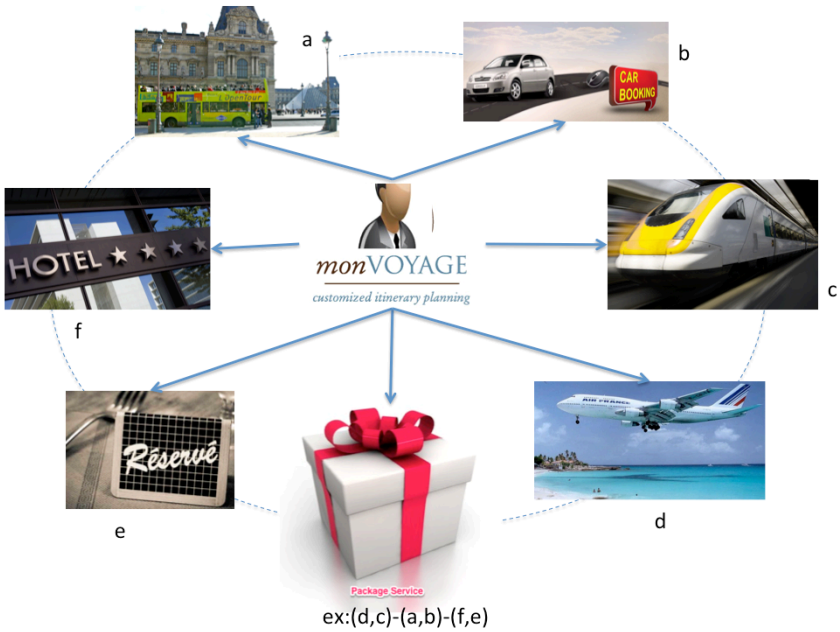


Fig. 1.3 : Scénario « mon voyage »

1.3 Nos objectifs

Après avoir décrit les différents contextes qui guident les avancements technologiques d'aujourd'hui et ceux de l'avenir, nous délimitons, comme le montre la figure 1.4, notre

périmètre d'étude dans cette thèse à l'intersection des trois contextes déjà évoqués : l'approche *user-centric* , l'hétérogénéité et la mobilité.

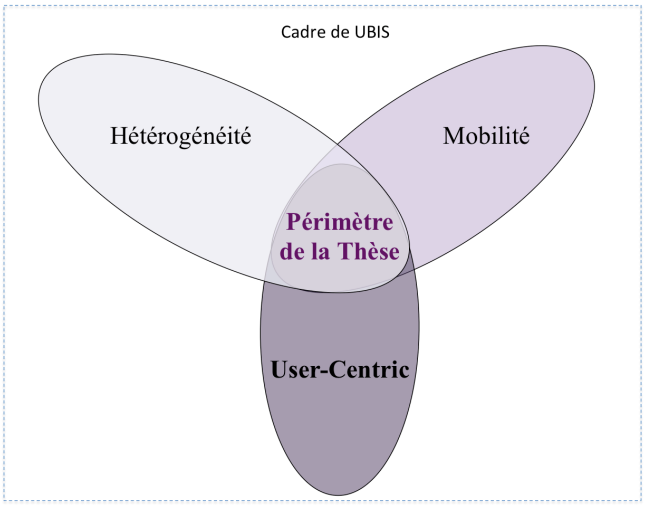


Fig.1.4 Périmètre de la thèse

La figure 1.5 illustre notre positionnement dans SOA.

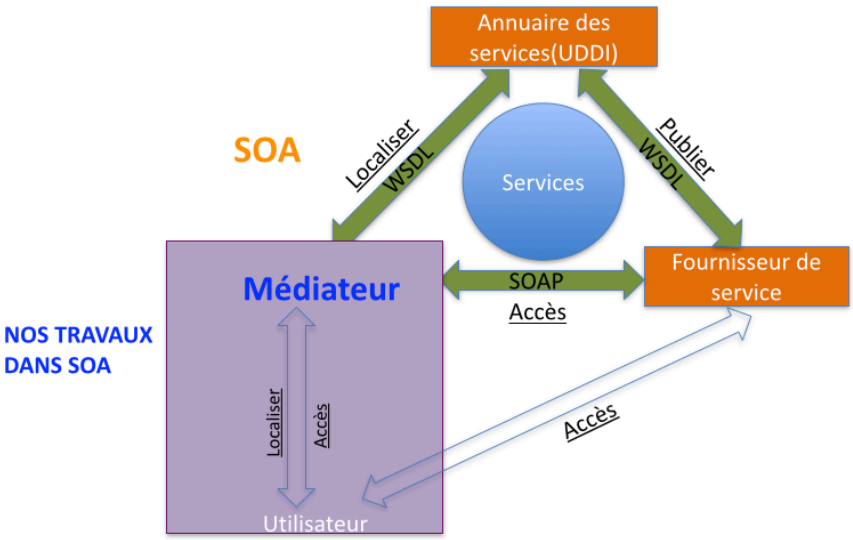


Fig. 1.5 Positionnement de nos travaux dans SOA.

Comme nous l'avons vu, il y a trois rôles dans le paradigme de SOA : consommateur de service, fournisseur de service et l'annuaire de service. Le fournisseur de services crée un service et fournit ses descriptions fonctionnelles à l'annuaire. L'annuaire est une entité centrale de fournisseur de services. Du point de vue des consommateurs, même si cela peut être physiquement distribué sur plusieurs plates-formes, les consommateurs de services doivent effectuer une recherche dans l'annuaire afin de trouver leurs services souhaités. Ils peuvent ensuite invoquer ces services. C'est-à-dire, que c'est cet annuaire qui joue le rôle d'intermédiaire entre eux. Les fournisseurs y enregistrent leurs services, et les clients y

cherchent le service satisfaisant leurs besoins. On a donc proposé un médiateur du coté du consommateur. Le médiateur devient le consommateur aussi bien du fournisseur que de l'annuaire, alors que l'utilisateur dans nos travaux peut communiquer avec notre médiateur ou bien avec le fournisseur de service dans SOA.

Les travaux existants ont une limite sur la composition de service qui répond aux exigences des utilisateurs. Les recherches dans ce domaine, aujourd'hui se trouvent soit orientées « opération centrique (WSDL/SOAP) », soit orientées « ressource centrique (RESTful) ». En raison de la limitation de l'approche existante mentionnée, nous allons proposer une médiation de composition dynamique de service web orientée *user-centric*.

Plus précisément, nous allons modéliser et affiner les exigences de l'utilisateur. Les utilisateurs doivent profiter des fonctionnalités des services à travers notre médiation vers une production de l'application. Ce modèle architectural de médiation va nous permettre de composer dynamiquement les services pour mieux répondre aux exigences de l'utilisateur et aux attentes des fournisseurs. L'approche de médiation orientée *user-centric* proposée doit accepter les services web indifféremment (SOAP et RESTful).

Une limite existe aussi, entre les travaux, qui tiennent au fait qu'ils ne prennent pas en compte l'adaptation au contexte et à la mobilité lors de la composition de service. Cette particularité permet d'assurer la flexibilité de notre solution. Notre architecture doit permettre aux utilisateurs d'avoir une expérience de service ubiquitaire et d'avoir une continuité de service pendant leur session. Selon les profils de service, le médiateur pourrait appeler le service le mieux adapté à la localisation de l'utilisateur pour assurer l'adaptation du contexte (ou l'ubiquité) et de la mobilité. Et cela nous permet facilement d'avoir les services virtuels et l'adaptation du contexte via la gestion de la mobilité réalisée dans UBIS.

1.4 Les Contributions

Je présente un modèle architectural de médiation pour la composition dynamique de service orientée *user-centric*. L'approche diffère de « opération-centrique » et de « ressource-centrique ». Nous considérons de façon différente les compositions des services de SOAP et des services RESTful, à travers la médiation. Selon les exigences de l'utilisateur, la médiation de service web peut nous fournir l'application de l'utilisateur selon deux types : 1) la médiation fournit un service composé comme une orchestration de service ; 2) l'utilisateur peut personnaliser et planifier les démarches de composition de service abstrait à travers notre médiation, la dernière retournant un environnement d'application de mashup pour l'utilisateur.

En considérant les défis de la mobilité, l'utilisateur peut profiter les meilleurs services fournis tout au long de sa session via les communautés virtuelles réalisées dans UBIS, donc notre médiation fournit à l'utilisateur une composition dynamique qui a la capacité de gestion de la mobilité. La programmation et un complément d'expérimentation montrent la faisabilité de notre architecture de médiation de service web.

1.5 La structure de thèse

Ma thèse est structurée de la manière suivante :

Le chapitre 2 dresse un état de l'art des travaux par rapport aux technologies de base sur le Service Web et les composants de service, on compare les avantages et les incontinents des Service SOAP et REST puis trouvé une exigence pour notre proposition. On discute des technologies et des approches de la composition de service web en précisant les détails sur les modes différents sur la composition de service. Dans le chapitre 3, on propose notre première proposition qu'il s'agit de modèle architectural de médiation et on précise les fonctionnalités des modules reliés. Dans le chapitre 4, on parle notre deuxième proposition sur le Mashup basé sur notre médiation. On prouve la faisabilité et l'expérimentation pour notre architecture dans la partie 5. Dans le chapitre 5 je parle de certaines extensions et applications fonctionnelles basées sur notre médiation de service. Le dernier chapitre termine ce mémoire par une conclusion qui discute les apports de notre travail, ainsi que les perspectives de recherche envisagées. Certains contenus de ma thèse ont été publiés dans mes publications.

Chapitre 2

État de l'art

2.1 Les services Web et ses APIs concernées

La technologie Web commence à être utilisée dans les années 1990 et connut très vite un grand succès. Le Web 1.0 est le Web constitué des pages web liées entre elles par des hyperliens, il a été créé au début des années 1990. Cette technologie, Web 1.0 a ensuite évolué vers Web 2.0. Le Web 2.0 s'est généralisé avec le phénomène des blogs, des forums de discussion agrégeant des communautés autour de sites internet et enfin avec les réseaux sociaux. L'expression Web 3.0 est utilisée en futurologie à court terme pour désigner l'internet qui suit le Web 2.0 et constitue l'étape à venir du développement du World Wide Web. Son contenu réel n'est pas défini de manière consensuelle, chacun l'utilisant pour désigner sa propre vision de l'internet du futur.

Un service web est un programme informatique avec des technologies web qui permet de communiquer et d'échanger des données entre application et systèmes hétérogènes dans les environnements distribués. Il s'agit d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine, de manière synchrone ou asynchrone.

Le concept de Service Web a été précisé et mis en œuvre dans le cadre de W3C particulièrement avec le protocole SOAP, cependant, le concept s'enrichit avec l'approfondissement des notions de ressource et d'état, dans le cadre du modèle de REST, et l'approfondissement de la notion de service, avec le modèle de SOA.

Dans cette partie, nous commençons nos recherches par étudier les technologies et le développement du Web [Box, Ehnebuske, Kakivaya, Layman, Mendelsohn, Nielsen, Thatte and Winer 2000, Fallside and Walmsley 2004, Fielding *et al.* 1999, Fielding 2000, Waldo *et al.* 1997] [Curbera, Duftler, Khalaf, Nagy, Mukhi and Weerawarana 2002, O'reilly 2007]. Ensuite, on analyse les caractéristiques et les fonctionnalités de SOA [Newcomer and Lomow 2004] [Erl 2005] et le paradigme de conception de SOA qui naturellement porte sur la capacité de la composition flexible[Zhao *et al.* 2012]. En plus, on étudie certaine application autour de technologie Web, par exemple le « *Mashup* » [Lawton 2008] [Liu *et al.* 2007, Yee

2008, Yu *et al.* 2008] et les architectures & les méthodes des intégrations des services dans le *Cloud Computing* [Hung *et al.* 2011, Miller 2008] [Vaquero *et al.* 2008, Zhang *et al.* 2010] [Frisch 2009] [Lenk *et al.* 2009] [Langenhan 2013] etc.

2.1.1 Le développement du Web et ses technologies

Le Web 2.0 est l'évolution du Web vers plus de simplicité (ne nécessitant pas de connaissances ni techniques ni informatiques pour les utilisateurs) et d'interactivité (permettant à chacun, de façon individuelle ou collective, de contribuer, d'échanger et de collaborer sous différentes formes). Dans le même temps, l'informatique distribuée a été largement développée dynamiquement par les langages et les paradigmes de programmations populaires. Un « Déjà vu » a été observée [Waldo, Wyant, Wollrath and Kendall 1997]: Le *Remote Procedure Call* (RPC) est apparu dans les années 1980. Avec les années des langages de procédure, et ensuite un RPC orienté objet est proposé dans les années 1990 après le développement de la programmation orienté objet (OOP). Le développement de méthodes de représentation de l'information influe également sur les technologies du web et de l'informatique distribuée. Par exemple, le SOAP est basé sur *Extensible Markup Language* (XML) ainsi que sur les normes et les techniques de web de l'informatique distribuée. La figure 2.1 illustre les événements importants liés à des langages de programmation. Dans les années 2000 les notions de blog, de wiki en 2001, et le réseau social (Myspace en 2003, Facebook en 2004) deviennent populaires. Le contenu généré par les utilisateurs se répand (Youtube en 2005, Twitter en 2006). L'expression Web 2.0, largement popularisé au milieu des années 2000, désigne cette transaction dans le flux de l'information et la manière d'utiliser le web. Le succès de Web 2.0 a conduit de nombreuses personnes à appeler Web 2.5, 3.0, et leur vision du Web de l'avenir. Alors de plus en plus des paradigmes de technologies web, les normes et les implémentations sont apparus.

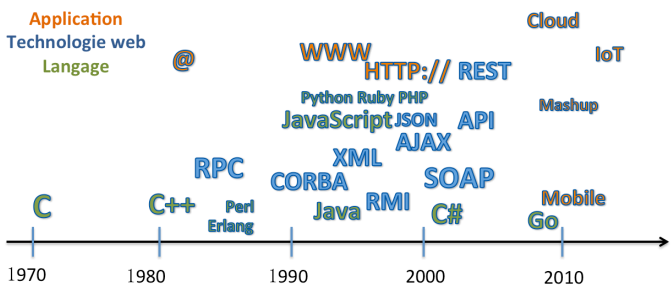


Fig. 2.1: Le développement du Web et ses technologies

L'expression « Web 2.0 » désigne l'ensemble des techniques, des fonctionnalités et des usages du *World Wide Web* qui ont suivi la forme originelle de web. Elle est utilisée par *Dale Dougherty* en 2003, diffusée par *Tim O'Reilly* [O'reilly 2007] en 2004 et consolidée en 2005 avec l'exposé de position « *What is Web 2.0* » s'est imposée à partir de 2007. Dans [o'Reilly 2009] l'auteur parle d'intelligence collective de services ou d'applications en ligne. Web 2.0 est alors redéfinie comme non seulement un médium (où les sites Web sont autant d'îlots d'informations) mais aussi comme une plate-forme. Les données sont considérées comme des connaissances implicites. Web 2.0 est donc en fait un socle d'échanges entre les utilisateurs. Web 2.0 permet aux utilisateurs d'interagir et de collaborer les uns avec les autres. Dans le Web 2.0, l'internaute devient acteur en alimentant les sites en contenu, comme les sites de réseaux sociaux, les blogs, les sites de partage vidéo, services hébergés (*hosted service*) ou de manière collaborative avec les *wikis*.

Les sites internet 2.0 permettent donc aux utilisateurs de faire plus que de retirer de l'information. En augmentant ce qui était déjà possible de faire avec le Web 1.0, ils apportent aux utilisateurs des nouvelles interfaces et des nouveaux logiciels informatiques. Les utilisateurs peuvent maintenant apporter des informations aux sites Web 2.0 et avoir le contrôle sur certaines de celles-ci. Le Web 2.0 se démarque par un certain nombre de traits distinctifs:

-- Ergonomie utilisateur plus conviviale ;

-- Architecture technique plus souple ;

-- Interopérabilité accrue entre les services et les applications qui le composent ;

--Développement des sites collaboratifs dans lesquels les internautes son mis à contribution.

Les internautes deviennent alors co-auteurs (« des web acteurs ») ou sont traités comme des co-développeurs.

En résumé, les Web 2.0 a marqué le développement d'un certain nombre de technologies et de langages dont voici les principaux éléments :

-- AJAX (*Asynchronous JavaScript and XML*) : désigne l'utilisation conjointe de technologie pour le développement d'application web ;

-- Microformats : désigne une manière de formater les données présentes dans du contenu HTML grâce à l'emploi de métadonnées ;

-- Interface riche (RIA ou « *Rich Internet Application* ») : désigne l'emploi de techniques comme Flash, Javascript et ActiveX pour créer des sites web dans lesquels des applications peuvent s'exécuter sans que l'utilisateur ait besoin de les installer ;

-- RSS (*Rich Site Summary*) : une branche de formats XML utilisé pour la syndication de contenu Web.

-- API (*Application Programming Interface*) : ensemble de fonctions, procédures ou classes mis à disposition des développeurs par un système d'exploitation ou un service web et qui va leur permettre de créer leurs propres applications : les mashups ;

-- Services de réseautage social : applications web servant à relier des amis, à créer des communautés virtuelles et qui posent le problème de notre visibilité ;

-- *Folksonomies* : système collaboratif de classification de ressources à l'aide des mots clés (ou tags, en anglais).

L'expression Web 3.0 est utilisée en futurologie à court terme pour désigner le Web qui suit le Web 2.0 et constitue l'étape à venir du développement de *World Wide Web*. En théorie et si le Web 2.0 est une plate-forme de connexion entre les documents, le Web 3.0 sera capable de connecter les données entre elles. Il est généralement admis qu'une solution Web 3.0 doit montrer certaines caractéristiques comme mobilité, universalité et accessibilité, c'est-à-dire qu'elle doit être indépendante de tout système d'exploitation, et de tout matériel (fabricant, marque, logiciel, ou de plugin) et strictement en conformité avec le W3C, ce qui permet de rendre d'autres logiciels accessibles à l'aide de micro format et ouverts aux bases de données diverses. La définition précise d'une application Web 3.0 n'est pas vraiment définie. L'expression est employée par tous les spécialistes pour expliquer ce que sera selon eux la prochaine étape de développement du Web. Les deux thèses dominantes sont de considérer le Web 3.0 comme l'internet des objets, qui émerge depuis 2008, l'autre thèse dominante est d'en faire le web sémantique.

Pour nous, imaginons que vous soyez à la recherche d'un restaurant, en région parisienne, qui propose des spécialités kurdes, offrant une atmosphère « typique », un service irréprochable et ce, pour une modique somme, une machine sera alors capable de vous servir sur un plateau l'adresse tant convoitée. C'en est donc terminé des longues consultations des multitudes pages de commentaires et de critiques laissées par les internautes sur l'endroit machin et le resto truc. L'application « Web 3.0 » est devenue intelligente et elle est maintenant capable d'analyser des multitudes de contenus pour en extraire les informations (« les sens ») qui correspondent parfaitement à vos attentes.

Du point de vue des technologies utilisées, les applications Web, Web 2.0/3.0 utilisent souvent les interactions du type de *machine-based* comme REST et SOAP. Les services sont « *exposés* » aussi bien à travers des APIs propriétaires que des APIs standard (par exemple, pour l'affichage sur un blog ou notification d'une mise à jour de blog). La plupart des

communications à travers des APIs impliquent XML, JSON et toute une série de spécification des services web.

2.1.2 Service Web

L'utilisation de Services Web connaît une popularité toujours grandissante auprès des développeurs de Web. Service Web est définie par le W3C comme « un système logiciel pour supporter les interactions de machine à machine au-dessus d'un réseau ». Il est un programme informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.

Un Services Web est donc invoqué à travers son API qui est accessible à travers un réseau, le service invoqué est exécuté à distance sur le serveur hébergeant le service demandé. Les interfaces sont entre les demandeurs et les fournisseurs de services. Service Web est en fait un premier dénominateur commun pour la création de service. Il existe plusieurs technologies derrière le terme service web :

1) Les services Web exposent les fonctionnalités sous la forme de services exécutables à distance. Leurs spécifications reposent sur le standard de SOAP [Box, Ehnebuske, Kakivaya, Layman, Mendelsohn, Nielsen, Thatte and Winer 2000] et *Web Service Description Langage* (WSDL) [Chinnici *et al.* 2007, Christensen, Curbera, Meredith and Weerawarana 2001] pour transformer les problématiques d'intégration héritée du monde Middleware en objectif d'interopérabilité. Ce genre de service est vu comme un genre de « opération centrique » qui a les opérations exposées dans le système. WSDL les services communiquent par échange de messages entre des terminaisons, constituées d'un ou plusieurs ports, chacun doté d'un type. Un fichier WSDL se lit du bas vers le haut. Il peut être écrit à la main, mais dans la plupart des cas il est automatiquement généré par les environnements de développement SOAP. Dans ma thèse, on travail tout d'abord avec le service web de ce type de SOAP/WSDL.

2) En outre, Les services web de type *Representational state transfer* (REST) exposent entièrement des fonctionnalités comme un ensemble de ressources (URI) identifiables et accessibles par la syntaxique et la sémantique du protocole HTTP. Les Services Web de type REST sont basés sur l'architecture du web et ses standards de base : HTTP et URI. Le WS RESTful est donc un style de « ressource-centrique » et vu selon la perspective de ressource.

Nous décrivons les principales caractéristiques et les protocoles des deux types dans la suite.

2.1.2.1 SOAP et WSDL

SOAP, WSDL et UDDI a été pris en charge par les fournisseurs majeurs de la solution informatique comme IBM, SUN, Microsoft et Oracle. Ce sont les trois standards qui construisent l'architecture de Service Web. Il existe plusieurs environnements qui supportent ces trois standards et par la suite qui supportent l'implémentation des WSs tels que :

- Serveurs HTTP IIS de Microsoft avec le Framework .NET (http://www.iis.net/)
- JAX-WS (http://jax-ws.dev.java.net/)
- Axis et le serveur Tomcat (http://ws.apache.org/axis/)
- Oracle WebLogic (http://www.oracle.com/appserver/weblogic/weblogic/weblogicsuite.html)
- WebSphere d'IBM (http://www-01.ibm.com/software/websphere/)
- JBoss (http://www.jboss.org/)

Ils les ont donc considéré comme et une solution globale pour l'intégration du système dans un environnement hétérogène [Pautasso *et al.* 2008, Richardson and Ruby 2008]. Sa technologie de pile comprend une série de normes basées sur XML et les protocoles définis par le *World Wide Web Consortium* (W3C) et *Organization for the Advancement of Structured information Standards* (OASIS) pour assurer l'interopérabilité. Les différentes normes et les protocoles sont les suivants:

--Protocole de Description: WSDL décrit les fonctionnalités (interfaces publiques et les opérations disponibles de WSs) et les messages XML[Part 2001] échangés entre les fournisseurs de services et clients. Une interface WSDL est un ensemble des opérations prises en charge, une opération est définie par les messages XML entrants et sortants. Les types de messages sont généralement définis à l'aide d'un schéma XML[Fallside and Walmsley 2004]. Services dans un fichier WSDL sont définis comme un ensemble des *endpoints* ou les ports. Chaque port associe à une adresse URI avec une liaison, où les opérations sont reliées à un protocole de communication concrètement (généralement SOAP).

Protocole de Communication : SOAP est un protocole de communication pour échanger les messages entre les applications différentes. C'est un protocole de messagerie pour WSs. Il relie les autres protocoles de couche application (HTTP généralement, mais c'est aussi possible pour les autres protocoles application comme FTP, SMTP etc.) afin de transférer les messages. Ces protocoles de couche application sont utilisés comme options pour le protocole de transport sous-jacent dans SOAP. Un document SOAP est essentiellement un document XML avec les informations suivantes dans son élément d'enveloppe de XML:

(1) Un élément de l'entête optionnel contient les informations d'en-tête, qui contient les informations spécifiques à l'application tels que l'authentification, la fiabilité, etc.

(2) Un élément de corps contient des informations de requête et de réponse, qui sont les charges utiles du message SOAP. Comme WSDL, XML Schéma est utilisé pour décrire la structure du message SOAP. XML et XML Schéma basé sur la représentation permet au protocole SOAP d'être interopérable entre les plates-formes hétérogènes.

(3) Un élément optionnel de la faute contient les erreurs et les informations d'état.

--Protocole de Publishing et découverte : *Universal Description*, *Discovery and Integration* (UDDI) [Curbera, Duftler, Khalaf, Nagy, Mukhi and Weerawarana 2002] est un registre (*registry*) basé sur le XML pour les fournisseurs de services , il représente la liste des services publiés et leurs descriptions, il permet à l'utilisateur de découvrir les services dont il a besoin. Il y a trois composants principaux dans un UDDI business registration

(1) Pages Blanches qui répertorient l'adresse et permettent d'obtenir les autres informations sur le fournisseur de services;

(2) Pages Jaunes qui répertorient les WSs disponibles pour les catégories industrielles basées sur des taxonomies standard;

(3) Pages Vertes qui donnent des informations techniques détaillées sur le WS particulier, qui peut généralement être rempli à partir des descriptions WSDL.

--Protocoles d'extensions: les trois protocoles ci-dessous constituent le fondamental de la construction les WSs WSDL/SOAP qui est opération-centrique. Mais dans les environnements d'applications plus complexes, d'autres exigences avancées (tels que les transactions atomiques, la sécurité, la confiance, la messagerie fiable) doivent également être accueillies. Comme ceux que l'on a vu dans la figure 2.2, OASIS a standardisé un ensemble de protocoles de WS-*[Andrieux et al. 2004]. Par exemple, *WS-AtomicTransaction* [Tai et al. 2004] est pour soutenir les transactions atomiques dans les interactions des WSs ; protocole de *WS-security* [Hirsch 2006] fournit un moyen pour appliquer le mécanisme de sécurité aux services Web à l'aide de *Security assertion markup language* (SAML)[Hirsch 2006], *Kerberos* [Neuman et al. 2005], et les formats de certificats comme *X.509* [Tuecke et al. 2004]. *WS-Trust* [El Maliki and Seigneur 2007] permettent aux applications de construire les échanges de messages SOAP avec confiance et le protocole de *WS-ReliableMessaging* [Pautasso, Zimmermann and Leymann 2008] qui permet aux messages SOAP d'être livrés de façon fiable entre les applications distribuées.

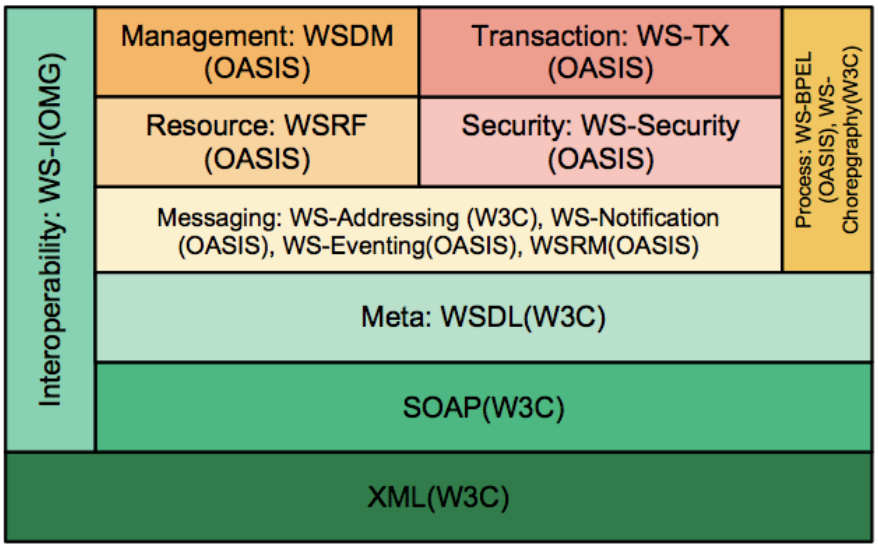


Fig. 2.2 : La spécification de Service Web et leurs dépendances

La plupart des implémentations de SOAP fournissent deux outils, l'un pour générer du code à partir de WSDL et l'autre pour générer WSDL à partir du code, ils sont appelés *wsdl2code* et *code2wsdl*. Ces outils permettent deux modèles de développement du service Web: Code-première ou bien WSDL-première. Dans la pratique, la plupart des services Web ont été développés de manière code-première, et puis les fichiers WSDL ont été générés. La rédaction d'un document WSDL correcte est plus difficile que d'écrire un programme correct. Les documents WSDL générés par la machine ont été trop compliqués pour être interprétés par les programmeurs, et par conséquent, ils sont rarement lisibles pour les humains.

2.1.2.2 L'architecture du Web

Le Web est à ce jour vu comme le plus grand système et « *a network of information resources* » [Raggett *et al.* 1999]. Une ressource est une clé d'abstraction du Web [Fielding 2000]. Pour identifier les ressources, sous la forme d'un *Uniform Resource Locator* (URL), à chaque niveau hiérarchique peut être résolus par une autorité de nommage correspondante. Par exemple, le nom de domaine peut être résolus par un *Domain Name System* (DNS), et le chemin d'une ressource peut être résolu par le serveur hébergé de la ressource. Une représentation des ressources est les données pour décrire l'état de la ressource avec des métadonnées. Plus précisément, la représentation d'une ressource Web est décidée par son URL, la méthode d'accès utilisée, les en-têtes de requête, et les négociations de contenu (le cas échéant).

2.1.2.3 Les standards et spécifications du Web

La pile de standard pour le Web est plus fine que celle de service web. Fondamentalement, il existe deux groupes de spécifications indépendantes : l'application et la représentation. Comme illustré dans la figure 2.3, le côté gauche est pour l'application ; et le côté droit est la représentation.

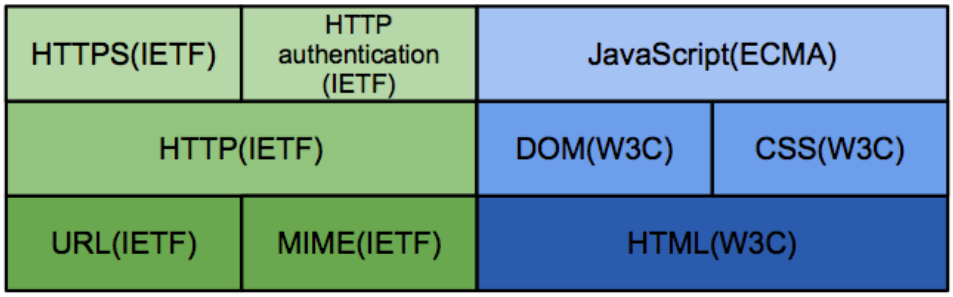


Fig. 2.3 : la pile des standards du Web

2.1.2.4 REST

Avec le développement de Web2.0/3.0, SOAP n'est pas le seul protocole d'échange des messages dans le monde du Web. De plus en plus des applications qui ont des interactions avec l'utilisateur se produisent. On trouve une grande tendance, c'est que les services proposés sont différents que les services SOAP dans le cadre de SOA, ils ont un type de REST plus léger est assez facile à utiliser, de nombreux fournisseurs de services tels que Google, Yahoo, Amazon etc., proposent une variété de service de REST. [Fielding 2000] Il a été décrite la première fois dans sa thèse. Tableau 2.1 sont certains services REST.

Fournisseur	Service RESTful
Google	*Youtube, Blogger, BookSearch, Calendar, Picasa, CodeSerach, DocumentsList, Finance Portlolio,Notebook, Health, Spreadsheets ...etc.*
Yahoo	*BOSS,Social, Address Book, HotJobs, Travel,Local, traffic, Maps,Music, Delicious, Flickr,...etc*
Amazon	*eCommerce, S3, ItemSearch*
Others	*FaceBook, Twitter, Digg, BBC, Wikipedia, MySpace*

Tableau 2.1 : Utilisation de REST dans les entreprises

La dérivation de REST architectural est illustrée dans la figure 2.4.

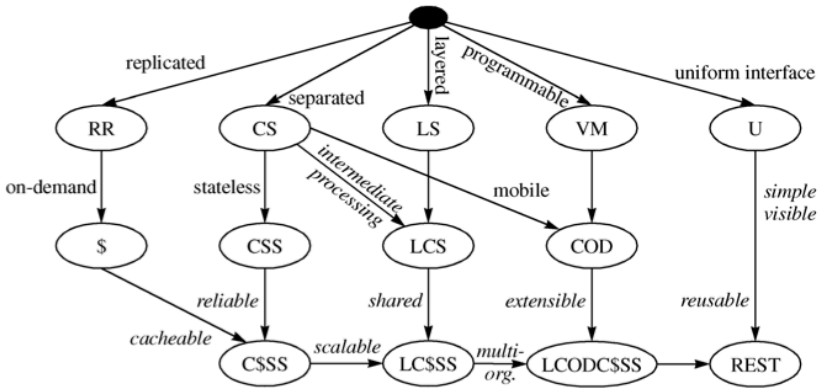


Fig. 2.4 : La dérivation de REST [Fielding 2000]

Un Service RESTful est donc un autre moyen de construire les services web. Ce type de WS suscite un grand intérêt croissant dans l'industrie et il a été largement adopté par certaines entreprises en raison de sa simplicité et sa légèreté (*light-weight*). L'idée de WS RESTful est d'appliquer les principes de REST dans le développement de WSs :

- Resource-centrique: les entités conceptuelles et les fonctionnalités sont modélisées comme des ressources identifiées par *universal resource identifier (URI).*
- L'interface uniforme : les ressources sont accessibles et manipulées via les opérations standardisées (*GET, POST, PUT*, *DELETE, TRACE, CONNECT*) dans le protocole HTTP1.1[Fielding, Gettys, Mogul, Frystyk, Masinter, Leach and Berners-Lee 1999] . GET est une opération pour prendre la représentation des ressources ; POST, PUT et DELETE sont utilisées pour créer, mette à jour, et supprimer les ressources respectivement.
- L'état de transfert : Les composants du système communiquent via ces interfaces d'opération standard et échangent les représentations de ces ressources (une ressource peut avoir multiples représentations). Dans un environnement de REST, serveurs et clients généralement transitent via l'état des représentations des ressources différentes en suivant les inter-liens entre les ressources.

L'architecture de REST introduit une série d'éléments architecturaux (agent d'utilisateur, *proxies*, passerelles et les serveurs originaux) qui sont destinés à être combinés pour construire un système *scalable*, il permet un grand nombre de clients (ou les agents d'utilisateur) pour accéder aux ressources publiées par un serveur d'origine unique illustré dans la figure 2.5. Chaque élément est relié à l'aide de protocole HTTP.

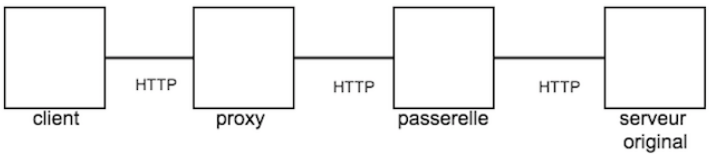


Fig. 2.5. REST basique : élément architectural et connecteur

Les *proxies* et passerelles sont optionnelles, ils sont généralement ajoutés à une architecture compatible avec le style REST pour effectuer le contrôle d'accès, la mise en cache, et une sorte de traduction de protocoles. Un proxy de contrôle d'accès peut être connecté à plusieurs agents d'utilisateur, il permet à un sous-ensemble d'entre eux d'accéder au serveur d'origine. Chaque demande de client peut donc passer par l'élément intermédiaire (d'ici proxy ou passerelle) et être entretenu de façon indépendante par un serveur d'origine.

En considérant l'existence de plusieurs serveurs, ils sont considérés comme des ressources autonomes et débranchés de l'information dont l'état évolue de façon indépendante. Les clients peuvent accéder séquentiellement à plusieurs serveurs (par exemple ils suivent les hyperliens de l'un à l'autre). Ce faisant, les clients peuvent aussi en quelque sorte recueillir les informations provenant de plusieurs serveurs et les regrouper localement. Ainsi, REST semble soutenir une forme de composition limitée au client. Aucun élément intermédiaire qui peut regrouper des informations à partir de plusieurs serveurs n'est explicitement prévu.

2.1.2.5 Comparaison entre SOAP et REST

Nous présentons ci-dessous la liste de différences entre SOAP et REST :

- SOAP adopte une approche de service donc chacun est représenté par un contrat formel pour définir l'interface de service web et un ensemble de méthodes spécifiques. Quand à REST, il adopte une approche de ressource qui est identifiée par des URIs.
- Dans SOAP, l'interface dépend de la méthode invoquée qui est spécifique pour chaque service. L'interface d'innovation dans REST est standardisée et uniforme pour toutes les ressources. SOAP est donc plus extensible mais moins simple.
- REST suit modèle de *stateless*, par rapport à SOAP, qui a des spécifications de mise en œuvre de *stateful*.
- REST adopte une interaction client/serveur et synchrone dans le Web. Quand à SOAP, il permet une composition de service synchrone ou asynchrone, une meilleure interopérabilité et plus de sémantique.
- SOAP utilise des interfaces et des opérations nommées pour exposer la logique de métier. REST utilise URI et des méthodes similaires (GET, PUT, POST, DELETE) pour exposer les ressources.

- SOAP a un ensemble de spécifications standard. Par exemple, *WS-Security* est la norme pour la sécurité dans la mise en œuvre. Il s'agit d'une norme détaillée fournissant des règles de sécurité dans la mise en œuvre de l'application. Pareillement, nous avons les spécifications particulières pour la messagerie, transactions etc.
- Contrairement à SOAP, REST repose principalement que sur les protocoles HTTP/HTTPS.
- REST est utilisé essentiellement par les développeurs de Web et surtout dans le cadre du Web 2.0. Il répond aux exigences d'intégration simples au niveau de l'interface d'utilisateur (*User interface*, UI). SOAP est plutôt utilisé dans les compositions de services complexes qui nécessitent de la sémantique.

En raison des points de vue différents de construire les WSs,

2.1.3 Conclusion

Dans cette partie là, on a discuté les développements et ses technologies de service Web. Surtout, on a discuté les services de SOAP et les services de REST et faire une comparaison dans le tableau 2.2 en raison des points de vue différents de construire les WSs. On a besoins d'une solution pour intégrer tous les deux types de service pour faciliter le usage au concepteur de web et au client final sur Web.

Critère	Service SOAP	Service RESTful	Commentaires
States	*stateful*	*stateless*	
Orientation	*Operation-centric*	*Resource-centric*	
L'indépendance de langage/ plate-forme	Oui	Oui	
Client-Server	default	default	
Simplicité	Non	Oui	
Basé sur Standard	Oui	Non	Cf. SOAP et WS-* spécifications
Sécurité	SSL, WS-Security	SSL	WS-Security fournit l'intégrité des messages de sécurité couvrant de bout en bout et l'authentification
Transactions	*WS-Atomic Transaction*	Non	
Fiabilité	*WS-ReliableMessaging*	application spécifique	
Performance	bien	mieux	*Caching* et *lower message payload* permet la performance de WS RESTful efficace et extensible
Caching	Non/Rare	Cache-friendly	
Protocole de transport	HTTP, SMTP, JMS etc.	HTTP	Pris en charge plusieurs protocoles de transport rend WSs WSDL/SOAP plus flexible
Taille de message	Lourd, qui a SOAP et WS-* *markup* spécifique	Light-weight, sans XML *markup* supplémentaire	
Protocole de communication du message	XML	XML, JSON, etc.	La flexibilité de REST est une caractéristique très utile en fournissant aux utilisateurs les messages *payloads* d'après leurs besoins spécifiques
Self-descriptive messages	*Suggest to put lots of metadata in SOAP header, and SOAP header entries are not standard*	*Encourage self-descriptifs messages, for exemple, by using HTTP*	
Description du service	WSDL	Pas de définition formelle	Dans l'étude REST, aucun moyen formel pour décrire une interface du service. Cela ne signifie pas une plus grande dépendance sur la documentation écrite
Manipulation through representations	*Use serialized parameters/objects*	*Encourage différent representations of the same resource*	
Outil du développement	Oui	Rare	La complexité du WSDL/SOAP dicte les besoins d'utiliser le cadre pour faciliter le développement des applications. En revanche, REST, en raison de sa simplicité, il peut être développé sans *framework*.
……			

Tableau 2.2 : Comparaison entre le service REST et le service SOAP

2.2 Le composant de service

2.2.1 Les services dans SOA

L'architecture orientée services (*Service Oriented Architecture*, SOA) [Newcomer and Lomow 2004] est une forme d'architecture de médiation qui est un modèle d'interaction applicative pour mettre en œuvre des services avec une forte cohérence interne (par l'utilisation d'un format d'échange pivot, le plus souvent XML) et des couplages externes "lâches" (par l'utilisation d'une couche d'interface interopérable, le plus souvent un service web). L'architecture de SOA est présentée généralement dans la figure 2.6.

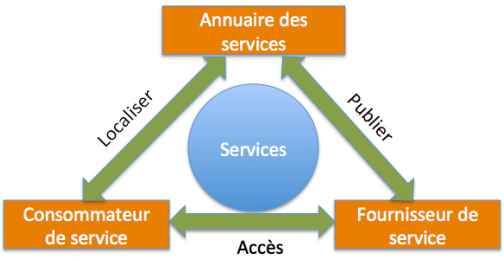


Fig. 2.6 : Modèle d'interaction orientée service

Comme ceux qu'on a vus dans la figure 2.6, il y a trois rôles dans les paradigmes de SOA : consommateur de service, fournisseur de service et le registre de service. Le fournisseur de services crée un service et fournit ses descriptions fonctionnelles à un registre. Le registre est une entité centrale de fournisseurs de services. Du point de vue des consommateurs, même si cela peut être physiquement distribué sur plusieurs plates-formes, les consommateurs de services peuvent effectuer une recherche dans le registre pour trouver leurs services souhaités. Ils peuvent ensuite invoquer ces services. C'est à dire, il y a un annuaire qui joue le rôle d'intermédiaire entre eux. Les fournisseurs y enregistrent leurs services, et les clients y cherchent le service satisfaisant leurs besoins.

La technologie de WS représente la technologie le plus utilisée pour migrer vers le SOA. SOA contribue donc à créer un meilleur alignement entre l'IT et le métier : SOA met en exergue, auprès des cadres de niveau métier, l'intérêt et la valeur ajoutée du travail effectué par l'IT. SOA est donc un soutien majeur à une plus grande flexibilité des métiers. Par conséquence, cette association de plus en plus proche entre le monde métier et l'IT pourra aisément justifier des nouveaux investissements au niveau de l'IT.

Aujourd'hui le SOA peut nous aider à mieux réutiliser nos investissements existants d'IT ainsi que les nouveaux services, c'est pourquoi le fait que les industriels développent SOA n'est pas simplement une mode. En effet, SOA est une solution métier qui est basée sur une entité technique nommée Service, elle se place plutôt dans la continuité logique des multiples tentatives de distribution des traitements, de répartition des données, d'intégration des applications, d'homogénéisation du système d'information etc. SOA permet d'intégrer les investissements IT plus facilement en utilisant des interfaces bien définies entre les services, c'est-à-dire le fournisseur permet l'accès à son service à travers une interface, le client désigne une personne, un serveur ou une autre application qui accède au service et l'invoque à travers l'interface proposée.

Selon *Thomas Erl* [Erl 2005], un service correspond à un ensemble d'opérations qui sont groupée logiquement et qui sont capables d'effectuer des unités reliées de travail., Une

exigence de base pour la réalisation des objectifs stratégiques de SOA, c'est que les services doivent être intrinsèquement composables. Comme un moyen de réaliser ces objectifs, le paradigme de conception de SOA est donc naturellement porté sur la capacité de la composition flexible [Zhao, Ren, Li and Sakurai 2012]. L'adoption de la SOA a été grandement facilitée par l'émergence opportune de la technologie des services web et leurs standards bien définis. Il y a huit principes à respecter et à suivre dans toute conception et déploiement d'une architecture SOA [Erl 2005]--Réutilisation, Couplage-lâche, *Stateless*, Autonomie, Interopérabilité, Abstraction, Description dans un contrat, Découverte et Composition.

- Réutilisation: la réutilisation des services partageant la logique entre plusieurs services avec l'intention de promouvoir la réutilisation. C'est à dire, il faut que la logique encapsulée dans chacune des opérations de service soit réutilisable afin qu'on considère le service comme réutilisable en soi-même.
- Couplage lâche: Les services sont connectés aux clients et autres services via des standards. Ces standards assurent le découplage, c'est à dire la réduction des dépendances. Ces standards sont des documents XML comme dans les services web. Il est assuré à l'aide des contrats de services qui permet aux services d'interagir selon des paramètres prédéfinis, et de partager certaines connaissances tout en restant indépendants les uns des autres.
- Autonomie : Un service doit exercer un contrôle fort sur son environnement d'exécution sous-jacent. Plus ce contrôle est fort, plus l'exécution d'un service est prédictible. Cela doit être pris en considération lors de la répartition du logique métier en plusieurs services et lors du choix des opérations à regrouper pour constituer un service.
- Interopérabilité : Elle permet aux services de communiquer et de collaborer entre eux afin d'offrir un service global, tout en gardant leurs autonomies.
- Abstraction : Le contrat d'un service ne doit contenir que les informations essentielles à son invocation. Seules ces informations doivent être publiées. C'est à dire, la seule partie qui reste visible à l'extérieur du service est celle décrite dans son contrat.
- Description dans un contrat : l'ensemble des services d'un même Système Technique est exposé au travers de contrats respectant les mêmes règles de standardisation. En plus, le contrat peut aussi contenir des informations plus sémantiques qui expliquent comment un service va accomplir un certain travail.
- Découverte : la découverte des services est depuis leur description extérieure. Le service est complété par un ensemble de métas donnés de communication au travers desquelles il

peut être découvert et interprété de façon effective. Elle se base sur un contrat de service, qui doit clairement décrire non seulement la logique de service, mais aussi la fonctionnalité offerte par chacune de ces opérations.

- Composition : Un service doit être conçu de façon à participer à des compositions de services, lesquelles permettent de représenter la logique sur différents niveaux de granularité et par suite faciliter la réutilisation et la création des couches d'abstraction.

Ces principes permettent aux services distincts d'interagir les uns avec les autres, ils fournissent donc une nouvelle façon pour la création de services – composition de service avec SOA. Le principe de composition dans SOA se consacre exclusivement à créer un « nouveau service » par une composition efficace. Donc, composition de service avec SOA est définie comme le processus consistant à combiner l'ensemble de services ou de capacités de fournir une nouvelle application. Ce processus est également connu comme "*Mashup*" [Murugesan 2007] pour les services Web. Evidemment, tous les autres principes pour soutenir la composition de service afin de réaliser cet objectif.

Par conséquent, dans les sections suivantes, on va analyser les différentes approches et méthodes de la composition de service qui nous aident à concevoir notre architecture de la médiation pour la composition.

2.2.2 Les services dans le *Clouding Computing*

Cloud computing est un terme vraiment « surchargé » et beaucoup plus que SOA aujourd'hui. Il décrit une nouvelle architecture de l'offre, de la consommation et de la prestation de service de l'IT basée sur l'Internet.

Cette architecture traite le service IT comme un service publique, comme l'offre d'énergie électrique[Hung, Shieh and Lee 2011, Miller 2008].Dans [Vaquero, Rodero-Merino, Caceres and Lindner 2008, Zhang, Cheng and Boutaba 2010], il propose une définition globale sur le *Cloud computing* en décrivant ses caractéristiques principales comme suit :

- À grande échelle: Le *cloud* généralement se compose des milliers de serveurs, qui offrent aux utilisateurs la puissance de calcul intensif.
- Virtualisation: *Cloud computing* permet de vritualiser les ressources informatiques dans le *cloud*.
- Haute fiabilité: Il utilise de multiples copies des données et des installations informatiques redondantes pour assurer la fiabilité de service et la sécurité de service contre la perte des données.
- Polyvalence: *Cloud computing* peut soutenir les développements des applications divers en

combinant des ressources informatiques hétérogènes.

- Service "*on-demand*": il contient d'énormes ressources informatiques, qui peuvent satisfaire divers exigences des utilisateurs. Les utilisateurs peuvent utiliser le *Cloud Computing* sur une base de paiement à l'utilisation (*pay-per-use*).
- Cout faible: *Cloud Computing* réduit les coûts associés aux ressources matérielles et logicielles sous-traitées.

L'architecture de *Cloud Computing* est généralement constituée en quatre conches [Zhang, Cheng and Boutaba 2010]. Premièrement, la couche de matériel est chargée de gérer les ressources physiques. Deuxièmement, la couche d'infrastructure (aussi connu comme la couche de virtualisation) crée un pool pour le stockage et la calcul de ressources, en divisant les ressources physiques à l'aide des technologies de virtualisation comme *Xen* [Von Hagen 2008], *KVM* [Frisch 2009] ou le *VMWare* [Langenhan 2013]. Troisièmement, la couche de plate-forme se compose des systèmes d'exploitation et des cadres d'application, dont le but est de réduire la charge au minimum en déployant des applications directement dans les conteneurs de la machine virtuelle comme illustré dans la figure 2.7.

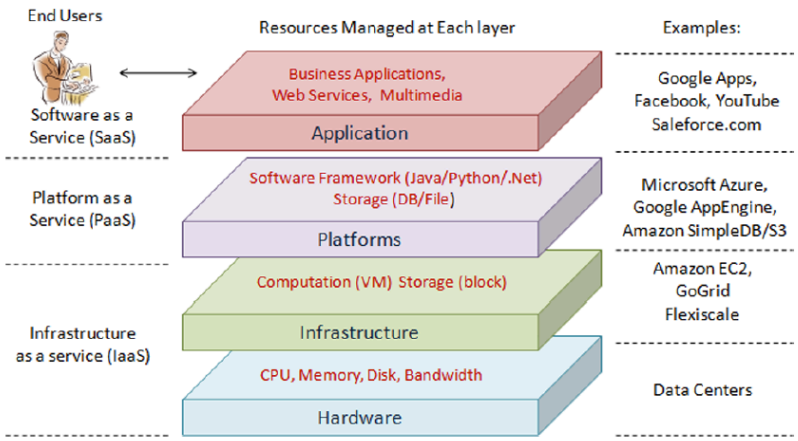


Fig. 2.7 : l'architecture de *Cloud Computing*

Dans le cadre de l'architecture de *Cloud*, dans [Buyya *et al.* 2009] il propose un génie de *cloud-service* comme une discipline qui puisse combiner les métiers et les technologies. Trois nouveaux aspects que le génie de *cloud-service* adresse en particulier comme :

- Tout est service.
- Service a des couts et des valeurs.
- Service constitue des réseaux de valeur.

Dans [Lenk, Klems, Nimis, Tai and Sandholm 2009], ils précisent que le *cloud* est une évolution « l'herbe-racine » des technologies de succès de Web 2.0/3.0. Il existe certains produits et *frameworks* de *cloud* commerciaux comme :

- *Microsoft Azure*: Il s'agit d'une offre d'hébergement (applications et données) et de services (*workflow*, stockage et synchronisation des données, bus de messages, contacts). Un ensemble d'API qui permet d'utiliser er d'accéder à cette plate-forme et aux services associés. Un environnement d'exécution permet une intégration étroite avec les principaux systèmes d'exploitation existant (Windows, OS X de MAC, et Windows Phone). Elle est composée des éléments des sites Web, Web rôle, machines/réseau virtuel, base de données de SQL etc. et d'autres services de type middleware et gestions d'identité tels que bus de service, contrôle d'accès de service etc.
- *Google App Engine* : c'est une plate-forme de conception et d'hébergement d'application web basée sur les serveurs de Google. Pour le moment il supporte les langages de Python, Java et Go. Pour programmer une application *Google App Engine* en Java, il existe un plugin de l'Eclipse qui permet de développer et de mettre en ligne l'application. Le support du serveur de développement est aussi disponible pour Netbeans.
- *Amazon Web Services*: c'est une collection de services fournis par Amazon.com, les offres Amazon Web Services sont accessibles en HTTP, sur l'architecture REST et par le protocole SOAP. Tout est facturé en fonction de l'utilisation, avec la valeur exacte variant selon le service, ou selon la zone géographie de l'appel.

2.2.3 Conclusion

Dans cette partie là, on a discuté les composants de service hébergé avec le Web. L'un part, auprès des cadres de niveau métier, l'intérêt et la valeur ajoutée du travail effectué par l'IT. SOA est donc un soutien majeur à une plus grande flexibilité des métiers. L'autre part, *Cloud computing* est un terme vraiment « surchargé » et beaucoup plus que SOA. Il décrit une nouvelle architecture de l'offre, de la consommation et de la prestation de service de l'IT basée sur l'Internet. Il offre la possibilité pour les services traditionnels tels que les services de SOAP, mais aussi il fourni la possibilité pour les service de style RESTful.

2.3 La composition de Service

Le WS, avec son interface autonome et « couplage lâche », nous fournit une approche intégrée idéale pour les mises en œuvre des services déployés sur le Web dans divers domaines de B2B/B2C/O2O. L'évolution des services introduira une évolution de l'architecture qui structure non seulement la syntaxe mais aussi la sémantique de l'écosystème. Par ailleurs, l'analyse des travaux dans l'état de l'art a permis de retenir qu'il existe plusieurs travaux sur la composition avec des degrés de maturité différents vis-à-vis de la coopération.

L'infrastructure de WS basique présentée dans la section précédente est suffisante pour les implémentations des interactions simples entre un client et un WS. Si une connexion d'un métier de WS implique l'invocation d'autres services web, il est nécessaire de combiner les fonctionnalités de certains services web. Dans ce cas-là, nous parlons de service composite.

Le processus du développement d'un service composite à son tour est appelé la composition de service. Composition de service peut être effectuée soit par la composition de services composites soit de services élémentaires. Services composites à leur tour sont définis récursivement comme une agrégation de service élémentaire et composite. Quand l'on compose les services web, le métier logique du client est implémenté par un ou plusieurs services. Ceci est analogue à la gestion de processus métier, où la logique de l'application est réalisée par la composition des services autonomes. Cela permet la définition des applications de plus en plus complexes en agrégeant les composants progressivement aux niveaux d'abstraction les plus élevés. Donc un client invoquant un service composite peut être lui-même exposé comme WS/WSC (un service web/ une composition de service web). Dans ce chapitre, on va parler des méthodes différentes pour la WSC.

Pour la composition de service, tout d'abord, notre travail concerne la conception de l'approche et de l'architecture du modèle. Nous avons commencé la recherche sur les contextes, les méthodes et les modèles pour la composition de service [Bucchiarone and Gnesi 2006, Cetin *et al.* 2007, Dustdar and Schreiner 2005, Pfeffer *et al.* 2008, Srivastava and Koehler 2003, Tang and Ai 2010, ter Beek *et al.* 2006]. On trouve que l'approche médiation [Wiederhold 1992] et l'approche Peer-to-Peer [Rolland and Kaabi 2007] sont intéressantes pour la composition. Ensuite on analyse les standards, les langages de composition basés sur les exigences différentes dans certaine situation particulière [Bertoli *et al.* 2010, Dey 2001, Dumas *et al.* 2006, Gruber 1995, Li *et al.* 2010, Ter Beek *et al.* 2007, Wang and Wu 2011, Weise *et al.* 2014]. A la fin , on analyse certaines approches de composition y compris *AI Planning* [McIlraith and Son 2002, Medjahed and Bouguettaya 2011, Medjahed *et al.* 2003, Sirin 2010, Strogatz 2001], les approches de web sémantique [Canfora *et al.* 2005, Ye *et al.* 2011], les approches de middleware [Zahreddine and Mahmoud 2005] [Brnsted *et al.* 2010, Vallée *et al.* 2005] [Nassar and Simoni 2011, Nassar and Simoni 2012] etc. Dans le même temps, nous avons étudié les travaux connexes menés par d'autres chercheurs qui travaillent dans les domaines industriels et dans la recherche académique similaire [Issarny *et al.* 2011, Milanovic and Malek 2004, Srivastava and Koehler 2003].

2.3.1 Les principes de la composition

La composition consiste à combiner les fonctionnalités de plusieurs WSs au sein d'un même processus métier (*business process*) dans le but de répondre à des exigences complexes qu'un service spécifique ne pourrait pas satisfaire. C'est à dire, la composition consiste à combiner plusieurs services web dans une nouvelle application afin de fournir aux utilisateurs des fonctionnalités avancées ou des valeurs ajoutées (planification de voyage, achats en lignes etc.). La composition comporte plusieurs étapes, qui généralement consistent à :

(1) la décomposition et précision de l'objectif de l'utilisateur de haut niveau en sous tâches.

(2) trouver les services web qui mettent en œuvre les fonctionnalités de chaque sous tâche.

(3) orchestrer les interactions entre WSs composés afin d'obtenir l'objectif de haut niveau de la composition et de satisfaire aux exigences de l'utilisateur.

2.3.1.1 Les standards

Les technologies de WS sont d'abord largement axées sur l'interopérabilité des systèmes hétérogènes basés sur le fondement des messageries pris en charge par WSDL et SOAP. La composition les WSs requière la capacité de spécifier le processus métier, les exigences de sécurité, la gestion des transactions et d'autres informations critiques liées au contexte des processus métiers , et ces informations sont généralement indiquées dans les modèles de services et de ses processus métiers. WSC, qui fait l'exécutable de processus, est concerné par les informations mentionnées ci-dessus. Une approche basée sur les normes est donc nécessaire pour la WSC afin de créer un processus métier de niveau haut.

La recherche bibliographique montre que le travail dans l'élaboration d'approches de composition appropriées se fait du côté de l'industrie ainsi que du côté de l'académie. La WSC actuelle n'a pas de modèle de référence. La WSC dans l'industrie se fait en adoptant la pile des services web généraux comme un standard pour les interactions des services web, et adopte des normes telles que BPEL4WS [Wohed *et al.* 2003] et WSCI [Arkin *et al.* 2002] pour relier ces services web entre eux pour former des processus métiers plus significatifs. Ces approches utilisent les techniques traditionnelles du langage de programmation pour réaliser le processus de la composition. Ils utilisent des structures spécifiques du langage et des modes de contrôle pour composer les services et effectuent la composition au niveau micro. Il s'agit purement d'une approche statique et il n'y a aucun moyen pour l'adaptation du processus et donc il ne peut pas changer au moment de l'exécution. Cela conduit la composition à se faire statiquement et manuellement. Dans ces approches, la composition

peut être considérée comme une composition de « fonction centrique » basée sur la raison pour laquelle les services composant le raisonnement derrière le raccordement de deux composants de service, ne dépend que de l'entrée de service et de sa sortie soit seulement de la description syntaxique de service. Ces types d'approches relèvent de la composition des services basée sur la syntaxe. Dans l'approche syntaxique actuellement, il y a deux écoles principales pour WSC, il s'agit de l'orchestration et la chorégraphie.

L'orchestration combine les services offerts par l'ajout d'un coordinateur qui est responsable de l'invocation et de combiner les activités (ex. dans BPEL4WS) et la chorégraphie qui est sans coordinateur, et qui définit les tâches complexes via la définition de la conversation qui devrait être entreprise par chaque participant; l'activité globale est alors réalisée par la composition des interactions *P2P* entre les services de collaboration.

La communauté de recherche académique utilise web sémantique pour résoudre les problèmes de la composition automatique et dynamique. Dans ces approches, la composition peut être considérée comme une composition centrée sur les processus fondés sur les *raisonnings* pour laquelle le raccordement de deux services dépend de IOPE à savoir la description sémantique de service. Les descriptions sémantiques fournissent des agents logiciels avec un moyen de raisonnement automatique sur la sémantique du service, c'est à dire les conditions et les effets. Dans un environnement de services sémantiquement annotés, les utilisateurs pourraient être aidés par des agents logiciels qui s'identifient automatiquement et, si nécessaire, composent dynamiquement des services afin d'atteindre les objectifs de l'utilisateur. Cela peut être soit explicitement déclarés ou issus de la situation de l'utilisateur actuellement en jeu. Ces approches sont regroupées dans la WSC basée sur la sémantique. WS technologies actuelles ne traitent que des aspects syntaxiques de services et fournissent ainsi un ensemble de WSs *on-the-fly* qui ne peuvent pas s'adapter à un environnement changeant sans intervention humaine. Actuellement, il existe deux modèles principaux disponibles pour WSC basée sémantique ie. OWL-S[Martin *et al.* 2004] et *Web Service Modelling Ontology* (WSMO) [De Bruijn *et al.* 2006] . OWL-S est un effort pour définir une ontologie pour le balisage sémantique de WS, destinée à permettre l'automatisation de la découverte de service, l'invocation, la composition, l'interopérabilité et le suivi d'exécution en fournissant des descriptions sémantiques appropriées des WSs. Le WSMO est un effort pour créer une ontologie pour décrire les différents aspects liés aux services web sémantique, visant à résoudre le problème de l'intégration. L'objectif de ces deux initiatives est de fournir un standard pour la description sémantique de WS.

2.3.1.2 Les exigences

Dans la composition traditionnelle, une hypothèse implicite, c'est que les services sont exécutés sur les serveurs *heavyweight* de l'entreprise et ils fournissent généralement les fonctions de calcul intensif. Notre but de cette partie est de regrouper les exigences des techniques basiques du domaine de la WSC qui se propagent à travers des recherches bibliographiques. Les exigences sont l'extension de celles qui ont été abordées principalement dans la littérature. Par exemple, l'environnement de WS est très dynamique et, afin d'effectuer la composition de services dans ce domaine, les approches de composition développées doivent répondre aux exigences suivantes du domaine de WS.

i. La complexité

- Exploratoire: Le nombre de services disponibles sur le web augmente énormément dans les dernières années et il est prévu d'avoir un référentiel de WS [Dustdar and Schreiner 2005].
- *Volatile*: Les WSs sont créés et mis à jour *on-the-fly.* L'environnement est caractérisé comme *volatile* [Dustdar and Schreiner 2005] qu'il s'agisse des services entrant et sortant du domaine dynamiquement. C'est à dire l'environnement ne cesse de changer. Les changements portent également sur la qualité de service (QoS) dont les valeurs pourraient évoluer dans le temps.
- Incertain/Non-déterministe: Les environnements de l'informatique distribuée sont incertains: l'invocation des WSs distants peut potentiellement produire des réponses inattendues [Maigre 2010, Portchelvi *et al.* 2012, Sirin 2010, Zhang *et al.* 2003]. Une réponse pourrait dépendre du fait que le WS fonctionne correctement et dépende des situations spécifiques du monde réel externes.
- Observable partiellement: Les services dans l'environnement sont observables partiellement, c'est à dire, ses statuts et variables internes [Bertoli, Pistore and Traverso 2010, Maigre 2010] ne sont pas accessibles.
- Hétérogénéité: Les WSs peuvent être développés par les organisations différentes qui utilisent divers modèles pour décrire les services. Il n'existe pas un langage unique pour définir et évaluer les WSs selon des moyens identiques.

ii. La performance

Le processus de composition est un processus exigeant des calculs. Les calculs sont requis pour composer un service en satisfaisant le but de l'utilisateur, cela pourrait conduire à une grande consommation du temps. Le nombre de services disponibles sur

le Web augmente énormément et il est prévu d'avoir une énorme *repository* de services Web pour être trouvé [Bartalos 2011].

iii. L'adaptabilité

Les WSs sont créés et mis à jour *on-the-fly*, ainsi le système de composition doit détecter la mise à jour en temps réel et une décision devrait être prise sur les informations mises à jour.

iv. La résilience de défaillance (échec)

Dans SOA, une idée très importante, c'est que les demandeurs de services et les prestataires de services peuvent provenir d'organisations/régions différentes. La probabilité d'erreurs telles que les défaillances et les changements de services dans les environnements distribués est donc plus élevée que dans un environnement mono-organisationnel. Si une exception [Vukovic *et al.* 2007] se produit, il aurait des possibilités de recomposer la composition de service pour que la nouvelle composition puisse fonctionner probablement avec des services différents sans défaillance.

v. L'optimisation

L'optimisation utilise les paramètres non-fonctionnels pour améliorer les comportements de la composition et ses composants. Bien que satisfaire les exigences fonctionnelles soit important pour la construction de WSC, l'optimisation des préférences non-fonctionnelles peut être tout aussi cruciale. Par exemple, une WSC, qui minimise les temps de réponse et les coûts de WS, et garantit la disponibilité dans un niveau de base, est nécessaire[Zhao and Doshi 2009].

vi. L'exactitude de la composition

L'approche de la composition doit être en mesure de garantir l'exactitude de la composition[Ter Beek, Bucchiarone and Gnesi 2007]. Le comportement de la composition développée doit se conformer à ses exigences.

vii. L'exécution de la composition

L'exécution de la composition peut se faire de la manière centralisée ou distribuée.

2.3.1.3 Les langages

Généralement, les langages de composition de service web sont classés en trois catégories.

Dans la première, il y a les langages de description/modélisation orientés processus, tel que le *Business Process Execution Language for Web Service*. *BPEL4WS* [Wohed, van der Aalst, Dumas and ter Hofstede 2003] qui est un langage largement utilisé dans le domaine industriel.

Il découpe un processus métier en processus abstrait et processus exécutable. En plus, il définit le modèle et le langage de description du comportement du processus métier. C'est un langage pour les processus métiers basés sur XML conçus pour permettre charger/partager les données distribuées, même à travers des organismes multiples, en employant une combinaison des WSs. Ainsi, cela facilite grandement la description de processus et son exécution comme illustrées dans la figure 2.8.

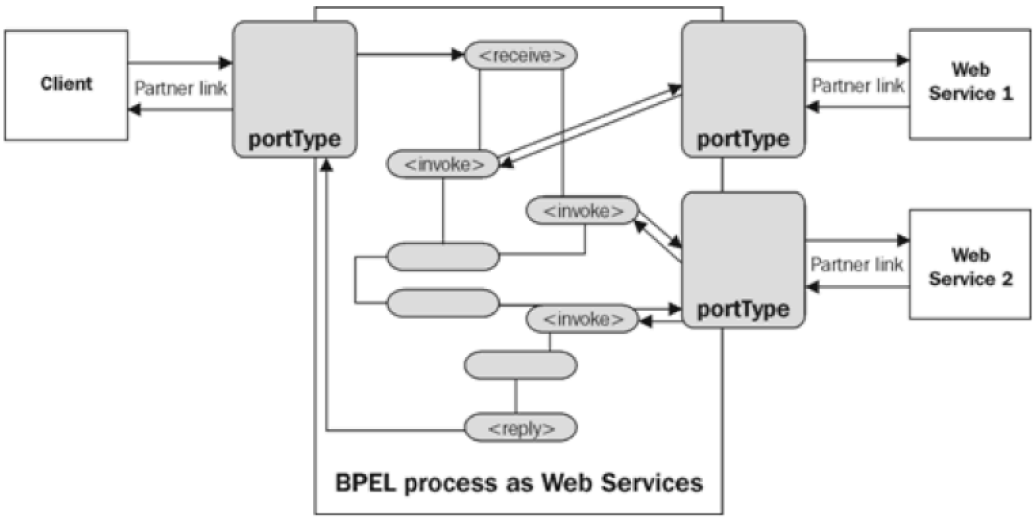


Fig. 2.8: un exemple de processus de BPEL

BPEL : BPEL décrit l'interaction des processus métiers basés sur les WSs, à la fois au sein des entreprises et entre elles. Les entreprises utilisant du langage BPEL pourront ainsi définir leurs processus métiers et en garantir l'interopérabilité non seulement à l'échelle de l'entreprise, mais également avec leurs partenaires commerciaux, au sein d'un environnement de WS. BPEL rend donc la possibilité des interopérabilités entre des activités commerciales basées sur des technologies différentes. Proposé par des créateurs des systèmes de BEA, IBM, et le *Microsoft*, la spécification BPEL combine et remplace le WSFL d'IBM et le XLANG du *Microsoft*. Parfois, BPEL est appelé aussi BPELWS ou BPEL4WS.

WSCI : *Web service Choregraphy Interface* [Arkin, Askary, Fordin, Jekeli, Kawaguchi, Orchard, Pogliani, Riemer, Struble and Takacsi-Nagy 2002] est un langage de haut niveau basé sur UML qui décrit la session de la couche métier. C'est un langage reposant sur l'XML aussi. Il propose de se focaliser sur la représentation des WSs en tant qu'interfaces décrivant le flux des messages échangés (la chorégraphie des messages) illustré dans la figure 2.9.

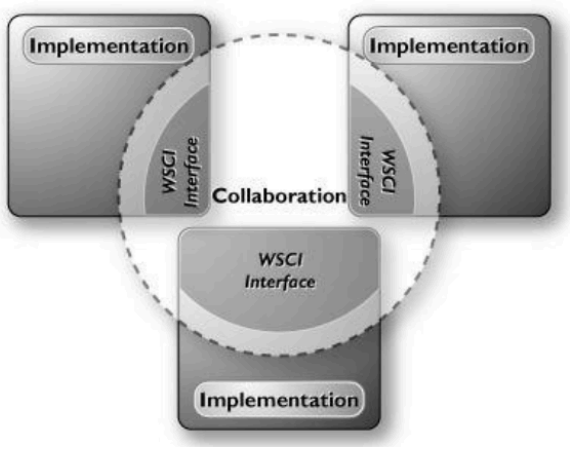


Fig. 2.9 : Architecture de WSCI défini par W3C

Il propose ainsi de décrire le comportement externe observable du service. Pour ce faire, WSCI propose d'exprimer les dépendances logiques et temporelles entre les messages échangés à l'aide des contrôles des séquences, la corrélation, la gestion des fautes et les transactions. On remarque que le WSDL et ses définitions abstraites sont réutilisées afin de pouvoir également décrire par la suite les modalités de concrétisation des éléments manipulées pour modéliser un service. Cependant, il n'offre pas la possibilité de la description d'un WS considéré isolément.

WS-CDL : [Mendling and Hafner 2005] *Web Services Choreography Description Language* est utilisé pour la description délicate entre processus métier, mais c'est un standard du processus métier abstrait non-exécutable.

BPML : *Business Process Modeling Language* (BPML) [Arkin 2002] est un langage de la modélisation des processus métiers. Il permet de définir un modèle abstrait d'interaction entre les 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 transaction. On peut ensuite lancer l'exécution du modèle et vérifier le fonctionnement théorique des processus différents.

WSCL : *Web Service Conversation Language* (WSCL) [Banerji *et al.* 2002] propose de décrire à l'aide de document XML et les services Web en mettant l'accent sur les conversations de ceux-ci. En outre, les messages à échanger sont pris en compte. WSCL a été pensé pour s'employer conjointement avec WSDL. Les définitions de WSDL peuvent être manipulées par WSCL pour décrire les opérations possibles ainsi que leur chorégraphie. En retour, le WSDL fournit les concrétisations vers des définitions de messages et des détails techniques pour les éléments manipulés par WSDL.

XLANG : [Thatte 2001] Créé par Microsoft, le XLANG est une extension de WSDL. Elle fournit en même temps un modèle pour une orchestration des services et des contrats de collaboration entre celles-ci. XLANG a été conçu 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. XLANG ajoute deux autres genres d'action: arrêts (date-limite et durée) et exceptions.

La deuxième catégorie est constituée des langages orientés données. On peut citer comme exemple *Active XML(AXML)* [Abitrboul *et al.* 2002] et *World-Wide Telescope (WWT)* [Gray and Szalay 2002]. Ces langages ont été utilisés surtout dans les domaines spécifiques, comme l'Astronomie ou la météorologie pour traiter les sources des données hétérogènes et massives.

La troisième catégorie est constituée des langages orientés sémantique, En ajoutant les informations sémantiques dans le WS en décrivant les données et fonctionnalités, il est possible d'automatiser la compréhension des fonctionnalités d'un WS et la portée des résultats fournis. Ceci peut conduire jusqu'à générer automatiquement un processus dynamiquement composés des divers WSs. Nous citons par exemple, *OWL-S (Semantic Markup for Web Service)*[Martin, Burstein, Hobbs, Lassila, McDermott, McIlraith, Narayanan, Paolucci, Parsia and Payne 2004] et *SAWSDL (Semantic Annotation Web Service Description Language)*[Kopecky *et al.* 2007].

WADL : *Web Application Description Langage* [Hadley 2006, Richardson and Ruby 2008] (WADL) est un format de fichier basé sur XML qui permet de décrire des services RESTfuls. Son premier but est de permettre de décrire les services proposés par une application sur l'internet. Cette spécification se heurte néanmoins à la spécification WSDL 2.0, qui elle aussi permet la description de web service RESTful. Le W3C contribue aussi à sa spécification. Voici un exemple de la recherche sur Yahoo Nouvelles décrit par WADL illustré dans la figure 2.10.

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
 xmlns:tns="urn:yahoo:yn"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:yn="urn:yahoo:yn"
 xmlns:ya="urn:yahoo:api"
 xmlns="http://wadl.dev.java.net/2009/02">
  <grammars>
   <include
    href="NewsSearchResponse.xsd"/>
   <include
    href="Error.xsd"/>
  </grammars>

  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
   <resource path="newsSearch">
    <method name="GET" id="search">
     <request>
      <param name="appid" type="xsd:string"
       style="query" required="true"/>
      <param name="query" type="xsd:string"
       style="query" required="true"/>
      <param name="type" style="query" default="all">
       <option value="all"/>
       <option value="any"/>
       <option value="phrase"/>
      </param>
      <param name="results" style="query" type="xsd:int" default="10"/>
      <param name="start" style="query" type="xsd:int" default="1"/>
      <param name="sort" style="query" default="rank">
       <option value="rank"/>
       <option value="date"/>
      </param>
      <param name="language" style="query" type="xsd:string"/>
     </request>
     <response status="200">
      <representation mediaType="application/xml"
       element="yn:ResultSet"/>
     </response>
     <response status="400">
      <representation mediaType="application/xml"
       element="ya:Error"/>
     </response>
    </method>
   </resource>
  </resources>
```

Fig. 2.10 : un exemple de recherche sur Yahoo Nouvelles décrit par WADL

Comme illustré dans la figure, le <*resources*> représente l'adresse de site, le <*resource*> représente les ressources utilisables fournis par ce site et paramètre de *path* pour représenter le URI, le <*method*> correspond la fonctionnalité de méthode e l'HTTP. On a donc un tableau 2.3 suivant pour voir les relations correspondant aux différences entre WADL et REST.

Elément WADL	Définition WADL	Règles correspond de REST
<*resources*> <*resource*>	La ressource de service REST et son URI	ressource abstraite et ressource centrique
<*method*>	Définition de méthode *GET* et *POST* etc.	interface uniforme

Tableau 2.3 : les relations correspondant entre WADL et REST

DSL : *Domain-Specific Langage (DSL)* [Fowler 2005, Klint *et al.* 2009, Maximilien *et al.* 2007] Un langage dédié est un langage de programmation dont les spécifications sont dédiées à un domaine de l'application précise. Le domaine de l'application peut être donc WS et sa WSC, ou bien le mashup. Il s'oppose conceptuellement aux langages de programmation classique comme le Java ou le C, qui tendent à traiter un ensemble de domaines. Néanmoins, aucun consensus ne définit précisément ce qu'est un langage dédié. Il est basé sur les concepts et les fonctionnalités du domaine en question. En tant que tel, un langage dédié est un moyen efficace de décrire et de générer des programmes dans un domaine spécifique.

Le DSL est un genre de « mini » langage construit au niveau haut d'un langage d'hébergement qui offre la syntaxe et la sémantique pour représenter des concepts et des comportements de WS dans un domaine particulier. En général, l'utilisation ou la conception d'un DSL permet d'atteindre les objectifs suivants:

- L'abstraction par la programmation à un niveau haut;
- Le code laconique;
- La simple et la syntaxe naturelle;
- La facilité de la programmation; et
- La génération de code par la conversion des phrases de DSL lors de l'exécution (*at runtime*).

Il existe deux types de langages dédiés : les langages dédiés internes et les langages dédiés externes. Les premiers sont destinés à être utilisé au sein du code source d'un programme, exprimé par ailleurs dans un autre langage qualifié de langage hôte. Les domaines d'application des langages dédiés sont très variés. Ces langages sont néanmoins créés pour effectuer une tâche précise. Par exemple, le Ruby est un langage dédié interne pour des applications Web. D'après [Cunningham 2007], Ruby est mis en œuvre comme un langage dédié interne car il a été développé sous *Emacs* qui lui-même s'appuie sur le langage *Emacs lisp*. Le langage d'Emacs *lisp* est donc considéré comme le langage hôte de Ruby. L'intérêt récent de Ruby est de reprendre techniquement la méta-programmation dans les langages dédiés.

En résumé, nous avons fait une analyse des différentes possibilités dans un premier temps pour choisir celle qui nous convenait le plus. Nous avons rejeté dès le début la possibilité d'utiliser XLANG, WSFL ou WSCL parce que l'union et l'extension de ces deux langages ont fait dériver le langage BPEL par conséquent on ne peut les considérer comme alternatifs. Dans le cas de WSCI, il permet d'organiser la chorégraphie des messages de services Web. Il doit travailler avec un autre langage complémentaire (tel que BPML) qui permettrait de décrire les processus métier. Outil complémentaire, BPML permet de décrire les processus métiers génériques en amont de l'enchaînement de messages WSCI. Dans ce cas il faudrait utiliser deux langages pour la conception d'un processus métier. Il est beaucoup plus simple d'utiliser un seul langage qui permet la conception de tout les processus, par exemple BPEL. Le langage BPEL serait donc le meilleur dans notre cas. Il est le résultat de l'unification et de l'évolution des trois différentes tentatives de standardisation des définitions des processus métier. Il est le standard pour la description des processus métier le plus complet existant. Mais la composition basée sur le moteur de BPEL est lourde et complexe. Nous avons donc choisi d'orchestrer nous-mêmes nos services web en fonction de scénarios plausibles, pour

avoir un outil plus léger et plus simple. Pour convertir le service REST à SOAP, il faut utiliser le WADL qui permet de décrire facilement le service RESTful et puis de faire la transition de WADL à WSDL pour formaliser les descriptions de service. Pour le traitement de fichier de script, c'est mieux d'utiliser un niveau plus haut, par exemple le DSL.

2.3.1.4 le Cycle de vie

Le cycle de vie de la composition contient les termes suivants pour développer une composition de service de bout-en-bout :

- Spécification: c'est à dire que l'utilisateur spécifie l'état initial, le but, y compris certaines contraintes de QoS.
- Planification: Techniques/Algorithmes sont utilisés pour générer un processus métier automatiquement/manuellement.
- Validation: C'est pour vérifier l'exactitude de la composition.
- Découverte: C'est pour trouver les composants des services actuels.
- Exécution et Surveillance (*Monitoring*): pour exécuter les services composés et pour surveiller les exceptions et les échecs.

Dans [Pessoa *et al.* 2008], les auteurs affirment que la plupart des recherches sur la WSC ont tendance à se concentrer sur une nouvelle technologie particulière (comme basé sur processus métier, basé sur *AI-planning*, ou basé sur le domaine-spécifique).

2.3.2 Les modes de la composition

La composition consiste à combiner les fonctionnalités de plusieurs WSs au sein d'un même processus métier (ou bien *business process*) dans le but de répondre à des demandes complexes, qu'un seul service ne pourrait pas satisfaire [Berardi *et al.* 2004]. Une composition comporte plusieurs étapes, qui généralement consistent :

(1) La décomposition et la précision de l'objectif de l'utilisateur de haut niveau en sous tâches.
(2) Trouver les services web qui mettent en œuvre les fonctionnalités de chaque sous tâche.
(3) Orchestrer les interactions entre services web composés afin d'obtenir l'objectif de haut niveau de la composition et de satisfaire les exigences de l'utilisateur.

Le processus métier est une représentation concrète des tâches à accomplir dans une composition. Le déroulement d'une composition nécessite la réalisation des étapes:

(1) La découverte des services web pouvant répondre aux besoins de la composition se fait généralement de manière manuelle par envoi de requête aux annuaires UDDI.
(2) L'organisation des interactions entre les services web composé est distinguée par les termes chorégraphie et orchestration [Peltz 2003].

(3) L'exécution de la composition est l'étape d'invocation effective des services web participant à une composition.

Tous ceux que nous venons de mentionnés fournissent un moyen de services web organisés dans les processus métiers significatifs. Donc dans un environnement de WS, un processus métier représente une WSC. On peut obtenir un WS composé lorsque chaque activité de processus métier est mise en œuvre par un composant de WS. Plusieurs WSs composés peuvent être associés à un même processus métier en fonction des composants de service attribués.

L'orchestration des composants de service est définie en spécifiant les dépendances entre eux. Ces dépendances sont définies par les modèles de processus métier associés et par les propriétés transactionnelles. Les premiers spécifient comment les WS sont couplés et comment le comportement de certains WSs influence les autres, tandis que les propriétés transactionnelles précisent le comportement de certains WSs en cas de panne. Le processus de production d'une WSC peut être généralement considéré au commencement avec certaines propriétés globales ou des dépendances (par exemple *workflow*) qui doivent être spécifiées. Vues les dépendances mondiales, les WSs individuels doit être orchestrés afin qu'ils répondent collectivement aux dépendances mondiales. Selon quand la connaissance des dépendances mondiales est connue. Il y a donc deux modes pour composer les services web à préciser, mode de médiation et mode de P2P.

2.3.2.1 Mode de médiation

Gio Wiederhold a introduit les travaux sur le médiateur en 1992, il l'a défini comme suit [Wiederhold 1992]:

« *A mediator is a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications*. »

Le médiateur est lié au domaine des bases de données, dans ma thèse, il s'agir de base de connaissance (BDC, en anglais *Knowledge base*), afin de clarifier la distinction entre les données et les connaissances, nous réaffirmons une définition de [Wiederhold 1993]:

*« **Data** describes specific instance and events. Data may gather automatically or clerically. The correctness of Data can be verified vis-a-vis the real world. »*

*« **Knowledge** describes abstract classes. Each class typically covers many instances. Experts are needs to gather and formalize knowledge. Data can be used to disprove knowledge. »*

Dans cette approche, toutes les dépendances globales sont connues pour au moins un service (appelé le médiateur) avant l'exécution. L'approche de médiation a suivi le modèle

d'orchestration, qui correspond à une partie exécutable d'un processus de service inter-organisationnel. Avec l'orchestration, vous pouvez définir la séquence des étapes dans un processus, y compris les conditions et exceptions, puis créer un contrôleur central pour mettre en œuvre la séquence. Dans une architecture SOA, les différentes étapes de la séquence sont mises en œuvre par les opérations des services. La séquence peut être mise en œuvre avec plusieurs techniques différentes. Normalement, les compositions relativement simples sont orchestrées dans le code, tel que Java ou C#, qui résident à l'intérieur du service composé. Mais pour les orchestrations plus complexes, on utilise normalement un outil pour créer un modèle visuel de la séquence, et ensuite générer le code qui exécute cette séquence, généralement dans un environnement en temps réel (*runtime*). C'est typiquement l'approche BPM.

Certain modèle d'orchestration existe, par exemple BPMN pour définir la représentation visuelle de la séquence et BPEL avec le « code » qui exécute la séquence. Presque toutes les infrastructures de SOA fournissent certain type de moteur de BPEL et la plupart des produits de BPM soutiennent déjà, ou sont en train de s'appuier sur ces normes dans leur modélisation et exécution. En outre, BPEL est exprimé en XML et défini par méta-data de Schéma XML qui est aligné avec les standards de SOA. BPEL lui-même utilise WSDL à deux niveaux : tout d'abord, les services web basés sur WSDL sont utilisés pour interagir avec les capacités requises par le processus. Deuxièmement, tous les processus de BPEL sont eux-mêmes des WS décrits en utilisant WSDL.

L'orchestration de service dépend d'un conducteur de service central qui fonctionne comme un médiateur et consume les services partenaires. Les services partenaires dans une orchestration de service n'ont pas besoins de savoir les détails de cette orchestration. Elle est normalement construite comme une médiation de la manière « *spoke-hub* » illustrée dans la figure 2.11.

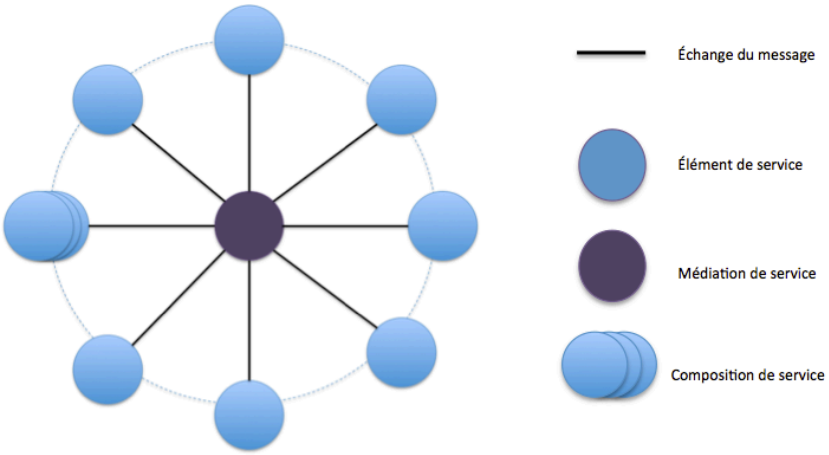


Fig. 2.11 : médiation de la manière « *spoke-hub* »

En résumé, l'orchestration :

- Définit un seul contrôle principal pour tous les aspects d'un processus (approche top-down).
- Supporte une vue graphique de la séquence.
- Permet un *Mapping* facile vers SOA.
- Est généralement plus simple pour commencer, mais souvent plus difficile à l'échelle des processus plus complexes.
- Est entrainée par le modèle de séquence graphique, c'est-à-dire la forme suit la fonction.

La popularité de cette approche est donc compréhensible, mais il ne prend pas en charge tous les types de processus. En plus, ce genre de processus métier implique des activités complexes et structurées, il a besoin d'un environnement *Stateful* pour l'invocation d'une chaine de WSs qui mettent en œuvre le processus métier. Les services représentant le noyau du logique métier pourraient participer aux processus complexes et être orchestrés par une médiation de service qui travaille comme un moteur d'orchestration. La médiation de service généralement n'a pas l'état de l'application, mais elle a l'état qui est lié au processus orchestré par lui-même. L'état du processus comprend des informations :

- Des participants au processus (les personnes et les services).
- Des entrées des participants.
- De la position réelle à l'intérieur du flux de processus.
- De certaines règles de base.

La médiation de service a une forte dépendance avec les autres services, elle fournit la colle pour lier les services. Elle doit coordonner les activités complexes qui peuvent s'étendre à plus d'une personne, ou plusieurs grandes entités commerciales, ou de longues périodes du temps. La médiation de WSC a un énorme potentiel en manière de rationalisation de B2B (*Business-To-Business*) et EAI (*Enterprise Application Intergration*). L'orchestration à

travers la médiation se réfère au processus métier exécutable qui peut interagir avec les WSs internes et externes. Les interactions se produisent au niveau du message. Elles comprennent le logique métier et l'ordre d'exécution des tâches et elles peuvent s'étendre aux applications et aux organisations pour définir un modèle de processus transactionnel, en plusieurs étapes et à long terme. L'orchestration représente toujours le contrôle du point de vue de l'une des parties.

2.3.2.2 Mode de Peer-to-Peer

Dans cette approche [Rolland and Kaabi 2007], chaque service individuel sait qu'un sous-ensemble, mais pas les dépendances mondiales complètement. Il suit donc le modèle de Chorégraphie. La chorégraphie correspond à des séquences de messages entre plusieurs services dans un processus inter-organisationnel, c'est-à-dire qu'elle fournit une approche différente qui gagne l'acceptation dans les scénarios qui ont les processus complexes avec de nombreuses parties, ainsi les systèmes basés sur un agent (*agent-based*) et les systèmes basés sur l'événement (*event-based*). Dans la chorégraphie, les règles qui déterminent le comportement de chaque participant dans les processus sont créées. Le comportement global du processus qui émerge est basé sur l'interaction des pièces individuelles, chaque manière autonome selon leurs propres règles.

Il existe actuellement deux sous approches principales à la chorégraphie, *message-based* et *work-component-based.*

L'approche de message est basée sur l'examen des les messages entre les participants dans un processus global. On peut définir les comportements par capture exhaustivement des contrats de message entre les parties participantes. C'est le mécanisme supporté par la norme WS-CDL et il est souvent utilisé pour les applications de B2B. Dans les inter-entreprises par définition, il est difficile de préciser la mise en œuvre de participants spécifiques, et il n'y a aucune autorité centrale pour l'ensemble des flux. L'approche basée sur message est attrayante parce qu'on doit spécifier les définitions des éléments du message (syntaxique, sémantique et comportement).

Dans l'approche de *work-component-based*, on peut définir le comportement de *work-component* individuel et laisser le comportement des processus émerger lorsque chaque instance de processus spécifique évolue. Par exemple, vous pouvez mettre en œuvre un comportement dans *work-component* individuel avec quelques règles simples telles que : quelles sont les capacités qu'un *work-component* doit compléter, quels rôles peuvent répondre à chaque exigence, ce qui provoque une exigence de devenir actif, et ce qui provoque une

exigence d'être considérée comme complète. Dans un système complexe, ces règles peuvent être spécifiées dans les métadonnées de *work-component*, puis mis en œuvre dans un conteneur de *work-component*.

En résumé, avec chorégraphie :

- Le comportement du processus global qui "émerge" vient du travail de ses parties (*bottom-up*). Aucune perspective globale n'est nécessaire.
- Les processus complexes sont décomposés en *agenda* où chaque élément autonome contrôle son propre *agenda.*
- Elle est facilement *mappable* aux systèmes basés sur agent ou basés sur événement (*agent-based and event-based systems*)
- Elle est généralement plus difficile à démarrer, mais souvent plus facile au passage à l'échelle (*scalability*) des processus complexes.
- Les représentations graphiques peuvent être obtenues par le processus, c'est-à-dire la forme suit la fonction.

2.3.3 L'approche de Middleware pour la composition

Tout d'abord, on trouve des recherches actuelles concernant l'élaboration de composition selon les exigences du domaine des services. La littérature montre le travail dans l'élaboration des approches de la composition de service comme par exemple [Issarny, Georgantas, Hachem, Zarras, Vassiliadist, Autili, Gerosa and Hamida 2011, Milanovic and Malek 2004, Srivastava and Koehler 2003] illustré dans la figure 2.14.

En plus, on met l'accent sur le mécanisme de la composition de service qui permet aux applications d'être adaptables et reconfigurables. De nombreuses approches différentes sont utilisées pour résoudre ce problème, mais la plupart repose sur le middleware qui permet la découverte et l'invocation de service. Il propose deux parties en général, le premier est un *registry* qui contient les informations sur les services existants, par exemple location, entrée et sortie. Le deuxième est le moteur qui compose les services distincts et il est responsable d'exécuter de l'application.

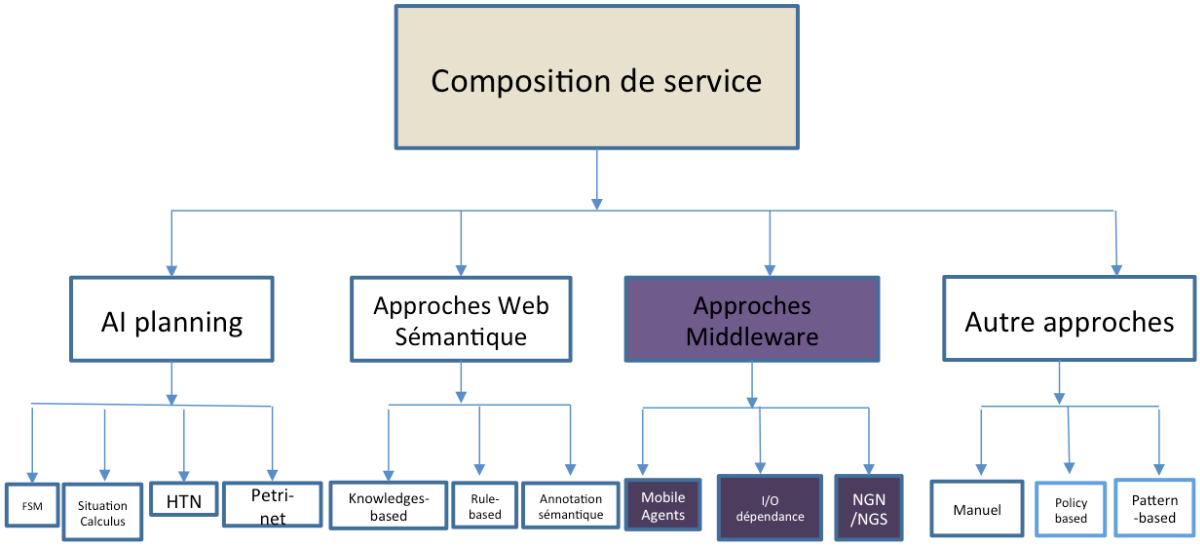


Fig. 2.12 : les approches de la composition de service

2.3.3.1 L'agent mobile

La première approche de middleware se trouve être la technologie de l'agent mobile. L'agent mobile est une entité autonome qui atteint un objectif soit seul, soit par collaboration. L'un de ses avantages est d'utiliser le langage sémantique pour supporter les interactions complexes. Comme celui proposé dans [Brnsted, Hansen and Ingstrup 2010, Vallée, Ramparany and Vercouter 2005], les applications de l'agent mobile sont initialement basées sur les composants, mais le langage sémantique permet les interactions entre les agents et les services. L'agent peut découvrir les services qu'il veut invoquer soit en cherchant un *registry* soit en se renseignant auprès des autres agents. Dans le même temps, les règles guidant les comportements d'agent peuvent être définies par l'utilisateur. L'avantage principal de cette approche est l'agent qui permet l'exécution du processus métier, en plus, il peut suivre les utilisateurs et ses actions et réagir aux changements.

2.3.3.2 Dépendances d'entrée/sortie

Une approche différente est présentée dans [Zahreddine and Mahmoud 2005], une manière de découverte dynamique des services est proposée. Chaque service est décrit par certains mots-clés et la liste de ses entrées et sorties. Chaque sortie peut être reliée à zéro ou une liste d'entrées afin que l'exigence de l'entrée produisant une sortie spécifique puisse être décrite. Le processus de la composition commence par la création des listes de services qui ont les entrées similaires et qui produisent les sorties similaires. Si un service n'est pas trouvé, la

recherche continue à créer une liste de services qui prend comme entrée, certaines entrées initiales et les sorties possibles de services dans la liste précédente. Ensuite la graphie d'invocation est établie. Si un service est en panne ou s'il a besoin d'être remplacé, le système utilise la liste de dépendance d'entrée/sortie pour trouver un remplacement.

2.3.3.3 NGN/NGS middlewares

2.3.3.3.1 Modèle de service

UBIS a proposé l'agent de QoS pour l'autogestion. Sur le Plan de Gestion illustré dans la figure 2.13, on trouve les fonctions d'autogestion d'un ES (*QoS monitoring, QoS contrat management, service deployment and accouting*), et les échanges avec sa communauté pour la gestion par le VSC. A ce niveau il traite les états associées à utilisation d'un ES.

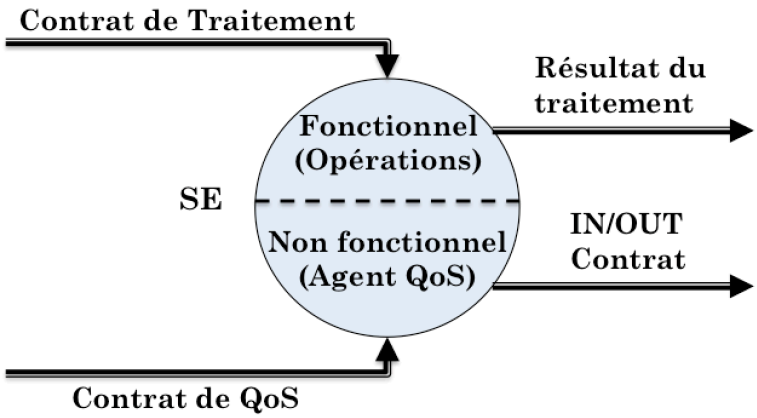


Fig. 2.13 Modèle de service

La ressource « élément de service » étant mutualisable par plusieurs utilisateurs, elle doit être allouée dynamiquement selon sa QoS courante. Cette dernière agrège les quatre critères du modèle de QoS(*Availability, Reliability, Delay and Capacity*) illustré dans la figure 2.14. Pour gérer la mutualisation de l'élément de service on propose d'associer à ce dernier une file d'attente qui contient les demandes utilisateurs acceptées par cet élément de service.

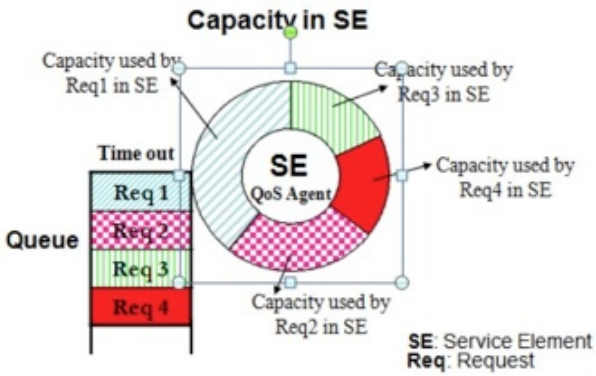


Fig. 2.14 Capacité dans SE

L'acception des demandes se fait selon la QoS courante de l'élément de service en termes de :

- Disponibilité : ou L'Accessibilité à l'élément de services, représente le nombre moyen de requêtes acceptées par L'ES. Ceci nous permet le dimensionnement au niveau des éléments de services ubiquitaires à déployer.
- Capacité : représente la capacité de traitement moyenne de l'ES pendant une unité de temps.
- Délai : représente le temps moyen de traitement de l'ES, il est géré par la file d'attente qui contrôle le délai d'attente dans la file pour chaque nouvelle requête.
- Fiabilité : représente le nombre de requêtes non servis par l'élément de service.

On prend en compte donc tous les paramètres de QoS pendant le processus du provisioning du service.

Dans [Nassar and Simoni 2011, Nassar and Simoni 2012], il propose un NGN/NGS Middleware illustré dans la figure 2.15 qui s'appuie sur l'architecture SOA/EDA et qui se base sur un ensemble de composants fonctionnels développés sous forme de services. Ce Middleware proposé suit une modélisation sous forme de *Middleware* (Service de Base, SEIB, Services Exposables) :

Les services exposables : ils peuvent être soit des services applicatifs élémentaires qui ne nécessitent aucune composition, soit des services applicatifs composables. Ils sont ainsi créés à l'aide d'un ensemble de services de base qui peuvent être composé avec d'autres services exposable.

SEIB : le *Service Elément Interconnexion Bus* a pour rôle d'interconnecter les éléments de services. C'est un ESB amélioré qui ajoute à sa fonction de médiation et d'interopérabilité entre les éléments de services une fonctionnalité d'interconnexion virtuelle.

Les services de base : Ils sont constitués d'un ensemble de composants fonctionnels développés en services tout en respectant le modèle de service introduit par les approches SOA améliorées.

Les services exposables seront exposés dans les catalogues des fournisseurs de services et des opérateurs pour que les utilisateurs puissent les solliciter. De plus tout utilisateur peut aussi jouer le rôle de fournisseur en exposant ses propres services.

La partie de composition horizontale de service s'appuie sur le concept de VPSN (*Virtual Private Service Nerwork*) qui est une abstraction des ressources à provisionner pour la session de services utilisateur. En effet, pour tout utilisateur, chaque service exposable demandé est parfois composé de plusieurs éléments de services. Le VPSN est un mécanisme de sélection et d'interconnexion virtuelle entre les éléments de services. Les éléments de services sont considérés comme des nœuds mutualisables qui sont enchainés par des liens lâches selon une logique de service personnalisée afin de constituer le réseau noté VPSN. Ce réseau est « virtuel » car il s'appuie sur des éléments de services mutualisables. De plus, le VPSN s'appuie sur des liens virtuels entre ces éléments de service, ce qui rend l'architecture du VPSN de plus en plus flexible. Ainsi, le VPSN est capable de s'adapter dynamiquement et sans couture à toutes modifications au niveau des capacités d'un service ou au niveau du contexte ambiant et des préférences de l'utilisateur. En outre, le VPSN est « privé » car il est constitué pour répondre à une demande de service d'un utilisateur donné.

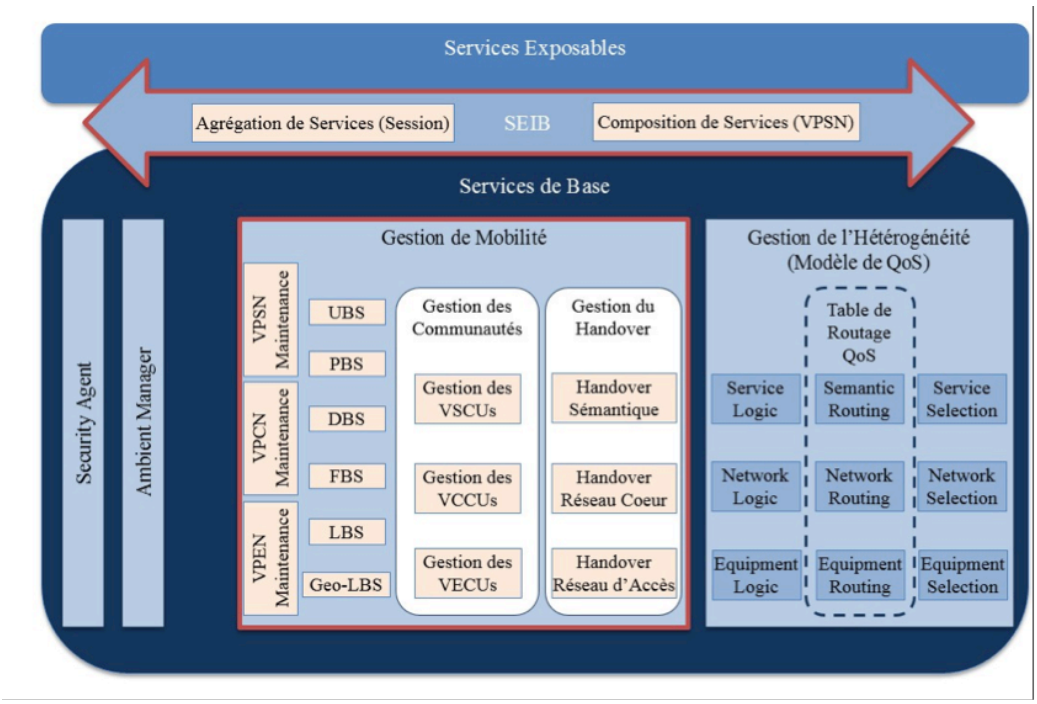


Fig. 2.15 : NGN/NGS middleware : Service de base

2.3.3.3.2 Les Communautés Virtuelles

Selon le journaliste politique Américain, *Norman Cousins*, la sagesse consiste à anticiper les conséquences. Cette stratégie s'appuie non seulement dans la politique mais aussi dans tout domaine menacé par erreurs. Dans notre approche *user-centric*, nous adoptons cette technique d'anticipation afin de prévenir toute discontinuité de la session de services de l'utilisateur. Puisque la cause de la coupure est soit le mauvais fonctionnement soit la dégradation de la QoS d'un des éléments de la session, nous basons notre anticipation sur le concept d'ubiquité. Ce dernier consiste à déployer d'une manière distribué des éléments ayant la même fonctionnalité et une QoS équivalente.

Ainsi, afin de garantir la continuité de services tout en tenant compte des préférences fonctionnelles et non fonctionnelles (QoS) de l'utilisateur, nous proposons une solution de gestion de la mobilité, qui est pilotée par l'aspect d'ubiquité, et qui s'appuie sur une anticipation dynamique et transparente des dégradations des services pré-provisionnés dans le VPSN de l'utilisateur. En effet, cette solution consiste à regrouper des services ubiquitaire dans des communautés virtuelle, dénommées VSCU (*Ubiquity-based Virtual Service Communities*). Par conséquent, quand un service sélectionné ne peut plus continuer à vérifier les préférences d'un certain utilisateur, il sera remplacé dans le VPSN de ce dernier par un autre service qui lui est ubiquitaire et qui appartient à sa VSCU. Ainsi, nous maintenons par anticipation et d'une manière et transparente la continuité de la session de services de l'utilisateur, tout en respectant ses préférences au niveau fonctionnel et au niveau de la QoS demandée.

Ce principe de gestion de mobilité par les communautés virtuelles n'est pas limité à la couche de service. Il est appliqué aussi sur la couche réseau et sur la couche des équipements. Ainsi, nous avons conçu, par analogie aux VSCUs, des VCCUs et des VECUs qui créent respectivement des communautés virtuelles au niveau des connectivités réseaux et au niveau des équipements. Cependant, dans le cadre de cette thèse, nous nous intéressons plus à la partie « *Service Delivery* ». Pour cela, dans la suite de cette partie, nous nous focalisons sur la couche service et nous traitons seulement le cas des VSCUs.

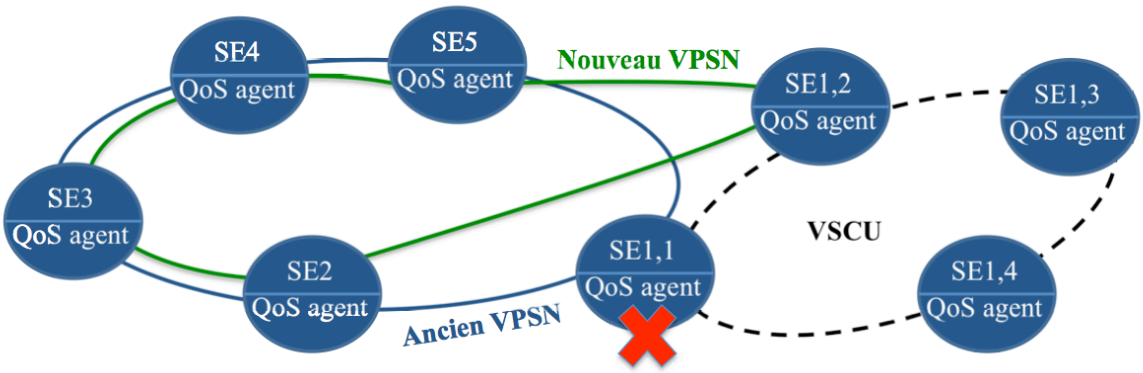


Fig. 2.16 : Gestion de la mobilité par les Communauté Virtuelles

Cette gestion de mobilité à l'aide des communautés virtuelles de niveau service est présentée dans la figure 2.16. Nous considérons le cas d'un utilisateur ayant un VPSN pré-provisionnant les services suivants : {SE1, 1 ; SE2 ; SE3 ; SE4 et SE5}, Nous supposons que le service SE1, 1 subit un mauvais fonctionnement ou une dégradation de sa QoS. Il n'arrive pas à respecter son contrat de SLA déjà négocié avec l'utilisateur. Afin d'empêcher toutes discontinuité de la session de services, nous anticipons en remplaçant le service dégradé par un autre service ubiquitaire, par exemple SE1, 2, appartenant à sa communauté VSCU créée en avance et contenant les services suivants {SE1, 1 ; SE1, 2 ; SE1, 3 ; SE1, 4}. Cependant la question qui se pose est la suivante : comment effectuer ce remplacement pour gérer la mobilité de la façon la plus rapide, la plus dynamique et la plus transparente ?

Pour répondre à cette question, nous adoptons une approche évènementielle qui s'appuie sur trois acteurs : un initiateur, un décideur et un exécuteur. Cette approche évènementielle est renforcée par une autre originalité qui consiste à avoir des éléments de service autogérés. En effet, l'autogestion de chaque service. Cet agent compare la QoS courante à la valeur seuil à ne pas dépasser. Si elle est supérieure, il envoie un *IN-Contract* qui montre que le comportement courant de service est toujours conforme à celui négocié dans le contrat de SLA, sinon il envoie un *OUT-Contract*. Dans ce deuxième cas, c'est l'agent de QoS du service dégradé qui détecte l'événement de non-respect du contrat signé et qui joue ainsi le rôle d'initiateur du processus de gestion par VSCUs. Ensuite, c'est lui qui va jouer le rôle de décideur en sélectionnant, parmi les membres de VSCU, un remplaçant ubiquitaire. Une fois la décision prise, le service VPSN Maintenance intervient pour jouer le rôle d'exécuteur en adaptant le VPSN en question. Cependant, la problématique qui se pose est de savoir comment le décideur va choisir le remplaçant ubiquitaire. Pour lever ce défi, deux algorithmes semblent possibles :

1) Le premier algorithme consiste à ce que le service dégradé envoie *l'OUT-Contract* en multicast, c.à.d. il l'envoie vers tous les services ubiquitaires qui se trouvent dans sa VSCU. Ensuite, il reçoit la réponse de chacun d'entre eux, et il choisit comme remplaçant le service ubiquitaire qui a répondu le premier.

2) Le deuxième algorithme consiste à ce que le service dégradé envoie *l'OUT-Contract* au premier service ubiquitaire de sa table de QoS, faisant partie de sa VSCU. Cette table classifie les services ubiquitaires de plus proche au plus loin d'un point de vue réseau, en se basant sur une QoS_{totale} qui est une agrégation de la QoS du service contenant cette table avec la QoS de service ubiquitaire ainsi que la QoS du lien qui les relie.

Il faut noter qu'un service peut être pré-provisionné dans plusieurs VPSN correspondant à différents utilisateurs. Par conséquent, un service qui ne peut plus vérifier le contrat de SLA signé avec un utilisateur donné, peut toujours rester conforme à d'autres contrats signés avec d'autres utilisateurs. Ainsi, le processus de maintenance ne sera exécuté que pour le VPSN vis-à-vis duquel le service est dégradé. La création des communautés virtuelles et comment structurer ces communautés virtuelles au niveau architectural ainsi que comment les créer et les gérer au niveau fonctionnel sont fait par d'autres chercheurs au sein de groupe de recherche de UBIS.

2.3.4 Le « Mashup »

Le Wikipédia définit le « mashup » ou « mash up » comme une application composite qui combine des contenus ou des services provenant de plusieurs applications plus ou moins hétérogènes. Dans le cas de site web, le principe d'un mashup est donc d'agréger du contenu provenant d'autres sites, afin de créer un site nouveau.

Le principe de mashup est sur la manière de participation de l'utilisateur interactif, la façon dont il rassemble et regroupe les données de la partie tiers. Il est dérivé de l'évolution de la technologie d'internet, en particulier de la popularité du Web 2.0. Web 2.0 permet que le page web du site ne soit plus un document statique (ex. HTML), mais soit un genre plus dynamique et interactif qui peut être consommé (par RSS, REST et ATOM). Pour AJAX et *Rich Internet Application* (RIA), ils ont changé la manipulation de la page web de *Document Object Model* (DOM) à *Web-Widget* (par exemple, DOJO [Lawton 2008]), les contenus et ses données dans la page web sont organisés de la manière de plus en plus composable. En outre avec ces « composants du web », les consommateurs peuvent même accomplir certaines logiques métiers dans le navigateur du Web au lieu d'accéder à la couche résidant dans le côté serveur.

Comme de plus en plus d'entreprises permettent le support commercial RSS/ATOM/REST dans leurs services, mashup se propage à travers le web, en s'appuyant sur le contenu et la fonctionnalité extraits des sources des données qui se trouvent en dehors de ses frontières organisationnelles.

Le mashup est généralement une manière de composer mais généralement effectuée au niveau du navigateur web, par « glisser-déposer » ensemble des applications provenant de sources différentes. Il peut être défini suivant deux genres : mashup du côté serveur, et mashup du côté client.

- Le mashup du coté serveur: tous les services de mashup doivent être intégrer du côté serveur, et puis le résultat doit être retourné vers le côté client. Comme dans la figure 2.8 gauche, les autres services dans les serveurs différents représentent peut-être service SOAP ou service REST.
- Le mashup du coté client: tous les services en mashup doivent être intégrer en particulier du côté clients. Comme illustré dans la figure 2.17 droite.

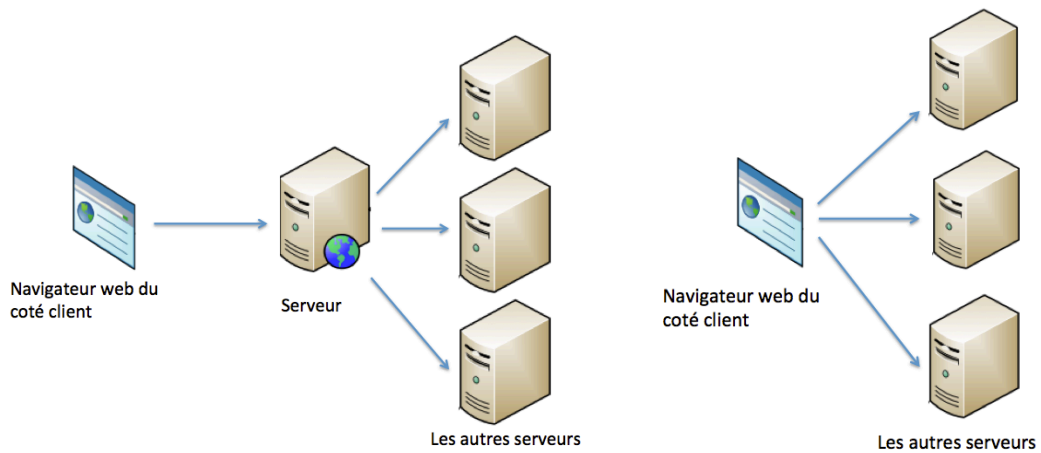


Fig. 2.17: mashup du côté client et mashup du côté serveur

Le « mashup » dans [Liu, Hui, Sun and Liang 2007, Yee 2008, Yu, Benatallah, Casati and Daniel 2008], il s'agit de composer certains services différents pour avoir un service plus puissant et mieux répondre aux exigences. C'est donc essentiellement un processus qui intègre les données/contenus provenant de sources différentes sur l'Internet. Par conséquent, il s'agit essentiellement d'un style de composition de service du point de vue SOA. En considérant le principe de Mashup et le rôle mentionné dans le paragraphe précédant, on peut étendre la base de SOA (*service provider, broker and consumer*) comme :

- Concepteur de composant mashup: les services doivent être fournis par plusieurs fournisseurs de service, chacun d'entre eux a sa propre spécification et méthode d'utilisation.

Par exemple, les service peuvent être service web (SOAP ou REST), JavaBeans de l'entreprise ou système hérité. Le premier problème de mashup est l'interopérabilité des services. D'autre part, mashup est principalement la composition au niveau de UI, tel que la composition actuelle au niveau de logique métier. Le modèle de composant mashup est l'extension de fournisseur de service. Le concepteur de composant mashup sélecte le service du catalogue de service (ex. UDDI) et puis prend la responsabilité d'encapsuler tous les services dans un modèle de composant standard avec l'interface utilisateur. Ensuite le composant de mashup est publié dans un *repository*.

- Serveur de Mashup: le serveur de mashup invoque trois composants. Le Catalogue de Service qui est le service d'UDDI pour publier les service du fournisseur. Le *repository* stocke tous les composants de mashup développés par le concepteur de composant mashup. Le Monitoring offre l'évaluation de performance (comme *Dashboard*) pour les composants de mashup. Il assure aux consommateurs le choix des composants mashup appropriés et les fournisseurs peuvent reconfigurer ceux qui sont inadéquates.
- Consommateur de Mashup: le consommateur de mashup sélecte les composants de mashup du coté serveurs et puis compose leur propre application dans le navigateur ou script de UI.

2.3.5 Conclusion

Dans ce chapitre, on commence par parler du développement des technologies du Web ainsi que des catégories de WS hétérogènes dans SOA. On analyse le service de mashup et le service dans le *cloud*. Ensuite, on analyse les méthodes et les technologies de composition dans le contexte de SOA. La composition de service est définie comme le processus de combinaison des services existants pour créer des nouveaux services. Elle est nécessaire dans de nombreux services appartenant à différentes entreprises/fournisseurs qui sont composés ensemble et exécutés pour réaliser des métiers de fonctionnalité complexe. Le paradigme des services Web est actuellement vu en tant que la conception du service et l'architecture de déploiement, donc la plupart des approches de composition de services se concentrent sur la WSC. On trouve le mode de médiation et le mode de P2P pour la composition de service ainsi que les techniques et les approches appliquées, il s'agit des approches middleware. Basé sur l'approche middleware et le contexte UBIS, on va proposer une solution de composition de service pour avoir une coexistence de service SOAP et service REST dans la même session illustrée dans la figure 2.18.

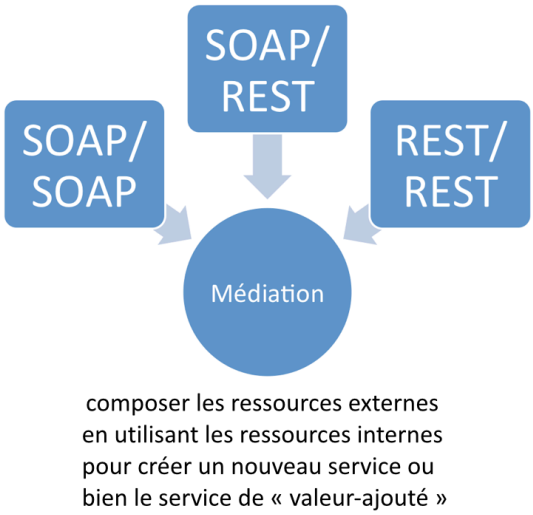


Fig. 2.18: coexistence de service SOAP et service REST dans la même session

Chapitre 3

Première proposition : le WS-médiateur

3.1 Introduction

Les méthodes et les techniques présentées dans l'état de l'art tiennent compte des différentes caractéristiques des services web pour fournir la WSC. Cependant, il existe certaines exigences de composition qui s'appliquent à toutes les techniques. Le but de ces méthodes est la combinaison de services d'une manière telle qu'ils puissent interagir entre eux en offrant une solution pratique. Au cours de la composition, toutes les caractéristiques fonctionnelles et non-fonctionnelles doivent être prises en compte afin de fournir une solution complète pour avoir une composition dynamique de service selon les exigences de l'utilisateur.

Dans ce chapitre, tout d'abord, on propose notre modèle architectural avec les modules fonctionnels, c'est une architecture de médiation pour la composition dynamique de service web de *user-centric*, dans le même temps, on a considéré la composition de services web différents (SOAP&SOAP, SOAP&REST et REST&REST). On fournit le mashup basé sur notre médiation pour que l'utilisateur puisse personnaliser les services d'après sa logique applicative. La mobilité de service et celle de l'utilisateur sont prises en compte par UBIS.

3.2 Une approche WS-médiateur pour la composition dynamique de service web

On conçoit une architecture globale pour pouvoir fournir une composition dynamique de service web afin de satisfaire les exigences de l'utilisateur. Nous présentons une médiation qui est *user-centric* pour la WSC en considérant les problématiques associées avec la composition pour l'ensemble des processus et les services différentes.

3.2.1 Les exigences

Notre modèle d'exigence de WS-Médiation est illustré dans la figure 3.1.

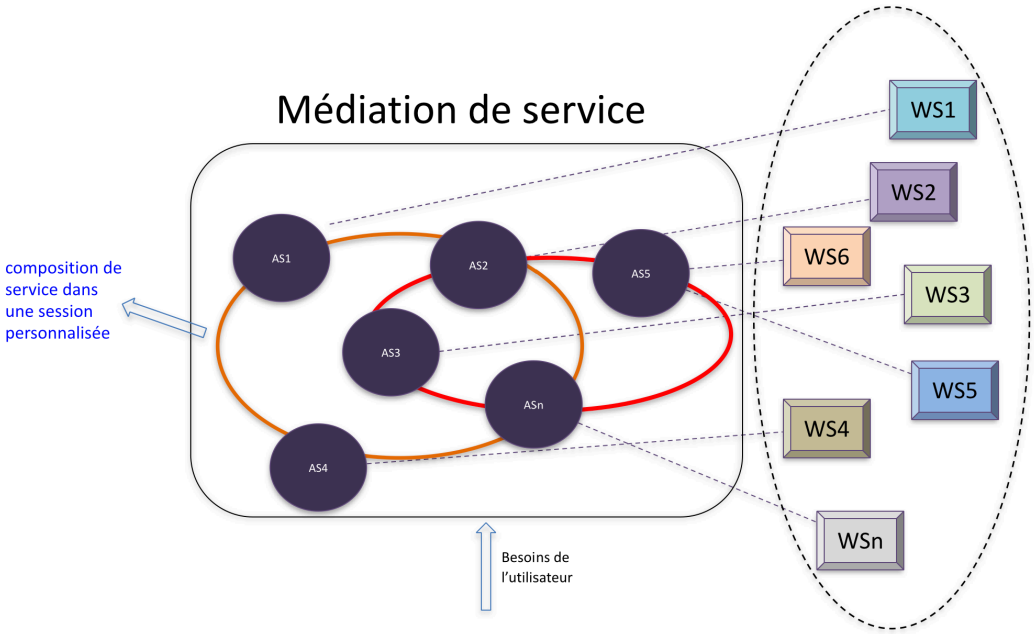


Fig. 3.1 Modèle d'exigence

Nous présentons notre architecture de médiation de la WSC. Dans notre médiation, on a conçu la composition des modèles de services. Chaque WS réel est adapté au modèle de service, l'adaptateur est compris dans notre médiation. Pour chaque modèle de service on a un profil de service correspondant et enregistré dans une base de connaissance.

D'après les exigences de l'utilisateur, notre médiation fonctionne de deux façons : la première, c'est quand la médiation va retourner un service composé fourni comme une orchestration de service. La deuxième façon, c'est quand l'utilisateur peut composer les services via notre médiation, puis la médiation va retourner une composition d'élément de service personnalisé pour application mashup de l'utilisateur.

L'application du client fournie, contient la composition dynamique de SE et le script du processus métier du coté client. Par rapport au style de mashup du coté client et du coté serveur, notre application du client fournie, est un genre de mashup mixte client-serveur, qui a les caractéristiques suivantes :

- *User centric*: l'application de client est considérée pour l'utilisateur (consommateur) mais pas pour le développeur (producteur). Tout consommateur peut composer son/ses propres applications de services puisque l'utilisateur peut réaliser le mashup par les actions "glisser-déposer" au sein d'un navigateur web.
- *Lightweight*: l'application de client est une intégration tactique légère des services/applications ou de contenu multi-provenant en une seule offre. Parce le « *mashup* » exploite les données et les services de sites Web publics et des applications Web, ils sont légers dans la mise en œuvre et construits avec un minimum de code. L'avantage de leur activité principale est qu'ils peuvent

répondre rapidement aux exigences tactiques avec des coûts de développement réduits et une meilleure satisfaction de l'utilisateur.

3.2.2 Modèle Architectural de WS-médiateur

Le WS-médiateur est implémenté comme une médiation de service qui permet de fournir les fonctionnalités demandées pour satisfaire les exigences des utilisateurs. Nous présentons ci-dessous les principaux éléments et conditions de réalisation de notre approche illustrée dans la figure 3.2.

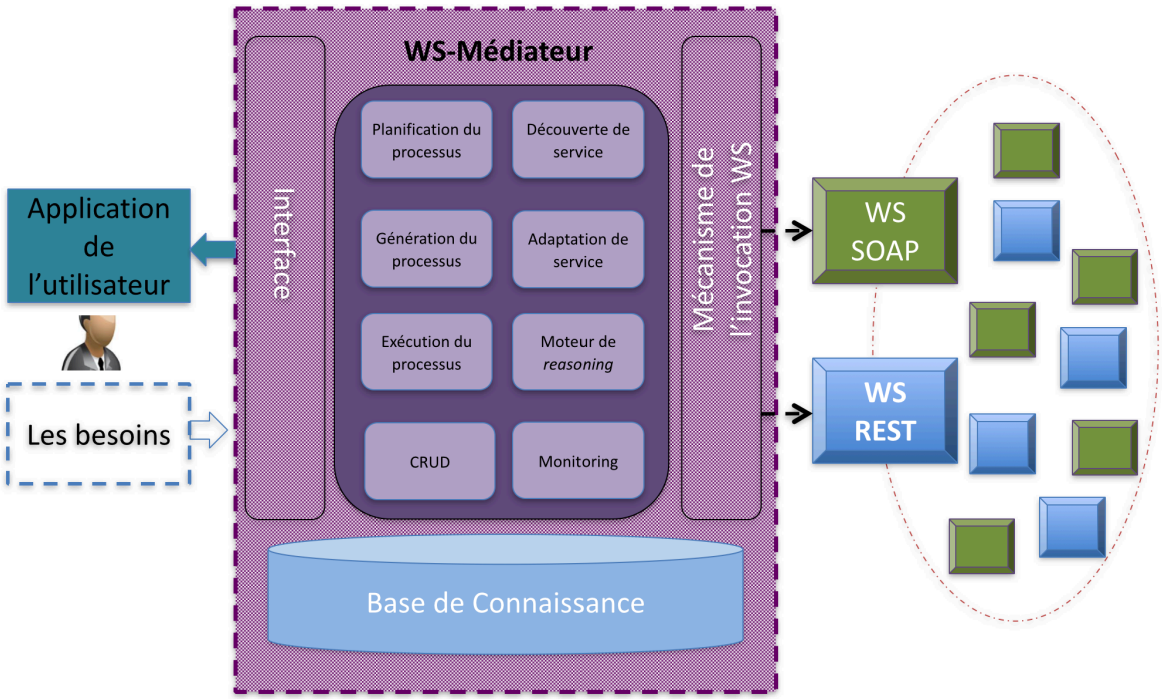


Fig. 3.2 Modèle Architectural de WS-médiateur

Nous arrivons à faciliter la médiation des données échangées entre service web composés. Lors de l'organisation des interactions entre les WSs, de nombreuses hétérogénéités peuvent apparaître entre les fonctionnements de ces services, il s'agit de suivre des types d'interactions différents, d'échanger les données selon différentes séquences d'échanges de messages, de prendre en charge à des degrés différents et de gérer les transactions, etc. C'est-à-dire que notre architecture de médiation adapte les services différents que ce soit le service WSDL/SOAP ou le service RESTful. Elle nous permet d'avoir une convergence de composition dynamique orientée *user-centric* au lieu de ressource centrique ou d'opération centrique. Précisément, nous proposons des composants qui peuvent interagir entre eux dans

notre WS-médiateur, afin que le médiateur puisse composer les services SOAP et SOAP/ SOAP et REST ainsi que REST / REST séparément.

3.2.3 Spécification de la fonctionnalité des modules de WS-médiateur

3.2.3.1 L'interface

Le médiateur interagit avec le client via une interface, essentiellement, l'interface devrait avoir les fonctionnalités suivantes :

- Accepter les requêtes de service d'un client pour la médiation de composition dynamique de service avec les candidats de WSs.
- Accepter les politiques de service telles que définies par le client.
- Accepter la soumission des informations par les services web.
- Accepter les requêtes d'un client pour les métadonnées de service/composition de service web.
- Renvoyer les résultats de la médiation au client.
- Stocker les métadonnées de service web du client dans un cache temporaire /base de connaissance permanent pour la réutilisation.

3.2.3.2 Mécanisme de l'invocation de WS

Le mécanisme de l'invocation de WS peut être fourni par une variété d'organisations et d'entreprises, qui met en œuvre des mécanismes différents définis dans les spécifications de WS. Comme le médiateur prend évidemment différentes méthodes de liaison (*binding*) des messages et des méthodes d'invocation etc., le mécanisme de l'invocation de WS est un agrégat de *binding* de messages et de méthode d'invocation pour répondre au WS différent. La méthode de liaison de message et le type d'invocation peuvent être définis dans les politiques de service, par exemple dans le contrat de service sous le *framework* de C#.NET.

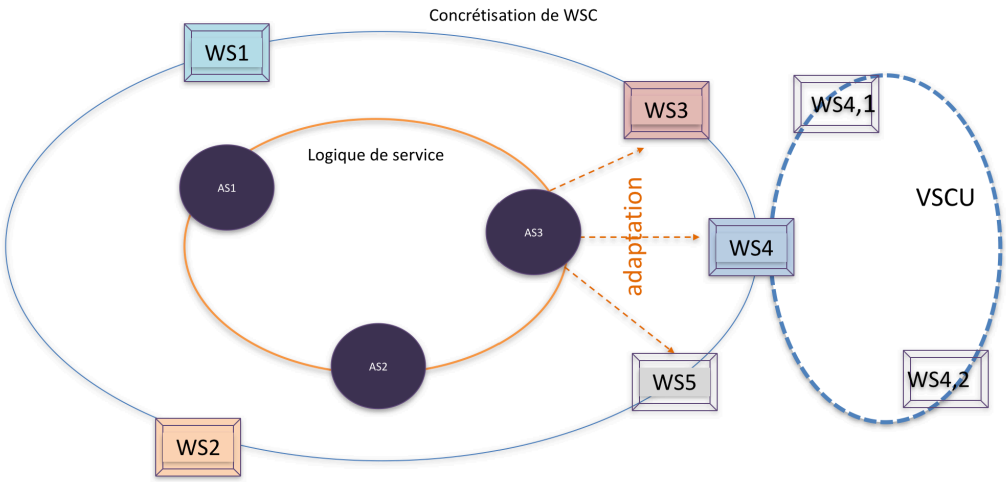


Fig. 3.3 l'invocation de service

Le médiateur prend évidemment différentes méthodes de liaison (*binding*) des messages et des méthodes d'invocation. On doit donc adapter les types de *binding* et les méthodes d'invocation pour appeler les WSs différents. La méthode de *binding* et le type d'invocation peuvent être définis dans les contrats de service.

Pour ce faire, on doit avoir un *mapping* entre les logiques de services abstraits et la concrétisation de la composition de service. C'est à dire, que pour chaque service abstrait, on fait une adaptation au service concret pour choisir le meilleur service demandé. En plus, il y a un autre sens de l'adaptation, c'est l'adaptation de l'aspect QoS grâce à la VSCU proposée par UBIS. Dans tous les cas, c'est pour choisir le meilleur service demandé.

3.2.3.3 Base de connaissance

Le médiateur se dote d'une base de connaissance (BDC) sur les WS disponibles qui peut être mis à jour dynamiquement. Cette base permet à notre architecture de fournir aussi bien une extension vers le Web sémantique qu'une capacité d'adaptation au contexte (ubiquité). En effet, d'une part, par l'identification des WSs (recensés dans la base) les mieux adaptés à la logique de l'utilisateur, le médiateur est capable de proposer une composition de service adéquate. D'autre part, nous pouvons également, selon la même logique, repérer l'entité de service la mieux adaptée à un service (abstrait) en fonction du contexte de l'utilisateur, ceci peut être couplé avec la fonction de découverte de services qui alimentera dynamiquement la base de connaissance.

Donc plus concrètement, la BDC consiste à rassembler toutes les informations (comme les URI, le fonctionnement, l'entrée / sortie à partir de WSDL) sur les services Web. Cette

«BDC» est utilisée pour afficher le catalogue disponible et pour connaître le modèle exact retenu. La «BDC» est constituée manuellement (hors ligne) dans la phase actuelle de notre travail, en partant de la description WSDL de chaque service. On définit deux sous bases dans notre BDC :

- Base de Modèles, qui recense les modèles de service et les modèles de processus. Pour les services couramment utilisés, un modèle synoptique peut être généré par le fournisseur de service.
- Base de QoS, qui stocke la description de QoS. Le médiateur peut trouver la description sémantique de QoS d'après WSDL pendant les périodes de découverte et de *matching*. Dans le processus de *matching* et d'optimisation, l'évaluation des performances QoS peut être associée à des paramètres de service pendant l'exécution, il faut sauvegarder les paramètres dans le médiateur.

Par exemple, pour le UI et le module de planification, il y aura aussi une interaction avec «BDC» pour connaître les apports de chacun. Nous nous assurons que chaque entrée de chaque service invoqué a été fait dans les règles. Chaque entrée est dans l'un des trois cas: 1) «constante»: dans ce cas, le médiateur invite l'utilisateur à fixer la valeur. 2) "*run-time*": dans ce cas, le médiateur va insérer, dans l'entité d'exécution autonome, un code spécifique permettant à l'utilisateur d'entrer la valeur en temps réel. 3) "d'un autre WS": dans ce cas, un lien est établi entre cette entrée et une sortie. Bien sûr, le médiateur a vérifié la compatibilité des liens établis par l'utilisateur (par exemple, une «chaîne de caractères » en sortie ne peut pas être mappée à une entrée de type "*integer*").

En plus, si le module de monitoring interagit avec cette BDC, les résultats de la surveillance sont utilisés pour générer dynamiquement et mettre à jour les métadonnées de la fiabilité de ces services Web, ce qui permet de réaliser l'adaptation dynamique explicite de la composition de services Web au moment de l'exécution. Le système peut être facilement utilisé par les applications, pour prévoir des mécanismes de tolérance aux pannes afin améliorer la fiabilité de la composition de service sans modification des composants de service. Cela est particulièrement bénéfique pour l'intégration des services Web autonomes.

3.2.3.4 Planification de service

Le module de planification du processus proposé fonctionne avec le composant de l'interface du médiateur pour l'utilisateur. Le UI et sa composante de service *binding* peuvent

être connectés et mis à jour dynamiquement. Ce module transforme les descriptions des tâches en précisant l'ensemble de la description des activités et de leurs associations, ces activités sont représentées par WS.

La planification du processus permet à l'utilisateur de composer les services au niveau de l'interface. Pour les utilisateurs, le composant d'UI est l'entité unique qui survit dans les applications clients et il peut immigrer vers les navigateurs web par un script.

L'utilisateur peut composer les services à travers l'interface, après « *opening session* », logique des services qui a été stockée dans un fichier XML, peut être envoyée par SIP (si l'infrastructure est basée sur IMS) et puis gérée par le VPSN (*Vitrual Private Service Network*) mentionné. Dans un premier temps, l'utilisateur peut choisir les services disponibles dans une liste de services avec certaines interactions.

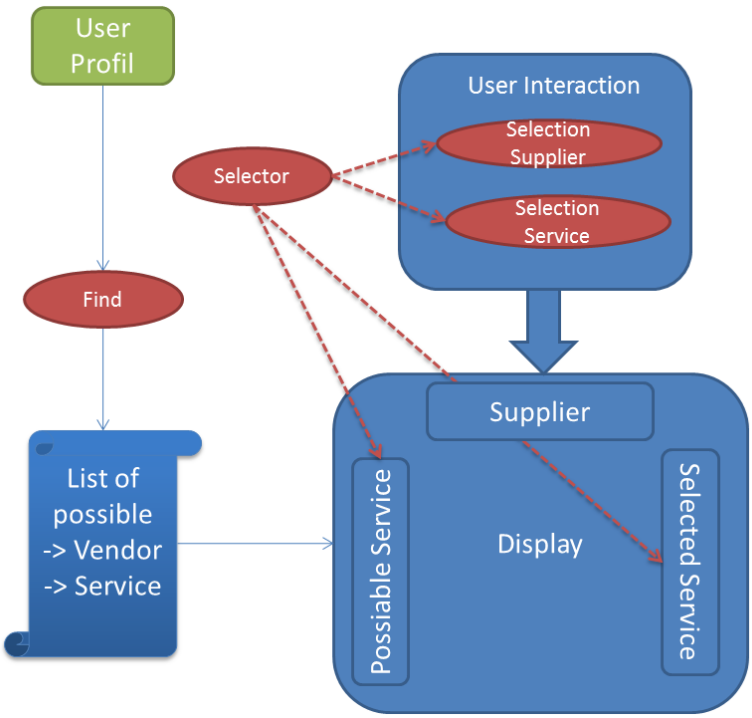


Fig. 3.4 IHM pour la planification

Et puis, l'utilisateur peut composer les services d'après sa logique pour avoir certaine transaction de services. C'est ainsi que l'on peut définir les logiques de transactions.

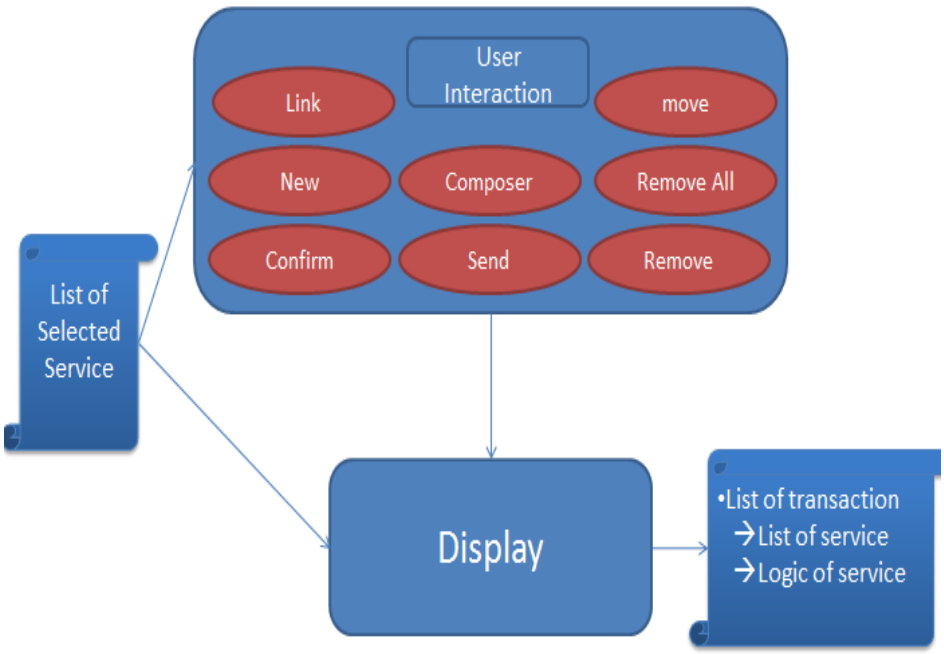


Fig. 3.5 Gérer les transactions de service

Une fois la « *opening session* » faite, la planification de processus va être générée et stockée dans un ficher XML pour pouvoir être utilisée.

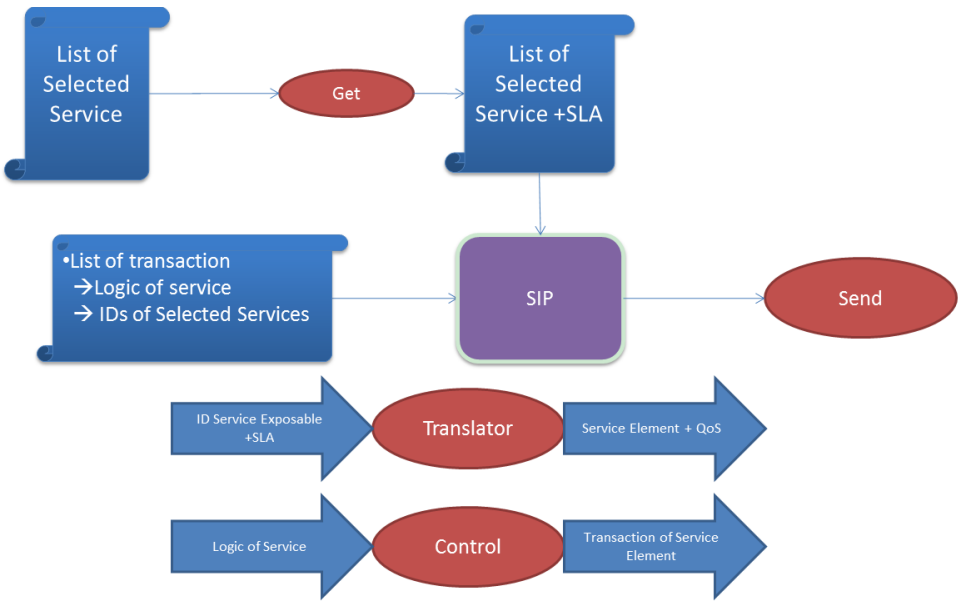


Fig. 3.6 *Opening session*

3.2.3.5 Découverte de service

Ce module prend la description des activités en Input, et aide l'utilisateur à travers l'acquisition de l'état d'un service correspondant, y compris sa disponibilité, la fonctionnalité

et l'interface d'interaction. Il fonctionne avec un répertoire de service et il désigne le processus par lequel les futurs utilisateurs d'un service recherchent manuellement/ semi automatiquement le service correspondant à leur exigence. Les WSs auront été préalablement créés et publiés dans les annuaires des WSs comme le UDDI. Le fonctionnement de recherche UDDI peut donc se focaliser sur un seul critère de recherche, comme le nom, le lieu ou la catégorie de l'affaire, ou bien le type de service par nom, l'identifiant métier, ou l'URL de découverte.

Dans le cadre de .NET pour tester ce module, on peut faire ce module à partir de l'URL d'un document de découverte résidant sur la BDC (Base de Connaissance). Le développeur d'une application cliente peut apprendre qu'un WS existe, connaître ses fonctions et comment interagir correctement avec lui. Les documents peuvent être des descriptions de service, des schémas XSD ou des documents de découverte. A l'aide de l'outil Wsdl.exe de .NET, vous pouvez créer une classe proxy pour le WS décrit par une description de service ou schéma XSD.

3.2.3.6 Génération du processus

Ce module cherche à résoudre les tâches de l'utilisateur en composant les WSs de SOAP/WSDL publiés. Quand un nouveau WS arrive, on doit le comparer avec ceux qui existent pour déterminer à quel type il appartient. Au moment de la comparaison, on doit utiliser les inputs et les outputs de service. Donc on utilise le parseur pour analyser le profil du service pour obtenir les identifiants ou URI des inputs/outputs. Le générateur de processus prend la description des activités et les résultats de la découverte de service en tant qu'entrées (Input) et délivre en sortie (Output) un modèle de processus au format XML qui décrit l'ensemble des services sélectionnés, le flux de contrôle et les flux des données entre ces services. Plus précisément, il analyse les requêtes de client et les politiques de service, il rassemble les processus métiers et réalise une série d'activités à exécuter. Le générateur reçoit le processus des éléments de service crée par l'utilisateur, il analyse ce processus au format approprié puis fait le *matching* et *binding* d'après les types et les étiquettes de service.

3.2.3.7 Exécution du processus

Le générateur peut recevoir le résultat approprié pour créer un processus métier exécutable décrit en XML. L'entrée de ce module est le modèle de processus. En configurant les fichiers WSDL et le processus métier, l'output est donc un fichier XML qui contient l'ensemble de la composition. Le module d'exécution reçoit le schéma de composition fournie par le module

de génération du processus. L'utilisateur peut être impliqué dans cette phase en introduisant les valeurs (si il y en a) du type d'entrée capturée dans les WSs diverses.

En considérant que chacun WS est autonome, quand nous voulons exécuter la composition de services, l'utilisateur n'a aucun contrôle sur les services. Même si les services eux-mêmes ne sont pas fiables, ils peuvent être modifiés (supprimés ou mis à jour) par les fournisseurs de services. Par conséquent, la possibilité de défauts est élevée lorsque la composition de service est exécutée. Nous nous occupons de ce problème au moment de la planification, mais pas au moment de l'exécution. Si nous voulons composer deux WSs, nous allons commencer par choisir les services que nous voulons utiliser, tester si les services sont disponibles, et vérifier les entrées et sorties de chacun d'eux. Une fois que les services sont prêts à être utilisés, la liaison sera implicite dans l'invocation des services composés.

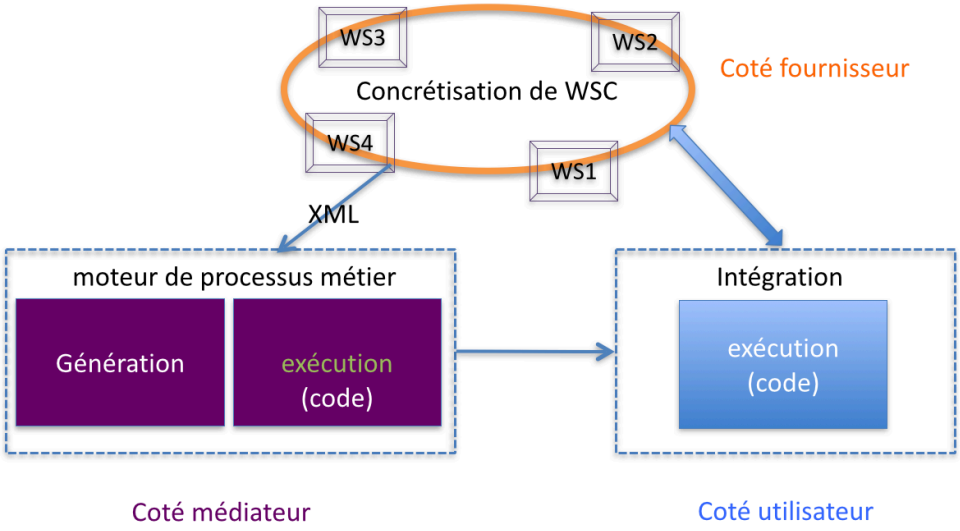


Fig. 3.7-a moteur de processus métier

Le moteur de processus métier peut recevoir le résultat approprié de concrétisation de WSC décrit en XML. L'entrée de ce module est le modèle de processus (schéma XSD et XML). Plus précisément, le générateur reçoit le processus, il analyse les requêtes de client et les contrats de service de ce processus puis fait le *matching* et *binding* d'après les types et les étiquettes de service fournis par l'adaptation. Puis passer au module d'exécution. Donc du coté médiateur, le moteur de processus métier peut donner un résultat de l'exécution à l'utilisateur, et dans le même temps le médiateur peut assurer la QoS fournie. Ou bien le médiateur peut fournir à l'utilisateur une intégration de code à exécuter, Mais dans ce cas, on ne peut pas assurer la QoS. Du point de vue de l'aspect fonctionnel, cela répond bien à notre positionnement dans SOA.

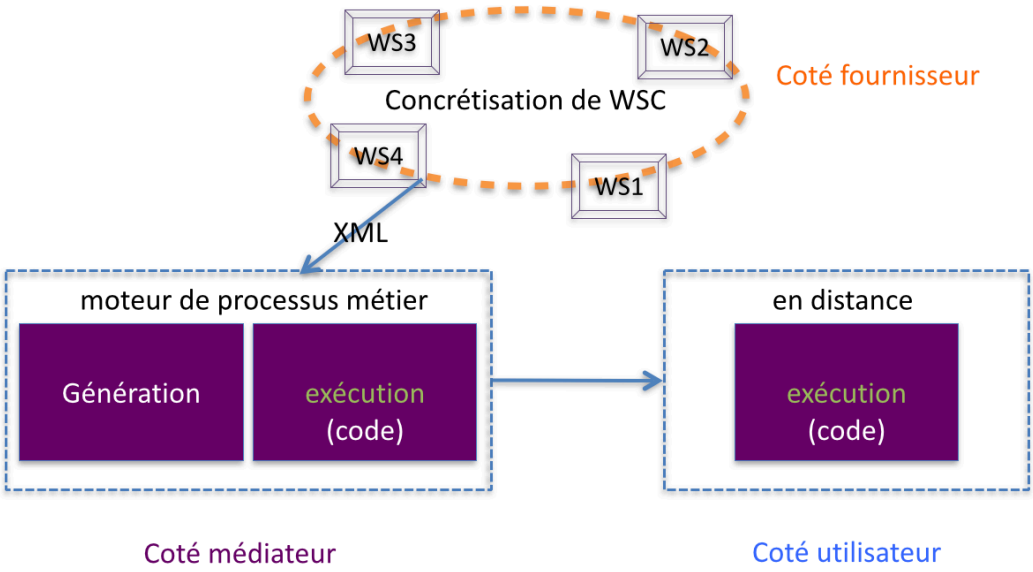


Fig. 3.7-b moteur de processus métier

3.2.3.8 Moteur de *Reasoning*

Le but de ce module est de prendre l'information contextuelle à partir des ressources diverses comme entrée, puis le procéder avec les règles dans la base de connaissance, et fournir les informations de décision aux modules de planification et de génération. Ce module est nécessaire pour soutenir les fonctionnalités sémantiques du web, par exemple l'IOPE (*Inputs, Outputs, Preconditions, Effects*) de *First-order-logic*. Dans ma thèse ce module est proposé théoriquement mais pas encore dans l'expérimentation.

3.2.3.9 Monitoring

Le but de ce module est de prendre l'information contextuelle à partir des ressources diverses comme entrée, puis le procéder avec les règles dans la base de connaissance, et fournir les informations de décision aux modules de planification et de génération. Ce module est nécessaire pour soutenir les fonctionnalités sémantiques du web, par exemple l'IOPE (*Inputs, Outputs, Preconditions, Effects*) de *First-order-logic*. Dans ma thèse ce module est proposé théoriquement mais pas encore dans l'expérimentation.

3.2.3.10 Adaptation de service

La plupart des applications basées sur SOA utilisent le processus métier pour la composition de service, mais les technologies basées sur le processus métier peuvent composer que des services SOAP mais pas les services RESTful. Le but de ce module de conversion de service

est d'intégrer le service SOAP et le service RESTful à travers un même processus métier. Il faut que le service RESTful ait un fichier de description pour pouvoir le transférer en un service SOAP. On choisit donc WADL mentionné dans l'état de l'art comme langage de description pour le service RESTful, grâce auquel le système peut produire un proxy pour notre processus de composition.

Dans le cas de service REST qui fournit nombres de fonctionnalités ainsi que plusieurs méthodes de HTTP, le ficher de WADL doit être plus compliqué, donc on a besoin d'un outil de conception visuel pour nous aider à rédiger de ficher WADL.

Il existe pas mal de projets qui sont en Open source dans *Google Code*, *REST Describe*, en est un, parmi eux, qui nous permet de développer le ficher WADL facilement et rapidement. Comme illustré dans les figures 3.8 ci-dessous.

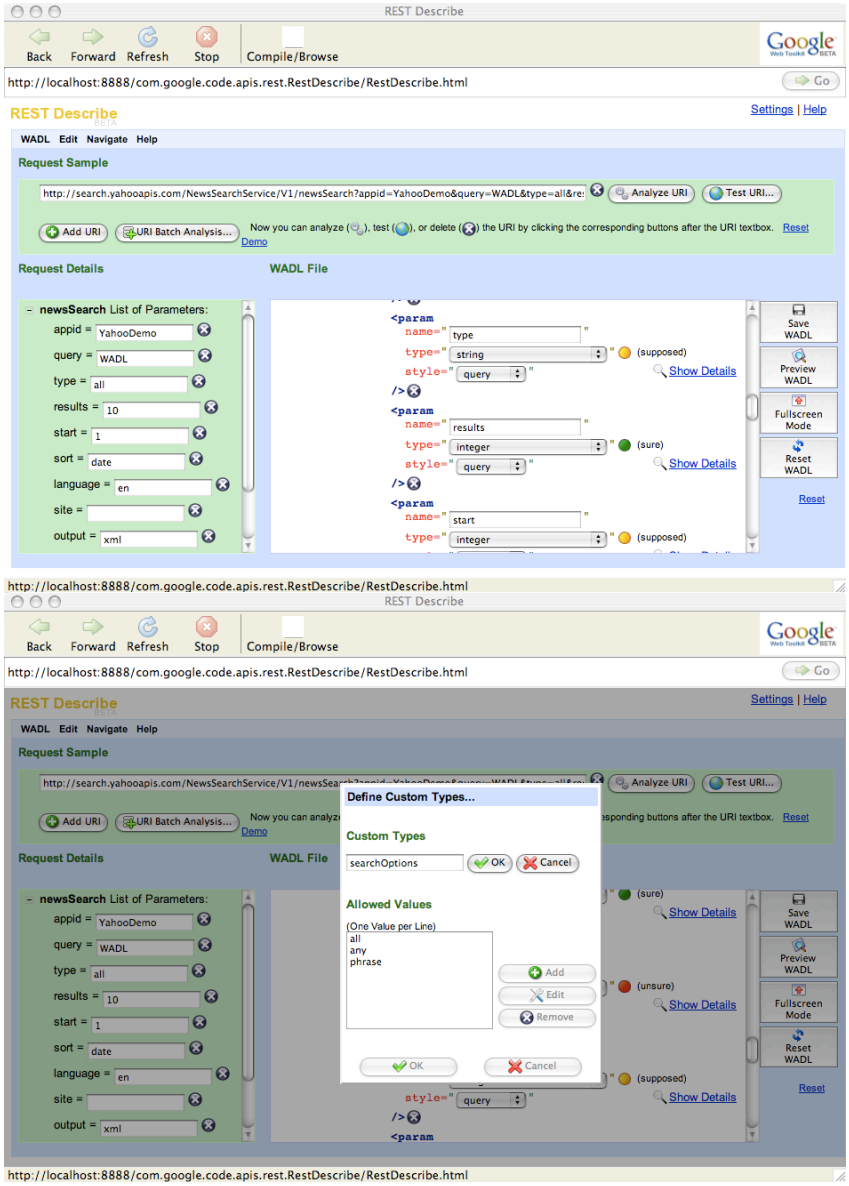


Figure 3.8 : Outil de REST *REST Describe*

Il faut que notre composant nous permettre de transférer le service RESTful en service SOAP et générer automatiquement le ficher de description de service SOAP. Il n'y a aucune différence par rapport à la rédaction de fichier XML, donc on peut intégrer tous les deux genres de service afin de fournir une composition de service au sein de processus métier pour l'utilisateur. Mais le développeur ou l'utilisateur peut transférer le service RESTful en service SOAP manuellement.

Lorsque le processus métier est basé sur les normes de WS-*, par exemple, le WS-BPEL et WSDL1.1, mais que le service RESTful ne peut pas être décrit par l'interface de service fourni par WSDL1.1, notre moteur de processus métier (module de génération et d'exécution) ne supporte pas les *bindings* et les invocations de service RESTful, sauf que l'on donne une interface de la manière de WS-* au service RESTful à travers un adaptateur de service web.

L'adaptateur de service web fournit une interface d'HTTP pour notre moteur de processus métiers. En plus le moteur de processus métier peut visiter le service RESTful pour invoquer le message SOAP à travers WS-adaptateur. Les fonctionnalités de WS-adaptateur sont :

1) De fournir l'interface de SOAP pour le moteur de processus métier, il supporte la norme de WSDL et il peut décrire l'interface de SOAP puis réaliser le *binding* par le moteur de processus métier.

2) De fournir un connecteur d'HTTP, il supporte l'invocation de l'interface d'HTTP puis réaliser la transformation de message SOAP et HTTP.

3) De réaliser la transformation entre le format de représentation de ressource de REST (par exemple JSON ou Atom) et le format de XML (par exemple, Plain Old XML, POX).

Le service RESTful fournit quatre genres de méthodes pour manipuler les ressources, donc le WS-adaptateur fournit quatre méthodes de standard correspondants basées sur les messages SOAP : onGET(), onPOST(), onPUT() et onDELETE() qui sont utilisés pour le *mapping* du message SOAP aux méthodes d'HTTP(GET POST PUT DELETE) illustrées dans la figure 3.9.

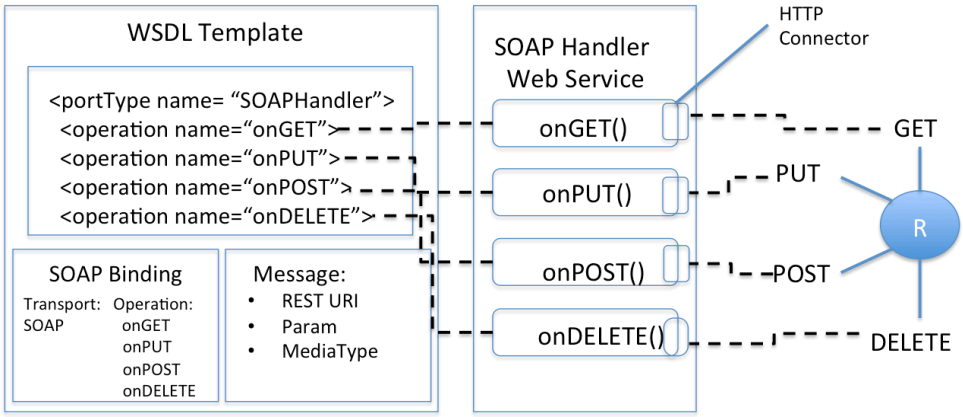


Fig. 3.9 Structure de WS-adaptateur

Pour chaque interface de message SOAP, on a un fichier de WSDL pour le décrire. Dans le Template de WSDL, <*message*> se trouve une tribu de paramètres, les valeurs de ces paramètres sont générées par les services RESTful concrètes. <*REST URI*> spécifie l'adresse URI du service RESTful. <*MediaType*> spécifie le format de ressource. <*Param*> spécifie le paramètre demandé par l'interface de service. Quand un service RESTful est publié, une instance de WSDL est générée, les paramètres sont décidés par les interfaces spécifiques. Les quatre standards de l'interface de SOAP et le ficher de WSDL seront utilisés pendant le *binding* et l'invocation de service lors de l'exécution de processus métier illustré dans la figure 3.10.

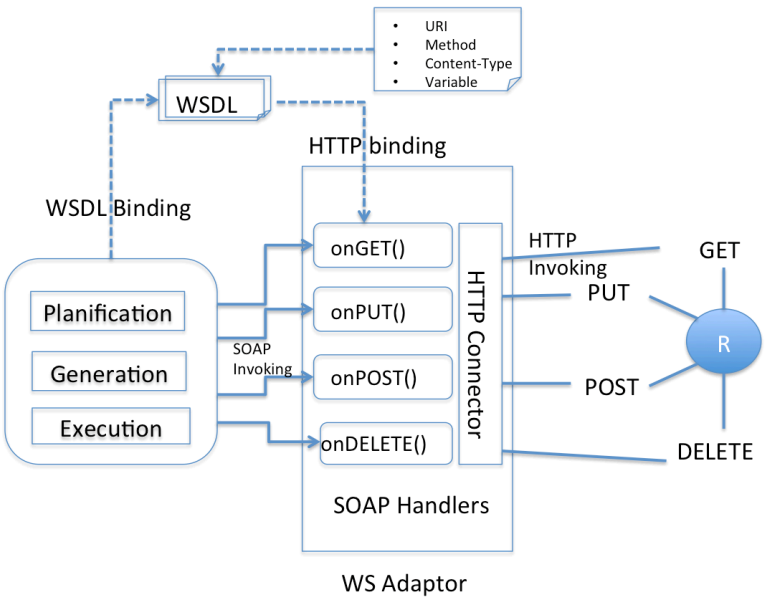


Fig. 3.10 : ressources *binding* et invocation basée sur service RESTful

3.3 Conclusion

Dans cette partie, nous avons présenté une approche de WS médiateur pour la composition dynamique de service Web en spécifiant les fonctionnelles des modules de dans. Nous arrivons à faciliter la médiation des données échangées entre les modules composés. Lors de l'organisation des interactions entre les WSs, de nombreuses hétérogénéités peuvent apparaître entre les fonctionnements de ces services, on l'a résolu par suivre des types d'interactions différents, d'échanger les données selon différentes séquences d'échanges de messages entres les modules fonctionnelles, de prendre en charge à des degrés différents et de gérer les transactions.

Chapitre 4

Deuxième proposition : le Mashup basé sur WS-médiateur

4.1 Introduction

Quand on parle de « Mashup », il peut avoir une la confusion pour distinguer le mashup et la composition de service traditionnel. Dans ma thèse, la composition de service est plutôt un assemblage de services existants du côté du médiateur, mais elle n'est pas basée sur le navigateur. En plus, les compositions qui sont généralement basées sur les WSs sont souvent tissées avec les processus métier et développées par les programmeurs professionnels. Ce genre de composition est principalement au niveau de l'« interface » au lieu d'être au niveau de l'« application ». Ce qui signifie que l'utilisateur doit avoir la pleine compréhension des interfaces de service. Ce qui est particulièrement difficile pour les utilisateurs sans connaissances professionnelles. Le mashup, d'autre part, utilise les techniques presque remarquablement simples pour connecter ensemble des services.

4.2 Les exigences

Le but de mashup basé sur la médiation, c'est pour fournir un environnement de développement de mashup à l'utilisateur afin de créer et de personnaliser les applications.

Donc tout d'abord, il faut fournir une méthode d'encapsulation/*wrap* pour les sources de données. Dans cette encapsulation, il faut réaliser la capacité de visualisation, configuration et interface uniforme pour permettre des intégrations de données diverses. Basé sur l'encapsulation des données, il faut traiter les données et les composer, c'est une démarche de mashup des données. Le mashup de source des données fournit en environnement de service mashup composé par des données de ressource différentes. Le mashup basé sur la médiation, est pour fournir des fonctionnalités de CRUD au service de mashup, dans le même temps, il fournit une interface uniforme au client et fournit une plate-forme de déploiement des services de mashup aux producteurs de service. D'autre part, en considérant l'agilité de la plate-forme, on va fournir l'interface de service RESTful pour que l'on puisse manager les services de mashup, puis les intégrer avec d'autres plates-formes (de *Cloud Computing* et *IoT*).

Pour supporter les interfaces ouvertes, on utilise REST et SOAP pour fournir les interfaces externes, donc les méthodes de GET, POST, PUT et DELETE dans REST pour réaliser le

CRUD de service mashup et nous avons défini les méthodes pour réaliser les mêmes fonctionnalités de service mashup.

Après avoir réalisé l'interface ouverte, on peut construire notre interface du web d'après les données fournies par les interfaces des services de SOAP et REST, puis fournir un environnement d'interaction à l'utilisateur à travers cette interface. L'interface construite de cette manière, sépare la couche de métier logique et la couche d'interface puisque toutes les couches sont indépendantes. Même si il y a des changements sur le logique métier, il n'a pas des influences sur la couche de l'interface, autrement dit, quand il y a des changements sur la couche d'interface, on n'a pas besoin de changer la couche du logique métier.

4.3 Le CRUD

4.3.1 Définir d'une Composition de service RESTful

En fait, une composition de service RESTful est un type spécial de l'élément intermédiaire, qui est différent de *proxies* et passerelles, elle ne fait pas simplement le *forwarding* des requêtes aux serveurs originaux, elle peut décomposer la requête afin qu'elle puisse être utilisée en invoquant le serveur d'origine.

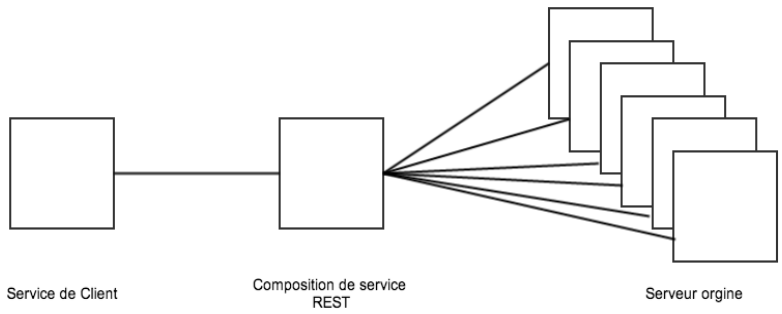


Fig. 4.1 Composition de service RESTful

Une composition de service RESTful, dans ma thèse, ne désigne pas simplement un composite de service RESTful ni une composition de service avec l'interface RESTful. L'architecture définit non seulement les interfaces externes pour les consommateurs de services et les éléments de services, mais aussi sa structure interne afin d'obtenir les propriétés souhaitées. Donc une composition de service RESTful illustrée dans la figure 3.10 ne doit pas être conçue comme une orchestration de service ni comme une chorégraphie de services. Par contre, une composition de service RESTful a une structure décentralisée

comme pour la chorégraphie mais elle fournit des interfaces centralisées à travers la médiation pour initier les instances de composition et suivre son statut comme une orchestration de service.

Compte tenu de la nature récursive de la composition, la composition de service RESTful est aussi un service RESTful. Ainsi, il publie un ensemble de ressources à ses clients, qui interagissent entre eux en suivant le principe de l'interface uniforme REST (c'est à dire, en utilisant les méthodes GET, POST, PUT, DELETE). En outre, le client va le faire, sans savoir que leurs requêtes sont desservies par une composition de service RESTful à travers la médiation de service (qui agit semblablement à tout autre élément intermédiaire).

Les clients ne peuvent pas savoir si le résultat d'une requête GET est calculé localement par la composition de service ou s'il vient du résultat de la combinaison des résultats d'un ensemble de requêtes GET envoyé aux serveurs d'origine. En plus, en tant que client l'initialisation (POST), la mise à jour (PUT), ou la suppression (DELETE), l'état de la composition de service, les requêtes peuvent entraîner des ressources en cours de création, de mise à jour ou de suppression locale, ou bien dans les actions similaires exécutés par la composition de service sur un sous-ensemble des ressources publiées par les serveurs d'origine composés.

Du point de vue de la mise en œuvre, on peut distinguer deux types de composition REST, le *stateless* et le *stateful*. Les compositions *Stateful* augmentent l'état des serveurs d'origines composés avec des informations qui sont gérées et stockées par la médiation. Ainsi, une partie de l'état des ressources composites est géré localement et le reste est partagé entre les serveurs d'origines. Par contre, les compositions *Stateless* ne permettent pas de maintenir un état local et le *mapping* de toutes les requêtes des clients à l'état des serveurs d'origines.

4.3.2 Client de composition de service RESTful

La composition RESTful fournit les services à ses consommateurs dans une manière RESTful. Cela nécessite que le client de la composition respecte les contraintes (cache, *stateless* et une interface uniforme) de REST. La contrainte de cache nécessite que le client comprenne le contrôle des métadonnées du cache dans les messages et également de mettre en œuvre le cache du coté client si possible. Pour les réponses de service de grandes charges utiles, le cache peut améliorer considérablement la performance perçue par l'utilisateur en réduisant le temps de réponse. La contrainte de *stateless* nécessite que le client mette suffisamment d'informations explicites dans chaque requête pour que la composition comprenne les requêtes sans maintien de l'état de la session des clients. La contrainte

d'interface uniforme nécessite que le client fasse usage de l'identification uniforme et des méthodes fournies par les services. En outre, il faut être en mesure de prendre part à des transitions d'état conduites par l'hypermédia.

4.3.3 Médiation RESTful

Un élément de service peut être soit RESTful soit non-RESTful, la médiation travaille comme une proxy RESTful, elle connecte le client RESTful aux éléments de service. La médiation communique au service non-RESTful avec SOAP et communique avec le client RESTful à travers son interface uniforme. A l'aide de la médiation, une composition RESTful peut traiter les services non-RESTful dans la manière RESTful au moins au niveau de l'interface.

Lorsque l'on ne peut pas obtenir des informations pour construire une interface RESTful de la description du service non-REST, par exemple, nous ne pouvons pas obtenir la mise en cache et l'idempotence d'une opération en analysant le WSDL du service SOAP. Par conséquent, les détails de mise en œuvre sont nécessaires afin de traiter la composition de service SOAP-SOAP et SOAP-REST qui sont les services non-RESTful. Si elle veut faire communiquer le client RESTful au service REST, tout simplement, elle doit impliquer la CRUD (*Create, Read/Retrieve, Update/Modify, Delete/Destroy*) pour échanger les informations dans la base de données, et les opérations de CRUD dans une relation de *mapping* séparée avec POST, GET, PUT, DELETE du protocole HTTP.

4.3.4 La ressource composite

Une ressource est une information qui peut être nommée. Une ressource composite est une ressource composée d'un ensemble d'autres ressources. Une opération d'une ressource composite est souvent mise en correspondance avec les opérations de ses membres correspondants. La représentation d'une ressource composite est normalement générée par la récupération, le traitement et la combinaison des représentations de ses membres de ressource. Un modèle de la médiation pour la composition de service RESTful est une opération qui est d'abord associée à un ensemble d'éléments de service pour donner le résultat. Ce modèle peut être mis en œuvre en tant que ressource composite.

4.3.5 Lien de dépendance fonctionnelle

L'abstraction la plus importante dans la composition de service RESTful est la dépendance fonctionnelle. C'est à dire, que l'on doit traiter les séquences des services dans la composition

d'après les relations de fonctionnalité pour réaliser une certaine dépendance entre les services, et automatiser le processus d'après les dépendances. La dépendance fonctionnelle peut être vue comme un domaine spécifique et peut être traitée par langage de DSL mentionné dans l'état de l'art. On peut réaliser ce DSL en méta-programmation comme les codes montrés dans l'Annexe C.

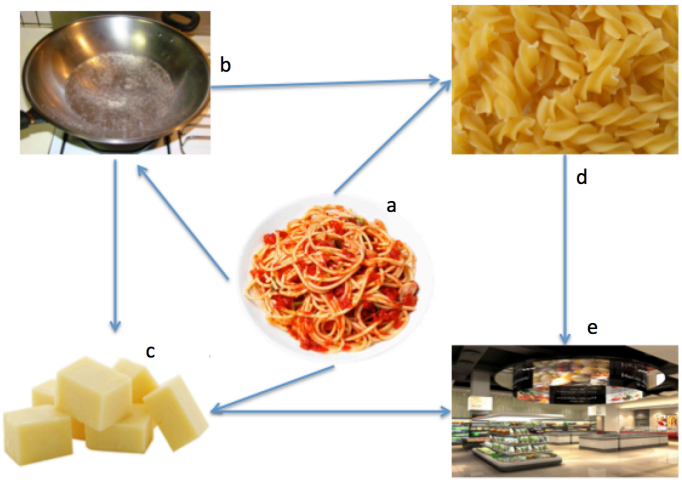


Fig. 4.2 Exemple de lien de dépendance implicite

Par exemple dans le cas d'un utilisateur illustré dans la figure 3.11, qui veut cuisiner les spaghettis avec du fromage, il y a certaines démarches à faire avant de commencer. En général, il faut impliquer les tâches (a)-« *make_mac_and_cheese* », (b)-« *boil_water* », (c)-« *buy_cheese* », (d)-« *buy_pasta* » et (e)-« *goto_store* ». Les tâches peuvent être des ressources comme les services ou bien les URIs. En particulier, le DSL :

- permet une représentation de l'interface commune entre les services de type différent.
- donne un modèle uniforme pour les données de service et les interactions entre les opérations de service.
- facilite le mashup de niveau haut.

Voici un exemple de la réalisation du lien de dépendance de service Cette partie est testée par méta programmation en utilisant le DSL illustré suivant dans les figures 4.3.

tasks.rb

```ruby
task "make_mac_and_cheese",
  [ "boil_water", "buy_cheese", "buy_pasta" ] do
  puts "Make Mac & Cheese"
end

task "boil_water",
  [ "buy_pasta", "buy_cheese" ] do
  puts "Boil Water"
end

task "buy_pasta",
  ["goto_store"] do
  puts "Buy Pasta"
end

task "buy_cheese",
  ["goto_store"] do
  puts "Buy Cheese"
end

task "goto_store" do
  puts "Goto Store"
end
```

Puis on peut l'interpréter par un DSL, ici tout est réalisé par langage Ruby.

micro-rake.rb

```ruby
class Task
  def initialize(name, deps, action)
    @name = name
    @deps = deps
    @action = action
  end

  def invoke
    return if @already_run
    @deps.each do |dep| TASKS[dep].invoke end
    execute
    @already_run = true
  end

  def execute
    @action.call
  end
end

TASKS ={}

def task(name, deps=[], &block)
  TASKS[name]= Task.new(name, deps, block)
end

require './tasks'

ARGV.each do |arg| TASKS[arg].invoke end
```

Donc le résultat

```
~/desktop/Ruby/hungry » ruby micro-rake.rb make_mac_and_cheese
Goto Store
Buy Pasta
Buy Cheese
Boil Water
Make Mac & Cheese
```

Fig. 4.3 : la réalisation de DSL en Ruby

4.4 La conception de Mashup Basé sur la médiation

Le mashup basé sur la médiation implique une architecture à trois « couches d'application » par rapport aux applications de B/S (*Browser/Server*) et C/S (*Client/Server*), nous avons la couche de source de données, la couche de service et la couche d'accès illustrées dans la figure 4.4.

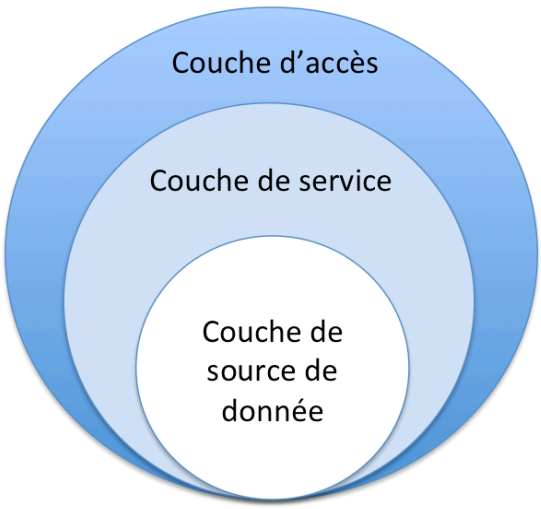


Fig.4.4: Les trois « couches d'application »

La couche de source de données s'occupe de l'encapsulation des données originales, elle encapsule les données comme le module configurable et exécutable. La source de données contient les sources diverses de la base de données, les sources de données locales, les sources de service web etc. La manière d'encapsulation/*wrap* peut résoudre les hétérogénéités de ressources puisque les sources différentes peuvent être exécutables uniformément, cela facilite le mashup des données diverses.

La couche de service s'occupe de mashup des modules de données encapsulés et fournis par la couche de source de données, puis génère le script de service mashup. Cette couche fournit essentiellement les fonctionnalités du CRUD pour le service mashup, c'est aussi elle qui supporte la couche d'accès.

La couche d'accès fournit essentiellement une interface uniforme exposée du service mashup, donc l'utilisateur peut invoquer, mettre à jour ou supprimer le service mashup déjà publié d'après son droit d'accès, en plus, il peut publier le nouveau service mashup. Tous les déploiements des interfaces exposées de service mashup sont fournis par SOAP et REST.

Donc dans le mashup basé sur notre médiation, il s'agit de deux niveaux en granularité:

- Niveau de médiateur (y compris couche de source de données et couche de service): c'est plutôt un mashup de niveau d'entreprise, c'est à dire le médiateur nous permet de créer et de déployer un environnement de mashup. Le producteur de service peut faire le mashup des services à travers le médiateur de la couche du logique métier avec les connaissances pour décrire les détails de représentation des tâches et la planification du processus afin de fournir un service mashup pour les utilisateurs.
- Niveau d'utilisateur (y compris couche de service et couche d'accès) : l'utilisateur peut faire le mashup des services à travers le navigateur web au niveau de "l'application" donc de la couche d'interface. Cette dernière peut cacher la complexité des compositions de service en fournissant un script (réaliser par Ruby, Php, Perl etc.) aux utilisateurs puisqu'elle peut tout simplement profiter des services Mashup (par exemple, les géo APIs de Google et Yahoo).

4.5 M-Mashup framework

Le M-Mashup dans ma thèse est une composition de services (REST, APP, SOAP, RSS, ou Atom) qui contient les informations sur comment représenter l'API de service et les données ainsi que comment composer les services et présenter une interface utilisateur (UI). Le M-mashup répond aux principes suivants :

i. Les APIs de service et les données concernées sont définies. Il permet un modèle uniforme ainsi que la représentation des services de type varié.
ii. La médiation de données, représente les manipulations des données comme combinaison, transformation et conversion.
iii. La médiation de protocole de service ou *wiring*, spécifie l'orchestration/chorégraphie de services.

Dans M-Mashup, on représente directement dans la syntaxique, les conceptions nécessaires pour couvrir les composants principaux de médiation pour notre modèle architectural de Mashup : 1) données et médiation; 2) APIs de service, ses protocoles, et chorégraphie ; et 3) un moyen pour créer des UIs personnalisées pour les mashups résultant. Les vues sont ajoutées en générant les templates de script par exemple RHTML de Ruby on rails (RoR). Donc notre M-Mashup contient :

- Les données : qui décrivent un élément de données utilisé dans un service. Un élément de

données correspond à un type complet de schéma de XML.

- Une API : qui nous donne la description complète de l'interface de service y compris les noms des opérations, les paramètres et les types de données.
- La médiation : qui décrit les transformations de un ou de multiples éléments de données pour créer un nouveau.
- Le service : qui relie une API de service avec un service concret pour indiquer le type de service.
- La recette : qui constitue une collection de services et mashups. Une recette contient les vues de chaque lien de dépendance (*wiring*). Certaines vues sont générées automatiquement et les autres sont personnalisées par les utilisateurs.

Donc dans M-Mashup :

Le « Mashup » est une composition de un ou de multiples services au niveau du navigateur. Il comprend une collection de médiation et des liens de dépendances.

La « Médiate » invoque une médiation de déclaration avec les instances des éléments de données pour le manipuler.

Le « Lien de dépendance ou *wiring* » comprend deux niveaux de granularités de connexion de services qui font partie d'un mashup :

1) Un protocole de haut-niveau de structure d'un mashup qui représente un ou multiples opérations de lien de dépendance et les étapes des invocations.
2) Une opération qui est le lien de dépendance des opérations de un ou plusieurs services. Ce genre d'opération inclut la possibilité d'invoquer les activités de services dans un mode asynchrone en définissant automatiquement des rappels (*callbacks*).

L'« étape de l'invocation » constitue une étape atomique dans une médiation de protocole. Une étape peut être invoquée plusieurs fois dans le cadre d'un protocole de lien de dépendance. Une étape est appelée par le nom de l'étape comme une invocation de méthode.

Nous proposons donc le *framework* de M-Mashup illustré dans la figure 4.5.

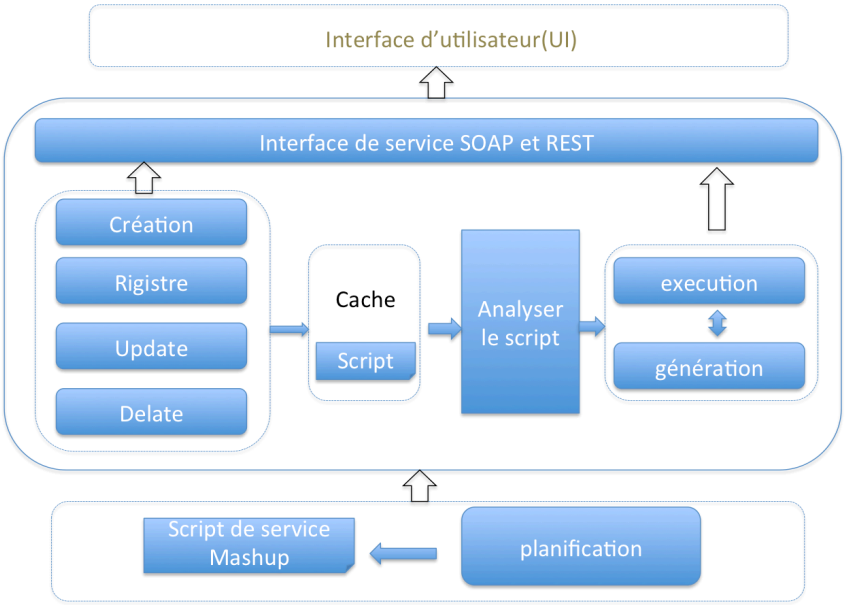


Fig. 4.5 : Framework de M-Mashup

Le ficher de script de service Mashup peut être généré par le module de planification. Ce module intègre la couche de source de données et la couche de service et fournit un environnement de génération de service mashup. La source de données contient des encapsulations des bases de données, ficher local, service RESTful et service SOAP etc., puis à travers la planification configure et génère les modules utilisés dans le service Mashup, réalise les intégrations des données à travers des filtrages, conversions et des combinaisons afin de générer un fichier de script de service Mashup. Dans ce fichier de script, il y a des informations de configuration sur la génération et l'intégration des sources de données.

Dans la couche de service, on a le module de CRUD (*Create, Registre, Update, Delete*) de service. Le CRUD est nécessaire pour interagir avec le cache et autre module de stockage. Le module de stockage contient des fichiers de script ainsi que les modèles de services qui sont les informations basiques de service Mashup stockées dans la BDC. Après avoir analysé le ficher de script, on peut l'exécuter puis obtenir le résultat d'exécution. Ici, il y a deux types d'interface ouvertes fournis, donc RESTful et SOAP qui augmentent la compatibilité du système en facilitant les délivrances de service (*service delivery*) du client REST et du client SOAP.

La couche d'interface fournit essentiellement une interface d'interaction avec l'utilisateur. L'utilisateur peut manipuler le service Mashup directement dans le navigateur. L'utilisateur fait les opérations de CRUD dans cette UI à travers l'interface de REST uniforme fourni par la couche de service pour l'interaction du côté UI et du côté du service mashup, puisque on

peut réaliser la séparation de niveau UI et de niveau logique métier. L'avantage de cette séparation facilite la mobilité de l'utilisateur du côté client qui avec l'ordinateur, la tablette et le Smartphone peut utiliser JSP, HTML(DHTML), Flex etc., donc des technologies de réalisation basées sur l'infrastructure de B/S. Ou bien avec Java et C basé sur l'infrastructure de C/S, on peut aussi réaliser les processus métier du côté serveur en impliquant multiple plates-formes de programmation. Globalement on peut construire un modèle de métier basé sur la médiation avec multi-terminal, multi-technologie et plate-forme indépendante et réutilisable.

4.6 La faisabilité de M-Mashup

4.6.1 Analyser le ficher de script de service

C'est l'analyse du fichier de script avant que l'on fasse le déploiement et l'exécution du service. On le fait après la réussite de la génération du fichier de script et on enregistre les informations de configuration concernées dans la structure de données correspondante. Ce module est essentiel pour vérifier le fichier pendant le CRUD de service ainsi que la génération de nœud de données. Ici on a illustré le fonctionnement de ce module dans un schéma de processus dans la figure 4.6.

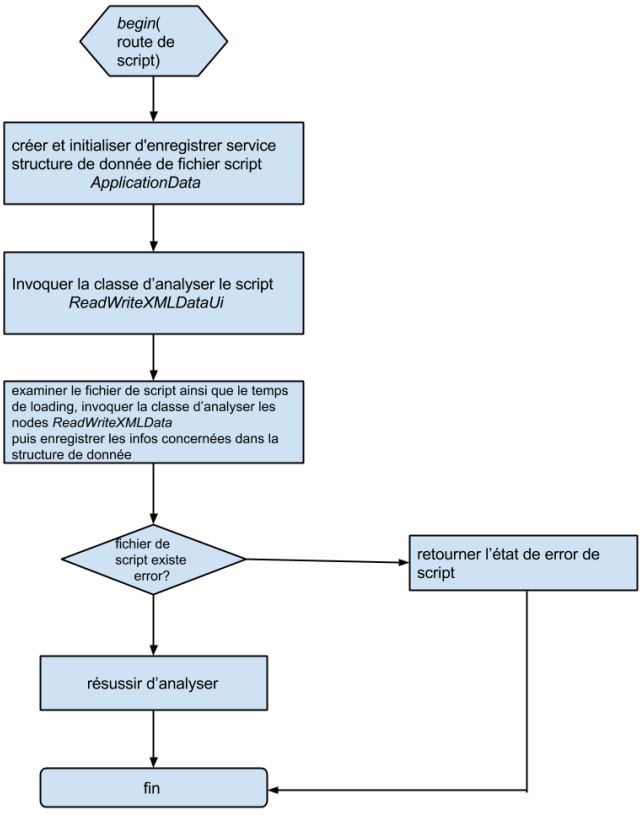


Fig. 4.6: Processus d'analyse du script de service

Les démarches concrètes donc sont de :

1. Commencer en prenant les paramètres d'input, ici c'est le nom de point d'accès (*endpoint*) complet du ficher de script.
2. Créer par exemple la structure de données *ApplicationData* pour enregistrer le fichier de script, *ApplicationData* qui est essentiellement utilisé pour enregistrer les informations d'état (*endpoint* de fichier de script de service, les nombres d'erreurs etc.) pendant le processus d'analyse du fichier. Puis initialiser ces informations et enregistrer les informations de configuration des nœuds (service ou source de donnée) dans la structure de données du projet, dans laquelle on trouve les informations de séquence de connexion entre eux, ou bien le lien de dépendance de DSL.
3. Invoquer la classe *ReadWriteXMLDataUi,* qui est utilisée pour les traitements correspondant avant d'analyser les nœuds (la démarche 4).
4. Vérifier la version du script de service, charger les informations concernées (les temps etc.). Démarrer la classe *ReadWriteXMLData* pour analyser les nœuds et commencer d'analyser le fichier de script.
5. *ReadWriteXMLData* qui analyse le script essentiellement selon les nœuds, les *arrows* et les étiquettes de fichier XML. Les nœuds enregistrent les types de source de données et les éléments de données ainsi que les informations de l'étiquette de projet. Arrow record sont les relations de séquence de connexion entre les nœuds. La classe *ProjetDatas* enregistre les informations de configuration de source de données différentes. Cette classe va enregistrer le résultat de l'analyse dans le projet, puis elle enregistre les informations d'état dans *ApplicationData.*
6. Examiner les informations d'état, si il y a des erreurs, il faut retourner l'état de ce fichier de script, sinon l'analyse est réussie.
7. Fin.

4.6.2 Le module d'exécution

Le module d'exécution s'occupe de l'exécution de service après avoir analysé le ficher, il faut sauvegarder le résultat du service mashup après l'exécution. Ce module est essentiellement pour fournir un environnement d'exécution pour les services REST. Ici on a illustré le processus dans la figure 4.7.

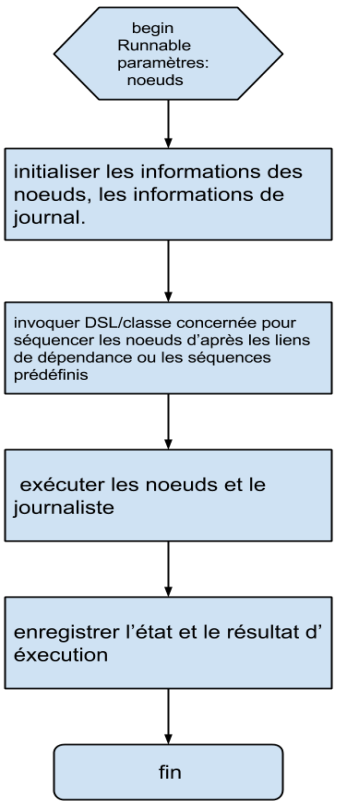


Fig. 4.7: Processus d'exécution

Les démarches concrètes sont donc :

1. Démarrer la classe *Runnable*, qui s'occupe de l'exécution de ce module. Les paramètres sont les nœuds de projet après avoir réussi dans le module d'analyse.
2. Initialiser les nœuds dans la collection <*AbstractNode*>, initialiser les informations de journalisation (*log*).
3. Invoquer le DSL ou bien la classe concernée pour séquencer les nœuds d'après le lien de dépendance ou bien les séquences prédéfinies dans la classe concernée.
4. Exécuter les nœuds concernés et les journaliser.
5. Enregistrer le résultat et l'état après la fin de l'exécution : après l'exécution tous les nœuds, les résultats et l'état d'exécution sont sauvegardés dans la structure de données concernée pour préparer l'obtention du résultat quand le module d'exécution invoquera les services.
6. Fin.

4.6.3 Le module de stockage

Le Module de stockage contient deux parties : le stockage de ficher de script et les services de mashup dans la base de connaissance. On peut réaliser le stockage de script tout simplement par création du chemin de l'enregistrement du *endpoint* du ficher concerné dans

le serveur. Le BDC enregistre essentiellement les autres informations de service mashup, y compris le nom de service, login de service enregistré, type de service, description de fonctionnalité de service, le temps d'inscription de service, le temps de modification (*last motified*) et le nombre de visite de service.

Il y a deux opérations de BDC concernées : *Service* et *DataBaseTool*.

--*Service* est utilisé pour enregistrer le service mashup. *DataBaseTool* est utilisé pour la manipulation de BDC. Donc concrètement les attributs compris dans la classe *service* : le nom de l'utilisateur (*username*), le nom de service (*mashupname*), le type de service (*type*), description de fonctionnalité de service(*description*), le nombre d'invocation de service(*count*). Pour chaque attribut, on fournit la méthode *set* et *get* pour configurer et obtenir la valeur de l'attribut correspondant.

--*DataBaseTool* est utilisé pour réaliser les manipulations sur la table de BDC, en plus, il fournit une interface d'invocation pour l'application qui utilise les données concernées. Ici on a défini quatre interfaces de l'opération :

--*String addService (String mashupname, String username, String type, String Descrip)* ; C'est la manipulation d'ajouter le service, la fonctionnalité principale est d'ajouter les informations au BDC d'après les paramètres de service. Les paramètres sont nom de service, nom de l'utilisateur, type de service et la description de service. La valeur retournée signifie si l'ajout est réussi.

--*String deleteService (String mashupname)* : C'est la manipulation de supprimer le service, la fonctionnalité est essentiellement la suppression du service dans le BDC. Le paramètre est le nom de service mashup, la valeur retournée signifie si la suppression est réussie.

--*String updateService(String mashupname, String username, String type, String descrip)*, C'est la manipulation de la mise à jour, c'est pour mettre à jour les informations de service dans la BDC. Les paramètres sont nom de service, nom de l'utilisateur, type de service et la description de service. La valeur retournée signifie si la mise à jour est réussie.

--List<service> *queryService* (*String type, String username, String mashupname*); C'est pour renseigner le service, la fonctionnalité est essentiellement pour les informations de service dans le BDC, les paramètres sont le type de service, nom de l'utilisateur et nom de service. La valeur retournée est la liste des services trouvés.

Quand on implémente la BDC, par exemple une base de données SQL, on doit précompiler les phrases de SQL, puis on peut tout simplement transférer les paramètres des phrases de

SQL pour faire l'exécution. Donc quand on fait l'exécution, on a besoins d'exécuter seulement la phrase de SQL.

4.6.4 Les éléments nécessaires à implémentation du M-mashup

4.6.4.1 Les ressources de service mashup

Puisque le service RESTful est une architecture orientée ressource, tout d'abord on doit diviser les données dans le M-mashup en ressource, puis le publier a travers de l'HTTP. D'après le style de l'architecture REST mentionné dans l'état de l'art, on peut considérer chaque donnée citée comme une ressource. Le M-mashup est essentiellement pour manager le service mashup. Le service mashup représente le service selon le script de service généré par le module de planification. Donc ici la ressource que l'on doit considérer, c'est le script de service mashup. Et pour concevoir le service SOAP, on fait correspondre à chaque opération une méthode de service web après la conception du service RESTful.

Les opérations correspondant à la ressource de service mashup sont :

--l'enregistrement de service : le concepteur peut fournir les informations de service, ajouter le script de service, enregistrer les ressources de service dans la base de données.

--le renseignement de service : renseigner les détails de service enregistré, y compris la liste de service, le nom de service, le temps de télécharger etc.

--l'exécution de service : après l'analyse du script de service, on fait l'exécution du service puis retourner le résultat à l'utilisateur.

--la mise à jour de service : modifier les informations de service qui est déjà enregistré (sauf le nom de service).

--supprimer le service : enlever le service déjà enregistré.

4.6.4.2 Le URI de service mashup

L'une des caractéristiques de REST est qu'il est adressable, l'adresse est représentée par l'URI. On peut donc représenter l'URI dans la manière de voies hiérarchiques, par exemple, la ressource de service mashup est représentée par l'URI *http://localhost:8080/mashup*, et pour certain service de mashup, on peut utiliser *http://localhost:8080/mashup/{servicename}* pour le représenter. En plus, on peut définir les opérations de l'enregistrement, l'exécution, la mise a jour et la suppression etc.

On peut utiliser *http://localhost:8080/mashup/services?mashupname=?&user=?&type=?* La manière du *endpoint* plus variable de renseignement pour exposer l'interface de

renseignement de service, et cette dernière est conforme aux habitudes de l'utilisateur. Pour le service SOAP, on peut utiliser par exemple l'URI comme *http://localhost:8080/soap/* pour exposer le service SOAP que l'on va déployer. Dans le même temps, il faut spécifier la requête correspondante et il faut encapsuler toutes les requêtes de cette méthode lorsque l'on veut visiter le site.

4.6.4.3 Interface Uniforme de l'HTTP

L'autre caractéristique du service RESTful est l'interface uniforme. Pour chaque opération de ressource, on a une interface correspondante, donc l'interface uniforme exposée est les méthodes de GET, PUT, POST, DELETE pour les ressources modifiables. Ici la figure 4.8 illustre l'interaction de service avec le médiateur.

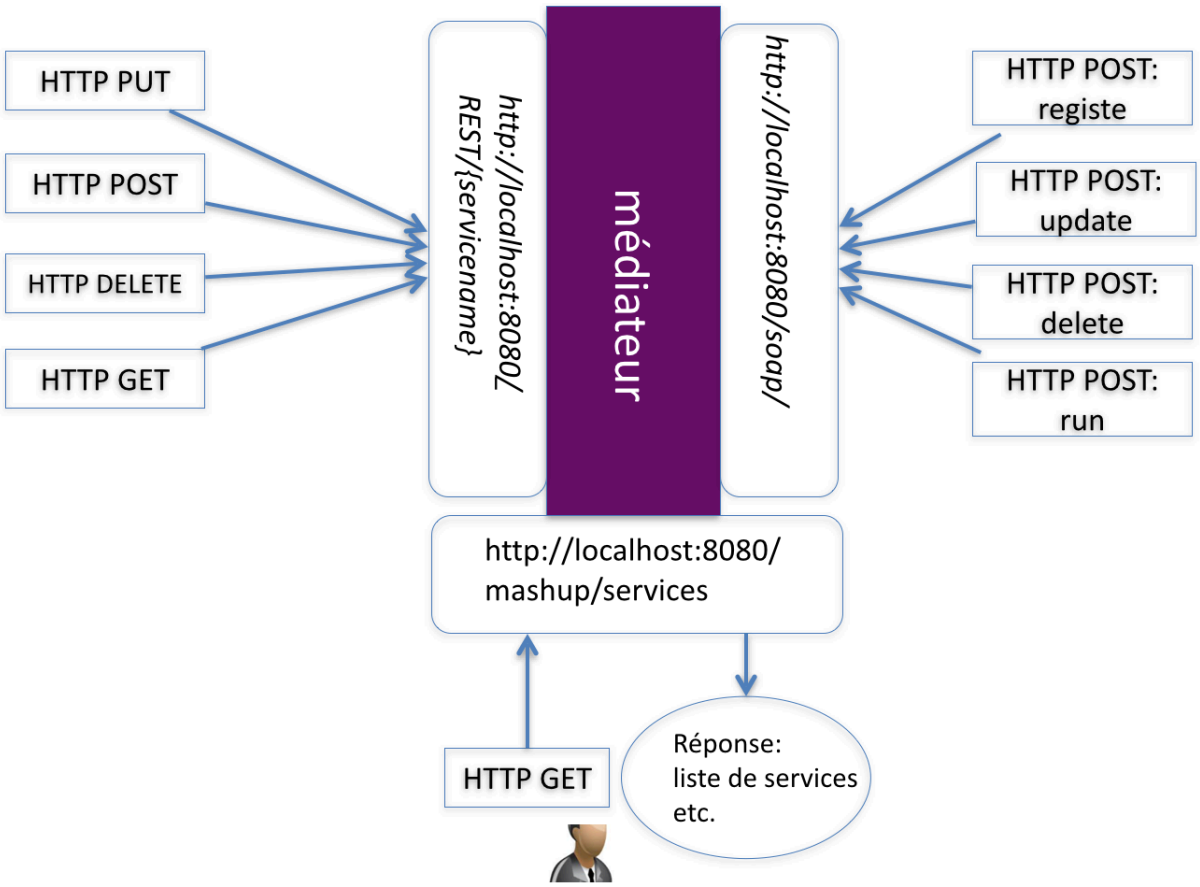


Fig. 4.8 : Mashup de services REST et SOAP basé sur le médiateur

Dans la partie gauche de la figure, ce sont les méthodes de visite de requête de l'HTTP, au milieu, c'est le médiateur ainsi que les URI exposés vers les ressources, la partie droite signifie que toutes les requêtes sont de type POST de l'HTTP quand on visite le M-médiateur avec les services SOAP, en plus les méthodes, les paramètres et les valeurs retournées sont encapsulés dans les messages SOAP.

4.6.5 La définition et la réalisation de l'interface de service RESTful

4.6.5.1 le registre

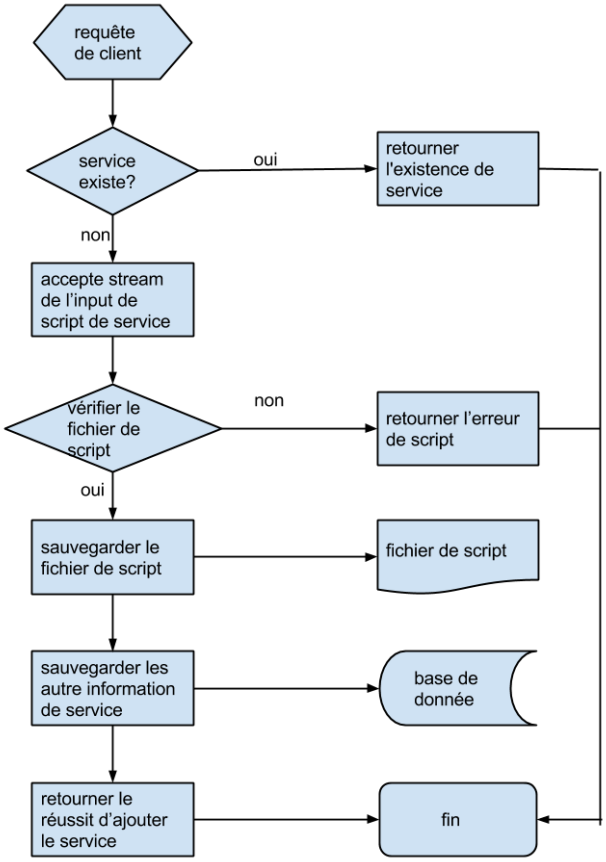


Fig. 4.9 : Processus de registre de service RESTful

Les démarches du registre illustré dans la figure 4.9 sont :

1. Le coté du client visite le *http://localhost:8080/mashup/{servicename}* dans la manière de l'HTTP POST;

2. D'après le paramètre du point d'accès *servicename,* il faut vérifier s'il y a un nom de service de *servicename* enregistré dans la base de données. Si il existe, on retourne l'état de l'existence de service, sinon passer à l'étape suivante.

3. Accepter le flux de l'input de fichier du script, vérifier ce fichier, si il y a des erreurs, retourner l'état de faute sinon passer à l'étape suivante.

4. Sauvegarder le fichier selon le nom de service, puis passer à l'étape suivante.

5. Sauvegarder les autres informations de service, y compris le type de service, la description des fonctionnalités de service etc. Retourner l'état de réussite.

6. Fin.

4.6.5.2 l'exécution

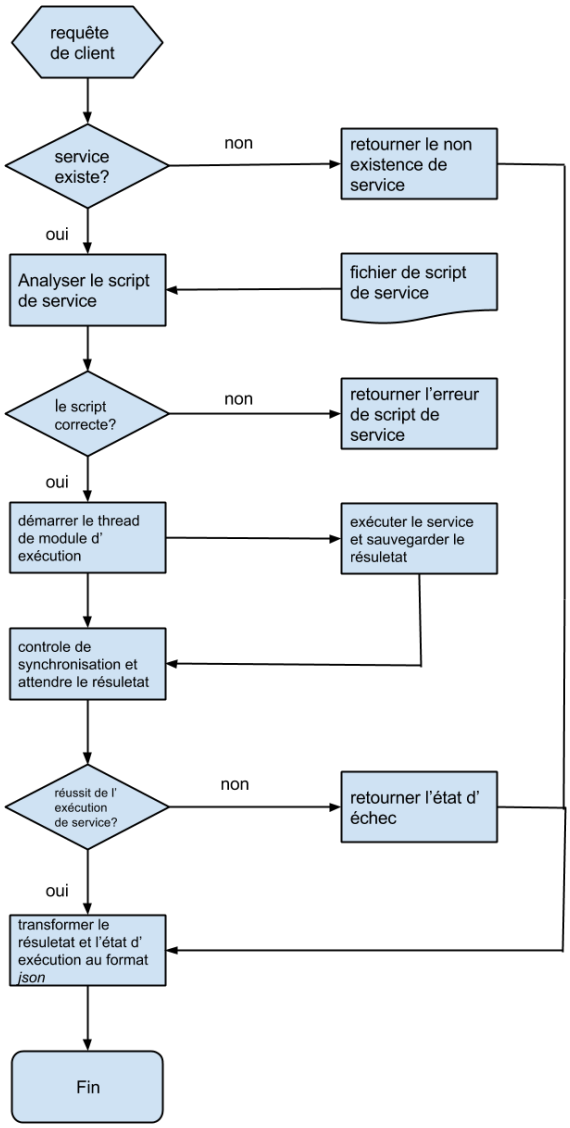


Fig. 4.10 Processus de l'exécution de service RESTful

Les démarches de l'exécution illustrée dans la figure 4.10 sont :

1. Le coté du client visite le site *http://localhost:8080/mashup/{servicename}* dans la manière de l'HTTP GET, ensuite passer à la démarche 2.

2. L'exécution de service peut obtenir le paramètre du point d'accès *servicename*, vérifier l'existence de service selon le nom de service. Si il n'existe pas, passer à la démarche 6, sinon passer à 3.

3. Analyser le fichier de script, précisément analyser les sources de données et les liens de dépendance et de connexion, sauvegarder le résultat de l'analyse dans la structure de données correspondante, si il y a des erreurs pendant l'analyse, il faut retourner l'état d'erreur et passer à l'étape 6, sinon passer à 4.

4. D'après les sources de données, après l'analyse, démarrer le thread du module d'exécution, le thread principal passe à la démarche 5 dans le même temps ; le thread d'exécution de service s'occupe des traitements des informations de l'environnement d'exécution, et sauvegarde le résultat et l'état de l'exécution. Ensuite passer à l'étape 5.

5. Le thread principal attend pour la synchronisation, après avoir retourné l'état et le résultat de l'exécution, vérifier si l'exécution est réussie. Si l'exécution a échoué, il faut retourner l'état de l'échec puis passer à l'étape 6, sinon passer directement à 6.

6. Transformer le résultat au format de JSON puis retourner le résultat. Passer à l'étape 7.

7. Fin.

4.6.5.3 La mise à jour

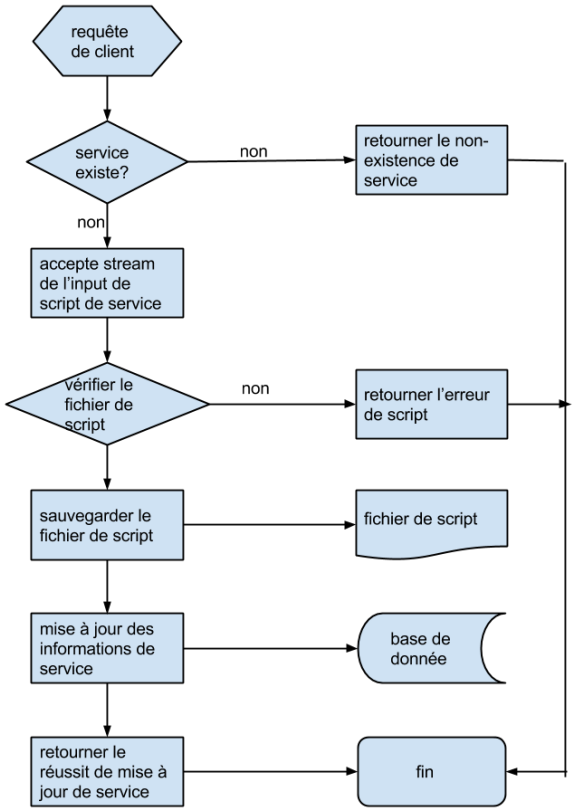


Fig. 4.11 : Processus de la mise à jour

Les démarches illustrées dans la figure 4.11 sont :

1. Le coté du client visite le site *http://localhost:8080/mashup/{servicename}* dans la manière de l'HTTP PUT, ensuite passer à la démarche 2.

2. Obtenir le nom de service *servicename,* vérifier l'existence de ce service selon le nom de service, si il n'existe pas, retourner l'état de non-existence. Sinon passer à la démarche 3.

3. Le médiateur accepte les flux des entrées de script de service, puis vérifier si c'est correct. S'il y a des erreurs, il faut retourner l'erreur de script, sinon passer à la démarche 4.

4. Sauvegarder le fichier de script dans le médiateur puis passer à l'étape 5.

5. Mette à jour les autres informations de service, retourné « la réussite » de la mise à jour puis passer à la démarche 6.

6. Fin.

4.6.5.4 La suppression

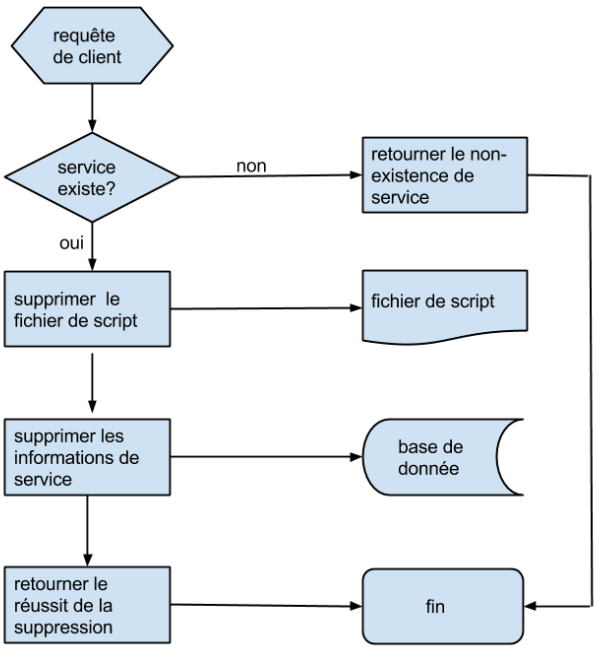


Fig.4.12 : Processus de suppression de service

Les démarches illustrées dans la figure 4.12 sont :

1. Le coté du client visite le site *http://localhost:8080/mashup/{servicename}* dans la manière de l'HTTP DELETE, ensuite passer à la démarche 2.

2. Le médiateur va obtenir le paramètre du point d'accès de *servicename,* vérifier l'existence de ce service selon le nom de service, si il n'existe pas, retourner l'état de non-existence. Sinon passer à la démarche 3.

3. Supprimer le fichier de script stocké dans le médiateur. Puis passer à la démarche 4.

4. Supprimer les autres informations dans la base de données, retourner l'état de réussite, puis passer à la démarche 5.

5. Fin.

4.6.5.5 Le renseignement

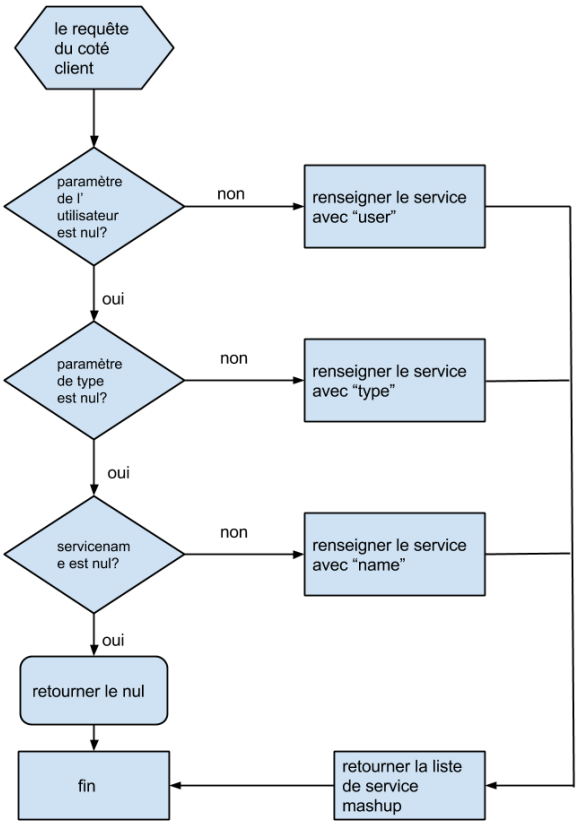


Fig. 4.13 : Processus de renseignement de service

Les démarches illustrées dans la figure 4.13 sont :

1. Le coté du client visite le site *http://localhost:8080/mashup/services?servicename=param1&users=param2&type=param3* les trois paramètres de renseignement peuvent être nuls, puis passer à la démarche 2.

2. Vérifier si le paramètre de renseignement de l'utilisateur *user* est nul ou pas, si c'est nul, renseigner son service *servicename,* si *servicename* est nul il faut renseigner tous les services de l'utilisateur *user*, puis passer à l'étape 5, sinon passer à 3.

3. Vérifier si le paramètre de renseignement de type *type* est nul ou pas, si c'est nul, renseigner son service *servicename,* si *servicename* est nul il faut renseigner tous les services de type *type*, puis passer à l'étape 5, sinon passer à 3.

4. Vérifier si le paramètre de renseignement de nom de service *servicename* est nul ou pas, si oui, il faut retourner nul puis passer à 6. Sinon il faut renseigne le service au nom de service *servicename*, puis passer à 5.

5. Retourner la liste de service mashup, puis passer à l'étape 6.

6. Fin.

Ce sont les processus de la réalisation de service RESTful, presque identiques aux fonctionnalités du service RESTful et au service de SOAP. La seule différence c'est la configuration de format de la valeur retournée, donc il faut remplacer le format de JSON par le format de XML quand il faut configurer la valeur de retour, ici on ne fait pas le processus de réalisation.

4.7 Conclusion

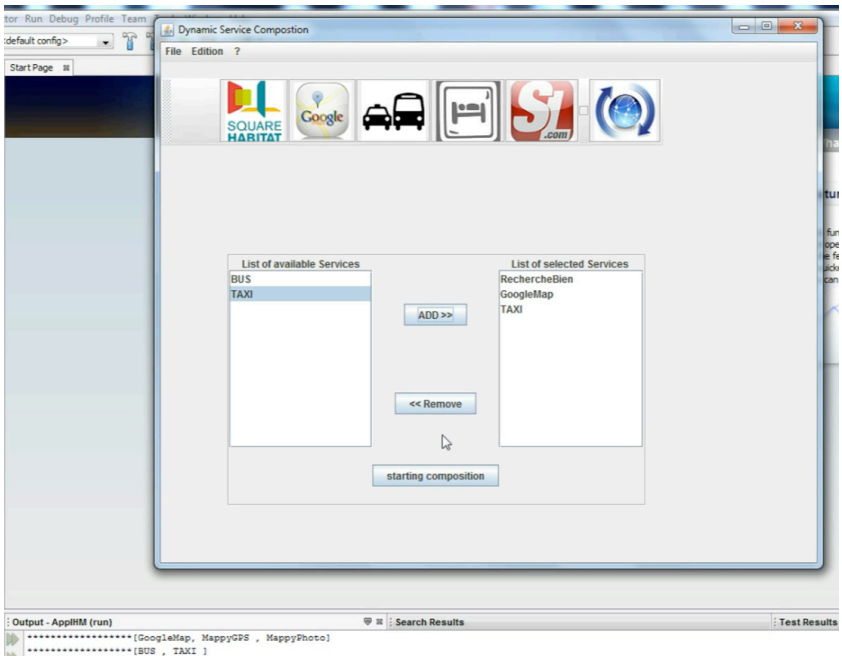
Il y a deux sens à l'aspect fonctionnel de notre médiation : le premier sens, c'est que la médiation va retourner un service composé fourni comme une orchestration de service, c'est déjà traité dans la partie précédente. Le deuxième sens qui est discuté dans cette chapitre, c'est que l'utilisateur peut composer les services via notre médiation, puis la médiation va retourner une composition de services abstraits personnalisés pour une application mashup de l'utilisateur. Nous pouvons ainsi concevoir un *framework* de mashup basé sur notre médiation et les processus de réalisation.

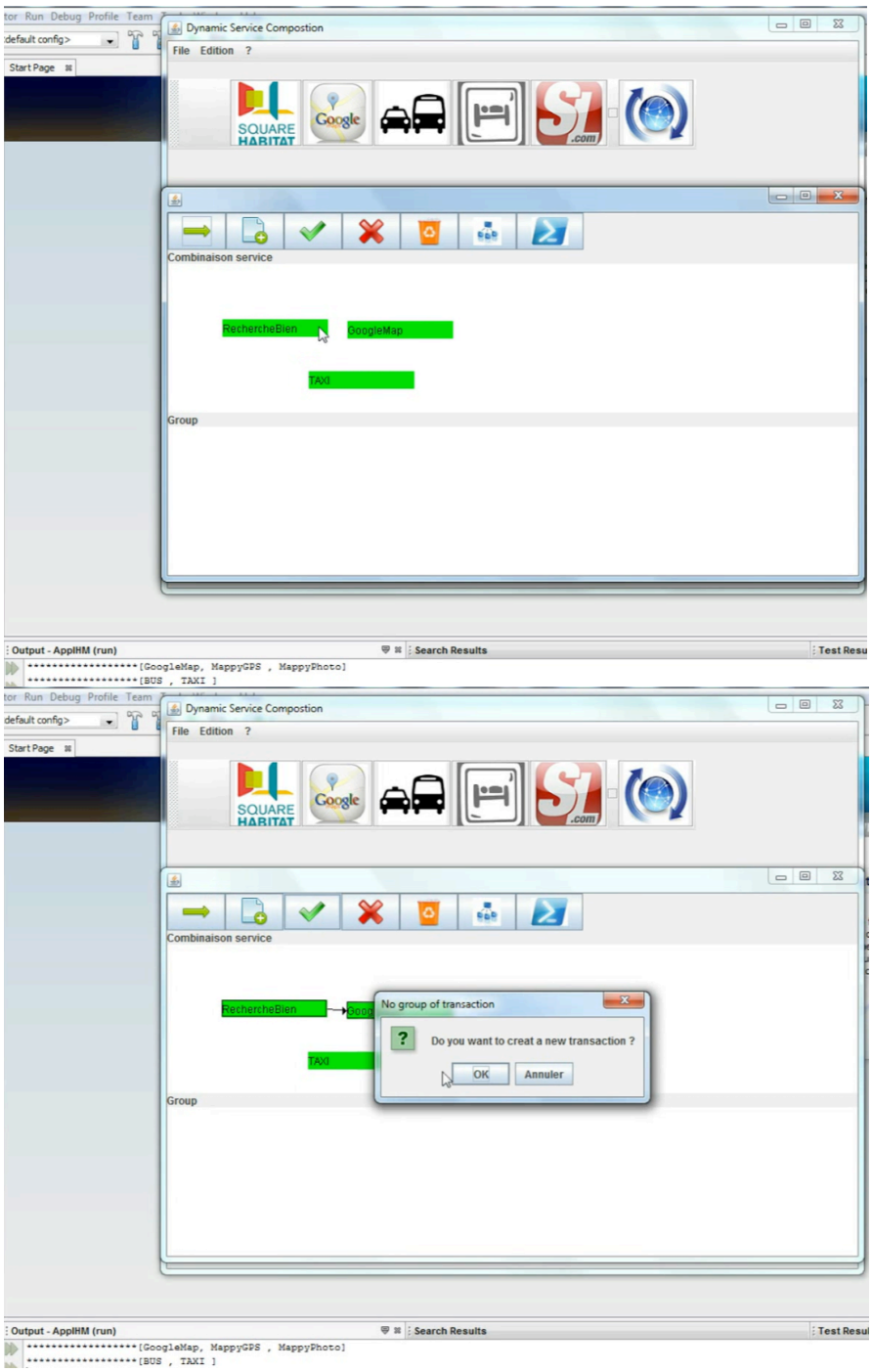
Chapitre 5

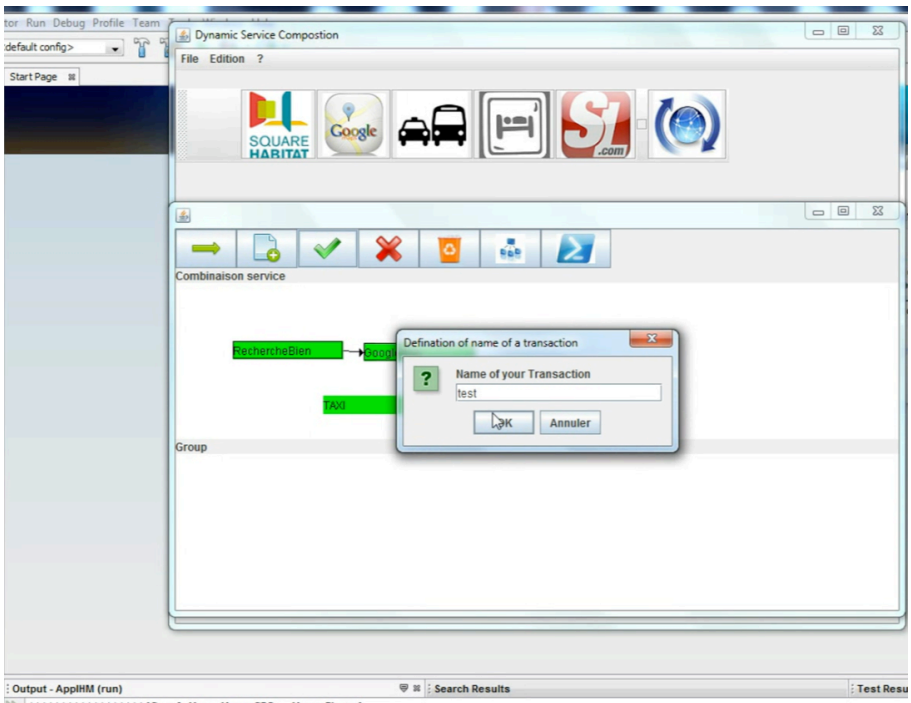
Expérimentation

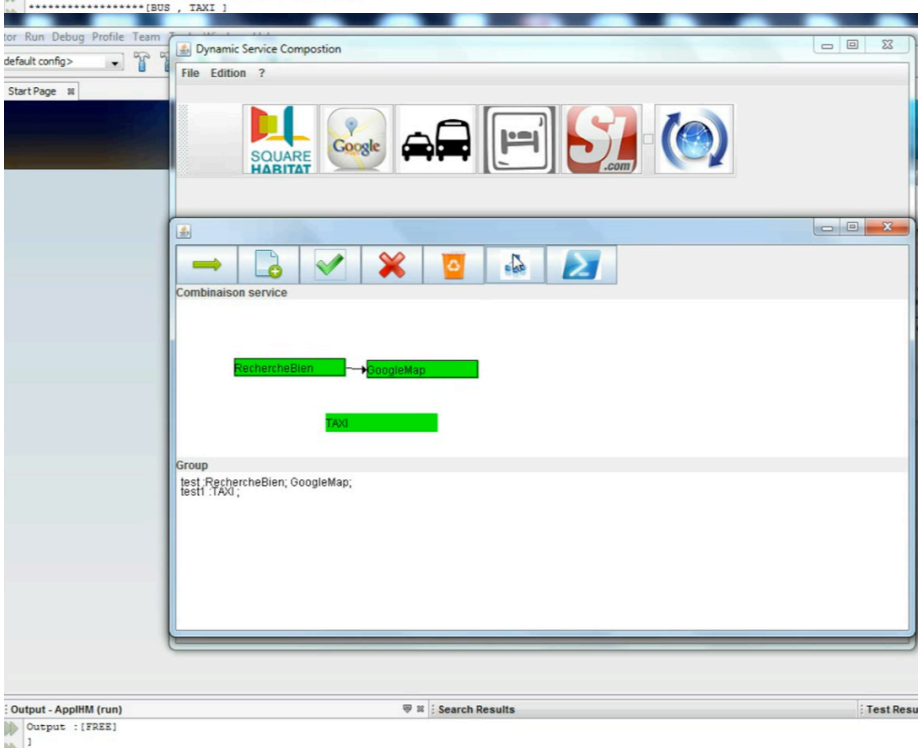
5.1 Module de planification

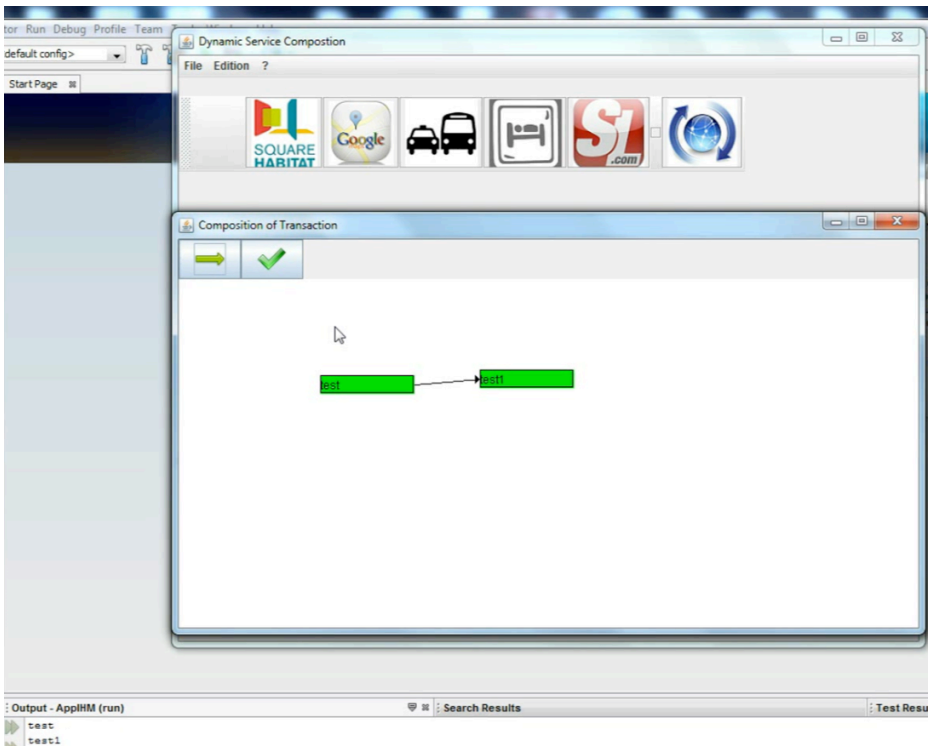
Le module de planification nous fournit un outil *draggable*, configurable et opérationnel. A travers ce module on peut obtenir un service de mashup du côté du médiateur puis ce service de mashup est enregistré dans un fichier XML, les démarches de capture d'écran sont illustrés suivants dans les figure 5.1.

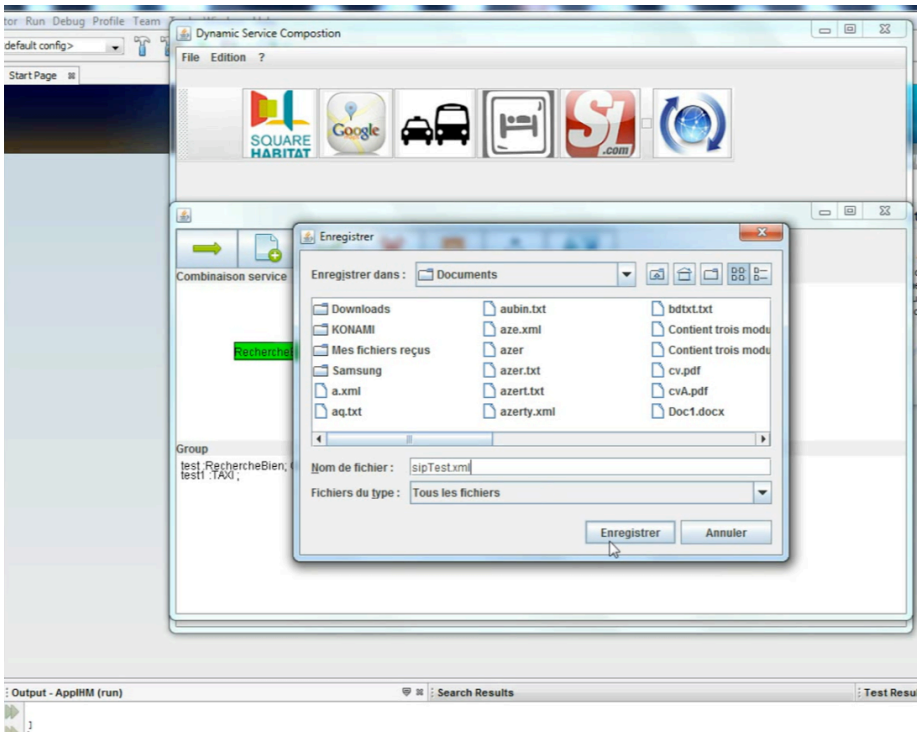



Dynamic Service Compostion
File Edition ?
Combinaison service
RechercheBien
GoogleMap
TAXI
Group
Output - AppIHM (run)
Search Results
[GoogleMap, MappyGPS , MappyPhoto]
[BUS , TAXI]
Dynamic Service Compostion
File Edition ?
Combinaison service
RechercheBien
TAXI
No group of transaction
Do you want to creat a new transaction ?
OK
Annuler
Group
Output - AppIHM (run)
Search Results
[GoogleMap, MappyGPS , MappyPhoto]
[BUS , TAXI]


Dynamic Service Compostion
File Edition ?
Combinaison service
RechercheBien
TAXI
Defination of name of a transaction
Name of your Transaction
test
OK
Annuler
Group
Output - AppHM (run)
Search Results
[GoogleMap, MappyGPS , MappyPhoto]
[BUS , TAXI]


Dynamic Service Compostion
File Edition ?
Combinaison service
RechercheBien
GoogleMap
TAXI
Group
test :RechercheBien; GoogleMap;
test1 :TAXI ;
Output - AppHM (run)
Search Results
Output : [FREE]


Dynamic Service Compostion
File Edition ?
Composition of Transaction
test
test1
Output - AppIHM (run)
Search Results
test
test1


Dynamic Service Compostion
File Edition ?
Enregistrer
Enregistrer dans : Documents
Downloads
KONAMI
Mes fichiers reçus
Samsung
a.xml
aq.txt
aubin.txt
aze.xml
azer
azer.txt
azert.txt
azerty.xml
bdtxt.txt
Contient trois modu
Contient trois modu
cv.pdf
cvA.pdf
Doc1.docx
Nom de fichier : sipTest.xml
Fichiers du type : Tous les fichiers
Enregistrer
Annuler
Combinaison service
Group
Output - AppIHM (run)
Search Results

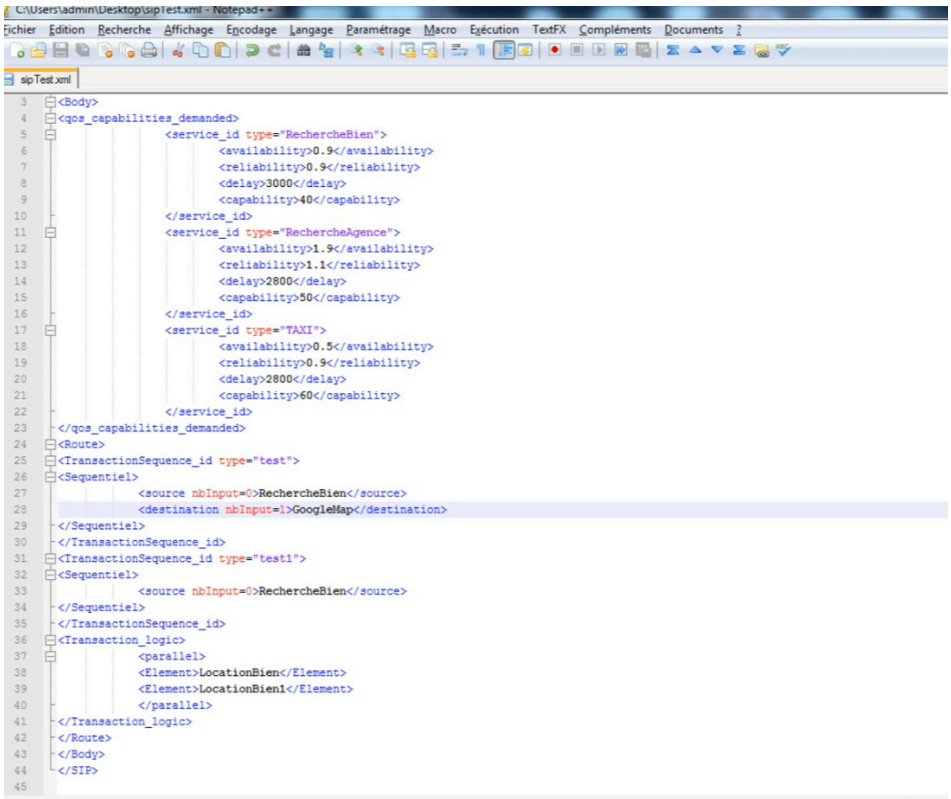


Fig. 5.1 les démarches de la planification de service

5.2 Mis en œuvre des services web

5.2.1 Présentation de WCF

Dans le cadre de notre proposition, nous avons choisi et commencé à réaliser une maquette du médiateur (module de générateur et exécuteur) visant à faciliter la composition dynamique d'une série de services personnalisés à l'aide d'un moteur de composition.

Cet outil permettra à un concepteur de « fabriquer » une application développée à l'aide d'une composition de service web. L'utilisateur peut choisir les services mais il ne pourra pas modifier cette composition.

L 'ensemble des composants fonctionnels sera développé dans un environnement .NET(C#, ASP.NET, …etc.), plus particulièrement en Framework 4.0 pour un développement de service en WCF (*Windows Communication Fondation*). L'ensemble sera développé sous « Visual Studio 2010 Professional ».

Les étapes de mise en œuvre de création d'un service WCF sont les suivantes :

- définition du contrat : On définit les signatures des méthodes composant notre service WCF, et le format des données à échanger.
- implémentation du contrat : On implémente les services.
- configuration du service : On définit les *endpoints*, autrement dit les points d'accès au service.
- hébergement des services dans une application .NET.

Puis, il ne reste plus qu'à créer une application cliente (quel que soit sa forme, Windows, Web, ou tout autre composant) qui pourra consommer le service WCF.

WCF repose sur trois éléments au sein desquels se trouvent les fonctionnalités de *matching* et *binding* du le médiateur.

- Une **Adresse** : adresse à laquelle le client doit se connecter pour utiliser le service.
- Un **Binding** : protocole à utiliser par le client pour communiquer avec le service.
- Un **Contrat** : informations échangées entre le serveur et le client afin que ce dernier sache comment utiliser le service.

5.2.2 Définition du contrat de service

Comme nous l'avons vu précédemment, le contrat de service est l'entité qui va :

- Etre échangée entre le serveur et le client.
- Permettre au client de savoir quelles sont les méthodes proposées par le service et comment les appeler.

L'élaboration d'un contrat de service s'effectue au travers les 3 métadonnées suivantes :

- ServiceContract: Cette métadonnée est attachée à la définition d'une classe ou d'une interface. Elle sert à indiquer que la classe ou l'interface est un contrat de service.
- OperationContract: Cette métadonnée est attachée aux méthodes que l'on souhaite exposer au travers du service WCF. Ainsi, il est techniquement possible de l'exposer au client comme certaines méthodes d'une classe.
- DataMember: Cet attribut se place avant les propriétés des classes qui définissent les objets que l'on va devoir passer en paramètre au service, ou que celui-ci va devoir nous retourner.

5.2.3 Hébergement du WS

Une fois que le service WCF est créé, nous allons l'héberger dans une application. Un service WCF peut être hébergé dans diverses applications :

- Dans une application cliente (Winform, service Windows…)
- Dans une application ASP .NET, elle-même hébergée sur un serveur IIS.

Pour notre moteur, nous allons juste chercher à héberger notre service sur WCF Service Host. WSH est le nom de l'hébergeur intégré à Visual Studio 2010. C'est lui qui héberge par défaut un service WCF créé avec le « *template* » de WCF *Service Library*.

La configuration d'un service WCF est stockée dans un fichier XML d'extension *.config* (*App.config* ou *Web.config* selon le type d'application). La configuration spécifique à WCF illustrée dans la figure 5.2 se trouve entre les balises *<system.serviceModel>* et *</system.serviceModel>*. On retrouve ici le nom de l'espace de nom spécifique à WCF :

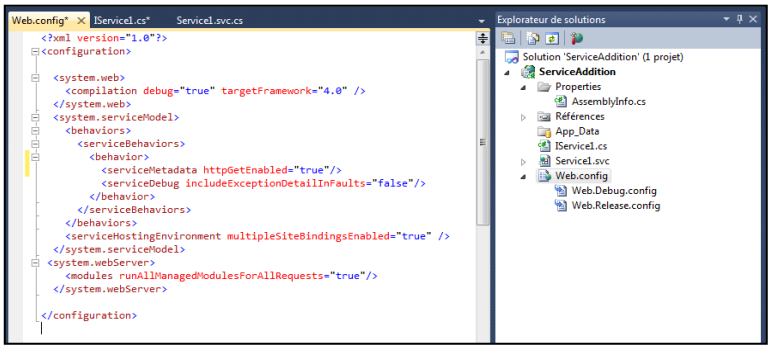


Fig. 5.2 La configuration spécifique à WCF

5.2.4 Consommation du WS

Nous avons vu que les échanges entre un service WCF et ses clients se font généralement par un mécanisme de sérialisation/désérialisation d'objets dans un langage dérivé du XML. De plus nous avons dit aussi que les clients obtiennent des informations sur un service donné grâce à l'appel d'un fichier d'extension .WSDL et que celui-ci contient la liste des méthodes exposées par le service ainsi que comment les utiliser (quels sont éventuellement les paramètres à fournir).

La combinaison de toutes les informations contenues dans le fichier WSDL va permettre à Visual Studio de générer ce que l'on appelle une « *classe proxy* ». Le rôle de cette classe proxy va être de masquer la complexité des mécanismes de communication entre le service web WCF et le client afin que le programmeur n'ait en réalité qu'un seul objet à utiliser (plus éventuellement les types complexes utilisés par le service qui seront alors eux aussi récupérés automatiquement). Cet objet exposera « localement » les méthodes proposées par le service distant. Comme son nom de « proxy » l'indique, le rôle de cette classe est de servir d'intermédiaire entre le client et le service.

Il existe deux méthodes pour générer une classe proxy :

- En ajoutant une référence de service dans le projet consommant le service WCF.
- En utilisant l'utilitaire svcutil.exe (méthode manuelle).

Nous allons maintenant voir comment écrire un client afin d'utiliser véritablement notre premier service WCF. Nous allons coder assez rapidement un client Winform illustré dans la figure 5.3 qui consommera notre service. Ce projet sera inclus dans la même solution que le reste du code et sera lancé en même temps que le service WCF lors du débogage.

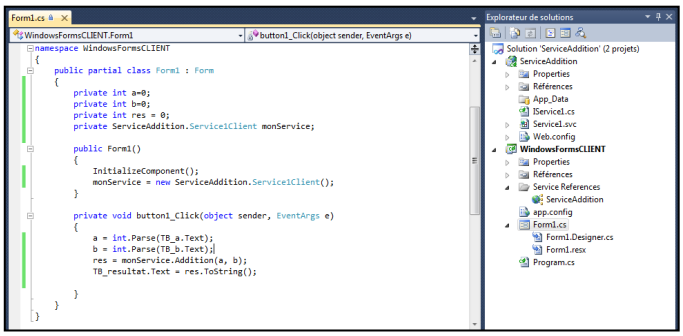


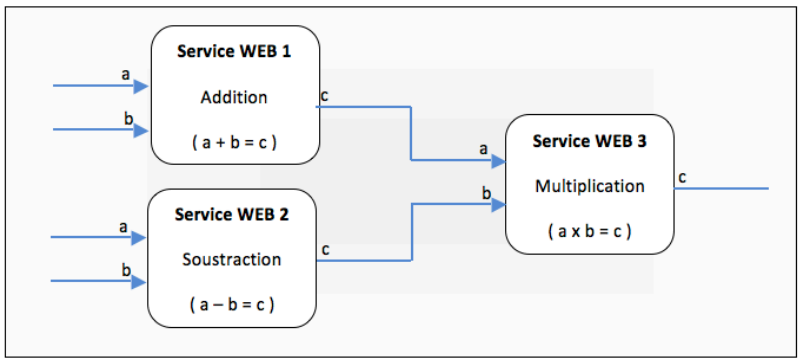
Fig. 5.3 Ajouter une référence de service dans le projet client

5.3 Mise en œuvre du médiateur

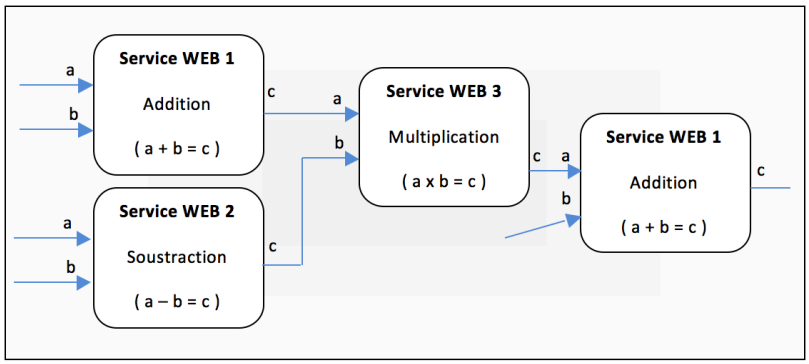
Le premier but de notre médiateur est de pouvoir orchestrer des services web. Nous devons donc être capable de définir un ordre d'exécution des services web et le chaînage qu'il pourrait y avoir. Avant de nous lancer dans le codage, nous avons effectué une étude sur les solutions déjà existantes. La prochaine partie présente ces solutions.

Pour la mise en œuvre de l'orchestration, nous avons décidé de créer deux projets différents : un pour le client et un pour le développeur. En effet, le but est de générer un code qui définira l'orchestration des services web, ce code sera mis dans un fichier nommé « structure.cs ». Ce fichier sera alors intégrer au projet client. Ainsi le client n'aura connaissance que du fichier « structure.cs » et du lien vers ces services web. Nous avons choisi d'effectuer tout d'abord une orchestration de services web très simple : Addition, Multiplication et Soustraction. Le but de la composition sera donc de pouvoir générer des chorégraphies comme celle-ci :

- Exemple 1:



- Exemple 2:



La difficulté est donc de pouvoir lier la sortie d'un service web vers un autre service et aussi de pouvoir le réutiliser une ou plusieurs fois. La deuxième difficulté est que ce chaînage sera codé dans un fichier pour que le client ne puisse pas changer à sa volonté la composition. Pour cela nous avons choisi de procéder de la manière suivante illustrée dans la figure 5.4:

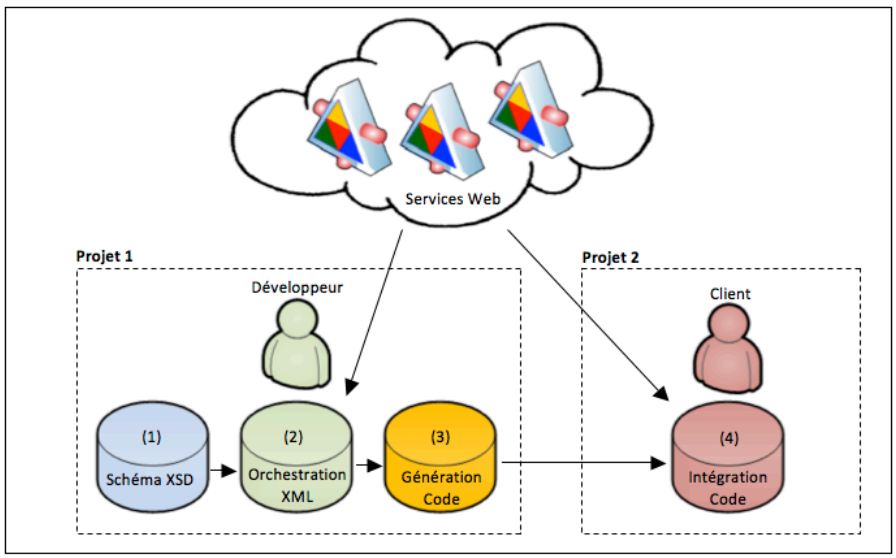


Fig. 5.4 Conception globale

(1) Il s'agit d'un schéma XSD construit par le développeur, permettant de définir le format du contrat XML.

(2) Il s'agit du contrat XML définissant l'orchestration des services web.

(3) Il s'agit du fichier « *structure.cs* » qui sera utilisé au sein de l'interface cliente. Ce code sera généré automatiquement en fonction du contrat XML.

(4) L'interface client ne tiendra compte que du fichier « *structure.cs* » et du lien vers les services web.

En conclusion, le projet 1 est la partie développement et servira pour le développeur. A l'aide d'un fichier XML il définira l'orchestration des services web et ainsi il générera un fichier « *structure.cs* » qu'il intégrera au projet 2 pour créer le produit final pour le client.

5.3.1 Contrat XSD

Le contrat XSD (*XML Schema Document*) désigne la norme « *XML Schema* » du W3C. Il s'agit d'une norme XML qui est utilisée pour décrire une structure de documents XML. Il permet donc de définir une structure de document XML, un format de document XML. Tout document XML respectant cette structure et référençant le XML Schéma est alors déclaré valide. C'est ce qu'on nous avons choisi d'utiliser pour définir notre composition.

Voici la structure de notre contrat que nous avons créé en correspondance avec notre projet :

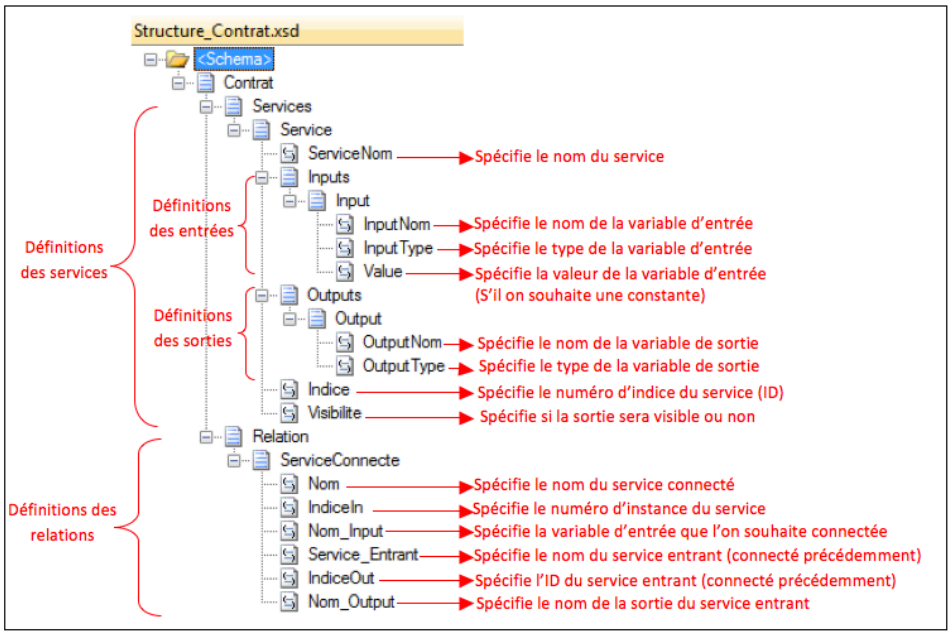


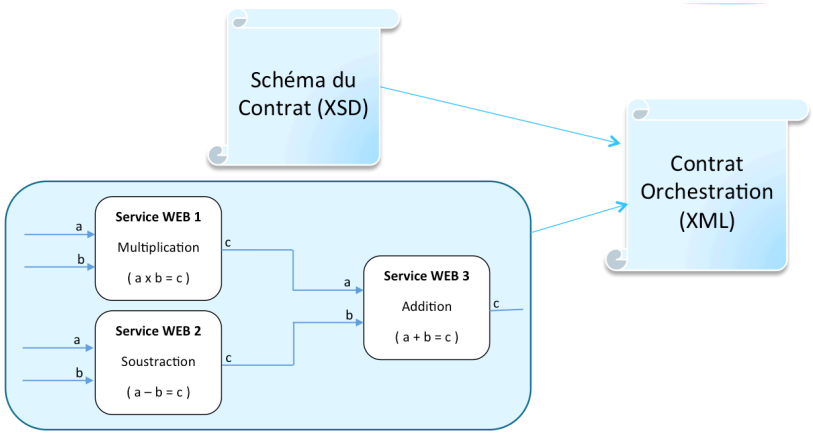
Fig. 5.5 Contrat d'orchestration

Notre fichier XML spécifiant la manière dont seront connectés nos services web devra donc suivre scrupuleusement ce schéma.

5.3.2 Contrat XML

Ce fichier correspond donc à l'écriture de la composition des services web que l'on souhaite, il doit suivre le schéma XSD vu précédemment.

Nous allons par exemple présenter la composition suivante dans l'exemple 1 :



Voici donc l'écriture de cette composition à l'aide du contrat XML illustré dans figure 5.6 :

```
<?xml version="1.0" encoding="utf-8" ?>
<Contrat xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="Structure_Contrat.xsd">
  <Services>
    <Service>
      <ServiceNom>Addition</ServiceNom>
      <Inputs>
        <Input>
          <InputNom>a</InputNom>
          <InputType>int</InputType>
        </Input>
        <Input>
          <InputNom>b</InputNom>
          <InputType>int</InputType>
        </Input>
      </Inputs>
      <Outputs>
        <Output>
          <OutputNom>RéponseAddition</OutputNom>
          <OutputType>int</OutputType>
        </Output>
      </Outputs>
      <Indice>1</Indice>
      <Visibilite>true</Visibilite>
    </Service>
    <Service>
      <ServiceNom>Multiplication</ServiceNom>
      <Inputs>
        <Input>
          <InputNom>a</InputNom>
          <InputType>int</InputType>
        </Input>
        <Input>
          <InputNom>b</InputNom>
          <InputType>int</InputType>
```

Spécifications du service 1 « Addition ».

Il a 2 entrées « a » et « b » de types « int »

Il a 1 sortie « RéponseAddition » de type « int ».

Il s'agit de l'indice numéro 1 (ID).

Sa sortie sera visible


```
        </Input>
      </Inputs>
      <Outputs>
        <Output>
          <OutputNom>RéponseMultiplication</OutputNom>
          <OutputType>int</OutputType>
        </Output>
      </Outputs>
      <Indice>2</Indice>
      <Visibilite>false</Visibilite>
    </Service>
    <Service>
      <ServiceNom>Soustraction</ServiceNom>
      <Inputs>
        <Input>
          <InputNom>a</InputNom>
          <InputType>int</InputType>
          <Value>5</Value>
        </Input>
        <Input>
          <InputNom>b</InputNom>
          <InputType>int</InputType>
        </Input>
      </Inputs>
      <Outputs>
        <Output>
          <OutputNom>RéponseSoustraction</OutputNom>
          <OutputType>int</OutputType>
        </Output>
      </Outputs>
      <Indice>3</Indice>
      <Visibilite>true</Visibilite>
    </Service>
  </Services>
  <Relation>
    <ServiceConnecte>
      <Nom>Addition</Nom>
      <IndiceIn>1</IndiceIn>
      <Nom_Input>a</Nom_Input>
      <Service_Entrant>Multiplication</Service_Entrant>
      <IndiceOut>2</IndiceOut>
      <Nom_Output>RéponseMultiplication</Nom_Output>
    </ServiceConnecte>
    <ServiceConnecte>
      <Nom>Addition</Nom>
      <IndiceIn>1</IndiceIn>
      <Nom_Input>b</Nom_Input>
      <Service_Entrant>Soustraction</Service_Entrant>
      <IndiceOut>3</IndiceOut>
      <Nom_Output>RéponseSoustraction</Nom_Output>
    </ServiceConnecte>
  </Relation>
</Contrat>
```

Fig. 5.6 Contrat XML

5.3.3 Génération du code

Nous devons maintenant générer le code qui permettra au client de pouvoir utiliser la composition des services sans connaître le fichier XML créé précédemment. Pour cela nous allons analyser le fichier XML à l'aide du code « *ParseurXML .cs*», nous allons ensuite générer le code à l'aide du fichier « *GénérationDuCode.cs* ». Ces 2 fichiers sont décrits en annexes.

Pour une meilleure efficacité nous avons décidé de créer une classe « Service » qui définit un service et une classe « *AttributService* » qui définit les types des interfaces d'entrées ou de sorties du service.

Voici les attributs de la classe « Service » (le contenu complet est défini en annexe) :

```
private BibService.ServiceMult.Service1Client serviceMult;
private BibService.ServiceAdd.Service1Client serviceAdd ;
private BibService.ServiceSoustr.Service1Client serviceSoustr;   } Définitions des services utilisables
private string _nom;          → Définitions du nom du service
private int _ordre;           → Définitions de l'indice du service
private List<AttributService> listeInput;   → Définitions des entrées
private List<AttributService> listeOutput;  → Définitions des sorties
private Boolean _Tete = true;     → Pour savoir si la composition commence par ce service
private Boolean _Visible = true;  → Pour savoir si la sortie, le résultat est visible
private Service _suivant;         → Pour connaître le service suivant connecté
```

Voici les méthodes de la classe « *Service* » :

```
public void MettreAjour()   → Pour mettre à jour les entrées du service suivant
public Boolean Pret()       → Pour tester si toutes les entrées possèdent une valeur
public Int32 Execution()    → Mise en œuvre du traitement du service
```

Voici les attributs de la classe « *AttributService* » (le contenu complet est défini en annexes) :

```
private string _nom;          → Nom de l'entrée ou sortie
private string _type;         → Son type
private object _value;        → Sa valeur
private AttributService _connextion;   → Son lien vers une autre entrée ou sortie
```

Ces 2 classes vont maintenant nous aider pour la génération du code. Nous devons aussi analyser le fichier XML que nous avons reçu. Pour cela nous avons créé un fichier « *ParseurXML.cs* » qui va nous permettre de retenir les informations dans le but de créer des objets *Service* et *AttributService*. Le code de ce fichier est définit en annexe.

Maintenant que nous avons créé nos objets *Service* et *AttributService* nous allons pouvoir générer le code qui sera utilisé par l'interface client. A l'aide de la classe « *GénérationDuCode* » nous allons créer un fichier « *Structure.cs* » dans lequel nous allons créer tous les objets dont nous avons besoins pour la configuration ainsi que leur composition pour le deuxième projet. L'ensemble du fichier « *GénérationDuCode.cs* » est présent en annexe.

Pour revenir à notre exemple précèdent avec nos 3 services web, nous avons créé le fichier XML, après exécution de la génération du code nous obtenons automatiquement le fichier « *structure* » suivant :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace PFE27012012
{
 public class Structure
{

private List<Service> _liste = new List<Service>();
public Structure()
{

/*-----------------------0--------------------*/
Service Addition1= new Service();
Addition1.nom="Addition";
Addition1.Indice=1;
Addition1.Visible=true;
AttributService i00= new AttributService();
i00.nom = "a";
i00.type = "int";
Addition1.Inputs.Add(i00);
AttributService i01= new AttributService();
i01.nom = "b";
i01.type = "int";
Addition1.Inputs.Add(i01);
AttributService O00= new AttributService();
O00.nom = "RéponseAddition";
O00.type = "int";
Addition1.OutPuts.Add(O00);
Addition1.ServiceTete=false;
/*-----------------------1--------------------*/
Service Multiplication2= new Service();
Multiplication2.nom="Multiplication";
Multiplication2.Indice=2;
Multiplication2.Visible=false;
AttributService i10= new AttributService();
i10.nom = "a";
i10.type = "int";
Multiplication2.Inputs.Add(i10);
AttributService i11= new AttributService();
i11.nom = "b";
i11.type = "int";
Multiplication2.Inputs.Add(i11);
AttributService O10= new AttributService();
O10.nom = "RéponseMultiplication";
O10.type = "int";
Multiplication2.OutPuts.Add(O10);
Multiplication2.ServiceTete=true;
/*-----------------------2--------------------*/
Service Soustraction3= new Service();
Soustraction3.nom="Soustraction";
Soustraction3.Indice=3;
Soustraction3.Visible=true;
AttributService i20= new AttributService();
i20.nom = "a";
i20.type = "int";
i20.value=5;
i20.box.Text="5";
i20.box.Enabled=false;
Soustraction3.Inputs.Add(i20);
AttributService i21= new AttributService();
i21.nom = "b";
i21.type = "int";
Soustraction3.Inputs.Add(i21);
AttributService O20= new AttributService();
```

```
O20.nom = "RéponseSoustraction";
O20.type = "int";
Soustraction3.OutPuts.Add(O20);
Soustraction3.ServiceTete=true;
_liste.Add(Addition1);
Multiplication2.Suivant = Addition1;
Addition1.getInput("a").connextion=Multiplication2.getOutput("RéponseMultiplication");
_liste.Add(Multiplication2);
Soustraction3.Suivant = Addition1;
Addition1.getInput("b").connextion=Soustraction3.getOutput("RéponseSoustraction");
_liste.Add(Soustraction3);
}
public List<Service> liste
{
get { return _liste; }
}

}
}
```

5.3.4 Partie Cliente

Une fois ce code généré nous pouvons intégrer le fichier « structure.cs » à notre projet client. Ce projet possède aussi l'accès aux 3 services web (addition, soustraction et multiplication). Nous avons décidé de créer une interface graphique construite automatiquement en fonction de la composition des services web. Nous avons donc créé un tableau qui comportera chaque service avec ses entrées et ses sorties. Voici la représentation de l'interface illustrée dans la figure 5.7 en reprenant notre exemple précèdent :

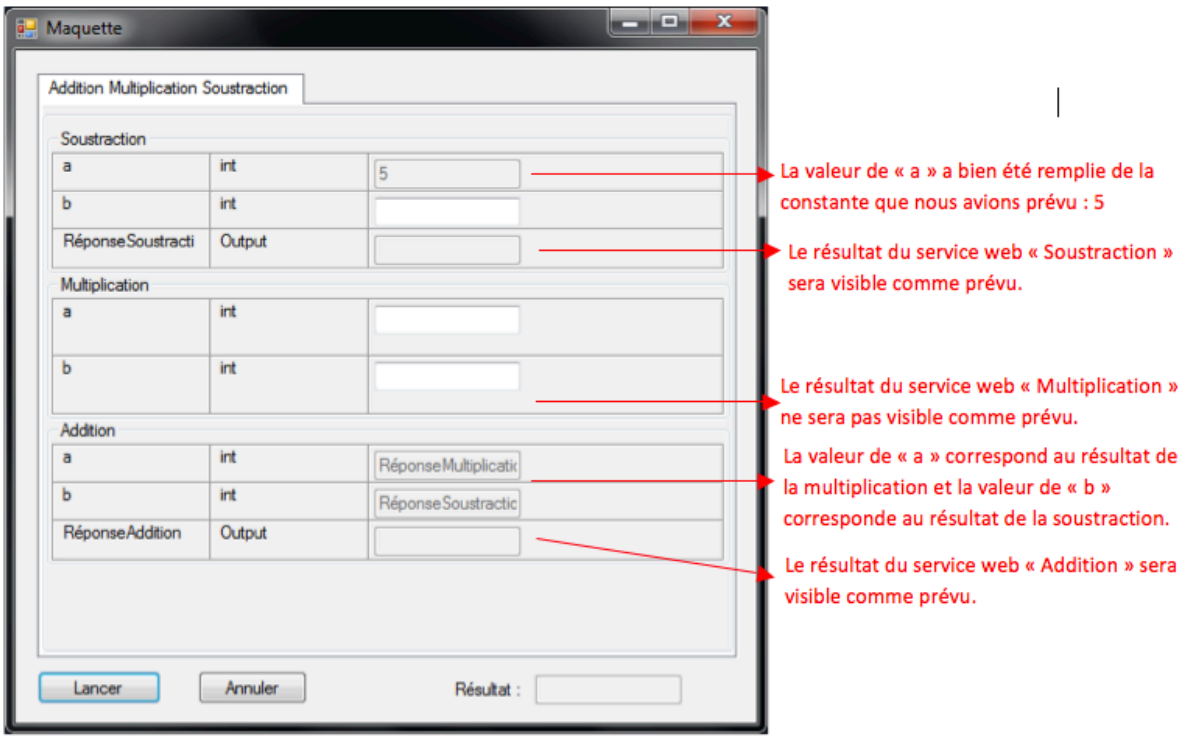


Fig. 5.7-a Interface graphique de la partie cliente

L'utilisateur doit remplir les champs obligatoires puis il clique sur le bouton « Lancer », ce qui aura pour effet de lancer l'orchestration des services web et d'afficher le résultat final dans le champ « Résultat ».

Voici la représentation de l'interface après avoir cliqué sur le bouton « Lancer » :

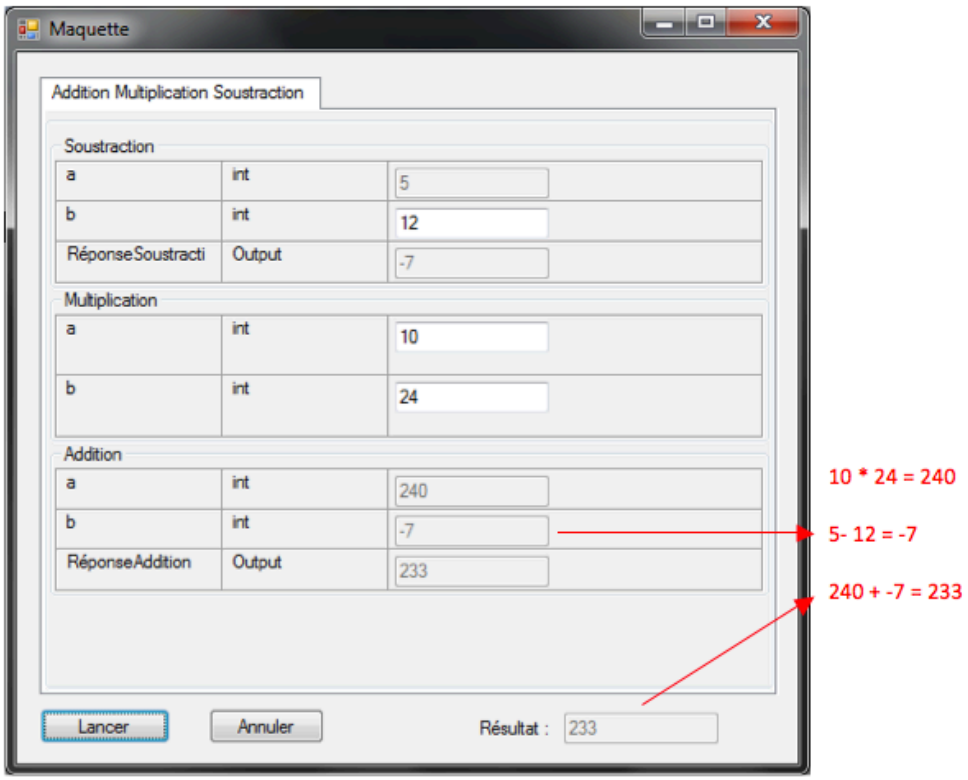


Fig. 5.7-b Interface graphique de la partie cliente

L'ensemble du code qui génère l'interface graphique est spécifié en annexe (*TabService.cs*).

5.3.5 Partie Développeur

Nous avons décidé de créer une interface graphique pour le développeur l'aidant ainsi à mieux gérer son projet. Il aura donc la même visibilité que le client, mais il aura en plus le choix de rajouter les contrats au format XML pour bien vérifier le fonctionnement. Ce projet comportera donc la génération du code en fonction du fichier XML et en plus la représentation graphique du fichier « *structure.cs* » généré. La représentation des différentes compositions sera visible sous forme d'onglets. Voici la représentation de l'interface graphique :

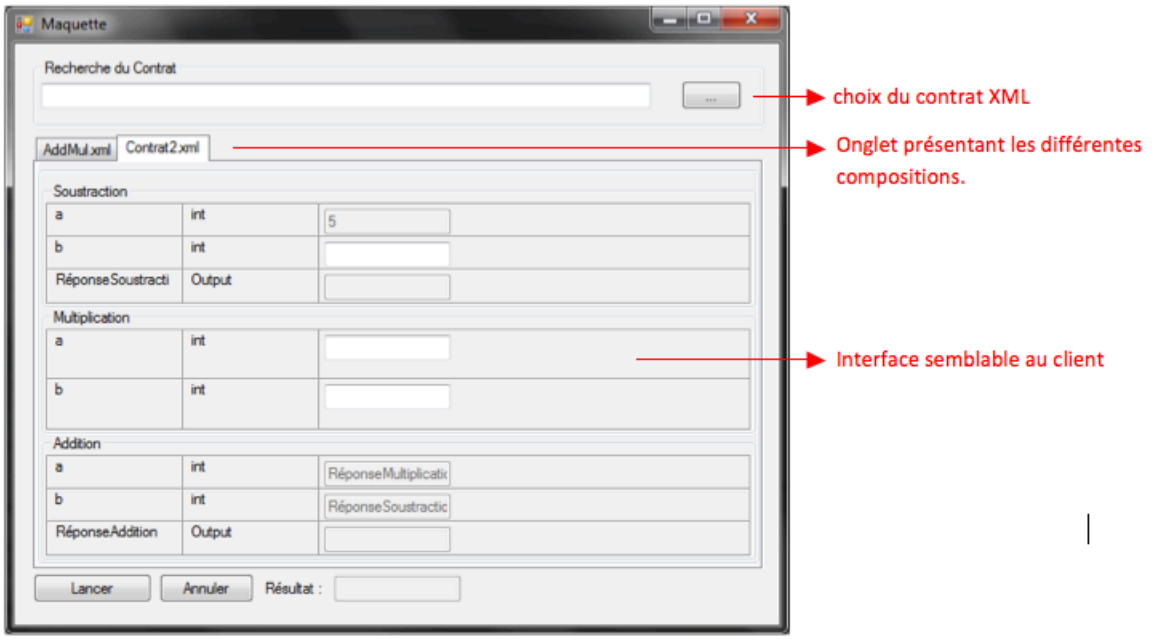


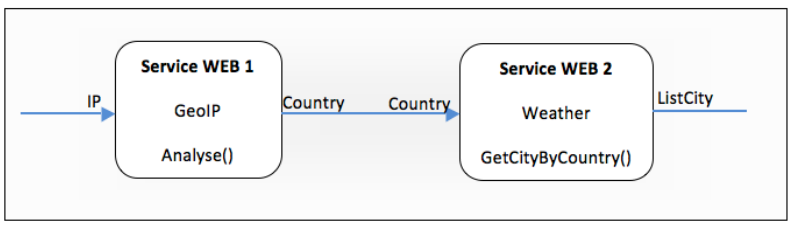
Fig. 5.8 Interface graphique de la partie du développeur

5.3.6 Mis en œuvre avec des services web réels

Nous avons ensuite effectué le même principe que précédemment avec 2 services web réels, c'est-à-dire 2 services web accessibles via le web dont on ne connait que le WSDL. Voici les 2 services web que nous avons sélectionné :

- GeoIp : a pour but de localiser le pays en fonction d'une adresse IP rentrée.
- Weather : a pour but de donner une liste de villes en fonction d'un pays rentré.

Nous avons effectué la composition très simple suivante :



Nous avons gardé notre schéma XSD, mais nous avons défini le contrat XML permettant d'effectuer la composition voulue, voici ce fichier :

```
<?xml version="1.0" encoding="utf-8" ?>
<Contrat xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="Structure_Contrat.xsd">
  <Services>
    <Service>
      <ServiceNom>geoip</ServiceNom>
      <Inputs>
        <Input>
          <InputNom>IP</InputNom>
          <InputType>string</InputType>
```

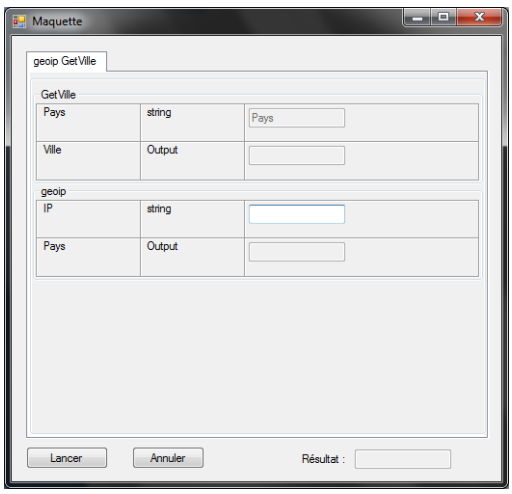
```
        </Input>
      </Inputs>
      <Outputs>
        <Output>
          <OutputNom>Pays</OutputNom>
          <OutputType>string</OutputType>
        </Output>
      </Outputs>
      <Indice>1</Indice>
      <Visibilite>true</Visibilite>
    </Service>
    <Service>
      <ServiceNom>GetVille</ServiceNom>
      <Inputs>
        <Input>
          <InputNom>Pays</InputNom>
          <InputType>string</InputType>
        </Input>
      </Inputs>
      <Outputs>
        <Output>
          <OutputNom>Ville</OutputNom>
          <OutputType>string</OutputType>
        </Output>
      </Outputs>
      <Indice>2</Indice>
      <Visibilite>true</Visibilite>
    </Service>
  </Services>
  <Relation>
    <ServiceConnecte>
      <Nom>GetVille</Nom>
      <IndiceIn>2</IndiceIn>
      <Nom_Input>Pays</Nom_Input>
      <Service_Entrant>geoip</Service_Entrant>
      <IndiceOut>1</IndiceOut>
      <Nom_Output>Pays</Nom_Output>
    </ServiceConnecte>
  </Relation>
</Contrat>
```

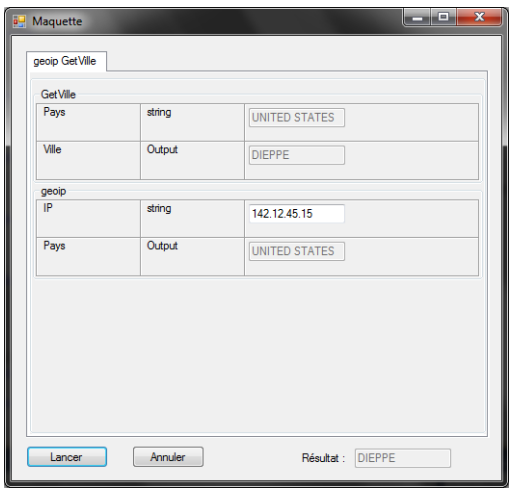
De la même manière que précédemment, nous avons généré le code permettant de construire le fichier « structure.cs » qui sera utilisé par le client, voici le code généré :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace PFE27012012
{
 public class Structure
{

private List<Service> _liste = new List<Service>();
public Structure()
{

/*----------------------0--------------------*/
Service geoip1= new Service();
geoip1.nom="geoip";
geoip1.Indice=1;
```

```
geoip1.Visible=true;
AttributService i00= new AttributService();
i00.nom = "IP";
i00.type = "string";
geoip1.Inputs.Add(i00);
AttributService O00= new AttributService();
O00.nom = "Pays";
O00.type = "string";
geoip1.OutPuts.Add(O00);
geoip1.ServiceTete=true;
/*-----------------------1--------------------*/
Service GetVille2= new Service();
GetVille2.nom="GetVille";
GetVille2.Indice=2;
GetVille2.Visible=true;
AttributService i10= new AttributService();
i10.nom = "Pays";
i10.type = "string";
GetVille2.Inputs.Add(i10);
AttributService O10= new AttributService();
O10.nom = "Ville";
O10.type = "string";
GetVille2.OutPuts.Add(O10);
GetVille2.ServiceTete=false;
geoip1.Suivant = GetVille2;
GetVille2.getInput("Pays").connextion=geoip1.getOutput("Pays");
_liste.Add(geoip1);
_liste.Add(GetVille2);
}
public List<Service> liste
{
get { return _liste; }
}
}
}
```

Enfin, l'interface graphique générée automatiquement a été illustrée dans les figures de démonstrations 5.9 suivantes :

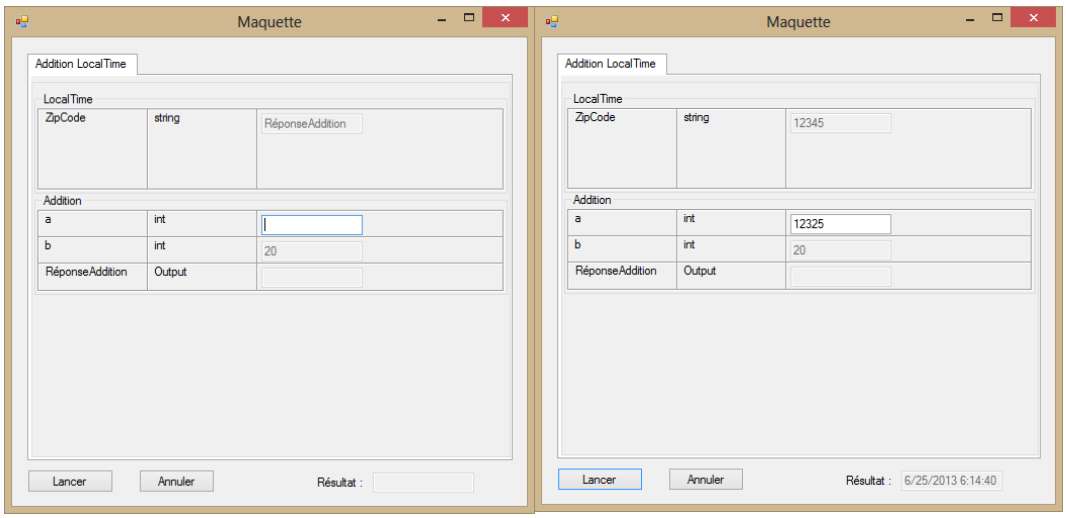




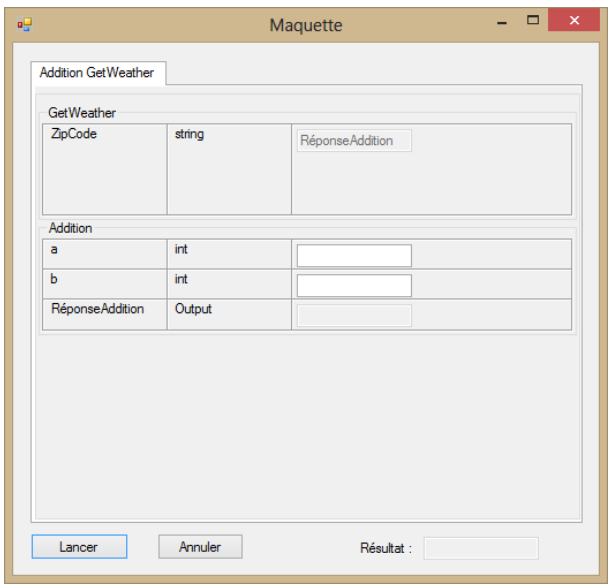
Nous réalisons quelques exemples de compositions dynamiques dans la démonstration.

Démo1 : Il s'agit d'un service local « Addition » et d'un service public « LocalTime »

`http://www.ripedevelopment.com/webservices/LocalTime.asmx`

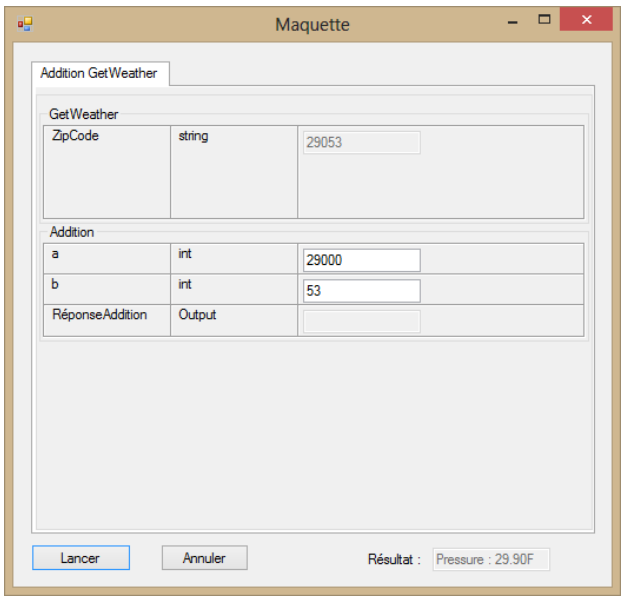


Démo2 : Il s'agit d'un service local « Addition » et d'un service public « GetWeather » :

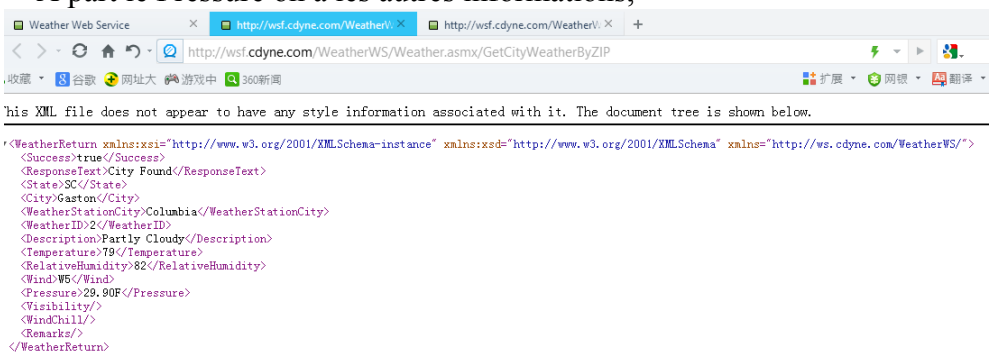


D'après le Zip code de US, On peut obtenir le Pressure sur les informations *Weather*

Par exemple, on fait l'addition 29000 et 53 pour avoir un zip code de US 29053 et puis d'après ce zip code on obtient les informations sur le weather.

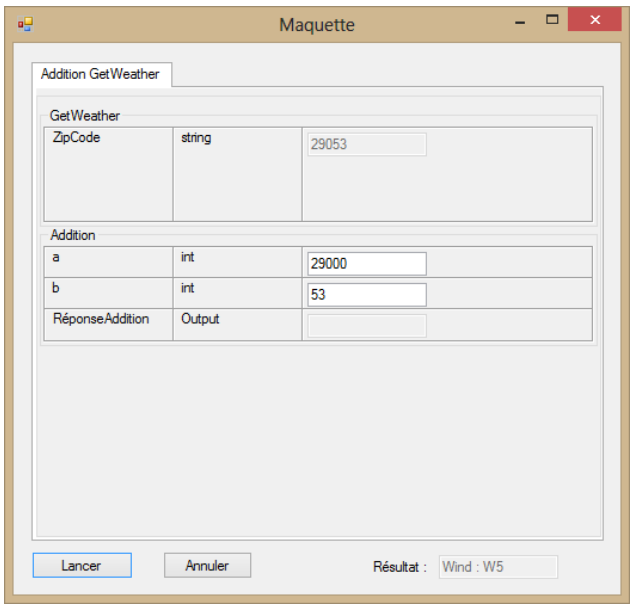


A part le Pressure on a les autres informations,



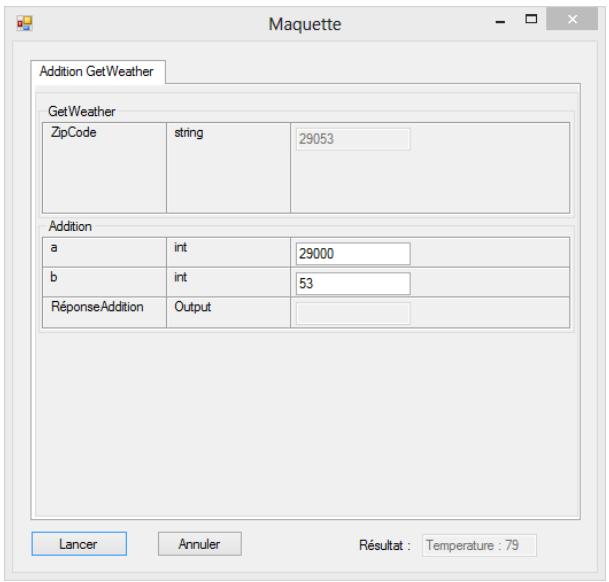
Et tout simplement on peut obtenir les autres informations d'après la fonctionnalité parsing de notre médiateur.

```
            return 0;
        }
        if (this.Suivant == null)
        {
            if (nom.Equals("GetWeather"))
            {
                return "Wind : "+((WeatherReturn)(OutPuts[0].value)).Wind;
            }
            return OutPuts[0].value.ToString();
```



Par exemple, on veut le *Temperature* au lieu de *Presure*

```
            return 0;
        }
        if (this.Suivant == null)
        {
            if (nom.Equals("GetWeather"))
            {
                return "Temperature : " + ((WeatherReturn)(OutPuts[0].value)).Temperature;
            }
            return OutPuts[0].value.ToString();
```



Démo3 : Il s'agit deux services publics « IpToCountry » et « GetCountry »

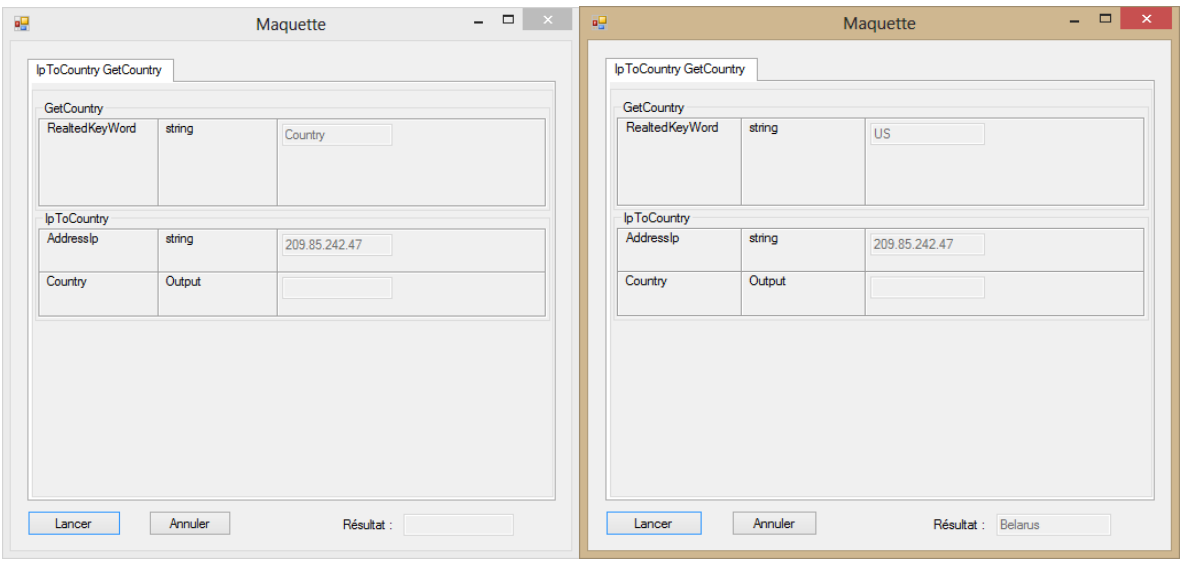


Fig. 5.9 Démonstrations

5.4 Conclusion

Dans ce chapitre, nous avons fait l'expérimentation de la maquette du médiateur et nous avons réalisé l'orchestration de service à travers notre maquette dans un premier temps suivre la planification de service et faire fonctionne les modules dans le médiateur.

Chapitre 6

Les extensions

6.1 L'architecture de *cloud* pour le WS-médiateur

6.1.1 Analyse de l'exigence

La composition de service s'appuie sur l'architecture SOA et fournit une technologie contemporaine pour développer des applications complexes à partir des composants des services existants. Surtout dans le domaine ubiquitaire, une solution est nécessaire pour accroître la disponibilité des services et fournir la composition de service à la demande (*on-demand*). Dans l'orchestration de service web traditionnel, il est difficile d'obtenir la composition de service à la demande dans la limite d'une organisation, à moins que l'organisation puisse nous fournir tous les services nécessaires pour la composition. Par conséquent, il ne peut probablement pas se permettre une composition dynamique de services dans la situation ubiquitaire. Bref, la composition dynamique de services *on-the-fly* et *on-demand* est nécessaire, nous sommes motivé à explorer des nouvelles possibilités pour la réalisation de la composition dynamique de service dans le domaine ubiquitaire.

La composition de service, pour l'orchestration et la chorégraphie de service traditionnelles, sont toujours réalisées dans un organisme particulier, ou dans un nombre limité d'organisations. La composition de service est limitée à ces organisations de services, celles qui sont essentiellement pour le *Pervsive Service Computing*. C'est à dire, celle des architectures de composition de service traditionnel. La communication entre les services dans une composition est généralement réalisée comme un simple modèle de client-serveur. Ce genre de composition de service est tellement « couplage fort », donc il ne correspond pas à des scénarios « couplage lâche » dans l'environnement ubiquitaire. Dans la suite, on va parler comment explorer la composition dynamique de service dans l'architecture de *cloud*, et viser à fournir aux utilisateurs une médiation basée sur le *cloud* pour la composition dynamique qui supporte la composition de service *on-demand* et *on-the-fly* dans le *Cloud*.

6.1.2 Médiation dans l'architecture Cloud

Dans le chapitre 3, on a déjà proposé une médiation pour la composition dynamique de service en précisant l'ensemble des fonctionnalités modules en fournissant un environnement de composition, dans lequel les utilisateurs peuvent décrire les exigences des tâches, faire des planifications, de la composition, découvrir les services, générer, exécuter et contrôler des compositions de service. Mais il y a un problème généralement pour toutes les approches centrales, c'est le point de panne unique (*single point of failure*), par conséquent, qui est critique pour la fiabilité de la composition de service.

On propose la « médiation as a service » qui fonctionne comme un middleware dans le *cloud* qui est une couche fonctionnelle pour la composition de service. Par conséquent, les systèmes et les applications basés sur cette médiation peuvent utiliser les ressources et les services de *cloud* puisqu'il nous permettre la composition dynamique de services *on-demand*.

En plus, le *Cloud Computing* nous fournit la possibilité pour le *pooling* des ressources des serveurs « *cluster* », il offre aussi la possibilité de l'attribution des ressources dynamiques et virtuelles pour les applications à la demande. On peut prendre la médiation comme une abstration de la composition, et chaque composition est une instance de la médiation. Chaque composition de service fournie par la médiation de service peut être considérée comme un service virtuel dans le *cloud*.

Pour le service dans le *cloud*, il se réfère aux matériels et logiciels accessibles via l'internet et hébergé par les centres de données. Les utilisateurs peuvent profiter des services du Cloud sur une base de paiement à l'utilisation (*a pay-per-use basis*).

Les services virtuels peuvent être développés dans une manière efficace par exemple les méta-modèles pour composer à partir des services existants. Pour répondre à ce but précis dans le *cloud*, une architecture avec la couche de middleware dans laquelle on peut trouver notre médiateur est nécessaire comme illustrée dans la figure 6.1.

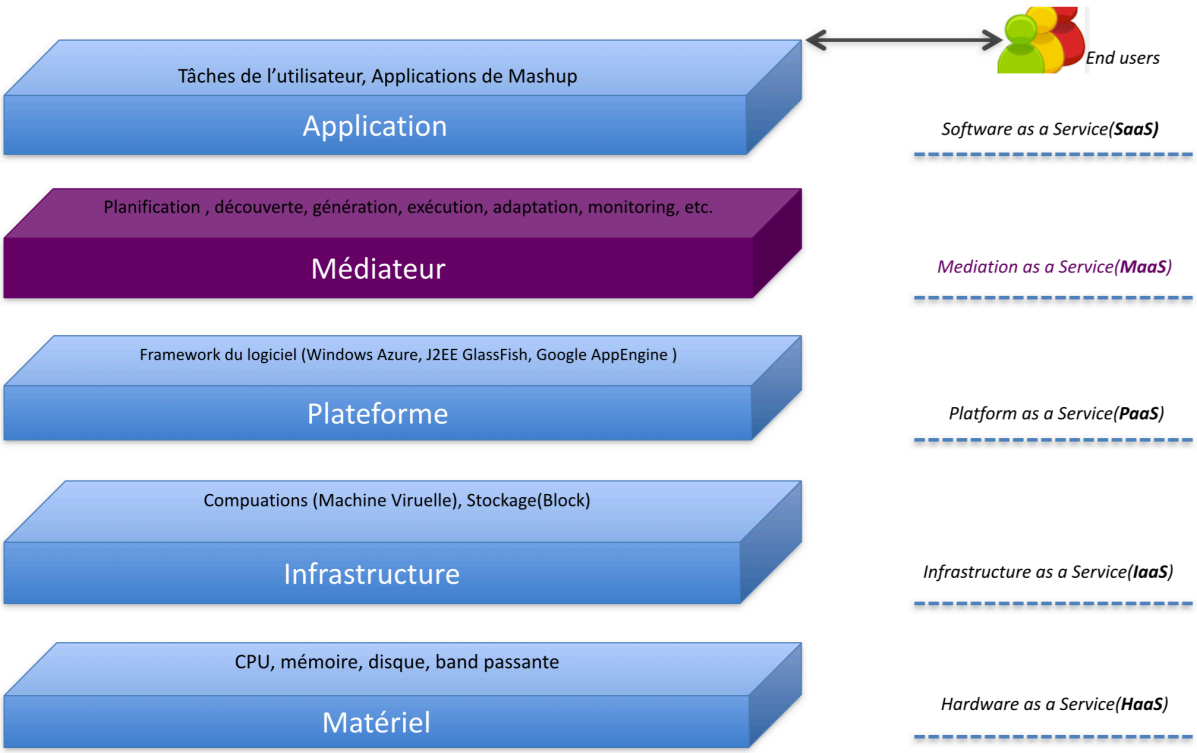


Fig. 6.1 médiation dans l'architecture de *cloud*

Nous appliquons « médiation as a service » qui est un médiateur dans la couche de middleware pour accélérer la composition dynamique de service pour étendre l'architecture de *cloud* existant. La couche de middleware se trouve entre la couche d'application et la couche de plate-forme, qui a pour but de créer et de gérer un *pool* de ressources de service pour accélérer la composition dynamique de service. Le middleware est généralement considéré comme un ensemble de services qui relie les applications de logiciel et le système d'exploitation, dans le but de réduire la complexité dans les applications et les services distribués.

Nous citons la « médiation as a service » dans le *cloud*, c'est pour relier la couche d'application et la couche de plate-forme pour le développement des services accélérées. Donc « composition as a service » est un instance de « médiation as a service » dans le *cloud*. En d'autre terme, c'est ce qui permet aux ressources de la composition virtuelle d'être gérées séparément par les utilisateurs et les fournisseurs. .

6.1.3 Les modules encapsulés et les conditions aux limites

D'après notre proposition mentionnée dans le chapitre 3, on va spécifier les modules encapsulés et les conditions aux limites et comment cela fonctionne dans cette architecture de

cloud. La figure 6.2 présente les fonctionnalités des modules dans la couche middleware ainsi que la relation avec la couche application et la couche de plate-forme. Notre médiateur dans le *cloud* fournit les services pour accélérer la composition de service à la couche d'application. La couche de plate-forme nous offre les stockages et les enregistrements des services pour le médiateur afin de réaliser les interactions et les découvertes de service « couplage lâche ». Les services dans le registre peuvent être le service SOAP ou le service RESTful. On pourrait voir le service SOAP et service REST comme des services exposables dans le cloud publique. La composition de service virtuelle, bien sûr, on peut la voir comme le service de Mashup pour l'utilisateur.

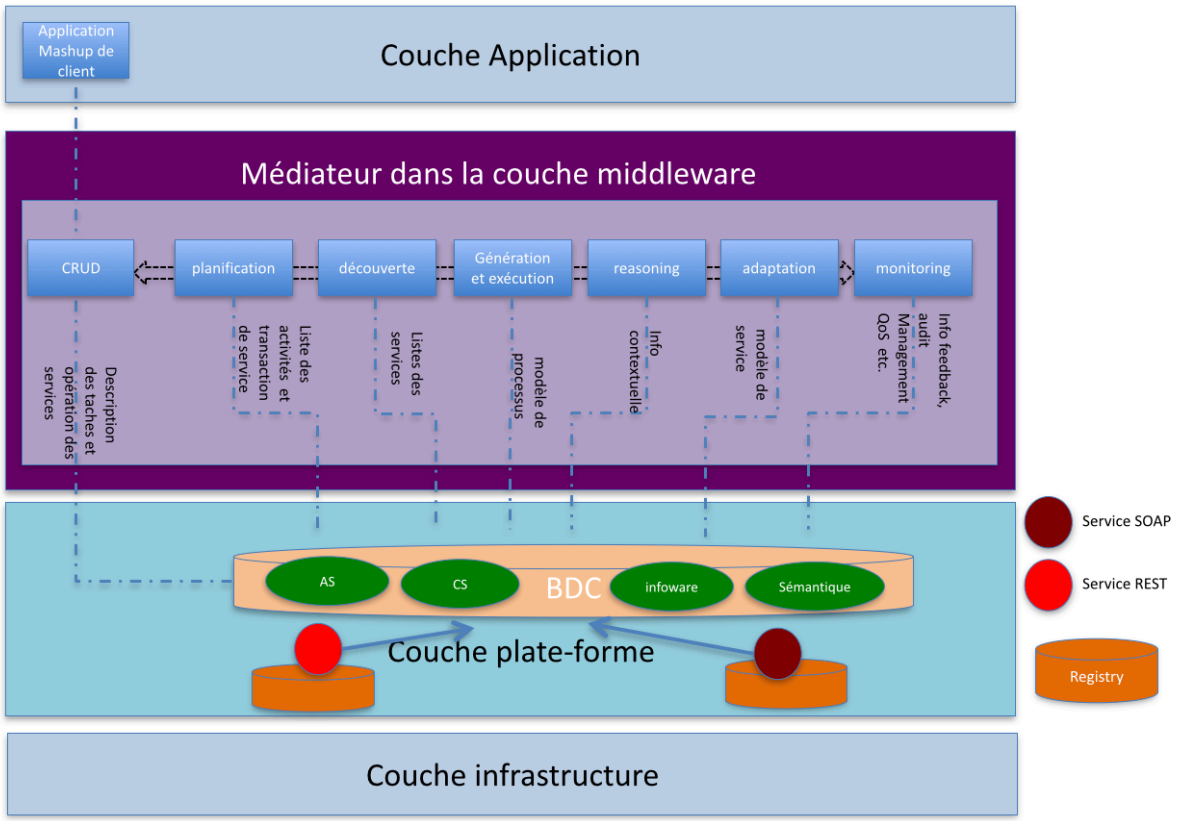


Fig. 6.2 les fonctionnalités des modules dans la couche middleware

6.1.4 La médiation basé sur la Plate-forme de Cloud AZURE

Le but de cette partie est de prouver que notre médiateur fonctionne dans la plate-forme cloud. On peut développer notre prototype de médiation en utilisant le Windows Azure Cloud plate-forme illustré dans la figure 6.3, Windows Communication Foundation (WCF) et Windows Workflow Foundation (WF). Le prototype est dans l'objectif de fournir une « médiation as a service » pour accélérer la composition de service dynamique. Ce prototype

permet la composition dynamique parce que l'utilisateur peut créer les nouvelles tâches on-the-fly et on-demand par composition des services dans le cloud.

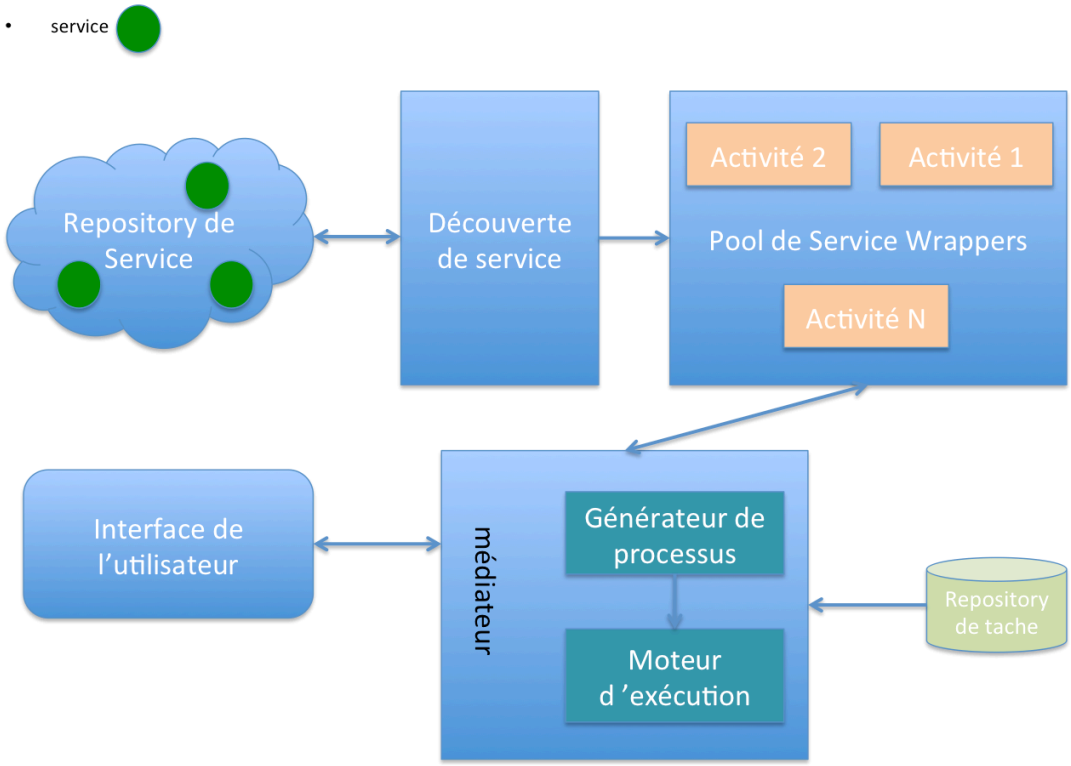


Fig. 6.3 le médiateur dans la plate-forme de Cloud AZURE

6.2 L'IoT basé sur WS-médiation (WMIoT)

6.2.1 Un environnement orienté WMIoT

Dans l'environnement de l'*Internet of Things* basé sur notre WS-médiation, comme illustré dans la figure 6.4, nous pouvons décrire quatre couches : couche d'abstraction et d'accès, couche de l'agrégation de la ressource, couche de support de service intelligent, et couche d'application. Cet environnement de l'architecture proposée est basé sur chaque équipement de l'IoT qui est considéré préalablement comme une ressource fondamentale du Web, mais en fonction des scénarios d'applications spécifiques, il permet aux développeurs tiers ou fournisseurs de service de créer des applications intelligentes à travers chaque un ou plusieurs couches de service.

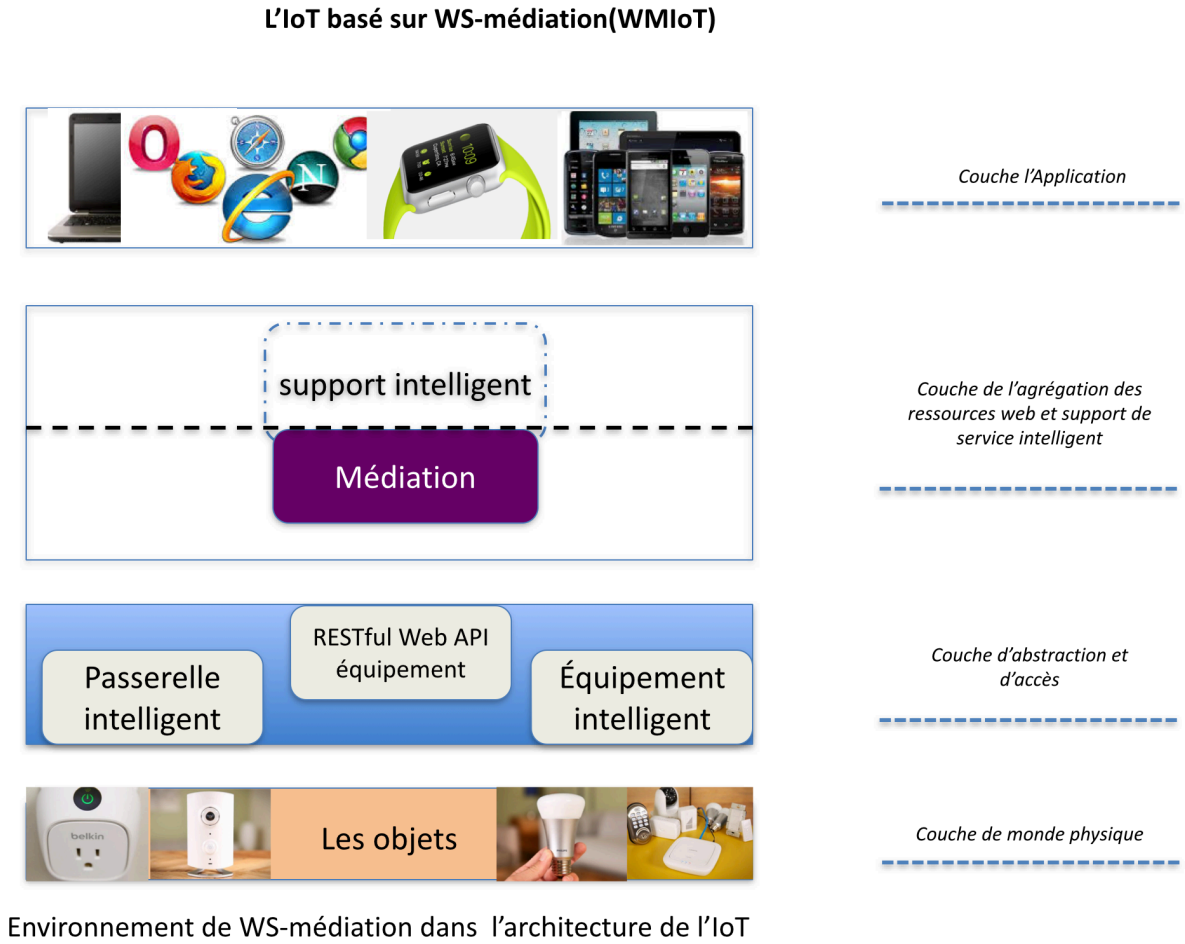


Fig. 6.4 Un environnement de WS-médiation dans l'architecture de l'IoT

Couche d'abstraction et d'accès : Du point de vue de l'application, on a besoin des applications pour accéder aux ressources ou aux données fournies par les équipements physiques différents. Donc on doit le faire selon les standards du Web et le style de REST mentionné dans l'état de l'art.

Pour ce faire, premièrement il faut pouvoir accéder aux objets internet pour que l'on puisse les considérer comme des ressources du Web. D'autre part, il faut abstraire les fonctionnalités des équipements comme WS selon les règles de REST et des normes de WS. Normalement on peut connecter les équipements au Web à travers les passerelles intelligentes ou bien les intégrées directement. En plus, cette couche doit fournir certaines fonctionnalités comme le management, par exemple le nom abstrait de la ressource, le cache des données et les ordonnancements etc.

Couche de l'agrégation de ressource et support de service intelligent: Cette couche est fondamentalement basée sur notre médiation de service, surtout pour les applications qui ont des processus métier complexes, elle baisse les difficultés de la composition. C'est à dire que

l'on peut réaliser l'agrégation et le mashup des applications et des équipements de l'IoT à travers notre médiation afin d'atteindre la collaboration intersectorielle et interrégionale. Les équipements de l'IoT peuvent obtenir les informations dans le monde physique. Ces informations représentent les activités des personnes selon une situation et un contexte. L'analyse et le traitement de ces informations sémantiques nous aide à fournir aux utilisateurs des services plus intelligents et plus agiles.

6.2.2 Connecter les équipements de IoT à Web : Stratégie de réalisation

Dans l'environnement mentionné, le noyau c'est la passerelle de WMIoT. Cette passerelle fournit la possibilité de connecter les équipements de l'IoT à Web. Le médiateur dans cette passerelle peut cacher la complexité et l'hétérogénéité de web pour qu'on puisse obtenir et manager les données et les ressources du système de capteur. Autrement dit, c'est un pont pour connecter les équipements de l'IoT à Web. En plus, le médiateur est un pont pour connecter les matériels « équipement » de la couche inférieure aux applications métier de la couche du haut.

6.2.2.1 L'architecture de WMIoT

Dans l'architecture de WMIoT illustrée dans la figure 6.5, on doit avoir : pilote de plugin, base de données embarquée, passerelle de base et WS-médiateur.

- Pilote de plugin: la passerelle de WMIoT doit fournir un chemin d'accès à l'internet pour les équipements qui ne supportent pas TCP/IP. Normalement ils ont WiFi, Bluetooth, NFC et Zigbee ou bien la communication en série. Le pilote de plugin peut fournir la fonctionnalité de communication avec ces équipements hétérogènes, en plus de l'analyse et du traitement des protocoles.
- Base de données embarquée : Elle peut cacher et stocker les données de l'équipement qui est connecté à la passerelle.
- Passerelle de base: on a DevREG (*device registrer*), DriREG (*Driver register*) et QM (*Queue management*). DriREG s'occupe du management sur les nouveaux enregistrements des pilotes de plugin. DevREG s'occupe du management des enregistrements des équipements au réseau et puis il va distribuer un identifiant uniforme par exemple <device.id>, ensuite générer l'URI correspondant à l'équipement. Le QM s'occupe du management des ordonnances sur le cache et les données des équipements connectés. En plus, la passerelle de base va fournir un pilote d'API qui invoque le pilote de plugin au WS-médiateur.
- WS-médiateur: le médiateur va fournir un environnement de Web et sa composition à la passerelle. Dans le même temps, il supporte le client Web pour visiter les services fournis par la

passerelle. Le médiateur fournit donc un *framework* pour les services web, surtout il fournit les opérations HTTP GET/PUT/POST/DELETE et un modèle de URI, pour que l'on puisse positionner les ressources d'après les paramètres de *endpoint* de l'URI.

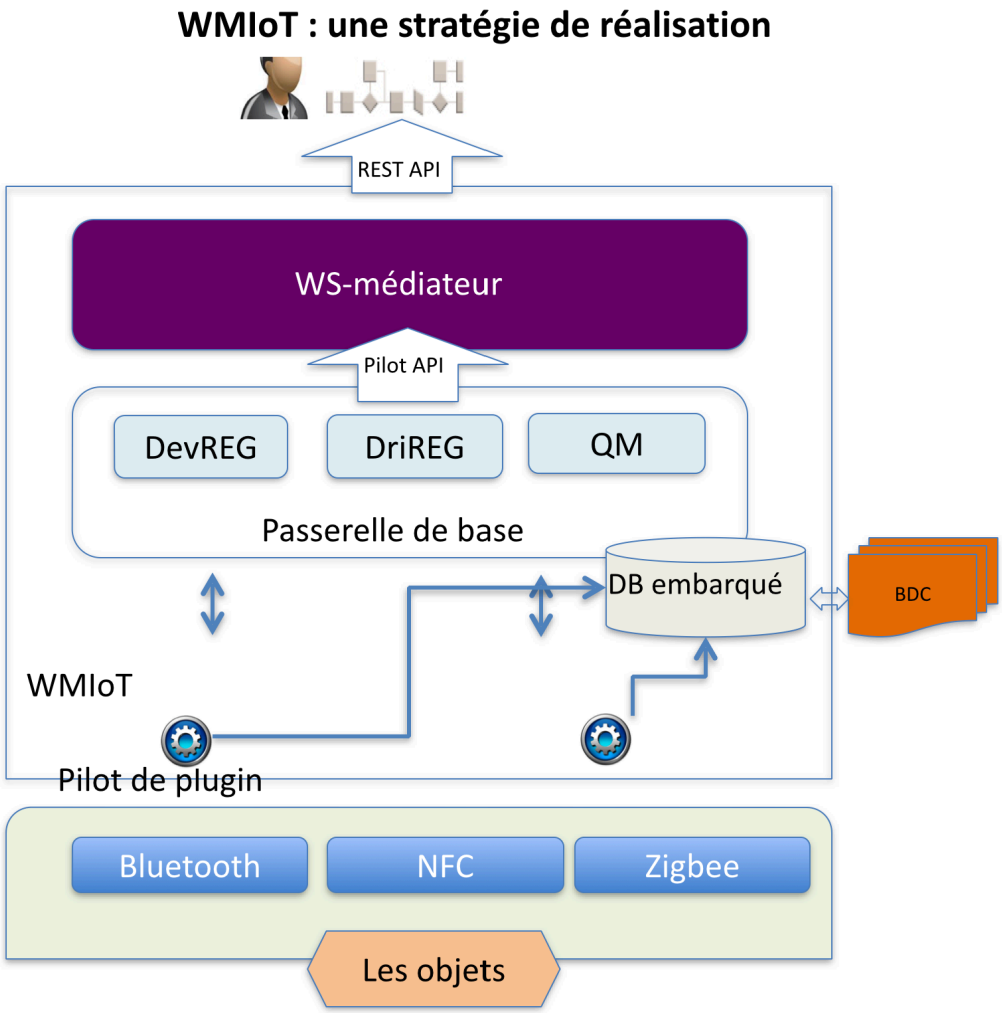


Fig. 6.5 WMIoT : Framework de WMIoT

6.2.2.2 Processus de fonctionnement de WMIoT

Comme illustré dans la figure 6.6, quand un client web veut obtenir les informations de l'équipement externe par l'HTTP, les démarches sont :

Le client web envoie une requête GET HTTP au WMIoT, le WS-médiateur va recevoir la demande puis analyser la requête et la router vers la ressource REST. La passerelle de base va invoquer les pilotes de plugin d'après les informations de routage, le pilote de plugin analyse la demande puis la transfère au format du protocole (Zigbee, Bluetooth, NFC et WiFi etc.) que les équipements supportent. Ensuite il envoie la commande de l'équipement, l'équipement concerné va retourner les données au pilote de plugin correspondant. Le pilote de plugin analyse les informations pour obtenir la valeur de l'équipement et puis il la stoke

dans la DB embarquée et envoie ces données au WS-médiateur. Le médiateur va encapsuler les données comme ressource Web avec le format XML ou JSON et puis les retourner au client web par la réponse HTTP. Le client web va finalement obtenir les informations correspondantes puis traiter la ressource de l'équipement au fur et à mesure.

En plus, lorsque les données sont stockées dans le cache de DB embarqué, donc quand le client web envoie la requête, le WS-médiateur va demander les données au DB directement puis obtenir les données dans le cache.

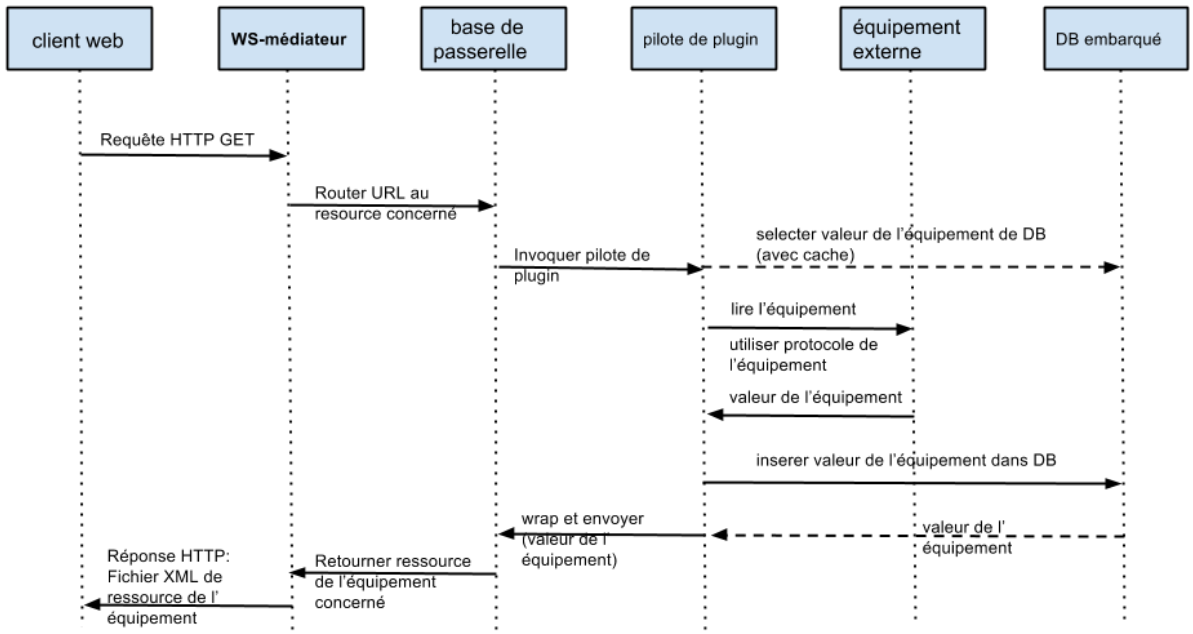


Fig. 6.6 : Processus de client web connecté à l'équipement de l'IoT externe à travers WMIoT

6.2.3 Discussion

L'intégration avec l'Internet : WMIoT est une architecture de l'intégration profonde avec Internet, en particulier avec le Web. Tous les périphériques réseaux sont extraits comme des ressources Web. C'est une bon avantage de la technologie d'héritage du Web, telles que ses caractéristiques standard (TCP/IP, HTTP, service Web, HTML5), déploiement à grande échelle de l'infrastructure Internet (Ethernet), la technologie de recherche sur l'Internet (moteur de recherche), le Web sémantique etc. Les modèles d'affaires sur Internet, tels que web 2.0, e-commerce, la publicité de recherche, services de *Cloud Computing* etc. Ils fournissent un espace élargi pour business modèle de l'IoT.

L'intégration et l'interopérabilité des ressources physiques : Avec WMIoT, une variété de dispositifs hétérogènes peut être intégrée dans le Web, ainsi des tiers peuvent se concentrer sur la façon de l'utilisation de l'interface Web pour la combinaison des ressources de service, sans la nécessité de faire l'accent sur la façon de résoudre l'hétérogénéité de ces dispositifs. En plus, nous avons le protocole HTTP comme l'interface uniforme pour accéder à la couche d'application, nous avons le format de données unifié pout les formats de données structurées (par exemple XML, JSON). Il améliore l'interopérabilité entre les appareils, il a également donné une base pour les combinaisons et les partages entre les ressources.

Nouveau business modèle pour l'industrie de l'IoT: Le fournisseur de service peut accorder plus d'attention à la conception des services et de l'expérience de l'utilisateur, il va développer son propre système matériel et logiciel aux coûts minimum. WMIoT fournit une solution de bout-en-bout pour les applications de l'IoT, tirée par les besoins de l'utilisateur. Cet écosystème permettra aux développeurs individuels et aux fournisseurs de service de développer plus de services et d'applications. Les services seront de plus en plus orientés utilisateur. Le nouveau marché des applications favorisera les fabricants d'équipement de l'IoT de plus en plus disposés à partager et ouvrir leurs services d'infrastructure pour générer des nouvelles valeurs business. La combinaison de service Cloud, l'e-commerce et les média sociaux apporteront de nouvelles réflexions pour le business modèle de l'IoT.

6.3 Conclusion

Dans ce chapitre, on parle tout d'abord de l'aspect de la sécurité de service pour notre médiation. Ensuite, basé sur notre proposition de médiation web dans le chapitre 3 et 4 , on parle de deux extensions, celle de l'architecture du *cloud* et celle de l'architecture de l'IoT. Les systèmes et les applications basées sur notre médiation peuvent utiliser les ressources et les services de *cloud* puisqu'il permet la composition dynamique des services *on-demand*. Dans WMIoT, tout est objet et tous sont vus comme des ressources web, pour que l'on puisse gérer les objets à travers la médiation de web et obtenir un environnement plus intelligent et agile.

Chapitre 7

Conclusion et Perspectives

7.1 Conclusion

Avec le développement rapide des technologies de l'information et le réseau, la recherche de l'interopérabilité est de nos jours une problématique centrale dans un environnement ubiquitaire et pour le *Pervsive Service Computing*. Ce domaine de recherche est favorisé par l'adoption de l'architecture orientée service comme modèle de développement, et particulièrement par les services Web qui combinent les avantages de ce modèle aux langages et technologies développés pour l'Internet. Les services Web ont permis une avancée significative dans la facilité des interactions et les prestations entres les systèmes distribués. Notamment, la composition de service Web est considérée comme un point fort, qui permet de répondre à des requêtes complexes en combinant les fonctionnalités de plusieurs services au sein d'une même composition.

Cependant, les services sont sujets à de nombreuses hétérogénéités, d'autant plus que dans le contexte dynamique et ubiquitaire d'Internet et la mobilité de l'utilisateur, il est difficilement envisageable d'imposer une représentation syntaxique/sémantique unique et partagée par tous les services Web. La représentation locale adoptée par chaque service doit être prise en charge, et les différences de représentation des données doivent être résolues par l'utilisation de mécanismes de médiation qui doivent être au sein de la composition.

C'est dans le but de répondre à ces problématiques que nous avons mené nos recherches. Nos travaux sont orienté vers *user-centric*, et plus particulièrement vers une proposition de médiation *user-centric* pour la composition dynamique de service Web. Nous avons étudié dans un premier temps les travaux existants relatifs à la médiation et à la description et représentation de services Web, afin d'établir notre proposition.

Le but de notre médiation est généralement de composer les ressources externes avec les ressources internes pour créer un nouveau service ou bien un service à « valeur-ajoutée ». Plus précisément, notre médiation *user-centric* est de composer les services exposables de

SOAP (opération-centrique) et de RESTful (ressource-centrique) avec les éléments de service (modèle de service abstrait) pour créer une application Web pour l'utilisateur. On a considéré trois types de composition.

La composition de service SOAP et SOAP : Les services basés sur SOA qui échange des messages en utilisant SOAP. Le SOAP est utilisé pour construire une architecture de systèmes distribués et pour réaliser l'intégration dynamique dans cette architecture. L'adoption de normes ouvertes permet d'avoir des services interopérables. Les systèmes construits avec des langages de programmation différents peuvent facilement être intégrés sur des plates-formes différentes. La composition de service SOAP et SOAP contient donc une série de services qui fournissent les modules basics exigés pour la composition de service standardisé.

La composition de service SOAP et REST : Lorsqu'il y a un grand nombre de services RESTful utilisé par Web 2.0, les applications orientées SOA, ont besoin de processus métier pour son exécution. Cependant, il n'existe actuellement aucun moyen d'être en mesure de soutenir les modes de REST, donc les applications de SOA ne peuvent pas utiliser les ressources du service RESTful. Nous proposons une solution pour intégrer les services RESTful à travers le processus métier, dans ce cas-là, le processus métier peut utiliser le WS-adaptateur pour appeler le service RESTful.

La composition de service REST-REST : une composition de service REST ne doit pas être conçue comme une orchestration de services ni comme une chorégraphie de services. Par contre, une composition de service REST a une structure décentralisée à la manière d'une chorégraphie mais elle fournit des interfaces centralisées à travers une médiation pour initier les instances de composition et pour suivre son statut comme l'orchestration de service.

En plus, notre contribution repose sur l'utilisation de la médiation qui tient compte des préférences fonctionnelles et non-fonctionnelles de l'utilisateur. Notre travail de recherche a abouti à plusieurs propositions, qui forment une architecture de médiation pour services Web composés. Selon les exigences de l'utilisateur, la médiation va retourner à l'utilisateur une orchestration de services exposables ou bien une composition des éléments de service pour application Mashup de l'utilisateur. Dans ces deux cas, l'utilisateur, dans notre médiation architecturale, est considéré non seulement comme un consommateur de service, mais aussi comme un producteur de service.

Grâce à la solution des communautés virtuelles, nous avons pu maintenir la continuité de services, tout en tenant compte des préférences fonctionnelles de l'utilisateur ainsi que de sa QoS demandée. Pour ce faire, nous avons proposé d'avoir le contexte d'adaptation de l'utilisateur en remplaçant tout service dégradé par un service ubiquitaire, ayant la même

fonctionnalité et une QoS équivalente. Afin de supporter cette approche, des communautés virtuelles sont créées pour combiner les services ubiquitaires et des agents de QoS qui sont intégrés dans ces différents services pour détecter toutes dégradations de la QoS ou du fonctionnement de chaque service.

7.2 Perspectives envisagées

Au niveau des études de sécurité, la confiance de service doit être considérée. Traditionnellement, la confiance a été utilisée avec la certification, en référence à une chaine de certificats qui peut être retracée par une autorité de confiance. Il se réfère à la croyance que le code sera exécuté d'une manière qui peut être garantie par une partie externe, avec attestation. Dans le contexte de notre architecture, la confiance se réfère à la relation entre le fournisseur de service et l'utilisateur. La « fiabilité » est un jugement sur la probabilité qu'un service puisse fonctionner comme prévu, y compris la priorité de sécurité, avec l'utilisation de paiement mobile par exemple avec NFC dans le domaine de *Cloud Computing* et l'IoT, la non-réputation de service qui devrait être considérer dans notre médiation de service Web au fur et à mesure.

Au niveau des études de sémantique, en effet, l'utilisation des langages naturels pour créer une composition qui sera une approche idéale pour les utilisateurs. Toutefois, ce type de composition de service est encore loin d'être industrialisé en raison de certaines questions non résolues, comme l'ambiguïté du langage naturel. Ainsi, on a besoin d'un autre travail de recherche sur le domaine de l'ontologie et de l'adoption d'intelligence artificielle pour le processus de création de service et le déploiement de module de *moteur reasoning*.

Au niveau des études de performances il faudrait rassembler les travaux d'ordre informationnel comme « la base de connaissances » et d'ordre non-fonctionnel comme « QoS agent » et effectuer des expérimentations en déroulant des scénarii afin d'évaluer les performances globales et afin de démontrer l'influence positive de nos propositions de gestion de la mobilité sur les performances du démonstrateur.

Dans la nouvelle génération de l'Internet et l'environnement ubiquitaire, il faut permettre à l'utilisateur d'utiliser les données à travers les APIs des services ouvertes, pour que chaque utilisateur ait la possibilité de définir une nouvelle application et l'ajout de nouvelles fonctionnalités pour leur permettre de participer à la construction des applications Web. Au fur et à mesure, les applications d'Internet sont enrichies non seulement par l'entreprise mais aussi par les utilisateurs eux-mêmes.

Liste des Abréviations

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
B2B	Business to Business
B2C	Business to Consumer
BPEL	Business Process Execution Language
CORBA	Common Object Requestor Broker Architecture
CRUD	Create, Read, Update and Delete
DOM	Document Object Model
DSL	Domain-Specific Langage
EAI	Entreprise Application Intergration
EJB	Enterprise Java Bean
ESB	Enterprise Service Bus
E2E QoS	End-to-End QoS
GSM	Global System for Mobile Communications
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Mark-up Language
IDE	Integrated Development Environment
IEEE	Institute of Electrical & Electronic Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IMS	IP Multimedia Subsystem
IM	Instant Messaging
IOPE	Input, Output, Precondition and Effect

JSON	JavaScript Object Notation
LBS	Location Basic Service
LTE	Long Term Evolution
NGN	Next Generation Network
NGI	Next Generation Internet
NGSI	Next Generation Service Interfaces
NLR	Noeud, Lien, Réseau
NFC	Near field communication
O22	Online to Offline
OOP	Object-oriented programming
OASIS	Organization for the Advancement of Structured Information Standards
OWL-S	Semantic Markup for Web Services
OWL	Web Ontology Language
P2P	Peer-to-Peer
QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer
RPC	Remote Procedure Call
RIA	Rich Internet Application
SE	Service Element
SEIB	Service Elements Interconnection Bus
SSH	Secure Shell
SAWSDL	Semantic Annotations for WSDL and XML Schema
SAWSD	Semantic Annotation for WSDL
SIP	Session Initiation Protocol
SLA	Service Level Agreements
SOA	Service Oriented Architecture

SOAP	Simple Object Access Protocol
TTM	Time to Market
TTL	Time-to- Live
UBIS	“User-centric”: uBiquité et Intégration de Services
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VoIP	Voice over IP
VPSN	Virtual Private Service Network
WADL	Web Application Description Language
WSDL	Web Service Description Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WS-CDL	Web Services Chroegraphy Description Language
WLAN	Wireless Local Area Network
W3C	World Wide Web Consortium
WSC	Web Service Composition
XML	eXtensible Markup Language
XML-RPC	eXtensible Markup Language – Remote Procedure Call

Liste des publications

1. **T.Zhang**, K.Chen, "Un système de composition et médiation dynamique de Services Web orienté *user-centric"* (Atelier ARC 2011(CDG MACS), Paris, France)
2. **T.Zhang**, K.Chen, C.Y.Yin, M.Akerma, "An experience of a lightweight user-centric dynamic service composition mechanism", IEEE UIC/USST (Ubiquitous Intelligence and Computing 8th International Conference/Ubiquitous Service Systems and Technologies (USST) Workshop), Sept 2-4, Banff, Canada
3. **T.Zhang**, K.Chen, M.Akerma, J.W.Yan, "A User-Centric WS-Mediator framework for on-the-fly Web Service Composition", IEEE 19th TELFOR 2011,22-24 November 2011,Belgrade, Serbia
4. **T.Zhang**, K.Chen, J.W.Yan, A.Bouhier "A new mediator-based architecture for the dynamic service composition", The Fifth International Conference on Advances in Human-oriented and Personalized Mechanism, Technologies, and Service. Centric 2012, 18-23 November, Lisbon, Portugal.
5. **T.Zhang**, K.Chen, "A User-Centric WS-Mediator framework for on-the-fly Web Service Composition" TELFOR Journal, Vol. 4, No.1, pp(s) 66-71, 2012

Références

ABITRBOUL, S., BENJELLOURN, O., MANOLESCU, I., MILO, T. and WEBER, R. *Active XML: Peer-to-peer data and web services integration*. VLDB Endowment (2002), 1087-1090.

ALCALDE, B., DUBOIS, E., MAUW, S., MAYER, N. and RADOMIROVIĆ, S. *Towards a decision model based on trust and security risk management*. Australian Computer Society, Inc. (2009), 61-70.

ANDRIEUX, A., CZAJKOWSKI, K., DAN, A., KEAHEY, K., LUDWIG, H., NAKATA, T., PRUYNE, J., ROFRANO, J., TUECKE, S. and XU, M. *Web services agreement specification (WS-Agreement)*. (2004).

ARKIN, A. Business process modeling language. *BPMI. org* (2002).

ARKIN, A., ASKARY, S., FORDIN, S., JEKELI, W., KAWAGUCHI, K., ORCHARD, D., POGLIANI, S., RIEMER, K., STRUBLE, S. and TAKACSI-NAGY, P. Web service choreography interface (WSCI) 1.0. *Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems* (2002).

BANERJI, A., BARTOLINI, C., BERINGER, D., CHOPELLA, V., GOVINDARAJAN, K., KARP, A., KUNO, H., LEMON, M., POGOSSIANTS, G. and SHARMA, S. Web services conversation language (wscl) 1.0. *W3C Note* 14 (2002).

BARTALOS, P. Effective automatic dynamic semantic web service composition. *Inf. Sci. and Technol. Bulletin ACM Slovakia* 3, 1 (2011), 61-72.

BARTOLETTI, M., DEGANO, P. and FERRARI, G. L. *Enforcing secure service composition*. IEEE (2005), 211-223.

BERARDI, D., CALVANESE, D., DE GIACOMO, G., LENZERINI, M. and MECELLA, M. *A foundational vision of e-services*, Springer2004).

BERNERS-LEE, T., FIELDING, R. and FRYSTYK, H. *Hypertext transfer protocol--HTTP/1.0*, May1996).

BERTOLI, P., PISTORE, M. and TRAVERSO, P. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence* 174, 3 (2010), 316-361.

BOX, D., EHNEBUSKE, D., KAKIVAYA, G., LAYMAN, A., MENDELSOHN, N., NIELSEN, H. F., THATTE, S. and WINER, D. *Simple object access protocol (SOAP) 1.1*2000).

BRNSTED, J., HANSEN, K. M. and INGSTRUP, M. Service composition issues in pervasive computing. *Pervasive Computing, IEEE* 9, 1 (2010), 62-70.

BUCCHIARONE, A. and GNESI, S. *A survey on services composition languages and models*. (2006), 51.

BUYYA, R., YEO, C. S., VENUGOPAL, S., BROBERG, J. and BRANDIC, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems* 25, 6 (2009), 599-616.

CANFORA, G., DI PENTA, M., ESPOSITO, R. and VILLANI, M. L. *An approach for QoS-aware service composition based on genetic algorithms*. ACM (2005), 1069-1075.

CETIN, S., ILKER ALTINTAS, N., OGUZTUZUN, H., DOGRU, A. H., TUFEKCI, O. and SULOGLU, S. *Legacy migration to service-oriented computing with mashups*. IEEE (2007), 21-21.

CHINNICI, R., MOREAU, J.-J., RYMAN, A. and WEERAWARANA, S. Web services description language (wsdl) version 2.0 part 1: Core language. *W3C Recommendation* 26 (2007).
CHRISTENSEN, E., CURBERA, F., MEREDITH, G. and WEERAWARANA, S. *Web services description language (WSDL) 1.1*2001).
CUNNINGHAM, H. C. Reflexive metaprogramming in Ruby: tutorial presentation. *Journal of Computing Sciences in Colleges* 22, 5 (2007), 145-146.
CURBERA, F., DUFTLER, M., KHALAF, R., NAGY, W., MUKHI, N. and WEERAWARANA, S. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE* 6, 2 (2002), 86-93.
CURBERA, F., LEYMANN, F., STOREY, T., FERGUSON, D. and WEERAWARANA, S. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*, Prentice Hall PTR Englewood Cliffs2005).
DE BRUIJN, J., BUSSLER, C., DOMINGUE, J., FENSEL, D., HEPP, M., KIFER, M., KÖNIG-RIES, B., KOPECKY, J., LARA, R. and OREN, E. Web service modeling ontology (wsmo). *Interface* 5 (2006), 1.
DEY, A. K. Understanding and using context. *Personal and ubiquitous computing* 5, 1 (2001), 4-7.
DRYDEN, I. and SESSION, I. http://www. stat. sc. edu/~ dryden/course/ild-ch-04. pdf. (1998).
DUMAS, M., SPORK, M. and WANG, K. *Adapt or perish: Algebra and visual notation for service interface adaptation*, Springer2006).
DUSTDAR, S. and SCHREINER, W. A survey on web services composition. *International Journal of Web and Grid Services* 1, 1 (2005), 1-30.
EL MALIKI, T. and SEIGNEUR, J.-M. *A survey of user-centric identity management technologies*. IEEE (2007), 12-17.
ERL, T. *Service-oriented architecture*, Prentice Hall New York2005).
FALLSIDE, D. C. and WALMSLEY, P. XML schema part 0: primer second edition. *W3C recommendation* (2004).
FAN, G., YU, H., CHEN, L. and LIU, D. *Aspect Oriented Approach to Building Secure Service Composition*. IEEE (2010), 176-185.
FERGUSON, D. F., LOVERING, B., STOREY, T. and SHEWCHUK, J. *Secure, reliable, transacted web services: Architecture and composition*. Technical report, MSDN Library (2003).
FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P. and BERNERS-LEE, T. *Hypertext transfer protocol–HTTP/1.1*, RFC 2616, June1999).
FIELDING, R. T. *Architectural styles and the design of network-based software architectures*, University of California2000).
FOWLER, M. Language workbenches: The killer-app for domain specific languages. (2005).
FRISCH, A. *Essential System Administration: Tools and Techniques for Linux and Unix Administration*, O'reilly2009).
GRAY, J. and SZALAY, A. *The world-wide telescope, an archetype for online science*. Technical Report MSR-TR-2002-75, Microsoft Research, June 2002, http://research. microsoft. com/research/pubs/view. aspx (2002).
GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies* 43, 5 (1995), 907-928.
HADLEY, M. J. Web application description language (WADL). (2006).

HIRSCH, F. Web Services Security SOAP Messages with Attachments (SwA) Profile 1.1. *OASIS Standard* (2006).
HUNG, S.-H., SHIEH, J.-P. and LEE, C.-P. Migrating Android applications to the cloud. *International Journal of Grid and High Performance Computing (IJGHPC)* 3, 2 (2011), 14-28.
ISSARNY, V., GEORGANTAS, N., HACHEM, S., ZARRAS, A., VASSILIADIST, P., AUTILI, M., GEROSA, M. A. and HAMIDA, A. B. Service-oriented middleware for the Future Internet: state of the art and research directions. *Journal of Internet Services and Applications* 2, 1 (2011), 23-45.
KENNEDY, M., LLEWELLYN-JONES, D., SHI, Q. and MERABTI, M. *A Framework for Providing a Secure System of Systems Composition*. (2011).
KLINT, P., VAN DER STORM, T. and VINJU, J. *Rascal: A domain specific language for source code analysis and manipulation*. IEEE (2009), 168-177.
KOPECKY, J., VITVAR, T., BOURNEZ, C. and FARRELL, J. Sawsdl: Semantic annotations for wsdl and xml schema. *Internet Computing, IEEE* 11, 6 (2007), 60-67.
LANGENHAN, D. *VMware vCloud Director Cookbook*, Packt Publishing Ltd2013).
LAWTON, G. New ways to build rich internet applications. *Computer* 41, 8 (2008), 10-12.
LENK, A., KLEMS, M., NIMIS, J., TAI, S. and SANDHOLM, T. *What's inside the Cloud? An architectural map of the Cloud landscape*. IEEE Computer Society (2009), 23-31.
LI, X., FAN, Y., MADNICK, S. and SHENG, Q. Z. A pattern-based approach to protocol mediation for web services composition. *Information and Software Technology* 52, 3 (2010), 304-323.
LIU, X., HUI, Y., SUN, W. and LIANG, H. *Towards service composition based on mashup*. IEEE (2007), 332-339.
LODDERSTEDT, T., BASIN, D. and DOSER, J. *SecureUML: A UML-based modeling language for model-driven security*, Springer2002).
MAIGRE, R. *Survey of the tools for automating service composition*. IEEE (2010), 628-629.
MARTIN, D., BURSTEIN, M., HOBBS, J., LASSILA, O., MCDERMOTT, D., MCILRAITH, S., NARAYANAN, S., PAOLUCCI, M., PARSIA, B. and PAYNE, T. OWL-S: Semantic markup for web services. *W3C member submission* 22 (2004), 2007-2004.
MAXIMILIEN, E. M., WILKINSON, H., DESAI, N. and TAI, S. *A domain-specific language for web apis and services mashups*, Springer2007).
MCILRAITH, S. and SON, T. C. Adapting golog for composition of semantic web services. *KR* 2 (2002), 482-493.
MEDJAHED, B. and BOUGUETTAYA, A. *Context-based Matching for Semantic Web Services*, Springer2011).
MEDJAHED, B., BOUGUETTAYA, A. and ELMAGARMID, A. K. Composing web services on the semantic web. *The VLDB Journal* 12, 4 (2003), 333-351.
MENDLING, J. and HAFNER, M. *From inter-organizational workflows to process execution: Generating BPEL from WS-CDL*. Springer (2005), 506-515.
MILANOVIC, N. and MALEK, M. Current solutions for web service composition. *Internet Computing, IEEE* 8, 6 (2004), 51-59.
MILLER, M. *Cloud computing: Web-based applications that change the way you work and collaborate online*, Que publishing2008).
MURUGESAN, S. Understanding Web 2.0. *IT professional* 9, 4 (2007), 34-41.
NASSAR, R. and SIMONI, N. *Cloud Management Architecture in NGN/NGS Context-QoS-awareness, Location-awareness and Service Personalization*. (2011), 629-635.
NASSAR, R. and SIMONI, N. *E2E mobility management in ubiquitous and ambient networks context*, Springer2012).

NEUMAN, C., HARTMAN, S., YU, T. and RAEBURN, K. The Kerberos network authentication service (V5). (2005).
NEWCOMER, E. and LOMOW, G. *Understanding SOA with web services (independent technology guides)*, Addison-Wesley Professional2004).
O'REILLY, T. What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, 1 (2007), 17.
O'REILLY, T. *What is web 2.0*, O'Reilly2009).
PART, X. S. 2: Datatypes. *W3C Recommendation* 2 (2001).
PAUTASSO, C., ZIMMERMANN, O. and LEYMANN, F. *Restful web services vs. big'web services: making the right architectural decision*. ACM (2008), 805-814.
PELTZ, C. Web services orchestration and choreography. *Computer* 36, 10 (2003), 46-52.
PESSOA, R. M., SILVA, E., VAN SINDEREN, M., QUARTEL, D. A. and PIRES, L. F. *Enterprise interoperability with SOA: a survey of service composition approaches*. IEEE (2008), 238-251.
PFEFFER, H., LINNER, D. and STEGLICH, S. *Dynamic adaptation of workflow based service compositions*, Springer2008).
PORTCHELVI, V., VENKATESAN, V. P. and SHANMUGASUNDARAM, G. Achieving Web Services Composition–a Survey. *Software Engineering* 2, 5 (2012), 195-202.
RAGGETT, D., LE HORS, A. and JACOBS, I. HTML 4.01 Specification. *W3C recommendation* 24 (1999).
REZNIK, J., RITTER, T., SCHREINER, R. and LANG, U. Model driven development of security aspects. *Electronic Notes in Theoretical Computer Science* 163, 2 (2007), 65-79.
RICHARDSON, L. and RUBY, S. *RESTful web services*, O'Reilly2008).
ROLLAND, C. and KAABI, R.-S. An intentional perspective to service modeling and discovery. *Relation* 1, 1 (2007).
SIRIN, E. *Automated Composition of Web Services using AI Planning Techniques*, Research Report, www. cs. umd. edu/Grad/scholarlypapers/papers/aiplanning. pdf,(accessed Sept 12, 2008)2010).
SRIVASTAVA, B. and KOEHLER, J. *Web service composition-current solutions and open problems*. (2003), 28-35.
STROGATZ, S. H. Exploring complex networks. *Nature* 410, 6825 (2001), 268-276.
TAI, S., KHALAF, R. and MIKALSEN, T. *Composition of coordinated web services*, Springer2004).
TANG, M. and AI, L. *A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition*. IEEE (2010), 1-8.
TER BEEK, M., BUCCHIARONE, A. and GNESI, S. *A survey on service composition approaches: From industrial standards to formal methods*. Technical Report 2006-TR-15 (2006).
TER BEEK, M. H., BUCCHIARONE, A. and GNESI, S. Formal methods for service composition. *Annals of Mathematics, Computing & Teleinformatics* 1, 5 (2007), 1-10.
THATTE, S. XLANG: Web services for business process design. *Microsoft Corporation* (2001), 13.
TUECKE, S., WELCH, V., ENGERT, D., PEARLMAN, L. and THOMPSON, M. *Internet X. 509 public key infrastructure (PKI) proxy certificate profile*. RFC 3820 (Proposed Standard) (2004).
VALLÉE, M., RAMPARANY, F. and VERCOUTER, L. *Dynamic service composition in ambient intelligence environments: a multi-agent approach*. Citeseer (2005), 1-6.

VAQUERO, L. M., RODERO-MERINO, L., CACERES, J. and LINDNER, M. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review* 39, 1 (2008), 50-55.
VON HAGEN, W. *Professional xen virtualization*, John Wiley & Sons2008).
VUKOVIC, M., KOTSOVINOS, E. and ROBINSON, P. *Application development powered by rapid, on-demand service composition*. IEEE (2007), 88-98.
WALDO, J., WYANT, G., WOLLRATH, A. and KENDALL, S. *A note on distributed computing*, Springer1997).
WANG, S.-L. and WU, C.-Y. Application of context-aware and personalized recommendation to implement an adaptive ubiquitous learning system. *Expert Systems with applications* 38, 9 (2011), 10831-10838.
WEISE, T., BLAKE, M. B. and BLEUL, S. *Semantic Web Service Composition: The Web Service Challenge Perspective*, Springer2014).
WIEDERHOLD, G. Mediators in the architecture of future information systems. *Computer* 25, 3 (1992), 38-49.
WIEDERHOLD, G. *Intelligent integration of information*. ACM (1993), 434-437.
WOHED, P., VAN DER AALST, W. M., DUMAS, M. and TER HOFSTEDE, A. H. *Analysis of web services composition languages: The case of bpel4ws*, Springer2003).
YE, Z., ZHOU, X. and BOUGUETTAYA, A. *Genetic algorithm based QoS-aware service compositions in cloud computing*. Springer (2011), 321-334.
YEE, R. *Pro Web 2.0 mashups: remixing data and web services*, Apress2008).
YU, J., BENATALLAH, B., CASATI, F. and DANIEL, F. Understanding mashup development. *Internet Computing, IEEE* 12, 5 (2008), 44-52.
ZAHREDDINE, W. and MAHMOUD, Q. H. *A framework for automatic and dynamic composition of personalized Web services*. IEEE (2005), 513-518.
ZHANG, Q., CHENG, L. and BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications* 1, 1 (2010), 7-18.
ZHANG, R., ARPINAR, I. B. and ALEMAN-MEZA, B. *Automatic Composition of Semantic Web Services*. (2003), 38-41.
ZHAO, H. and DOSHI, P. A hierarchical framework for logical composition of web services. *Service Oriented Computing and Applications* 3, 4 (2009), 285-306.
ZHAO, L., REN, Y., LI, M. and SAKURAI, K. Flexible service selection with user-specific QoS support in service-oriented architecture. *Journal of Network and Computer Applications* 35, 3 (2012), 962-973.