REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Mentouri de Constantine
Faculté des sciences de l'ingénieur
Département Informatique

THÈSE

Présentée par

Mme Hemam née Hioual Ouassila

Pour obtenir le diplôme de :

Doctorat en sciences de l'université Mentouri de Constantine

Spécialité:

« Informatique »

Composition sémantique des services web dans un contexte d'ebXML

Thèse soutenue le : 09/10/2011

Devant le jury composé de :

Nacereddine Zarour (Pr)	Univ. Mentouri de Constantine	Président
Zizette Boufaida (Pr)	Univ. Mentouri de Constantine	Rapporteur
Ramdane Maamri (MC)	Univ. Mentouri de Constantine	Examinateur
Okba Kazar (MC)	Univ.Mohamed Khider de Biskra	Examinateur
Amar Balla (Pr)	Ecole Supérieure d'Informatique, Alger	Examinateur

Ma chère famille ***Maman***, ***Papa***, *mes* ***sœurs*** *et* ***frères***, *mon petit* ***Kamel Ouassim*** *chéri, mon mari* ***Sofiane*** *et mes adorables neveux et nièce* ***Louai Eddine*** *et* ***Imène*** *pour m'avoir fourni un cadre agréable et chaleureux qui m'a toujours aidé à me ressourcer et qui m'a permis quand mes poèmes devenaient lugubres de m'apercevoir que la vie est belle.*

Ma belle famille….

À tous mes amis, ceux que je vois encore et ceux que le vent a emmené vers d'autres rives ….

REMERCIEMENTS

On passe tous par des moments de doute, d'incertitude, de petites et grandes crises …et on ne peut les dépasser que par le soutien de tous ceux qui nous relèvent de nos petites peines. Pour cela et pour bien d'autres raisons je tiens à remercier :

Monsieur **MAHMOUD BOUFAIDA**, Professeur à l'université Mentouri de Constantine, de m'avoir accueillie dans son équipe SIBC du laboratoire LIRE, d'être toujours disponible en m'offrant ses conseils méthodologiques.

Madame **ZIZETTE BOUFAIDA**, Professeur à l'Université Mentouri de Constantine, ma directrice de thèse, qui m'a initiée à la recherche dans un domaine qui m'a toujours motivée. Méticuleuse et perfectionniste, toujours disponible, elle m'a prodigué des conseils inestimables, dans tous les domaines, tout au long de ma thèse. Ses commentaires sur mon travail m'ont chaque fois permis de m'orienter dans les bonnes directions. Je tiens à lui exprimer toute ma reconnaissance et gratitude.

Les membres du jury qui m'ont fait l'honneur de bien vouloir évaluer mon travail, et plus précisément :

Monsieur **NACEREDDINE ZAROUR**, Professeur à l'Université Mentouri de Constantine, pour l'honneur qu'il m'a fait, en acceptant la présidence de ce jury.

Monsieur **AMAR BALLA**, Professeur à l'Ecole Supérieur de l'Informatique, Alger, Monsieur **OKBA KAZAR**, Maître de Conférence à l'Université Mohamed Khider de Biskra, et Monsieur **RAMDANE MAAMRI**, Maître de Conférence à l'Université Mentouri de Constantine qui m'ont fait l'honneur d'accepter de faire partie de mon jury en tant qu'examinateurs.

L'équipe SIBC du laboratoire LIRE, au sein de laquelle cette thèse est née et a évoluée, qui m'a procuré les conditions de travail les plus favorables. C'est une seconde famille que j'ai trouvée au sein de cette équipe.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

1. Contexte du travail

L'Architecture Orientée Services (AOS ou SOA – *Service-Oriented Architecture* en anglais) permet aux concepteurs de systèmes d'information d'organiser un ensemble de logiciels isolés en un ensemble de services interconnectés, accessibles par une interface et des protocoles standard [87]. Un *service web* est une unité discrète, publiée sur le web, qui remplit une collection de tâches répondant au(x) même(s) objectif(s). L'AOS est un paradigme architectural qui peut être utilisé pour concevoir des infrastructures permettant aux *clients* et aux *fournisseurs* d'échanger des services, malgré la disparité des domaines, des technologies, et des fournisseurs.

L'intérêt des services web est de permettre à une entreprise d'exporter au travers du réseau internet ses compétences et son savoir-faire, d'interagir avec ses partenaires, de rechercher de nouveaux marchés et de nouveaux supports de vente. En juillet 2002, Amazon.com a ainsi été l'une des toutes premières à obtenir un fort écho médiatique en rendant sa base de données d'articles accessible par un service web. Ces architectures orientées services qui trouvent leurs origines dans l'informatique distribuée en prenant le réseau internet comme plate-forme d'exécution de composants logiciels interopérables conduisent à des interactions complexes à grande échelle et des défis nouveaux. En effet, contrairement aux interfaces de programmation (Application Programming Interface, API) « classiques », les services web sont conçus pour découvrir et invoquer d'autres services et tirent leur versatilité de leurs interfaces qui sont des abstractions n'imposant aucune contrainte en matière de mise en œuvre, e.g., langage de programmation, système d'exploitation, etc.

Actuellement, les services web reposent principalement sur des standards XML (*cf.* Figure I.1) : WSDL (Web Services Description Language) permet une description syntaxique des services en termes d'entrées, sorties ; OWL-S [25] a pour objectif de faire une description « sémantique », c'est-à-dire explicitant le « *profile* » du service (quelles sont les informations nécessaires à l'exécution du service ? quelles sont les informations renvoyées ?), son « *process model* » (comment fonctionne le service ?) et son « *grounding* » (de quelle façon le service doit-il être utilisé ?). La convergence entre web sémantique et services web [9] a pour but d'augmenter l'expressivité des descriptions et de rendre plus efficace la gestion, la découverte, la composition et l'invocation des services au travers d'un protocole de communication et d'un répertoire de services UDDI (Universal Description, Discovery and Integration). Ce protocole permet à un « fournisseur » d'enregistrer son service et à un « consommateur» de trouver le service adéquat. Finalement, les services web s'appuient sur SOAP (Simple Object Access Protocol), un protocole d'échange de messages entre services fondé sur HTTP.

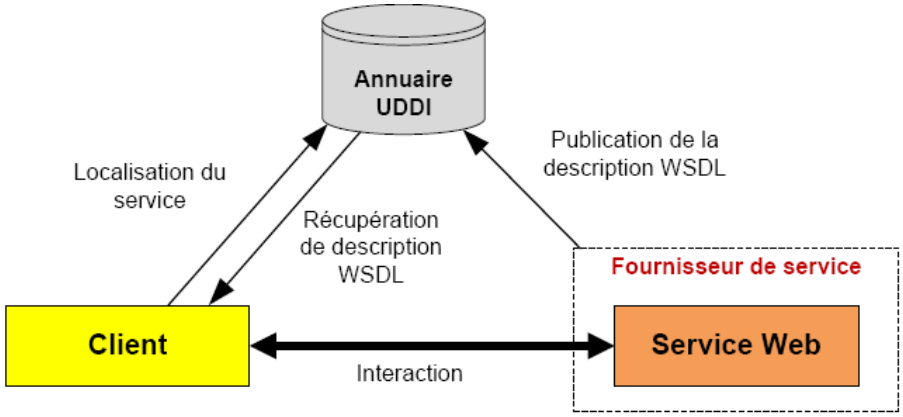


Figure I.1. *Déploiement, recherche et invocation de services web.*

Mais pour exploiter au maximum les avantages de ce nouveau type d'applications, la communauté des services web doit se doter d'outils donnant la possibilité de pouvoir assembler différents services entre eux. Cet assemblage a pour but, d'effectuer des tâches qu'un simple service web ne saurait résoudre. C'est la *composition des services web*. Le résultat de cette composition est un enchaînement de services qui permet de définir la façon dont les données calculées par les uns sont consommées par les autres.

Il existe deux grands courants de composition de service web. Le premier, principalement utilisé par le monde industriel, permet de définir des procédés réutilisables sous la forme d'orchestration et de chorégraphie en utilisant des langages de gestion de procédés. Le second courant cherche à définir un enchaînement dynamique de services web permettant de résoudre un but spécifique fourni par l'utilisateur, en tenant compte de ses préférences. La composition dynamique, qu'aucune technologie n'implémente à l'heure actuelle, peut se justifier par les points suivants :

- du fait de l'évolution permanente de l'offre de services et de leurs capacités, une description statique de la composition est difficile à maintenir ;
- elle permet d'adapter la composition aux attentes de l'utilisateur et donc de ne pas être contraint par une composition à priori ;
- elle permet de faire d'Internet une zone de compétences composables et évolutives et s'inscrit ainsi dans le cadre des SAAS (Software As A Service).

Notre objectif est de proposer un modèle de composition s'appuyant sur la planification multi-agent. En effet, les caractéristiques d'un service Web sont proches de celles d'un agent dans le cadre de la planification multi-agent : il est *autonome* et *peut communiquer* avec d'autres services web. De plus, nous verrons qu'il est possible, en lui ajoutant une description sémantique de ses fonctionnalités, de raisonner sur ses capacités.

La planification multi-agent est une extension du domaine de la planification en intelligence artificielle aux systèmes multi-agent. En planification classique on cherche, par exemple, à contrôler les mouvements d'un robot. Alors qu'en planification multi-agent on s'intéresse à la coordination des mouvements entre chaque robot : la planification est distribuée.

Il existe deux approches de la planification distribuée. Dans la première, chaque agent produit un plan. Le but est de coordonner ces différents plans afin d'éviter les conflits : c'est la coordination de plans. Dans la seconde approche, le but est de permettre aux agents de co-construire un plan en tenant compte des compétences de chaque agent au fur et à mesure de la co-construction : c'est la synthèse dialectique de plans. Dans cette approche, les agents ont la possibilité d'émettre des hypothèses afin de ne pas bloquer le processus dialectique lorsqu'ils se trouvent dans une situation de blocage. Par exemple, un agent peut sortir un carton d'une pièce à condition que la porte de la pièce soit ouverte. Si l'agent sait que la porte est fermée ou bien qu'il ne dispose pas

de cette information, il émettra comme hypothèse que la porte est ouverte. Cette hypothèse formera un nouveau but qui devra être résolu par un autre agent, etc.

La méthode de composition dynamique de services web que nous proposons s'appuie sur la synthèse dialectique de plans. Les agents managers (services web) proposent leurs compétences à l'agent compositeur afin de réaliser les objectifs fixés par l'utilisateur, en émettant, si besoin, des hypothèses sur les données manquantes.

2. Problématique et objectifs

Comme nous avons déjà mentionné dans la section précédente, la composition de services Web a pour but de produire une description spécifiant une séquence d'appels à des services ainsi que la façon dont ces services sont liés entre eux, dans le but de résoudre un objectif donné. Cette opération se déroule en trois étapes :

1. les services Web sont recherchés et sélectionnés à partir d'un annuaire UDDI (dans notre cas du registre ebXML) en fonction des besoins à réaliser ;
2. la composition est effectuée en utilisant la description sémantique des services sélectionnés ;
3. une description du service composite, c'est-à-dire, l'enchaînement des appels aux services sélectionnés, est créée.

Puisque nous nous sommes intéressés, dans un premier travail de recherche, à la négociation des paramètres des CPP d'ebXML [107], nous avons choisi les services Web du registre ebXML comme continuité à notre domaine de recherche.
L'objectif de notre travail est d'ajouter un composant à la spécification fonctionnelle d'ebXML dont le rôle est de *Combiner, sémantiquement, des services Web pour répondre aux besoins de l'entreprise cliente.*

L'ensemble des services web des entreprises utilisant ebXML pour participer à des marchés globaux, est regroupé dans un registre ou répertoire (Registry/Repository). Le problème de la composition des services Web dans un contexte ebXML apparaît: lors de la première phase du scénario de collaboration entre deux partenaires commerciaux. Quand une entreprise cherchant un partenaire potentiel consulte le registre, le service demandé avec certains paramètres peut ne

pas exister mais en même temps, il peut exister des services, qui combinés ensemble, peuvent répondre à ses besoins. Pour résoudre ce problème, nous proposons une architecture à base d'agents, permettant de composer des services web, conformément à la spécification fonctionnelle d'ebXML. Cette architecture définit plusieurs niveaux de responsabilité et utilise comme modèle de composition *« la planification sous hypothèse avec synthèse dialectique de plans »*. Le modèle de composition dynamique de services web que nous proposons permet de passer de l'état initial à l'état final afin de produire le plan solution, selon le domaine de planification. Les services web (les agents manager) proposent leurs compétences à l'agent compositeur afin de réaliser les objectifs fixés par ce dernier, et de ce fait par l'utilisateur, en émettant, si besoin, des hypothèses sur les données manquantes. L'approche que nous développons concerne uniquement la composition : nous ne traitons pas des étapes en amont et en aval de la composition, i.e., la recherche de services web et l'invocation des services intervenant dans cette composition.

3. Organisation de la thèse

Le manuscrit est structuré en six chapitres et une conclusion générale.

1. Dans le chapitre 1, nous introduisons les concepts de services web et de web sémantique, et nous présentons comment le web sémantique a été intégré dans la communauté des services web (par le biais des langages sous-jacents au Web sémantique, tels que RDF et OWL) et comment il a apporté, entre autres, un niveau sémantique dans la description des services web (par le biais des langages OWL-S et SAWSDL).
2. Le chapitre 2 est consacré à un état de l'art sur les travaux liés à la composition de services. D'abord, nous proposons une catégorisation selon laquelle nous décrivons et analysons les approches existantes de composition de services web, à savoir les approches dites *statiques* et celles dites *dynamiques*. Nous analysons, ensuite, les approches présentées et concluons ce chapitre par les limites que les travaux existants en composition de services présentent pour la modélisation du problème de la chorégraphie dynamique de services (composition dynamique de services).
3. Nous consacrons le chapitre 3 à un état de l'art sur les travaux liés au mariage de la composition de services avec la coordination multi-agent en général et la planification en

particulier. Nous les présentons en deux temps. D'abord, nous commençons par présenter les limites des services web, et, avant de présenter la pertinence de la combinaison des technologies agent et services web, nous faisons une brève présentation des concepts clé des systèmes multi-agent ainsi que les classes de modèles de coordination. Ensuite, nous présentons un panorama des modèles de coordination dans les systèmes multi-agent ainsi que les travaux les implémentant pour le problème de la composition de services. Et finalement, nous consacrons la dernière partie de ce chapitre aux problèmes liés à la coordination de plans du fait que nous nous intéressons aux travaux liés à ce type de coordination multi-agent.

4. Nous donnons une vue d'ensemble de notre modèle de composition et nous présentons notre architecture combinant technologies services web et systèmes multi-agent, dans le quatrième chapitre. Il présente les motivations pour le choix de l'approche agent, la spécification de l'architecture en termes de structures d'agents, de comportements des agents, ainsi que la communication inter-agents.
5. Dans le chapitre 5 de ce manuscrit, nous proposons notre modèle de composition de services web basé sur la planification multi-agent, et sur l'architecture présentée dans le chapitre précédent, dans lequel les services sont vus comme des agents manager capables d'échanger des propositions et des contre-propositions, avec l'agent compositeur, et d'émettre des hypothèses.
6. Le chapitre 6 illustre notre modèle par une étude de cas de composition de services web aboutissant à l'organisation d'un voyage. Cette étude a mis en avant le dialogue de trois agents dans le but de proposer un plan d'exécution des services qui répond favorablement à la requête de l'utilisateur.
7. Enfin nous terminons ce manuscrit par une conclusion générale qui récapitule les travaux réalisés et propose quelques travaux en perspective.

CHAPITRE 1

SERVICES WEB ET WEB SEMANTIQUE

L'évolution des réseaux informatiques combinée à celle des systèmes informatiques a permis la mise en œuvre de systèmes distribués de plus en plus performants et le développement d'applications de toute sorte s'appuyant sur ces infrastructures. Les systèmes informatiques distribués recouvrent un champ très vaste dont font partie les grilles de calcul, le stockage d'informations dans des réseaux de pairs, le déploiement d'applications web, etc.

L'évolution de plateformes à composants distribués (ex. les EJB de Sun ou encore .NET de Microsoft) a donné naissance à un nouveau mode de développement logiciel appelé SOC.

> ***SOC**, acronyme de Service Oriented-Computing, désigne ce paradigme de programmation émergeant, pour le développement de systèmes d'informations distribués, dans lequel les concepts de distribution, d'ouverture, de messagerie asynchrone et de faible couplage tiennent un rôle primordial [88], [90].*

Dans ce contexte, les applications sont construites à partir de services individuels exposant leurs fonctionnalités à travers des registres et s'abstrayant complètement de leur implémentation sous-jacente. Leurs interfaces publiées peuvent alors être recherchées et invoquées par des utilisateurs ou d'autres services. L'avantage de SOC est qu'il permet de créer des systèmes d'information en agrégeant des services individuels.

1. SOA ou les architectures orientées services

Les besoins de SOC pour la satisfaction des précédents cas d'usage peuvent être plus facilement satisfaits à travers une architecture répondant aux propriétés requises. Une telle architecture est appelée SOA ou architecture orientée services.

> ***Une architecture orientée services**, calque de l'anglais Service Oriented Architecture, ou SOA, est une architecture logicielle s'appuyant sur un ensemble de services simples. Son objectif est de décomposer une fonctionnalité en un ensemble de fonctions basiques (les services) fournies par des composants et de décrire le schéma d'interaction entre ces services [86].*

Dans une SOA, les services peuvent communiquer entre eux. Cette communication peut soit consister en un simple passage de données ou impliquer la coordination de deux ou plusieurs services pour l'accomplissement d'une activité. Il faut alors définir des moyens pour connecter les services entre eux.

Beaucoup considèrent que la première SOA est apparue avec l'utilisation des DCOM[1] ou ORB[2] basés sur les spécifications de CORBA.

Cependant, la principale différence entre SOA et les autres architectures distribuées, telles que CORBA, est le faible couplage des services par l'expression des "interactions" entre les services. En effet, le point de départ des spécifications SOA a été le besoin croissant de sélectionner et d'intégrer, au fil de l'eau, des services hétérogènes et inter-organisationnels, aussi bien à travers le web qu'au sein d'environnements intelligents et pervasifs. Toutefois, là où CORBA se concentre sur les objets pour créer un environnement de programmation distribué, SOA se focalise sur les documents et l'interopératibilité des processus métiers entre partenaires et consommateurs à travers l'Internet moyennant des transactions à long terme (et pas seulement des transactions à court terme).

1. DCOM pour Distributed Component Object Model.
2. ORB pour Object Request Brokers.

2. Services web

Afin de mettre en œuvre une SOA, il est essentiel d'avoir une compréhension claire de ce qu'est un service web.

À l'origine, la technologie des services web a été initiée par IBM et Microsoft, puis en partie normalisée sous l'égide du W3C[3], l'organisme chargé de standardiser les évolutions du web. Elle est maintenant acceptée par l'ensemble des acteurs de l'industrie informatique, faisant des services web une technologie révolutionnaire [55].

Un service web est concrètement un ensemble de fonctionnalités exposées sur un réseau. Ces fonctionnalités sont bien définies, auto-contenues et ne dépendent ni du contexte ni de l'état d'autres services.

> *Un **service web** est un composant logiciel accessible à travers des intranets, des extranets et l'Internet moyennant des technologies web et un système standard de messagerie basé sur XML [51].*

L'avantage des services web, par rapport aux autres approches de systèmes distribués tel que RMI (pour Remote Method Invocation), réside dans leur support des pare-feux, mais surtout dans leur articulation autour d'XML (standard promulgué par le W3C), ce qui leur procure l'avantage d'être non propriétaire et ainsi multiplateforme.

Les services web varient en complexité de simples opérations telles que la vérification en ligne du solde d'un compte bancaire, des services domotiques d'enregistrement et de diffusion de programmes audiovisuels, des services en ligne d'agences de voyages, à des systèmes complexes de type CRM (customer relationship management) [56], ERP (enterprise resource planning) [57], etc.

---

3. Le World Wide Web Consortium, abrégé W3C, est un consortium fondé en octobre 1994, par Tim berners Lee, pour promouvoir la compatibilité des technologies du World Wide Web telles que HTML, XHTML, XML, CSS, PNG, SVG et SOAP. Le W3C n'émet pas des normes, mais des recommandations.

Techniquement parlant

Les travaux dans le domaine des services web ont rendu possible la *publication*, la *découverte* et l'*invocation* d'applications à travers le web. Ainsi, les services web sont actuellement basés sur la triade de fonctions illustrées sur la figure 1.1. Leur architecture repose sur les principes et standards suivants :

– Un service web est identifié par une chaîne de caractère appelée URI (Unified Resource Identifier), dont la syntaxe respecte une norme d'Internet mise en place par le W3C.

– L'interface publique du service et les liaisons sont définies et décrites en XML, généralement en WSDL [34]. Un document WSDL, pour Web Service Description Language, est une description basée sur XML indiquant le protocole de communication et le format de messages requis pour communiquer avec un service.

– La définition du service peut alors être découverte par d'autres systèmes logiciels au travers de registres tels qu'UDDI [138]. Ce modèle, acronyme d'Universal Description Discovery and Integration, est une technologie d'annuaire basée sur XML permettant de localiser sur le réseau un service web recherché.

– Les applications et systèmes peuvent alors interagir avec le service web d'une manière prescrite par les protocoles Internet [20], usuellement SOAP [140] (pour Simple Object Access Protocol) qui est un protocole permettant l'échange d'informations structurées dans un environnement décentralisé et distribué. Il a été conçu indépendamment de tout modèle de programmation et autre sémantique spécifique d'implémentation.

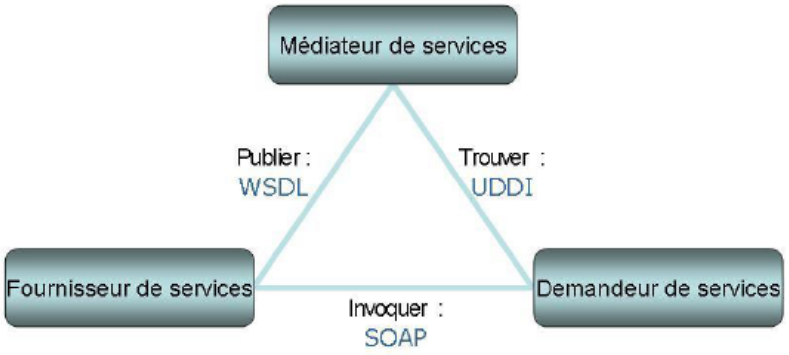


Figure 1.1. Services web fondés sur les fonctions *Publier*, *Trouver*, *Invoquer*.

3. Services web et web sémantique

Quelles que soient les versions (1.1 ou 2.0) de WSDL, la description du service reste uniquement au niveau fonctionnel, c'est-à-dire qu'elle contient la manière dont on peut utiliser le service et non ce que fait le service. Par conséquent, la description WSDL reste insuffisante lors du processus de sélection. Des travaux proposent des moyens de décrire des services Web sémantiques afin de pallier cette difficulté et pour mettre en œuvre d'autres aspects tels que la découverte automatique de services.

Cette section présente tout d'abord la définition des services Web sémantiques, puis étudie les langages émergeants qui permettent de décrire ce type de services Web.

3.1. Services Web sémantiques

L'objectif premier du Web sémantique est de définir et lier les ressources du Web afin de simplifier leur utilisation, leur découverte, leur intégration et leur réutilisation dans le plus grand nombre d'applications [5]. Le Web sémantique doit fournir l'accès à ces ressources par l'intermédiaire de descriptions sémantiques exploitables et compréhensibles par des machines. Cette description repose sur des ontologies. Selon [135], une ontologie est *une spécification explicite d'une conceptualisation.* Une *conceptualisation* est un modèle abstrait qui représente la manière dont les personnes conçoivent les choses réelles dans le monde et une *spécification explicite* signifie que les concepts et les relations d'un modèle abstrait reçoivent des noms et des définitions explicites [135]. Le Web sémantique est devenu un domaine à part entière, preuve en est la création en 2001 du groupe de travail sur ce sujet par le W3C.

À ce jour, le standard de description WSDL ne supporte pas la description de services Web comme une ressource utilisable dans le contexte du Web sémantique. Or, l'automatisation des processus d'enregistrement, de recherche et d'acquisition peut faciliter, à terme, la tâche des concepteurs de systèmes à base de services Web qui doivent faire face à l'augmentation du nombre de services Web disponibles.

Afin d'automatiser les processus d'enregistrement (action d'identification du service Web), de recherche (action issue de la requête du client) et de sélection (action de choix) des services Web, des travaux académiques ont été initiés, principalement dans le domaine du Web sémantique [136]. Les services Web issus des travaux de ce domaine sont appelés des **services Web sémantiques** (par opposition à notre appellation de service Web classique). D'après [125], les services Web sémantiques sont la combinaison de deux technologies (*cf.* Figure 1.2) : celle des services Web et celle du Web sémantique. Les services Web sémantiques sont des services Web dont la description est améliorée par des langages empruntés au Web sémantique, tel que RDF [52] et OWL [33]. Cet emprunt au Web sémantique permet à ces services Web d'être découverts et sélectionnés automatiquement par des machines ou d'autres services Web distants. Ceci permet aux services Web sélectionnés de répondre au mieux à la requête du client.

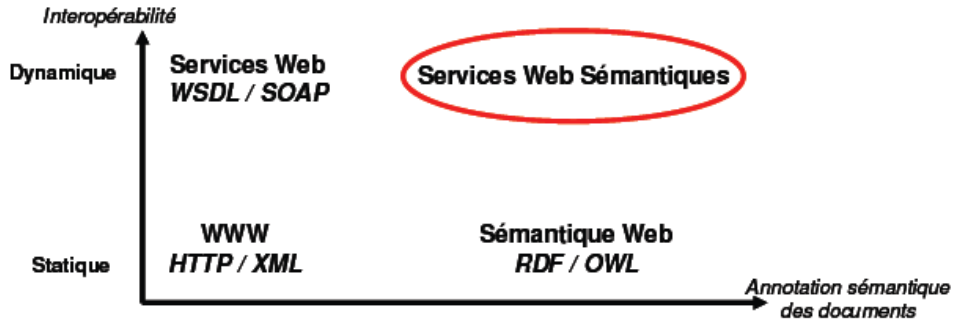


Figure 1.2. Les services Web sémantiques se situent entre deux problématiques liées au Web : l'interopérabilité et l'annotation sémantique des ressources [22].

[15] introduit un ensemble de concepts-clés inhérents à cette technologie hybride que sont les services Web sémantiques. Nous n'évoquons ici que les concepts de *notion de service* et de *représentation de service* qui aident la compréhension des travaux portant sur la description de services Web sémantiques.

La notion de service. La notion de service par [15] intègre trois idées principales. Les deux premières respectent des notions sous-jacentes à la description d'un service Web classique : **un service est une interaction entre deux parties** (communément appelées dans le domaine des services Web *fournisseur* et *client*). **Un service doit être considéré à différents niveaux d'abstraction** : le service concret (suite spécifique d'actions) et le service abstrait (ensemble de services concrets sans les précisions techniques utiles lors de la sélection de services). La

troisième idée sous-jacente à la notion de service est que cette dernière a un sens dans la mesure où **le service est inclus dans un domaine d'application** (par exemple, le domaine du tourisme). [15] définit ce domaine comme étant le *domaine de la valeur* du service.

La représentation du service. L'intérêt de combiner le Web sémantique et les services Web est de proposer une représentation intégrant les valeurs fournies par les services sous une forme compréhensible et interprétable automatiquement par les machines. Dans ce contexte, la description de services repose sur les techniques issues de la représentation de connaissances.
Étant donné que ce domaine a développé des langages et techniques formels pour décrire les connaissances et raisonner sur ces dernières, la représentation de connaissances s'impose d'elle-même comme une solution pertinente. La description des services Web sémantiques repose sur deux choix. Tout d'abord, le choix du langage de représentation de connaissances à utiliser. Ensuite, le choix des concepts et relations à prendre en compte dans la description et leur(s) signification(s). Ceci implique la création d'ontologies ou la sélection d'ontologies existantes qui puissent fournir un vocabulaire ontologique structuré, c'est-à-dire un ensemble de concepts et de relations qui puissent être utilisés pour décrire des entités dans le domaine de la valeur [15].

3.2. Langages de description de services Web sémantiques

Des travaux, tels que [23], [30], proposent une manière de représenter de manière sémantique des services Web (autrement dit proposent de décrire des services Web sémantiques). Malgré l'abondance de ce type de travaux, aucun ne s'est imposé comme une solution de description de services Web sémantiques. Nous avons choisi d'aborder ici seulement les travaux issus du W3C qui tentent d'apporter une solution standard en termes de description de services Web sémantiques. Deux travaux sont présentés : OWL-S et SAWSDL.

3.2.1. OWL-S

OWL-S [25] (*Ontology Web Language for Services*) est un langage issu des travaux de la DARPA[4] et de son programme *Agent Markup Language* (DAML) et prend la suite de DAML-S (*DARPA Agent Markup Language Service*)[5].

4. DARPA est l'acronyme de Defence Advanced Research Projects Agency http://www.darpa.mil/
5. http://www.daml.org/services/

Il a été intégré au consortium W3C en 2004, au sein du groupe d'intérêt sur les services Web sémantiques[6], lors de la recommandation du langage OWL.

Le but initial du langage OWL-S est de mettre en œuvre des services Web sémantiques. Cette mise en œuvre inclut un grand nombre d'objectifs, rendus possibles par le biais de l'expressivité héritée d'OWL et de l'utilisation de la logique de description [43]. Ces objectifs sont :

- **la description de services Web sémantiques**. Cet objectif est étudié dans ce chapitre ;
- **la découverte automatique** de ces services (non prise en compte à ce jour) ;
- **l'invocation automatique** de ces services, par le biais de la détection et de l'interprétation automatique de la localisation et des paramètres d'entrée/sortie (étant donné que cet objectif a déjà été atteint par le standard de description de services Web classiques – WSDL, nous ne traitons pas ici cette partie du langage OWL-S) ;
- **la composition automatique** de services (description et invocation) et **la surveillance** de l'exécution de la composition.

La Figure 1.3 illustre le fait qu'une ressource fournit un service. Ce service présente un profil de service (*ServiceProfile*), est décrit par un modèle de service (*ServiceModel*) et supporte un accès de service (*ServiceGrounding*).

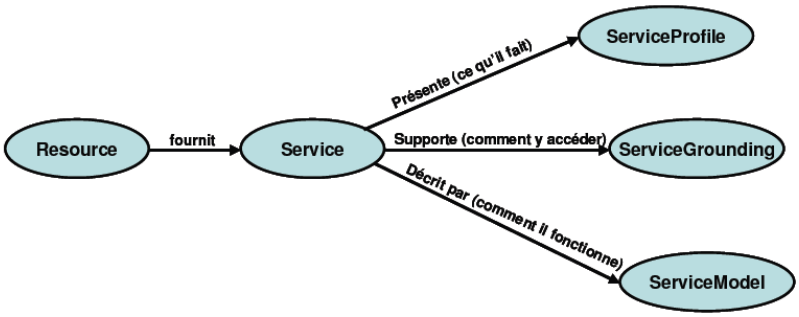


Figure 1.3. Diagramme fonctionnel de l'ontologie de service [25].

6. http://www.w3.org/2002/ws/swsig/

La description sémantique en OWL-S d'un service Web *GlobalWeather*, est présentée par la Figure 1.4.

```
1  <!DOCTYPE uridef[
2      <!ENTITY service "http://www.daml.org/services/owl-s/1.0/Service.owl">
3      <!ENTITY my_profile "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/profile.owl">
4      <!ENTITY my_process "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/process.owl">
5      <!ENTITY my_grounding "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/grounding.owl">
6  ]>
7  <rdf:RDF
8      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
9      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema"
10     xmlns:owl="http://www.w3.org/2002/07/owl"
11     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
12     xmlns:service="http://www.daml.org/services/owl-s/1.1/Service.owl"
13     xml:base="http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/service.owl">
14     <owl:Ontology about="">
15         <owl:imports rdf:resource="&service;" />
16         <owl:imports rdf:resource="&my_process;" />
17         <owl:imports rdf:resource="&my_profile;" />
18         <owl:imports rdf:resource="&my_grounding;" />
19     </owl:Ontology>
20     <service:Service rdf:ID="GlobalWeather">
21         <service:presents rdf:resource="&my_profile;#GlobalWeatherProfile"/>
22         <service:describedBy rdf:resource="&my_process;#GlobalWeatherProcess"/>
23         <service:supports rdf:resource="&my_grounding;#WsdlGrounding"/>
24     </service:Service>
25 </rdf:RDF>
```

Figure 1.4. Ontologie *Service* représentant l'ensemble des éléments composant la description sémantique du service Web *GlobalWeather* par le biais d'OWL-S.

L'ontologie *Service* est l'élément principal de la description d'un service Web sémantique (*cf.* Figure 1.4). Elle référence les trois autres parties constituant la description. Tous les espaces de noms nécessaires sont décrits dans l'élément racine *rdf*. L'URL de la localisation de cette ontologie se situe dans cet élément racine (*cf.* ligne 13). L'élément service présente les éléments formant la description OWL-S du service *GlobalWeather* : le profil du service (GlobalWeatherProfile, lignes 3 et 21), le processus (GlobalWeatherProcess, lignes 4 et 22) et l'accès au service (GlobalWeatherGrounding, lignes 5 et 23).

L'élément *Profile* apporte une description du service et de son fournisseur (description abstraite du service). Le *ServiceProfile* du service est utilisé lors de la publication et la recherche d'un service. Il inclut trois types d'informations (décrites au format RDF) : le fournisseur, le comportement fonctionnel et les attributs fonctionnels. Le *ServiceProfile* du service *GlobalWeather* est illustré par la Figure 1.5.

```
1  <!DOCTYPE uridef[
2  <!ENTITY concept "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/GlobalWeatherConcept.owl">
3  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
4  ]>
5  <rdf:RDF
6      xmlns:process= "http://www.daml.org/services/owl-s/1.1/Process.owl"
7      xmlns:profile= "http://www.daml.org/services/owl-s/1.1/Profile.owl"
8      xml:base= "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/profile.owl"
9      ...
10     >
11     ...
12     <profile:Profile  rdf:ID="AddServiceName">
13         <profile:serviceName> GlobalWeather </profile:serviceName>
14         <profile:textDescription>  Get weather report for all major cities around the world.
15         </profile:textDescription>
16         <profile:contactInformation>
17             <profile:Actor rdf:ID="WebServiceX"/>
18         </profile:contactInformation>
19         <profile:hasInput>
20             <process:Input rdf:ID="GlobalWeatherSoap_GetWeather_parameters_IN">
21                 <process:parameterType rdf:datatype="&xsd;#anyURI">
22                     &concept;#GetWeatherTypeDeclaration
23                 </process:parameterType>
24             </process:Input>
25         </profile:hasInput>
26         <profile:hasOutput>...</profile:hasOutput>
27         <profile:hasInput>...</profile:hasInput>
28         <profile:hasOutput>...</profile:hasOutput>
29     </profile:Profile>
30 </rdf:RDF>
```

Figure 1.5. *ServiceProfile* du service Web *GlobalWeather* défini par l'élément Profile.

Le fournisseur. Cette description est une représentation abstraite des acteurs intervenant dans la vie du service Web (le fournisseur et le client de service). Une liste d'informations permettant d'entrer en contact avec le fournisseur compose cet élément de description du service Web (telle que son adresse physique, l'URL du site Internet, son nom, son téléphone, son adresse électronique, son fax ou encore le contact de prestataires intervenant pour la maintenance du service). Dans le *ServiceProfile* du *GlobalWeather*, le fournisseur est défini par l'instance WebServiceX de la classe Actor (<profile:Actor rdf:ID="WebServiceX"/>, *cf.* ligne 17 de la Figure 1.5).

Le comportement fonctionnel. Cet élément de description permet de rendre publiques les opérations disponibles que propose le service, ainsi que leurs paramètres d'entrée et de sortie. De plus, le *ServiceProfile* inclut la description des pré-conditions nécessaires à l'appel des opérations et les effets attendus à la suite de l'exécution du service. Le service *GlobalWeather* (*cf.* Figure 1.5) propose deux opérations (dont *GetWeather*), possédant chacune un paramètre d'entrée (<profile:hasInput>, *cf.* lignes 19 à 24 et ligne 27) et un paramètre de sortie (<profile:hasOutput>, *cf.* lignes 26 et 28). La définition des types de données est opérée dans l'ontologie *Concept* (lignes 21 à 23 dans la Figure 1.5). Dans la Figure 1.6 est illustrée la déclaration du type

(GetWeatherTypeDeclaration) du paramètre d'entrée de *GetWeather* est constituée de deux éléments (CityName et CountryName, *cf.* respectivement lignes 4 à 7 et 8 à 11) de type *string*.

```
1  <owl:Class rdf:ID="GetWeatherTypeDeclaration">
2      <rdfs:subClassOf rdf:resource="&owl;#Thing"/>
3  </owl:Class>
4  <owl:DatatypeProperty rdf:ID="CityName">
5      <rdfs:range rdf:resource="&xsd;#string"/>
6      <rdfs:domain rdf:resource="#GetWeatherTypeDeclaration"/>
7  </owl:DatatypeProperty>
8  <owl:DatatypeProperty rdf:ID="CountryName">
9      <rdfs:range rdf:resource="&xsd;#string"/>
10     <rdfs:domain rdf:resource="#GetWeatherTypeDeclaration"/>
11 </owl:DatatypeProperty>
```

Figure 1.6. Extrait de l'ontologie *Concept* pour la description des types de données du service Web *GlobalWeather*.

Les attributs fonctionnels. Ce dernier type d'information contenu dans le *ServiceProfile* apporte des informations supplémentaires concernant le service. Tout d'abord, on peut trouver la **catégorie du service** (par exemple, en utilisant le système de classification nord-américain UNSPSC[7] qui encode les produits et les services pour leur utilisation commerciale sur le Web.
Ce système a l'avantage de contribuer au Web sémantique en formalisant ces catégories en ontologies via OWL et RDF). Ensuite, le futur client du service peut trouver le **niveau de qualité** déterminé par le fournisseur du service. Ce niveau de qualité peut être représenté, par exemple, par une appréciation (telle que « bon », « digne de confiance » ou encore « temps de réponse rapide »).

Enfin, la troisième catégorie d'information stockée dans le *ServiceProfile* est constituée de **tout autre information jugée pertinente par le fournisseur** (telle que le coût du service, sa localisation géographique, sa disponibilité) afin que son service soit sélectionné par de futurs clients. La description du service Web *GlobalWeather* ne contient pas ce type d'information.

7. UNSPSC – *Universal Standard Protocol and Services Classification*, http://www.unspsc.org/

Le *ServiceModel*, dans le contexte d'un service Web élémentaire, présente le fonctionnement du service et définit comment interagir avec ce dernier. Dans le cadre de la description d'une composition de services, le ServiceModel permet aussi de décrire l'activité du service dans une composition. À l'instar du *ServiceProfile*, le *ServiceModel* apporte une description abstraite du service Web sémantique. Dans cette entité d'OWL-S, le service est vu comme un processus (*Process*). Un processus est une spécification de la manière dont le client peut interagir avec le service. Il est composé d'un ensemble d'informations sur ce dernier : son nom, les participants à son exécution (un client simple ou d'autres services Web), ce qu'il fait, les conditions d'utilisation et ses effets, son résultat, ses paramètres (entrée, sortie). Il existe trois types de processus : atomique (qui possède un accès), simple (qui fournit une vue sur d'un autre processus), et composé (qui est décomposable en processus simple). Étant donné que, dans cette section, nous ne nous préoccupons que de la description d'un service Web élémentaire (et non composé), nous n'étudions ici que le processus *atomique* (les deux autres processus sont présentés dans le deuxième chapitre de ce manuscrit concernant la composition de services Web).

Un processus atomique est un processus qui peut être directement invoqué par l'intermédiaire d'un accès (utilisation du *GroundingProfile*). Le service *GlobalWeather* possède deux processus atomiques (dont GlobalWeatherSoap_GetWeather – *cf.* Figure 1.7, lignes 12 à 15). La description du processus atomique contient les paramètres d'entrée et sortie et fait référence à l'ontologie *Concept* (ligne 2) pour la définition de la structure des données.

Le *ServiceGrounding* (élément grounding) d'une description OWL-S d'un service Web apporte une description concrète du service. La définition de l'accès du service (protocole d'accès au service, format de messages, appel aux opérations et type de transport) est contenue dans le *ServiceGrounding*.
Pour décrire ces informations, le *ServiceGrounding* fait référence à la description standard du service en WSDL. Le *ServiceGrounding* permet de concrétiser la définition du processus dans le *ServiceModel*.

```
1  <!DOCTYPE uridef[
2  <!ENTITY concept "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/GlobalWeatherConcept.owl">
3  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
4  ]>
5  <rdf:RDF
6      xmlns:concept= "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/GlobalWeatherConcept.owl"
7      xml:base= "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/process.owl" >
8      ...
9      <process:Input rdf:ID="GlobalWeatherSoap_GetWeather_parameters_IN">...</process:Input>
10     <process:Output rdf:ID="GlobalWeatherSoap_GetWeather_parameters_OUT">...</process:Output>
11
12     <process:AtomicProcess rdf:ID="GlobalWeatherSoap_GetWeather">
13         <process:hasInput rdf:resource="#GlobalWeatherSoap_GetWeather_parameters_IN"/>
14         <process:hasOutput rdf:resource="#GlobalWeatherSoap_GetWeather_parameters_OUT"/>
15     </process:AtomicProcess>
16
17     <process:Input rdf:ID="GlobalWeatherSoap_GetCitiesByCountry_parameters_IN">...</process:Input>
18     <process:Output rdf:ID="GlobalWeatherSoap_GetCitiesByCountry_parameters_OUT">...</process:Output>
19     <process:AtomicProcess rdf:ID="GlobalWeatherSoap_GetCitiesByCountry">...</process:AtomicProcess>
20 </rdf:RDF>
```

Figure 1.7. *ServiceModel* du service Web *GlobalWeather*.

La Figure 1.8 illustre l'accès au service *GlobalWeather* par l'intermédiaire de son *ServiceGrounding*. Le service *GlobalWeather* possède deux accès (GlobalWeatherSoap_GetWeather_Grounding et GlobalWeatherSoap_GetCitiesByCountry_Grounding) aux deux opérations du service (*cf.* respectivement lignes 15 à 31 et ligne 32). Ces accès aux services sont définis dans le document en les liants aux processus atomiques définis dans le *ServiceModel*. Cette liaison est possible via la description concrète par des références au document WSDL :

- vers les opérations (<grounding:wsdlOperation>, lignes 17 à 22),
- vers les messages échangés (<grounding:wsdlInputMessage> – ligne 23 et <grounding:wsdlOutputMessage> – ligne 25),
- vers les paramètres d'entrée et sortie (respectivement <grounding:wsdlInput> – ligne 24 et <grounding:wsdlOutput> – ligne 26).

```
1  <!DOCTYPE uridef[
2      <!ENTITY pm_file "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/process.owl">
3      <!ENTITY local_service "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/service.owl">
4      <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
5  ]>
6  <rdf:RDF
7      xml:service="http://www.daml.org/services/owl-s/1.1/Service.owl"
8      xml:base= "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/grounding.owl"...>
9      ...
10     <grounding:WsdlGrounding rdf:ID="WsdlGrounding">
11         <service:supportedBy rdf:resource="&local_service;#GlobalWeather" />
12         <grounding:hasAtomicProcessGrounding rdf:resource="#GlobalWeatherSoap_GetWeather_Grounding"/>
13         <grounding:hasAtomicProcessGrounding rdf:resource="#GlobalWeatherSoap_GetCitiesByCountry_Grounding"/
14     </grounding:WsdlGrounding>
15     <grounding:WsdlAtomicProcessGrounding rdf:ID="GlobalWeatherSoap_GetWeather_Grounding">
16         <grounding:owlsProcess rdf:resource="&pm_file;#GlobalWeatherSoap_GetWeather"/>
17         <grounding:wsdlOperation>
18             <grounding:WsdlOperationRef>
19                 <grounding:portType rdf:datatype="&xsd;#anyURI">...</grounding:portType>
20                 <grounding:operation rdf:datatype="&xsd;#anyURI">GetWeather</grounding:operation>
21             </grounding:WsdlOperationRef>
22         </grounding:wsdlOperation>
23         <grounding:wsdlInputMessage rdf:datatype="&xsd;#anyURI">...</grounding:wsdlInputMessage>
24         <grounding:wsdlInput>...</grounding:wsdlInput>
25         <grounding:wsdlOutputMessage rdf:datatype="&xsd;#anyURI">...</grounding:wsdlOutputMessage>
26         <grounding:wsdlOutput>...</grounding:wsdlOutput>
27         <grounding:wsdlDocument  rdf:datatype="&xsd;#anyURI">
28             http://www.webservicex.com/globalweather.asmx?wsdl#
29         </grounding:wsdlDocument>
30         <grounding:wsdlReference rdf:datatype="&xsd;#anyURI">...</grounding:wsdlReference>
31     </grounding:WsdlAtomicProcessGrounding>
32     <grounding:WsdlAtomicProcessGrounding rdf:ID="GlobalWeatherSoap_GetCitiesByCountry_Grounding">
33         ...
34     </grounding:WsdlAtomicProcessGrounding>
35 </rdf:RDF>
```

Figure 1.8. *ServiceGrounding* du service Web *GlobalWeather*.

La localisation de la description WSDL du service *GlobalWeather* est contenue dans le *ServiceGrounding* (<grounding:wsdlDocument>, lignes 27 à 29).

OWL-S permet à travers trois ontologies (*ServiceProfile*, *ServiceModel* et *ServiceGrounding*) de décrire de manière sémantique un service Web. Le standard de description WSDL est toujours utilisé en vue de décrire les aspects fonctionnels du service. OWL-S, bien qu'il ne soit pas standardisé par le W3C, commence à être largement utilisé dans les travaux du Web sémantique.

3.2.2. SAWSDL – Annotations sémantiques pour WSDL

À l'initiative du groupe de travail d'annotations sémantiques pour WSDL (*Semantic Annotations for WSDL Working Group*[8]), SAWSDL (*Semantic Annotations for WSDL and XML Schema*) [67] est une recommandation du W3C depuis août 2007. Ce groupe de travail prend part aux activités du W3C portant sur les services Web. L'objectif de SAWSDL est d'ajouter de la sémantique à la description WSDL des services et des schémas XML [59].

8. http://www.w3.org/2002/ws/sawsdl/

SAWSDL est la suite de WSDL-S (*Web Service Description Language – Semantic*) [112]. SAWSDL définit un mécanisme d'annotation en vue de spécifier les éléments de WSDL à l'aide d'ontologie(s). Cette annotation repose sur la définition d'attributs étendant le standard de description. Les annotations sémantiques référencent des ontologies pré existantes. Le mécanisme d'annotation de SAWSDL est indépendant de tout langage de représentation d'ontologies.

SAWSDL propose deux sortes d'annotations sémantiques : une première pour identifier le concept sémantique (représentée par l'attribut modelReference) et une seconde pour faire le lien entre le concept et le document WSDL (représentée par les attributs liftingSchemaMapping et loweringSchemaMapping).

SAWSDL propose deux sortes d'annotations sémantiques : une première pour identifier le concept sémantique (représentée par l'attribut modelReference) et une seconde pour faire le lien entre le concept et le document WSDL (représentée par les attributs liftingSchemaMapping et loweringSchemaMapping).

SAWSDL permet d'ajouter facilement une description sémantique aux descriptions WSDL déjà existantes. L'avantage de cette proposition est qu'elle permet de modifier des descriptions WSDL existantes sans trop alourdir le fichier de description. Cependant, SAWSDL ne permet pas d'annoter tous les éléments XML du document WSDL (par exemple, la description du fournisseur de service ne peut pas être annotée) et ne permet pas d'apporter de la sémantique aux éléments n'étant pas décrits par WSDL (tels que la qualité de service).

4. Conclusion

La technologie des services Web permet à des applications de dialoguer à distance via Internet, indépendamment des plates-formes et des langages sur lesquelles elles reposent, en s'appuyant sur des protocoles standards (WSDL, SOAP, UDDI). Les services Web, de plus en plus utilisés dans le monde industriel, posent de nouveaux problèmes : comment composer ces services ?

L'intégration du Web sémantique dans la communauté des services Web (par le biais des langages sous-jacents au Web sémantique, tels que RDF et OWL) a apporté, entre autres, un niveau sémantique dans la description des services Web (on parle alors de services Web

sémantiques) par le biais des langages OWL-S et SAWSDL. Cependant, la communauté des services Web n'a pas aujourd'hui adopté l'un ou l'autre de ces langages comme standard de description de services Web sémantiques.

En ce qui concerne OWL-S, nous pouvons mettre en évidence le fait que les acteurs du domaine de la composition de services Web et du Web sémantique ne sont pas encore à ce jour d'accord sur le rôle d'OWL-S. Les acteurs du premier domaine pensent qu'il s'agit d'un langage de composition alors que les acteurs du domaine du Web sémantique voient plutôt OWL-S comme une ontologie de description. En effet, Ankolekar et al, dans une étude comparant OWL-S avec les différents langages de la technologie des services Web, mettent en parallèle OWL-S tant avec des langages de description (tels que WSDL) qu'avec des langages de composition (tels que WS-CDL et BPEL4WS). De notre point de vue, l'utilisation de OWL-S aujourd'hui en tant que langage de description de services Web sémantiques, permettra, à terme, qu'il soit aussi utilisé en tant que langage de composition.

CHAPITRE 2

COMPOSITION DE SERVICES WEB : ETAT DE L'ART

Ce chapitre est consacré à un état de l'art sur les travaux liés à la composition de services D'abord, nous proposons une catégorisation selon laquelle nous décrivons et analysons les approches existantes de composition de services web, à savoir les approches dites *statiques* et celles dites *dynamiques*

Nous analysons les approches présentées et concluons ce chapitre par les limites que les travaux existants en composition de services présentent pour la modélisation du problème de la chorégraphie dynamique de services (composition dynamique de services).

1. Composition de services Web

1.1. Composition de services web : définition

Un des défis des SOA est l'intégration de services ou de systèmes distribués pour l'approvisionnement de nouveaux services personnalisés, plus riches et plus intéressants aussi bien pour des applications, d'autres services ou plus communément pour des utilisateurs humains.

En effet, si une application ou un client requièrent des fonctionnalités, et qu'aucun service n'est seul apte à les fournir, il devrait être possible de combiner ou de composer des services existants afin de répondre aux besoins de cette application ou de ce client [87] [14]. C'est ce que l'on appelle la *composition de services* web [4].

> ***La composition de services*** *vise à faire interopérer, interagir et coordonner plusieurs services pour la réalisation d'un but.*

1.2. Composition de services web : état de l'art

La composition de services est à l'origine d'un nombre considérable de travaux, aussi bien dans la recherche académique que dans l'industrie.

En dépit de tous les efforts déployés, la composition de services reste un problème très complexe. Cette complexité tient au fait que les solutions de composition de services doivent tenir compte du nombre croissant de services déployés sur le web, de leur mise à jour continuelle et de leur *hétérogénéité* [74]. En conséquence, elles nécessitent de traiter les problématiques de la coordination des services, leurs transactions, leur exécution et leur modèles de conversation (ou d'interaction) [8].

Ces dernières années, plusieurs communautés se sont intéressées à la problématique de la composition de services. Des solutions ont notamment été proposées dans les communautés des bases de données [11][6], du web sémantique [141][24] [100][58], et plus récemment dans celles des agents [122] [66] [139][60]. Les solutions proposées peuvent être classifiées selon deux axes:
1. En fonction du degré de participation de l'utilisateur dans la définition du schéma de composition, ces propositions sont *manuelles*, *semi-automatiques* ou *automatiques*.
2. Selon que la sélection des services et la gestion du flot soient faites ou non à priori, une approche sera dite *statique* ou *dynamique*.

> *On dit qu'une* ***composition*** *est* ***statique*** *lorsqu'elle prend place à l'étape de conception, au moment où l'architecture et la conception du système logiciel sont planifiées. Les composants (ou services) qui seront utilisés sont préalablement choisis et reliés, et la gestion du flot est effectuée a priori [123].*

Les approches statiques de composition de services sont celles adoptées par l'industrie. Elles s'inspirent de la gestion de processus métiers quant à la description des données et des flots de contrôle pour le processus de composition.

> *Par opposition, une **composition** de services est dite **dynamique** si les services sont sélectionnés et composés à la volée en fonction des besoins formulés par l'utilisateur [132].*

Une approche dynamique pour la composition de services offre le potentiel de réaliser des applications flexibles et adaptables en sélectionnant et en combinant les services de manière appropriée sur la base de la requête et du contexte de l'utilisateur. Ce type de composition peut engendrer de nombreuses applications utiles qui n'ont pas été prévues à l'étape de conception. Par conséquent, la composition dynamique de services est propice dans un environnement tel que le web et l'informatique pervasive où les composants disponibles sont dynamiques et les attentes des utilisateurs variables et personnalisées. C'est précisément ce type d'approche qui fait l'objet de notre travail.

1.2.1. Composition statique des services web : orchestration et chorégraphie

A. Chorégraphie

> ***La chorégraphie** décrit la collaboration entre une collection de services dont le but est d'atteindre un objectif donné. L'accomplissement de ce but commun se fait alors par des échanges ordonnés de messages [16].*

La chorégraphie de services dépeint les interactions dans lesquelles les services engagés participent afin d'atteindre cet objectif, ainsi que les dépendances entre les interactions telles que le flot de contrôle (ex. une interaction doit précéder une autre), le flot de données, les corrélations des messages, les contraintes de temps, les dépendances transactionnelles, etc.

La figure 2.1 représente la chorégraphie des services *client*, *fournisseur* et *entrepôt* pour la commande et la livraison d'un produit. Elle est représentée sous la forme d'un diagramme d'activités UML. Les actions élémentaires dans ce diagramme représentent des activités impliquant l'envoi et la réception de messages. Par exemple, l'action ***Commander produit***, entreprise par le client, résulte en l'envoi d'un message au fournisseur.

Un modèle de chorégraphie n'est pas exécutable. Il constitue en fait un accord entre des parties, où celles-ci spécifient comment leur collaboration doit se dérouler. Cette convention peut être établie par une consultation commune ou conçue par un comité de standardisation.
WS-CDL [102] est un exemple de langage pour la description de chorégraphies.

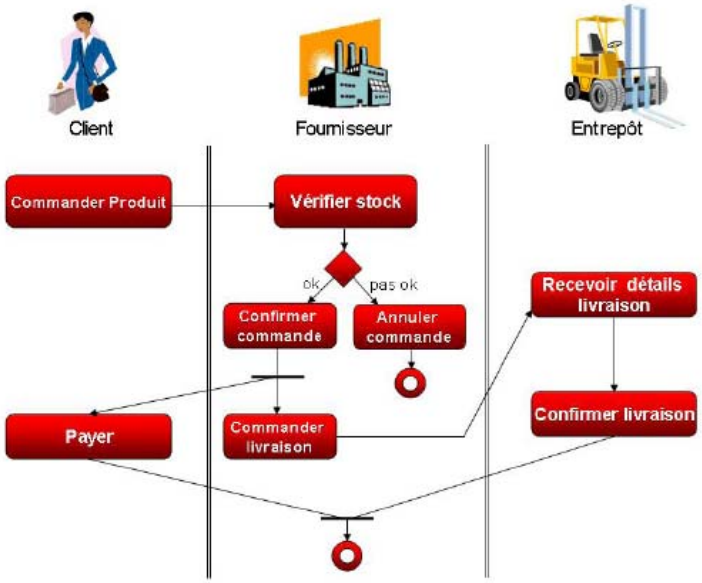


Figure 2.1. Chorégraphie de services pour la commande et la livraison d'un produit.

La chorégraphie est aussi appelée composition dynamique [13]. En effet, l'exécution n'est pas régie de manière statique comme dans une composition de type orchestration. Dans une chorégraphie, à chaque pas de l'exécution (*i.e.* à chaque étape de la composition), un service Web choisit le service Web qui lui succède et implémente ainsi une partie de la chorégraphie. La composition de type chorégraphie n'est pas connue, ni décrite à l'avance.

Le W3C compte depuis 2002 parmi ses groupes de travail, le groupe de travail sur la chorégraphie de services Web (*Web Services Choreography Working Group*). Pour ce dernier, la chorégraphie des services Web concerne les interactions observables des services avec leurs utilisateurs (appelés aussi clients) [17]. Ces utilisateurs, automatisés ou non, peuvent être d'autres services Web, des applications, ou des concepteurs d'applications. Cet ensemble spécifique d'interactions peut être comparé à une collaboration entre un ensemble de services Web et leurs clients. La description d'une chorégraphie est un contrat multi-parties qui décrit, à partir d'un

point de vue global, le comportement observable externe entre plusieurs clients (généralement des services Web). Chaque comportement externe observable est défini comme la présence ou l'absence de messages échangés entre un service Web et ses clients.

B. Orchestration

> ***L'orchestration*** *décrit, du point de vue d'un service, les interactions de celui-ci ainsi que les étapes internes (ex. transformations de données, invocations à des modules internes) entre ses interactions [12].*

Dans une orchestration, le chef d'orchestre est donc le service dont les interactions sont dépeintes. Celles-ci comprennent l'envoi et la réception de messages à/de la part de partenaires sélectionnés. L'orchestration permet ainsi à un service d'être enchaîné à d'autres d'une manière prédéfinie. Elle est ensuite exécutée par des *scripts d'orchestration* qui décrivent les interactions entre les applications en identifiant les messages, la logique et les séquences d'invocations. Le composant exécutant les scripts d'orchestration est appelé *moteur d'orchestration*. Celui-ci agit comme une entité centralisée pour coordonner les interactions entre les services.

La figure 2.2 illustre, du point de vue du service *fournisseur*, l'orchestration des services de l'exemple précédent. Notons que les noms d'action dans ce schéma d'orchestration (ex. ***Vérifier disponibilité***) peuvent différer de ceux apparaissant dans le schéma de chorégraphie (ex. ***Vérifier stock dans la figure 2.2***). En effet, les actions dans un schéma d'orchestration sont instanciées en fonction de la structure du service dont l'orchestration est spécifiée.

Par ailleurs, un schéma d'orchestration organise des interfaces de comportements de plusieurs actions et dépeint en plus les actions internes du service (en pointillés sur la figure 2.2).

Il est à noter qu'une interface de comportement peut être utilisée comme point de départ pour générer un squelette d'orchestration qui sera ensuite complété par d'autres actions et des détails d'actions internes. WS-BPEL [64] et ebXML [70] sont des exemples de langages d'orchestration.

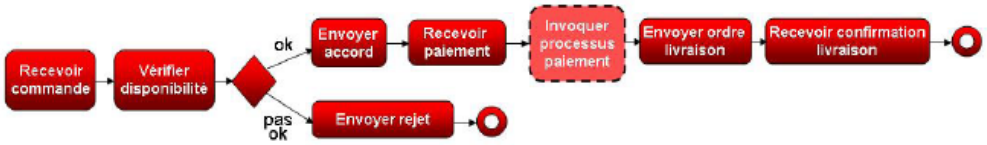


Figure 2.2. Orchestration de services pour le rôle *fournisseur*.

Analyse

Les approches et systèmes sus-décrits permettent ainsi à un utilisateur d'écrire le plan de composition de services, afin que ce dernier soit :

1. directement exécuté. On dit alors que la composition est entièrement statique ;
2. instancié et ensuite exécuté. La composition n'est pas entièrement dynamique, puisque le profil des services à composer est préalablement spécifié. Néanmoins, le moteur de composition est dynamique, puisque la composition varie en fonction des instances des services trouvées.

Par ailleurs, la plupart de ces approches permettent de traiter aussi bien les procédures génériques que des besoins ouverts d'utilisateurs. Cependant, elles présentent plusieurs inconvénients :

– Ces approches exigent de l'utilisateur d'avoir des connaissances bas-niveau du système. Dans le cas de WS-BPEL (Web Services Business Process Execution Language) [64], BPWS4J [63] et WS-CDL (TheWeb Services Choreography Description Language) [102] par exemple, l'utilisateur est censé construire un workflow au niveau XML. Ces systèmes ne s'adressent donc qu'à des usagers spécialistes en ingénierie de l'information.

– Les langages de chorégraphie permettent de décrire une collaboration entre services pour l'accomplissement d'un but. Toutefois, n'étant pas exécutables, ils ne suffisent pas seuls à l'implémentation d'une chorégraphie automatique de services.

– Dans ces travaux, les services à composer et le flot entre services sont préalablement définis. En conséquence, si l'un des services à composer n'est pas disponible, l'exécution échoue.

– Des inconsistances peuvent survenir à cause de l'approvisionnement de nouveaux services ou du remplacement d'anciens services par d'autres. Dans ces cas, il est inévitable de changer d'architecture logicielle et de la relier à d'autres services ou même, dans le pire des cas, changer la définition du processus et reconcevoir le système.

Par conséquent, une sélection et une composition statiques (ou partiellement dynamiques) de services conviennent tant que l'environnement des services (les partenaires et les composants services) reste inchangé, ou change rarement. L'utilisation de ces approches peut être très restrictive dans des environnements à large échelle tels que le web où il faut envisager des composants capables de s'adapter automatiquement aux changements imprévisibles.

1.2.2. Composition dynamique des services web

Les différentes approches existantes pour la composition dynamique de services Web peuvent être regroupées en deux courants.

Le premier se base sur le fait qu'un service Web composite est définit par un ensemble de services atomiques et par la façon dont ils communiquent entre eux. Cette définition est de même type que la manière dont est défini un processus métier.
Ce courant propose donc d'adapter les méthodes d'orchestration et de chorégraphie afin de les rendre dynamiques.

Dans le second courant, la composition est vue comme la génération automatique d'un plan d'exécution des services Web. Les approches envisagées sont des approches du domaine de l'intelligence artificielle et plus particulièrement de la planification. Ces approches sont basées sur le fait qu'un service Web peut être spécifié par ses pré-conditions et ses effets.

A. Approches orientées workflow.

D'une certaine façon, un service composite est similaire à un processus métier [44]. Un service composite inclut un ensemble de services atomiques ainsi que les contrôles et échanges de données entre ces services. De la même façon, un processus métier est composé d'un ensemble d'activités élémentaires structurées, ainsi que l'ordre d'exécution entre elles.

Ainsi Casati présente Eflow [45], une plateforme pour la spécification, la création et la gestion de services composites en utilisant les méthodes de génération de workflow statiques mais représentés en interne sous la forme d'un graphe qui peut être modifié dynamiquement lors de l'exécution. Ce graphe contient, en plus des nœuds représentant les services, des nœuds de décision et d'évènements qui permettent une plus grande robustesse du système : en cas de non

disponibilité d'un service Web, celui-ci peut être automatiquement remplacé par un service Web équivalent.

Khalaf [116] propose de regarder les chorégraphies créées avec BPEL4WS en tant que services Web, c'est-à-dire créer des patrons de composition qui pourront être composés, formant ainsi de nouveaux patrons de composition. Cette solution facilite la composition par une intégration plus simple de patrons d'assemblage prédéfinis dans un nouveau patron mais ne règle pas le problème de la dynamicité de la composition de services Web.

Laukkanen et Helin [85] identifient deux solutions possibles pour composer dynamiquement des processus métier : remplacer un service Web dans un processus métier existant par un autre service ayant des fonctionnalités similaires, ou définir un nouveau workflow à partir des services Web disponibles. Les auteurs proposent d'utiliser les descriptions sémantiques des services Web pour pouvoir comparer les fonctionnalités en utilisant les notions de pré-conditions et d'effets d'OWL-S. La solution proposée (représentée dans la figure 2.3) peut être résumée ainsi :
1. Identifier les fonctionnalités requises
2. Trouver les services adaptés grâce à leur description sémantique
3. Créer ou modifier le workflow
4. Exécuter le workflow et vérifier son exécution.

Dans le même esprit, Osman [133] propose de limiter les points négatifs des techniques de composition de type workflow en utilisant les technologies du Web sémantique, plus particulièrement OWL : il ne s'agit pas directement de composition automatique mais le but est de limiter le nombre de services Web disponibles afin de créer des workflows plus facilement. Pour cela, les auteurs proposent de regrouper les services Web en domaines spécifiques (e.g., domaine des services Web d'hôtels, de transports, etc.) Ainsi le choix des services à utiliser est restreint au domaine des services ayant comme fonctionnalités les objectifs du workflow.

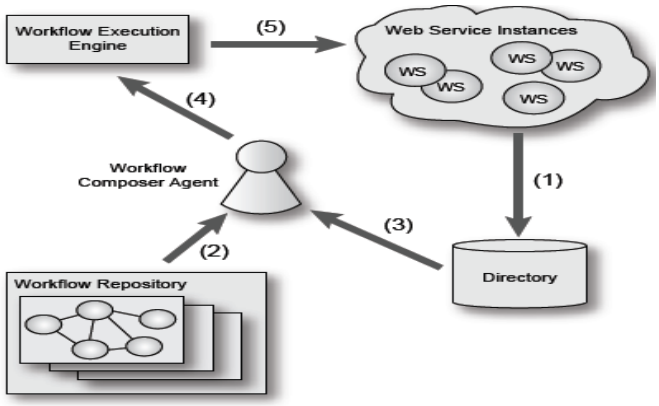


Figure 2.3. Composition dynamique de workflows

B. Approches orientées intelligence artificielle

La composition dynamique de services Web par des techniques d'intelligence artificielle, et plus particulièrement par des techniques de planification, est la voie qui semble la plus prometteuse [74]. En effet, les concepts de OWL-S sont tr`es fortement inspirés de ceux de la planification : les pré-conditions présentent les conditions qui doivent être satisfaites pour garantir la bonne exécution d'un service Web. Les effets sont le résultat du succès de l'exécution d'un service. Dans ce qui suit, nous présentons quelques axes de recherche actuels concernant la composition par planification et par d'autres techniques d'intelligence artificielle.

B1. Calcul situationnel (situation calculus). McIlraith et al [126] proposent d'adapter et d'´etendre le langage Golog pour la construction automatique de services Web. Golog est un langage de programmation logique qui permet de faire du calcul situationnel (i.e., langage logique qui sert à représenter des changements ou évolutions en termes de situations, d'actions et d'objets). Le problème de la composition de services Web est abordé de la façon suivante : la requête de l'utilisateur et les contraintes des services sont représentées en termes de prédicats du premier ordre dans le langage de calcul situationnel. Les services sont transformés en actions (primitives ou complexes) dans le même langage. Puis, à l'aide de règles de déduction et de contraintes, des modèles sont ainsi générés et sont instanciés à l'exécution à partir des préférences utilisateur. Ces recherches ont été étendues dans [127] : les prédicats nécessaires aux calculs situationnels sont tirés de DAML-S et représentés sous la forme de réseaux de Petri.

B2. Theorem proving. Waltinger [118] a élaboré une approche pour la composition de services par preuve de théorèmes. Cette approche est basée sur la déduction automatique et la synthèse de programmes. Les services disponibles et les requêtes utilisateur sont traduites dans un langage du premier ordre. Puis des preuves sont produites à partir d'un prouveur de théorèmes. La composition est obtenue à partir de preuves particulières. Dans le même genre, Rao et al [75] ont introduit une méthode de composition automatique de services Web sémantiques en utilisant des preuves de théorèmes logiques linéaires (Linear Logic Theorem Proving).

B3. Système multi-agent. Du fait de leur autonomie et leur hétérogénéité, les services Web peuvent être vus comme des agents. Ainsi Cheng et al [152] proposent le langage ASDL (Agent Service Description Language) qui a pour objectif de décrire le comportement externe des services Web. Une spécification ASDL décrit les messages compris par un service, ainsi que les protocoles d'interactions utilisés. Puis la composition est effectuée à l'aide du langage ASCL (Agent Service Composition Language) qui permet de décrire la logique avec laquelle un nouveau service est composé à partir de services existants. L'avantage de ces travaux est de mettre l'accent sur les interactions entre services au sein de la composition et d'adapter celle-ci en fonction des contraintes résultant des interactions. Cependant cette méthode n'est pas totalement dynamique. En effet, elle consiste à assembler et adapter au contexte d'exécution des patrons de composition définis a priori par le concepteur. Plus récemment, Müller et Kowalczyk [62] travaillent sur un système multi-agent pour la composition de services basé sur la concurrence entre coalitions de services : les agents représentants les services se contactent les uns les autres pour proposer leurs services en fonction de leur capacité de raisonnement et ainsi former des coalitions d'agents capables de résoudre les buts fournis par l'agent utilisateur. Puis les différentes coalitions vont faire une offre la plus compétitive possible. Chaque solution reçoit une note de l'agent utilisateur. C'est donc la solution ayant le plus haut score qui sera choisie. Ainsi, seuls les services Web correspondant le plus aux attentes de l'utilisateur seront utilisés dans la composition.

B4. Planification. Un intérêt croissant de la communauté planification pour la composition de services Web peut être expliqué par la similarité entre les descriptions OWL-S et les représentations en planification (cf. chapitre 3_planification en intelligence artificielle). En effet, la communauté du Web sémantique s'est fortement inspirée du langage PDDL lors de la conception du langage de description OWL-S. Ainsi une description OWL-S peut être facilement traduite en une représentation PDDL : les services Web sont représentés par des opérateurs ayant des pré-conditions et produisant des effets. Puis un planificateur peut alors être utilisé pour produire un plan qui correspond à la composition des services Web. Carman et al. [83], mais également Constantinescu et al [61] ont utilisé cette correspondance pour composer un ensemble de services Web de manière intuitive : les services Web sont choisis et composés uniquement à partir des types de données de leurs entrées et sorties.

Pour avoir un état de l'art complet de la composition dynamique par planification, nous pouvons nous reporter aux travaux de Peer publiés en 2005 [72], mais nous allons donner dans les prochaines sous sections un aperçu des principales pistes de la composition de services Web par planification.

B4.1. Planification « classique ». De nombreux travaux ont été produits ces dernières années. Les travaux de Sheshagiri et al. [91] proposent l'utilisation d'ontologies de domaines afin d'ajouter des contraintes pour optimiser la recherche dans un espace de plans. Ceux de McDermott [26] proposent d'utiliser la planification par régression estimée (Estimated-Regression Planning) qui permet d'utiliser des heuristiques pour guider la recherche dans un espace de situations. Le principe est le suivant : un agent qui désire interagir avec un service Web va construire un plan puis l'exécuter. Si l'exécution échoue, l'agent aura appris de nouvelles informations de cette interaction, qui seront utilisées dans le prochain cycle de planification et exécution. Sirin et al. [40] présentent une méthode semi-automatique pour la composition de services Web. A chaque fois qu'un utilisateur doit sélectionner un service Web, tous les services disponibles qui correspondent au service sélectionné, sont présentés à l'utilisateur qui fait alors le choix à la place du système. L'idée de la composition semi-automatique est assez intéressante car il est très difficile de capturer le comportement des services Web dans leurs moindres détails, puis de les composer automatiquement. Bien que cette méthode soit simple, elle permet de voir comment un planificateur automatique et un humain peuvent travailler ensemble pour répondre à

la requête de l'utilisateur. Enfin Peer [71] propose une approche basée sur le langage PDDL dans laquelle, après création du domaine de planification à partir de la description sémantique, un planificateur est choisi parmi plusieurs en fonction des instructions PDDL utilisées dans le domaine ou de la complexité du but à atteindre.

B4.2. Planification basée sur les règles (Rule-based planning). Medjahed [9] présente une technique pour générer des services composites à partir de descriptions déclaratives de haut niveau. Cette méthode utilise des règles de composabilité pour déterminer dans quelle mesure deux services sont composables. L'approche proposée se déroule en quatre phases :

1. La phase de spécification. Elle offre une spécification de haut niveau de la composition désirée en utilisant le langage CSSL (Composite Service Specification Langage).
2. La phase de correspondance. Elle utilise des règles de composabilité pour générer des plans conformes aux spécifications du service demandeur.
3. La phase de sélection. Si plus d'un plan est généré, la sélection est effectuée par rapport à des paramètres de qualité de la composition.
4. La phase de génération. Une description détaillée du service composite est automatiquement générée et présentée au demandeur.

La principale contribution de cette approche est la notion de règles de composabilité. Les règles de composabilité considèrent les propriétés syntaxiques et sémantiques des services Web. Les règles syntaxiques incluent des règles pour les types d'opérations possibles et pour les liaisons protocolaires entre les services (i.e., les bindings).

Les règles sémantiques incluent des règles concernant la compatibilité des messages échangés et la compatibilié des domaines sémantiques des services, mais également des règles de qualité de la composition. Cette notion de règles de composabilité met en avant les attributs possibles des services Web qui peuvent être utilisés dans la composition de services et peut ainsi jouer le rôle de directive pour d'autres méthodes de composition par planification.

B4.3. Planification hiérarchique. Dans [32], Wu et al préconisent l'utilisation du planificateur SHOP2 pour la composition automatique de services Web à partir de leur description sémantique. SHOP2 est un planificateur HTN (Hierachical Task Network).

D'après les auteurs, le principe de décomposition d'une tâche en sous-tâches dans la planification hiérarchique est très similaire au concept de décomposition de processus composites dans OWL-S. Afin de rendre leur proposition réalisable, les auteurs font deux hypothèses concernant le modèle de processus :

– Les processus atomiques ont, soit des sorties, soit des effets, mais pas les deux en même temps.

– Les structures de contrôle Split et Split+Join ne sont pas utilisées dans les processus composites

Sirin et Parsia [39] sont allés plus loin en intégrant un raisonneur sur les langages de descriptions dans le planificateur SHOP2. La planification HTN est très puissante pour les domaines où la connaissance est complète et détaillée, ce qui n'est pas, d'après Klush et al [9], le cas avec les services Web.

2. Analyse

L'utilisateur peut être dans une configuration ou, au contraire, il a besoin d'émettre des demandes personnalisées et totalement imprévisibles par un expert. Par exemple, organiser un emménagement (où pourraient être composés des services de décoration, de vente de meubles et de livraison), se procurer un ensemble de matériels informatiques (qui pourraient présenter contraintes de compatibilité, de coût ou de marque que les services à composer doivent respecter), ou même personnaliser sa maison intelligente (où pourraient être composées les fonctionnalités d'un téléphone et d'une télévision pour afficher l'identité des appelants), etc. Contrairement aux procédures génériques, ces cas d'usage sont des exemples de demandes d'utilisateur peu communes (car personnalisées) et imprévisibles par un expert.

D'un point de vue synthétique, les requêtes de l'utilisateur peuvent :

– porter sur toutes sortes de questions, commandes, demandes de devis, etc.

– comporter diverses contraintes sur les services demandés, qu'elles soient locales comme une marque ou une capacité de stockage, ou globales telles que le coût total d'une commande ou la durée totale d'un itinéraire, etc.

– présenter des interdépendances entre les sous-requêtes, que ce soit à propos de ressources, de contraintes de simultanéité, des relations entre des tâches et sous-tâches, etc.

C'est dans ce cadre que rentrent les travaux sur la composition dynamique de services web. Or, les travaux existants ne fournissent ni un modèle pour représenter de tels besoins (qu'ils décrivent à travers des schémas), ni une architecture de services capables de raisonner sur ces besoins, et encore moins des modèles de collaboration dynamique entre les services pour la satisfaction de ces besoins (puisque la composition dans les approches existantes est le résultat de l'exécution d'un schéma).

En conséquence, dans le contexte d'environnements dynamiques tels que le web ou l'informatique pervasive, où les services changent et les besoins des utilisateurs varient et ne peuvent pas tous être prévus à l'avance par un expert, la découverte des services doit se faire à la volée et la chorégraphie de ceux-ci, au lieu d'être préalablement fournie à travers un schéma de composition, doit s'effectuer de manière dynamique en fonction des besoins énoncés par l'utilisateur, des caractéristiques fonctionnelles et structurelles des services découverts et des interactions résultantes de ceux-ci.

3. Conclusion

Nous venons de présenter les principaux travaux en composition de services ; approches utilisant des scripts de workflows, des éditeurs, des techniques de planification, ou encore celles issues de l'intelligence artificielle.

Pourtant, aucune de ces approches ne peut être considérée comme étant pleinement dynamique. Certes, elles présentent pour chacune des points forts dans certains aspects de la composition, tels que des moteurs de composition dynamique. Ceci au détriment d'autres aspects, tels que des plans abstraits prédéfinis, des moteurs de composition centralisés, et une sélection statique des services à composer.

Comme nous l'évoquions plus haut, ces limitations proviennent du fait que ces travaux implantent des orchestrations de services. Elles décrivent la composition du point de vue d'un

service ou d'un moteur de composition centralisé pour un but prédéfini (i.e. procédure générique). Ce qui conduit inévitablement à des solutions statiques et ad hoc.

Au contraire, la chorégraphie de services se fonde sur le principe d'une composition de services dynamique et décentralisée, où chaque service doit être capable de collaborer avec d'autres services. Une telle approche doit tenir compte de la distribution et de la flexibilité des services, des besoins ouverts de l'utilisateur et des interactions flexibles des services sur la base de leurs fonctionnalités.

Les systèmes multi-agent semblent être un paradigme particulièrement bien adapté pour pallier les limites intrinsèques aux services web en général et des approches existantes de composition de services web en particulier. Ce volet de recherche fera l'objectif du prochain chapitre de ce manuscrit.

CHAPITRE 3

ENTRE COMPOSITION DE SERVICES ET PLANIFICATION MULTI-AGENT

Les systèmes multi-agent semblent être un paradigme particulièrement bien adapté pour pallier les limites intrinsèques aux services web en général et des approches existantes de composition de services en particulier.

Ce chapitre est consacré à un état de l'art sur les travaux liés au mariage de la composition de services avec la coordination multi-agent en général et la planification en particulier. Nous les présentons en deux temps. D'abord, nous commençons par présenter les limites des services web, et, avant de présenter la pertinence de la combinaison des technologies agent et services web, nous faisons une brève présentation des concepts clé des systèmes multi-agent ainsi que les classes de modèles de coordination. Ensuite, nous présentons un panorama des modèles de coordination dans les systèmes multi-agent ainsi que les travaux les implémentant pour le problème de la composition de services. Finalement, nous consacrons la dernière partie de ce chapitre aux problèmes liés à la coordination de plans du fait que nous nous intéressons aux travaux liés à ce type de coordination multi-agent.

1. Limites des services web

Comme nous le citions précédemment (dans le deuxième chapitre), il existe des langages pour la spécification de composition (orchestration ou chorégraphie) de services. Cependant, aucun standard n'a été promulgué pour la composition de services. En comprendre les raisons nécessite de revenir aux sources en identifiant les limites intrinsèques aux services web [98] :

- Les services web sont passifs jusqu'à ce qu'ils soient invoqués ;

- Un service web a seulement connaissance de lui-même, mais pas de ses applications ou utilisateurs clients ;
- Un service web n'est pas adaptable, et il n'est pas capable de bénéficier des nouvelles capacités de l'environnement afin d'apporter des services améliorés ;
- Enfin, la plus grande limite des services web est qu'il n'existe actuellement aucun standard pour supporter la composition de leurs fonctionnalités.

L'autonomie, l'adaptation et la coopération, points faibles des services web, sont par ailleurs des domaines qui ont largement été explorés par la recherche dans les systèmes multi-agent. Plusieurs études décrivent à ce propos la pertinence de la combinaison des technologies agent et services web [82][65] [99] [10] [131] [37].

2. Systèmes Multi-Agent

Les agents et systèmes multi-agent ont beaucoup été étudiés dans la littérature [68] [97] [103]. Le paradigme agent a été très influencé par le paradigme objet, mais s'en distingue par trois aspects qui sont l'autonomie, la rationalité et l'interaction.
Les systèmes multi-agent ne sont pas qu'un simple groupe d'agents dans un environnement commun, mais une réelle organisation avec des règles sociales et des interactions permettant la coopération et la collaboration pour la résolution de problèmes que les systèmes centralisés ne pourraient pas résoudre.

Les systèmes multi-agent sont l'une des dernières générations de systèmes informatiques intelligents. Émergeant de la recherche en intelligence artificielle distribuée dans les années quatre-vingt, les systèmes multi-agent constituent aujourd'hui une grande part de la recherche et du développement de l'intelligence artificielle et de la programmation distribuée.

> ***Un système multi-agent*** *est défini comme un ensemble d'agents autonomes ayant un ou plusieurs buts à accomplir et qu'ils œuvrent à atteindre en exécutant concurremment une ou plusieurs tâches.*

La notion d'agent, sur laquelle se fondent les systèmes multi-agent, s'est vue donner des définitions par plusieurs auteurs [68] [101] [95]. Ce qu'il faut en retenir c'est qu'un agent, contrairement à d'autres programmes, doit au moins simultanément être :

1. *situé* : il perçoit le monde/environnement dans lequel il se situe ;
2. *autonome* : il prend des décisions et agit sur son environnement en vue d'atteindre son objectif ;
3. *interactif* : il a la capacité d'interagir avec d'autres agents ;

Les modèles d'agents peuvent être classés selon un axe qui va des agents dits purement ***réactifs***, dans lesquels le comportement des agents est décrit uniquement sous la forme de règles "perceptions!action", aux agents ***cognitifs*** qui ont une représentation globale de leur environnement et des autres agents avec lesquels ils communiquent et qui s'organisent autour d'un mode social d'organisation, en passant par les agents ***hybrides***, combinaison des deux précédentes catégories.

Les SMA constitués d'agents cognitifs ou hybrides utilisent des ACL (Agent Communication Language) pour la communication inter-agent. Les principaux ACL utilisés, i.e. KQML [129] et FIPA-ACL [48], sont fondés sur la théorie des actes de langage [76] [31] et ont été conçus pour l'échange d'information, de connaissances ou encore de services entre agents. C'est la communication entre agents qui fait qu'un groupe d'agents forme un SMA. La communication permet la coopération et la coordination entre agent [68] [93].

3. Coordination d'agents autonomes

Partageant un espace commun, usuellement appelé environnement, les agents sont amenés à coopérer pour s'aider mutuellement dans l'accomplissement de leurs objectifs, partager leurs ressources, résoudre les conflits éventuels ou les éviter, etc. Ceci amène les agents à se coordonner afin de pouvoir évoluer en complète synergie dans cet environnement [120]. En conséquence, des interactions prennent place afin que ces agents déterminent l'organisation la plus appropriée (qui fait quoi ?) et communiquent les résultats de leur coopération (quand ? et à qui ?).

> ***La coordination*** *est un processus d'agencement et de répartition des actions d'un système distribué en vue d'atteindre un objectif déterminé ou d'obtenir une meilleure rentabilité des composants du système.*

Selon Malone et Crowston, la coordination peut également être définie comme étant la gestion des interdépendances entre les activités [137]. Ces interdépendances peuvent concerner des ressources partagées, des contraintes de simultanéité, des relations entre producteur/consommateur, des tâches/sous-tâches, etc.

Dans ce contexte, le rôle d'un agent est de construire des solutions, seul ou en coopération avec d'autres agents, qui tiennent compte des interdépendances et des contraintes spécifiques au problème traité et qui soient en plus en mesure d'assurer une cohésion avec l'ensemble des agents qui partagent son environnement.

Au vu de la contrainte de distribution dans les systèmes multi-agent, c'est-à- dire qu'il n'existe aucun contrôle centralisé et que les agents sont totalement autonomes dans leur choix d'actions et de décisions en général, la coordination est ainsi assurée de façon complètement distribuée au travers de mécanismes tels que les négociations automatiques [120] [79].

Par ailleurs, il existe actuellement diverses formes et mécanismes de coordination; de la coordination implicite, comme dans les communautés biologiques (par exemple, les fourmis), à la coordination explicite à travers des mécanismes de raisonnement formalisés et surtout plus élaborés. Ces mécanismes recourent à l'utilisation de la communication explicite par envoi de messages dont la syntaxe et la sémantique sont régies par des standards de communication tels que KQML [129], FIPA ACL [48], etc.

4. Modèles de coordination d'agents, et applications à la composition de services

La coopération multi-agent, en vue d'engendrer un comportement global cohérent du SMA, requiert des mécanismes élaborés de coordination afin d'éviter des conflits potentiels et de favoriser la synergie des activités des agents [7]. La coordination multi-agent fait souvent référence au processus qui contrôle la prise de décision et guide le comportement global inhérent à l'exécution d'une collection d'agents.

On distingue deux grandes classes de modèles de coordination :

Les modèles orientés tâches Il s'agit de modèles applicables dans un contexte de résolution distribuée de problèmes (Distributed Problem Solving). Dans ces modèles de coordination on présuppose l'existence d'un but global partagé par un groupe d'agents au sein d'une organisation.

C'est par exemple le cas de la métaphore des agents déménageurs, où il est question d'un ensemble d'agents (A_1;A_2; : : :) en charge d'assurer le déménagement d'une maison (tâche T) constituée de plusieurs pièces. Le but global des agents est alors d'accomplir la tâche T, et le but individuel de chaque agent A_i est d'accomplir une sous-tâche T_j selon l'allocation d'un gestionnaire.

Les modèles orientés agents Il s'agit de modèles applicables dans le contexte d'agents (semi-) autonomes ne partageant pas forcément un but global sinon la cohérence et la viabilité du système. Dans ce cas de figure, les agents en tant qu'entités individuelles cherchent à optimiser leurs propres objectifs ou utilités (par exemple, gagner le maximum dans une situation de compétition des agents déménageurs).

Ces deux classes de coordination reposent sur les mécanismes détaillés dans les modèles que nous présentons ci-après.

Notons d'ores et déjà que les modèles de coordination orientés tâches correspondent mieux aux caractéristiques du problème de la chorégraphie de services. Nous présentons alors les principales approches de coordination multi-agent et donnons des exemples de travaux les ayant employées pour la composition de services.

4.1. Structures organisationnelles

Une structure organisationnelle est une distribution fonctionnelle (de compétences, de rôles), spatiale (affectation à des régions délimitées) et temporelle d'agents.

Une organisation peut être statique ou dynamique. Elle est dite statique lorsqu'elle fige *a priori*, lors de la conception du SMA, les comportements des agents ainsi que les règles de comportement qu'ils doivent respecter pour éviter des conflits potentiels. Ainsi, dans une coordination par règlementation, plusieurs conflits sont résolus *a priori*.

Une organisation est dite dynamique lorsque ses fondements peuvent changer au cours du temps (ex. formation et dissolution de coalitions). Le problème de coordination dans ce cas consiste à spécifier l'organisation et la répartition dynamique des tâches à résoudre, la sauvegarde des compétences et connaissances du système lorsque l'organisation change et la représentation des connaissances organisationnelles [81].

La formation de coalitions

Un domaine très significatif pour les organisations dynamiques est celui de la formation de coalitions dans les SMA. Les coalitions sont des regroupements ponctuels d'agents compétitifs pour la réalisation d'un projet commun [110]. Une coalition peut être définie comme une organisation à court terme basée sur des engagements spécifiques et contextuels, ce qui permet aux agents de bénéficier de leurs compétences respectives. Dans un contexte économique par exemple, plusieurs compagnies s'allient virtuellement pour répondre à des offres nécessitant des compétences variées.

Le principe de la formation de coalition a été proposé pour l'allocation de tâches pour la résolution de problèmes. Les travaux du domaine ont porté sur la recherche de consensus par formation de coalitions [121] [108] [53]. Par exemple, dans [121], l'approche consiste à coordonner des agents lors de l'allocation de tâches.
Les agents considérés peuvent être coopératifs ou compétitifs. Cette approche se fonde sur une agrégation multicritère basée sur des techniques d'aide à la décision.
L'approche décrite dans [53] propose quant à elle un algorithme de consensus qui privilégie les libertés des agents où chacun peut changer de stratégie durant la phase de recherche du consensus et peut n'échanger que ses préférences.

Pour la composition de services

Plusieurs travaux exploitant les méthodologies agent et SMA pour la composition de services proposent d'utiliser des mécanismes de formation de coalitions.

L'approche de [139] propose par exemple de composer des services à travers la coalition d'agents médiateurs qui ont pour rôle de réaliser les tâches requises par les services. Les besoins à satisfaire sont exprimés en terme de contraintes (ex. budget=1500$) et de préférences (ex. petit déjeuner continental). La décomposition de ces besoins est ensuite faite au niveau des agents médiateurs qui consultent une ontologie de tâche [117] pour déduire qu'une tâche à réaliser est complexe et implique la réalisation de plusieurs sous-tâches.

> ***Une tâche*** est dite ***complexe*** si elle nécessite l'utilisation des capacités de plusieurs agents.

Dans l'approche de [139], les agents contractent en priorité les agents crédibles et de confiance, c'est-à-dire ceux dont ils savent qu'ils ont auparavant déjà effectué le type de tâches demandées.

Aussi, les travaux de [60] emploient un agent utilisateur qui publie dans un *blackboard* le but de composition à satisfaire. Ce but contient la description d'un workflow générique référençant des "types" de services (ex. service de location de voiture, d'hôtellerie, etc). Les agents services existants, au delà d'une deadline d'inscription, récupèrent alors du *blackboard* les sous-buts à satisfaire, et procèdent à un appariement entre les types de services recherchés et les services qu'eux-mêmes proposent afin d'évaluer s'ils peuvent ou non contribuer à une coalition. Chaque agent service peut ensuite proposer ou accepter la demande d'un autre agent service pour former une coalition. Pour qu'un nouvel agent service rejoigne une coalition, il doit être sujet au vote de tous les agents services la formant.

4.2. Planification

L'approche de coordination par planification distribuée s'intéresse aux interactions qui existent entre les plans des agents [46]. Dans ce type d'approches, la coordination peut s'exprimer par la résolution de conflits entre les activités des agents. Celle-ci peut être vue comme un processus de négociation [54] au cours duquel les agents doivent parvenir à un consensus, un processus d'argumentation [124] où chaque agent essaie de convaincre les autres du bien-fondé de sa proposition, une synchronisation entre les plans [3] ou une fusion de plans [114].

Pour la composition de services

Nous avons déjà présenté plusieurs approches de composition utilisant des techniques de planification dans le deuxième chapitre de ce manuscrit.

4.3. Protocoles d'interaction

Un protocole d'interaction permet aux agents d'avoir des conversations, sous forme d'échanges structurés de messages [96]. Il décompose une tâche en sous-tâches et les distribue, suivant une stratégie donnée, aux agents du système. Cette stratégie doit par exemple permettre [41] : d'allouer les tâches aux agents en fonction de leurs capacités, d'allouer les responsabilités aux agents de manière couvrante pour assurer la cohérence globale, etc.

On compte d'ores et déjà plusieurs protocoles d'interaction tels que le CNP (Contract Net Protocol) [119] qui relève des réseaux contractuels et qui propose des séries épisodiques pour des actes d'intercommunication (annonces, enchères, messages de récompense) ; le protocole Rubinstein [5] alternant proposition et contre-proposition, FIPA-Request [49] permettant aux agents d'accepter ou de refuser d'exécuter une requête ou encore les protocoles de ventes aux enchères [47] [14].

Pour la composition de services

À notre connaissance, il existe peu de travaux proposant des protocoles d'interaction conçu spécifiquement pour le problème de la composition de services.

Ceux qui s'en rapprochent sont par exemple les travaux de [128] qui transcrivent, pour la négociation entre services, le protocole Rubinstein dans la description XML des services, ou encore le protocole WS-Agreement [1] qui spécifie des conversations à deux étapes, une offre suivie d'un accord, pour l'établissement de contrats et d'accords entre un service fournisseur et un consommateur.

Enfin, l'approche de [122] sépare la description et la composition des services de la spécification des formats de données et du protocole de transport (notamment décrits en WSDL). Les contributions de ces travaux consistent en la proposition d'un langage de description de services

ainsi qu'un protocole de composition. L'approche conçue se base sur un agent en charge de contacter des services à propos des tâches qu'il souhaite accomplir. Chaque fois qu'un service accepte de réaliser une tâche demandée, l'agent met à jour la construction de son workflow. Celui-ci est ensuite exécuté pour l'accomplissement de la tâche globale.

4.4. Discussion

Après analyse des différents modèles de coordination, il apparaît que les principales limites de leur application au problème de la chorégraphie dynamique de services est l'hypothèse faite que les tâches à satisfaire sont décomposées en amont de la coordination. Au contraire, une tâche devrait pouvoir être décomposée sur la base des fonctionnalités disponibles (et non en amont) en sous-tâches interdépendantes en terme d'exécution.

Cela requiert aux agents, avant tout, de pouvoir raisonner sur leurs actions afin de déterminer si une tâche est complexe et nécessite ainsi le concours d'autres agents pour son accomplissement. Or, les plates-formes de développement d'agents actuelles, telles que JADE [78], DIMA [143] ou JACK [77], ne permettent pas de doter les agents de capacités leur permettant de raisonner sur leurs propres compétences. De même, les langages de communication entre agent actuels (ACL) [48] [129] ne permettent pas d'exprimer des questions ou des assertions à propos des capacités des agents [2] [111] [94]. Enfin, les protocoles d'interaction existants ne prennent pas en compte les dépendances entre les différentes sous-tâches issues de la décomposition d'une tâche globale [73], considérant que la tâche a été préalablement décomposée.

5. Coordination de plans

Dans ce paragraphe, nous nous intéressons à la coordination coopérative de plans. On parle alors de planification distribuée. Le terme de planification distribuée est toutefois ambigu car il n'explicite pas ce qui est distribué. En effet, les plans peuvent être construits de manière centralisée puis distribués aux agents, ou bien chaque agent peut construire localement son propre plan puis le coordonner de manière distribuée. Dans le premier cas, seule l'exécution du plan est distribuée. En revanche, dans le second, la synthèse de plans, le processus de coordination ainsi que l'exécution sont réalisés de manière complètement distribuée.

Nous présentons les mécanismes de coordination centralisée de plans, puis les trois grands mécanismes de coordination décentralisée de plans qui sont la planification partiellement globale, la synchronisation distribuée de plans et la fusion incrémentale de plans.

Coordination centralisée de plans. La coordination par planification centralisée repose toujours sur l'existence d'un agent coordinateur. Cet agent centralise l'ensemble des plans des agents du système et résout les conflits potentiels entre leurs activités en introduisant des actions de synchronisation. L'agent coordinateur peut : soit planifier pour l'ensemble des agents et, dans ce cas, il doit décomposer le plan global en sous-plans synchronisés pouvant être exécutés par les agents ; soit chaque agent peut planifier localement et, dans ce cas, le rôle de l'agent coordinateur se limite à la synchronisation des plans reçus. Ces deux approches peuvent être résumées ainsi :

– ***approche par synchronisation*** : la synchronisation de plans s'appuie sur la notion de plan non-linéaire. En effet, un plan non-linéaire [36] est un plan dont les actions ne sont pas strictement ordonnées. Par conséquent, il est possible de contraindre les relations d'ordre entre les actions en ajoutant des actions de synchronisation et garantir ainsi que le plan sera exempt de conflits ;

– ***approche hiérarchique*** : Corkill [21] propose une implémentation distribuée de noah (Nets of Action Hierarchies) [38]. Cette implémentation est fondée sur l'allocation de sous-buts d'un certain niveau à des agents. Les plans sont synchronisés niveau par niveau, c'est-à-dire que les agents négocient et résolvent les conflits de chaque niveau avant de raffiner le plan à un niveau inférieur.

Planification partiellement globale. La planification partiellement globale, PGP « Partial Global Planning » [35] est un schéma générique de coordination distribuée pour la résolution de problèmes qui s'appuie sur une architecture de type tableau noir (zone de mémoire commune et partagée entre les agents). Ce schéma de coordination ne vise pas à résoudre des conflits entre les plans des agents, mais à permettre un gain de performance en terme de temps de calcul (i.e., en réduisant l'inactivité de certains agents, en minimisant les tâches redondantes). Chaque agent est capable de percevoir les modifications de la zone du tableau dont il est responsable, et d'interagir avec les autres pour confronter ces connaissances. Les interactions entre agents reposent sur la diffusion d'une structure de données, appelée *plan partiel global*.

Synchronisation distribuée de plans. Les travaux basés sur la synchronisation voient les activités des agents comme des processus concurrents qui doivent être synchronisés. Dans ce cadre, El Fallah Seghrouchni et Haddad [3] proposent un algorithme distribué qui permet de réaliser une telle coordination. Ces travaux s'appuient sur une représentation du monde par états de manière similaire au formalisme STRIPS [115]. La production des plans est réalisée par les agents en fonction de leurs buts locaux. Chaque agent est alors responsable de la synchronisation de ses plans avec les autres agents. Pour qualifier un plan local validé dans un contexte multi-agent, i.e., exempt de conflits, El Fallah Seghrouchni introduit la notion de « plan structuré ». La construction d'un plan structuré se décompose en deux phases. Tout d'abord, l'agent élabore un plan de manière indépendante sans tenir compte des plans produits par les autres agents. Puis, il essaie de synchroniser son plan en le diffusant aux autres agents.

Fusion incrémentale de plans. L'insertion incrémentale de plans (« Plan- Merging Paradigm », PMP) [113] permet aux agents de planifier, de coordonner leurs plans et de les exécuter d'une manière distribuée et incrémentale sans être contraints de suspendre l'exécution des plans précédemment coordonnés. PMP permet à un grand nombre d'agents de concilier leurs plans individuels, afin d'éliminer tout risque de conflit durant la phase d'exécution. La validation d'un plan individuel est réalisée par insertion incrémentale dans un graphe de contraintes établi à partir des différents plans en cours d'exécution. Cette insertion transforme le plan individuel en un plan coordonné exécutable, tout en imposant des contraintes d'ordre entre les plans en cours d'exécution et le plan à valider.

Les différents mécanismes présentés sont des mécanismes de coordination et de coopération dans le cadre des systèmes multi-agent. Ils permettent de garantir l'intégrité fonctionnelle du système, i.e., que les activités des agents soient exemptes de conflits, mais ils n'ont pas pour objectif de produire un plan global. Dans le paragraphe suivant, nous présentons un mécanisme permettant cette production d'un plan global : la ***synthèse dialectique de plans***.

5.1. Synthèse dialectique de plans

La synthèse dialectique de plans est un processus collectif de délibération permettant à un groupe d'agents de raisonner sur un but à atteindre. Elle repose sur une approche conventionnelle du dialogue [92], c'est-à-dire que tous les agents impliqués dans la synthèse doivent respecter un

certain nombre de règles. L'approche proposée par Pellier [29] est basée sur des cycles de propositions et contre-propositions des agents, appelés cycles de conjecture - réfutation. Cette approche introduit le concept de conjecture. Une conjecture peut se définir comme un plan sous hypothèses, i.e., un plan qui peut s'exécuter si certaines conditions sont vérifiées (cf. § 6.2).

Définition. 3.1 (Conjecture)

Une conjecture est un tuple

$$\chi = (A, <, I, C)$$

- $A = \{a_0, \ldots, a_n\}$ est un ensemble d'actions ;

- $<$ est un ensemble de contraintes d'ordre sur les actions A de la forme $a_i < a_j$, i.e., a_i précède a_j ;

- I est un ensemble de contraintes d'instanciation portant sur les variables des actions A de la forme $x = y$, $x \neq y$, ou $x = cst$ tel que $cst \in D_x$ et D_x est le domaine de x ;

– C est un ensemble de liens causaux de la forme $a_i \xrightarrow{p} a_j$ tels que a_i et aj sont deux actions de A, la contrainte d'ordre $a_i < a_j$ existe dans $<$, la propriété p est un effet de a_i et une pré-condition de aj et finalement les contraintes d'instanciation qui lient les variables de a_i et de a_j portant sur la propriété p sont contenues dans I.

Principes. Contrairement à la formalisation « classique » des dialogues argumentatifs [80], les interactions entre agent prennent la forme d'un débat. Chaque agent possède un tableau de démonstration dans lequel il enregistre les propositions des autres agents (figure 3.1). Ce tableau peut être vu comme un graphe orienté dont les nœuds représentent des conjectures. Il définit l'état du dialogue mais également l'espace de recherche co-construit par les agents. Les agents s'échangent des messages au travers d'actes de dialogue (tableau 3.1) de deux types : les actes de niveau informationnel qui permettent aux agents d'échanger des informations sur les conjectures contenues dans le tableau de démonstration ; et des actes de niveau contextuel qui permettent de modifier le contexte de l'interaction.

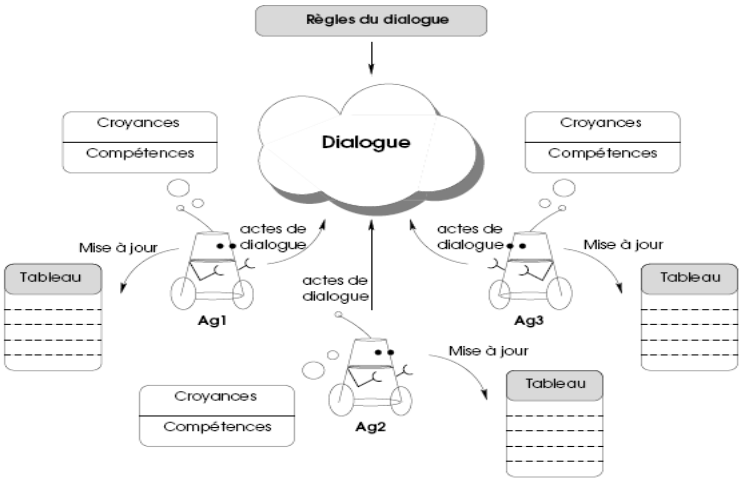


Figure 3.1. Schématisation du modèle de construction dialectique de plans

Cycle de conjecture - réfutation. Le mécanisme guidant les interactions entre les agents est la boucle de *raisonnement par conjecture - réfutation* (figure 3.2). Cette boucle, qui s'inspire des recherches dans un espace de plans, dirige les propositions des agents. A chaque itération, un agent choisit une conjecture et cherche à lui appliquer un opérateur dialectique faisant ainsi progresser collectivement l'élaboration d'un plan solution. L'exploration de l'espace des conjectures, représenté par les tableaux de démonstration des agents, est obtenue en appliquant itérativement la procédure suivante :

1. choix d'une conjecture χ non encore explorée ;
2. choix de l'opérateur dialectique à appliquer à χ ;
3. calcul des modifications à apporter à χ ;
4. soumission des modifications aux autres agents.

Niveau	Actes
Informationnel	*refine, refute, repair, failure*
Contextualisation	*prop.solve, prop.failure, prop.success, ack.failure, ack.success*

Tableau 3.1. Actes de dialogue classés par niveau

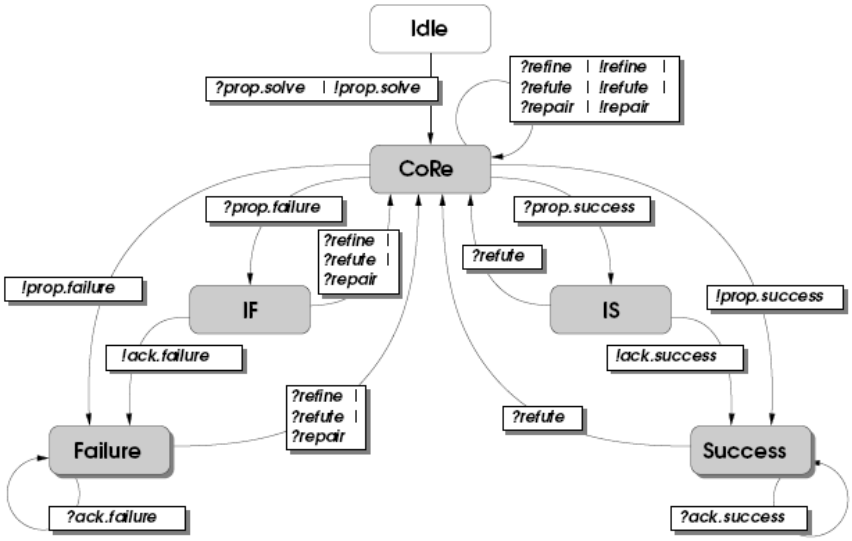


Figure 3.2. Automate de contextualisation du dialogue

Le raisonnement se poursuit tant que l'espace de recherche ne contient pas au moins une conjecture qui respecte les propriétés d'un plan solution : les ensembles de contraintes d'ordre et d'instanciation doivent être cohérents ; la conjecture ne doit plus formuler d'hypothèses et il doit exister une linéarisation du plan atteignant le but.

Les opérateurs dialectiques. Les mécanismes permettant aux agents de s'échanger des informations sur les conjectures sont le raffinement, la réfutation et la réparation :

– l'opérateur dialectique de raffinement est utile pour démontrer qu'une hypothèse peut être vérifiée. Le raffinement peut se faire par ajout d'un lien causal entre deux actions, s'il existe déjà au sein de la conjecture une action produisant l'effet correspondant à l'hypothèse émise, ou par ajout d'une sous-conjecture, i.e., une ou plusieurs actions dont le but est de démontrer la validité de l'hypothèse ;

– l'opérateur dialectique de réfutation cherche à démontrer qu'une conjecture est incorrecte, i.e., qui contient des séquences d'actions non valides (conflits d'accès aux ressources, hypothèses non vérifiées, ...) ;

– l'opérateur dialectique de réparation calcule les modifications à apporter aux conjectures lorsque celles-ci ont été préalablement réfutées. Une conjecture peut être réparée par ajout de contraintes d'ordre, par modification des contraintes d'instanciation, ou encore par ajout d'une conjecture.

5.2. Planifier sous hypothèses

La synthèse dialectique de plans repose sur la capacité des agents à produire des conjectures, i.e., des plans qui peuvent être exécutés si certaines propriétés du monde sont vérifiées. Le modèle de planification sous hypothèses (*Assumption-based Planning*) [27] est le mécanisme qui permet de produire les raffinements nécessaires à la démonstration des propriétés supposées par les agents.

5.2.1. Modèle de la planification sous hypothèses

Le modèle de planification sous hypothèses, sur lequel est fondée la production des raffinements, est basé sur le modèle de la planification hiérarchique. Ce type de planification emploie un modèle d'actions qui suppose que toutes les pré-conditions des opérateurs *o* sont satisfaites dans l'état *s* précédant son application. Par définition, ce modèle ne permet pas le déclenchement d'opérateurs dont certaines pré-conditions ne sont pas vérifiées. Or, dans le cadre de la planification sous hypothèses, ce sont justement ces pré-conditions particulières qu'il est intéressant de considérer, car elles expriment les propriétés du monde manquantes. Mais il n'est pas intéressant de considérer toutes les conjectures car seuls les raffinements formulant un petit nombre d'hypothèses ont une chance d'aboutir à un plan solution. Par conséquent, planifier sous hypothèses consiste à construire un plan tout en minimisant le nombre d'hypothèses introduites.

Définition. 3.2 (Hypothèse)

Soit une conjecture $\chi = (A, <, I, C)$. Une hypothèse formulée par χ est définie comme une pré-condition *p* d'une action $a_j \in A$ telle que pour toutes actions $a_i \in A$, le lien causal $a_i \xrightarrow{p} a_j \notin C$

Exemple de méthode faisant une hypothèse (Nous représentons les hypothèses à l'aide d'astérisques).

;; move a container *c* from *x* to *y* (with a robot *r*)

move-container(*c*, *x*, *y*)

precond: at(*c*, *x*), *at(*r*, *x*)

reduction: load(*r*, *c*, *x*), move(*r*, *x*, *y*), unload(*r*, *c*, *y*)

La méthode fait l'hypothèse que, au moment du chargement du container *c*, le robot *r* sera à l'endroit *x*.

5.2.2. Génération des hypothèses

La génération des hypothèses repose sur le fait qu'il n'est pas nécessaire que l'ensemble des pré-conditions soient vérifiées dans l'état initial pour qu'un opérateur ou une méthode soit applicable. Les pré-conditions non vérifiées forment alors un ensemble particulier de propriétés du monde : les hypothèses.

Le calcul des hypothèses repose sur l'identification de l'ensemble de substitutions possibles pour appliquer un opérateur ou une méthode. Soit *precond* les pré-conditions d'un opérateur ou d'une méthode, *s* les croyances d'un agent et σ une substitution définissant un ensemble de contraintes d'instanciation, l'algorithme calcule les substitutions possibles à partir de *s*. Puis, pour chaque substitution, l'algorithme teste si $\sigma(p)$ existe ($p \in precond$). Si ce n'est pas le cas, alors $\sigma(p)$ est une hypothèse.

5.2.3. Production des raffinements

L'algorithme de planification sous hypothèses s'appuie sur une recherche dans un espace d'états. Cet espace d'états est stocké dans un arbre appelé arbre de raffinements. Chaque nœud de l'arbre représente un état du monde atteignable après l'application d'une action. Les arêtes représentent les transitions possibles entre les différents états du monde. Une transition se caractérise par l'opérateur, ou la méthode, appliquée ainsi que les éventuelles hypothèses nécessaires à son exécution.

La procédure qui produit les raffinements se découpe en deux phases : l'expansion de l'arbre de raffinements et l'extraction de la conjecture.

La construction de l'arbre de raffinements consiste à décomposer récursivement les actions complexes contenues dans les nœuds de l'arbre jusqu'à ce qu'une feuille soit produite, i.e., un nœud possédant une séquence d'actions vide. Cette feuille représente l'état du monde qui sera atteint après l'exécution du raffinement.

6. Conclusion et analyse

Nous avons vu dans la première partie de ce chapitre que l'idée de présenter des services web sous forme d'agents était déjà entamée dans plusieurs travaux de recherche et ce à cause des limites que présente le paradigme de services web. Nous avons étudié les différents travaux liés à la coordination multi-agent. Nous avons ainsi passé en revue les principaux modèles de coordination ainsi que les travaux en composition de services implémentant ces méthodes dans leurs propositions.

Après analyse de ces différents travaux, il en ressort que la difficulté pour la conception d'une approche dynamique de chorégraphie de services réside principalement dans :
– La formalisation du but de composition. La plupart des approches existantes ne traitent pas ce problème puisqu'elles s'appuient sur un workflow, écrit par un expert dans un langage tel que WS-BPEL, destiné à être instancié et exécuté d'une part, et manipulé par des spécialistes en technologie de l'information d'autre part.
– La découverte dynamique des services candidats à la composition par rapport au but de composition. En effet, les approches existantes décomposent en amont le but de composition qu'ils décrivent à travers un plan spécifiant les profils des services à découvrir. Les méthodes de découverte se réduisent alors à une simple procédure d'appariement sémantique entre les profils abstraits et les services concrets trouvés.
– La conception d'un modèle dynamique d'interaction de services, qui tient compte à la fois du but de la composition et des fonctionnalités des services.
En effet, les buts de composition, décrits à travers des plans dans les approches actuelles, spécifient le profil des services à composer ainsi que la séquence de leur invocation. Pour implémenter une coordination dynamique, il s'agira d'abord pour les services de prendre dynamiquement part à la décomposition et l'allocation du but de composition et ensuite utiliser des mécanismes de coordination tenant compte de leurs fonctionnalités afin d'interagir de manière flexible de couvrir adéquatement ce but.

Par ailleurs, notre étude nous a conduits à la conclusion que les modèles de coordination multi-agent étaient un bon candidat pour pallier les limites des services web en général, et de la chorégraphie de services en particulier car ils apportent des solutions et modèles pour tenir

compte de plusieurs contraintes telles que la distribution des services, l'ouverture et la flexibilité de leur environnement. Cependant, certaines des difficultés précédemment soulevées restent insatisfaites, notamment : (a) la modélisation de buts de composition complexes, où il s'agit de formaliser des besoins complexes d'utilisateurs pouvant regrouper des requêtes sur des services, des contraintes, et des interdépendances entre les requêtes, (b) la décomposition et l'allocation dynamique des tâches en fonction des compétences des agents/services découverts.

C'est sur ces points que notre travail porte. Nous avons pour objectif de proposer un modèle de composition orienté tâche, i.e. applicable dans un contexte de résolution distribuée de problèmes et dans lequel un but global à satisfaire (i.e. les besoins de l'utilisateur) sera décomposé en des sous buts à attribuer à un ensemble d'agents capables de les réaliser. C'est pour cette raison, la deuxième partie de ce chapitre, a été plus particulièrement consacré à traiter les aspects associés à la coordination de plans.

CHAPITRE 4

UNE ARCHITECTURE BASEE AGENT POUR LA COMPOSITION DE SERVICES WEB

Les systèmes multi-agent sont l'une des dernières générations de systèmes informatiques intelligents. Émergeant de la recherche en intelligence artificielle distribuée dans les années quatre-vingt, les systèmes multi-agent constituent aujourd'hui une grande part de la recherche et du développement de l'intelligence artificielle et de la programmation distribuée. Ces derniers offrent de nombreux avantages pour la conception et le contrôle des systèmes complexes. Cependant leur développement reste difficile. Les difficultés de développement inhérentes aux systèmes multi-agent, sont la conséquence de la diversité et de la complexité des concepts d'agent et des mécanismes de coordination, d'interaction, d'organisation, etc. Cette complexité fait que l'utilisation de la plupart des outils et des plates-formes existants est difficile aux non spécialistes en multi-agent [130].

Nous donnons dans ce chapitre une description générale de notre modèle de composition dynamique de services web et présentons l'architecture combinant technologies agent et services web sur laquelle elle s'appuie. Nous commençons la description de notre architecture par la motivation de l'approche agent. Ensuite, nous décrivons les structures internes des différents agents impliqués, ainsi que leurs rôles dans le processus de composition. Nous discutons également comment s'effectue la communication entre les différents types d'agent.

1. Motivations de l'approche agent : entre coordination d'agents et composition de services

Les services web partagent avec les systèmes multi-agent plusieurs propriétés dont les méthodologies agents tiennent explicitement compte dans leur ébauche de solutions. De telles propriétés sont [38][18] :

La distribution géographique et/ou logique d'entités, données ou informations hétérogènes ;

La complexité du problème global à résoudre, celui-ci n'étant manipulable que par des stratégies heuristiques utilisant des données ou connaissances locales ;

La flexibilité des interactions il n'y a pas d'affectation a priori des tâches et les mécanismes de résolution de problèmes ne sont pas préalablement assignés à telle ou telle autre entité ;

L'environnement dynamique nécessitant, pour la résolution de problèmes, des entités réactives et adaptatives ;

L'ouverture il n'est pas possible de donner une spécification complète du problème à résoudre ni de définir une fonction d'utilité globale.

En outre, les services web sont souvent des interfaces de programmes orientés objet (développés via J2EE ou .NET) traités pour générer des descriptions WSDL afin de pouvoir communiquer avec des messages SOAP. Ils bénéficient alors des caractéristiques d'abstraction du paradigme orienté objet (encapsulation, héritage, passage de messages, etc). L'une des majeures évolutions de SOA et SOC consiste à remplacer le noyau orienté objet des services par un noyau orienté agent [20][98] [99] [10]. Les services tiendraient alors compte du contexte d'ouverture et de distribution de l'environnement, bénéficieraient des caractéristiques de proactivité, flexibilité et d'adaptativité des agents, et deviendraient de plus réellement interactifs. En outre, comme les agents peuvent potentiellement participer à des interactions complexes tout en maintenant la coordination avec leurs accointant, les services pourraient être dotés des pré-requis pour la modélisation d'une collaboration dynamique entre services.

En conséquence, les systèmes multi-agent semblent alors être particulièrement bien adaptés à la modélisation des problématiques (de distribution, d'ouverture, de flexibilité et de collaboration des services) sous-tendues par la composition dynamique de services. C'est la raison pour

laquelle l'objet de notre étude va porter sur l'étude de solutions SMA pour la modélisation des problématiques de composition de services. Plus précisément, notre travail a pour but de définir un mécanisme de coordination multi-agent, supportant des interactions flexibles des agents pour la satisfaction de besoins en services énoncés par des utilisateurs, dans les contextes ouverts et décentralisés du web et de l'intelligence artificielle et plus particulièrement de la planification.

2. Définition du problème

La composition de services Web a pour but de produire une description spécifiant une séquence d'appels à des services ainsi que la façon dont ces services sont liés entre eux dans le but de résoudre un objectif donné. Cette opération se déroule en trois étapes :

1. les services Web sont recherchés et sélectionnés à partir d'un annuaire UDDI (dans notre cas du registre ebXML) en fonction des besoins à réaliser ;
2. la composition est effectuée en utilisant la description (syntaxique ou sémantique) des services sélectionnés ;
3. une description du service composite, i.e., l'enchaînement des appels aux services sélectionnés, est créée.

La description de la composition permet alors d'exécuter les services Web et ainsi réaliser les objectifs de l'utilisateur. En effet, de la même manière que la description d'un service spécifie la façon d'y accéder, la description du service composite explicite comment exécuter les différents services ainsi que la façon dont les données produites par un service seront consommées par les services suivants.

Dans cette thèse, nous traitons beaucoup plus en détail le deuxième point de l'algorithme de composition, i.e., trouver l'ordre d'exécution des services Web présélectionnés à partir de leur description sémantique pour réaliser un but fixé par l'utilisateur (l'entreprise).

Nous définissons un agent pour chaque service retenu dont le but est de simuler l'exécution du service. Cet agent est une entité autonome qui contient un *planificateur* et qui est capable de communiquer avec les autres agents dans le but de co-construire un plan d'exécution des services Web. Le modèle de composition que nous proposons s'appuie sur une architecture multi-agent dont les *agents manager* représentent les services Web (Figure 4.2). Les *agents manager* et

l'agent compositeur sont initialisés avec la description sémantique du service et avec une base de connaissances. La description OWL-S permet de définir le domaine de planification de l'agent (manager ou compositeur) tandis que la base de connaissances (issue d'une base de données par exemple) apporte les connaissances qui seront nécessaires à l'agent pour raisonner.

Où nous voulons intégrer l'architecture proposée ?

Puisque nous nous sommes intéressés, dans un premier travail de recherche, à la négociation des paramètres des CPP d'ebXML [107], nous avons choisi le contexte d'ebXML comme continuité à notre domaine de recherche.
L'objectif de notre travail est d'ajouter un composant à la spécification fonctionnelle d'ebXML dont le rôle est de : ***Combiner, sémantiquement, des services Web pour répondre aux besoins de l'entreprise cliente.***

L'ensemble des services web des entreprises utilisant ebXML pour participer à des marchés globaux, est regroupé dans un registre ou répertoire (Registry/Repository). Le problème de la composition des services Web dans un contexte ebXML apparaît: lors de la première phase du scénario de collaboration entre deux partenaires commerciaux (cf. Figure 4.1). Quand une entreprise cherchant un partenaire potentiel consulte le registre, le service demandé avec certains paramètres peut ne pas exister mais en même temps, il peut exister des services, qui combinés ensemble, peuvent répondre à ses besoins.

L'architecture que nous proposons permet de composer des services web, conformément à la spécification fonctionnelle d'ebXML. Cette architecture définit plusieurs niveaux de responsabilité et utilise la planification classique avec synthèse dialectique de plans (planification sous hypothèse).

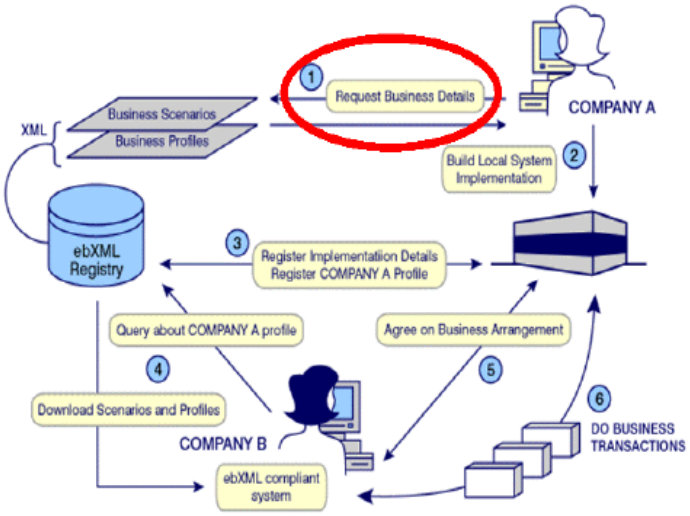


Figure 4.1. Interaction entre deux partenaires selon ebXML[42]

3. Description de l'architecture

Généralement, une architecture exprime la structure fondamentale du système à analyser et à concevoir. L'architecture définit un ensemble de composants fonctionnels, de sous-systèmes ou de modules décrits en termes de leurs comportements et interfaces [88] [109]. Elle définit aussi comment ces composants interagissent et comment ils doivent être interconnectés pour accomplir correctement les buts du système. Ainsi, une description architecturale est principalement requise pour la spécification de la structure du système.

En principe, le terme composant inclut les éléments fonctionnels qui peuvent être des objets répartis ou bien des agents autonomes tel est le cas dans notre travail. En prenant en considération ces définitions, nous allons présenter dans les sections suivantes de ce chapitre les spécifications des différents composants, ainsi que les concepts liés à leur fonctionnement.

3.1. Présentation de l'architecture proposée

Comme le montre la figure 4.2, l'architecture que nous proposons est une architecture à base d'agents qui définit plusieurs niveaux de responsabilité. Quatre types d'agents la composent : *l'agent re-constructeur de requête*, *l'agent manager-raisonneur*, *l'agent compositeur* et un ensemble *d'agents manager*.

Selon les différents types d'agents définis, notre architecture permet de définir plusieurs niveaux de responsabilité:

- La responsabilité de construire sémantiquement la requête utilisateur assurée par l'agent re-constructeur de requêtes en se basant sur l'ontologie globale des BPs (Business Process),
- La responsabilité d'affecter une tâche à l'agent compositeur ou directement à l'agent manager assurée par l'agent manager-raisonneur en se basant sur la vue globale qu'il a sur le système,
- La responsabilité de composer et coordonner l'exécution du service composé assurée par l'agent compositeur.
- Et la responsabilité de représenter un service Web au moment de la composition assurée par l'agent manager.

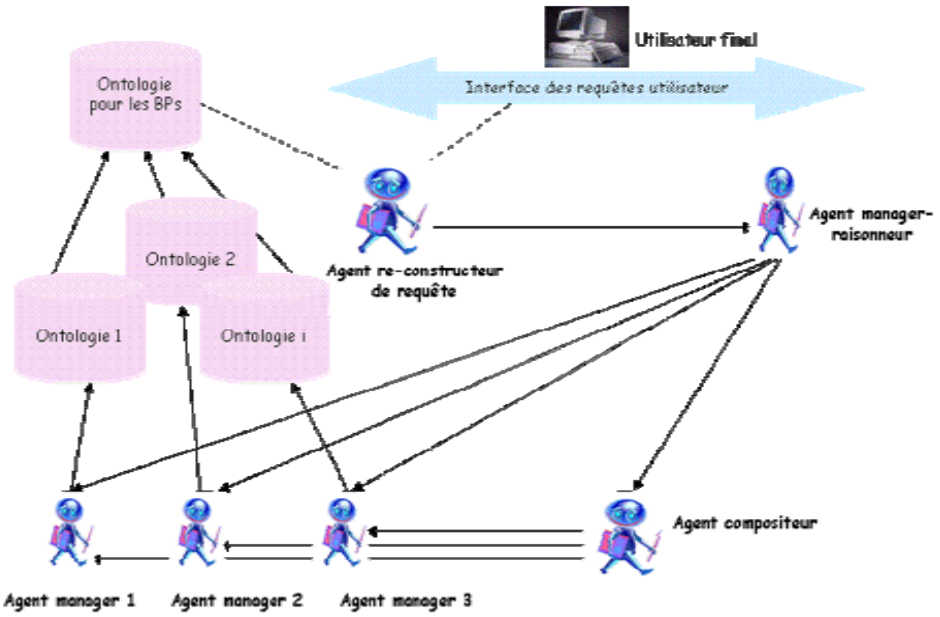


Figure 4.2. Une architecture multi-agent pour la composition des services Web dans un contexte d'ebXML [105]

Un autre composant important permet de spécifier la sémantique des différents services Web existant au niveau du registre ebXML. Il s'agit d'une ontologie globale en langage OWL-S, pour l'annotation sémantique de ces services. Notre architecture dispose également d'un sous ensemble d'ontologies OWL-S.

3.2. Structure des différents composants de l'architecture proposée

Dans cette section, nous présentons la structure interne des différents composants de notre architecture (différents types d'agent et ontologies).

3.2.1. Structure de l'agent reconstructeur de requêtes

Comme le montre la figure 4.3, l'agent reconstructeur est composé de quatre principaux modules :

- Le module de connaissances inclut une base de données et un ensemble d'actions. Ce module intègre une ontologie regroupant les concepts et noms d'actions manipulés par l'agent.

  Le rôle de ce module est de mémoriser les informations, qui sont nécessaires pour le bon déroulement de la phase de reconstruction de requête (cf. Section 5 et chapitre 5). Parmi ces informations se trouvent les classes obtenues du fichier XML ainsi que les relations entre ces classes. Ces informations sont utiles pour une éventuelle réutilisation d'une transformation (XML vers OWL-S).

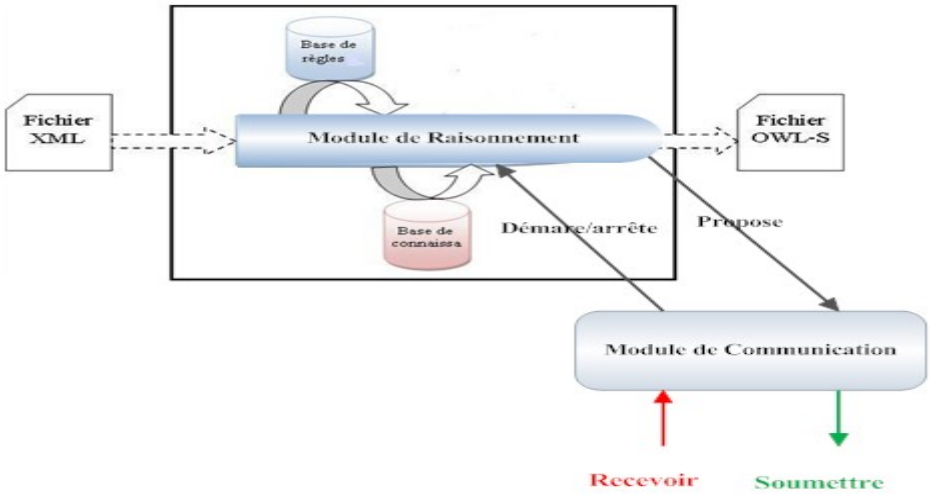


Figure 4.3. Structure de l'agent reconstructeur de requêtes [106]

- Le module incluant l'ensemble des règles que le module de raisonnement utilise pour effectuer les transformations utiles pour obtenir à la fin le fichier OWL-S attendus.

- Le module de raisonnement qui constitue le cerveau de l'agent, il permet de raisonner en utilisant les différentes règles ainsi que sa base de connaissances. Il se base principalement sur deux règles (cf. Chapitre 5).
- La couche de communication permet le transport des messages. Dans notre travail, nous utilisons le langage FIPA-ACL [50] comme langage de communication entre les différents agents. Celui-ci a pour rôle de structurer les messages construits par l'agent. Par ailleurs, les agents peuvent communiquer sur le web via le protocole de transport standard SOAP.

3.2.2. Structure de l'agent manager-raisonneur

L'agent manager-raisonneur est représenté par une structure (cf. Figure 4.4) comportant deux principaux modules : un module de raisonnement et un gestionnaire de dialogue.

- Le module gestionnaire de dialogue permet d'échanger des messages entre l'agent manager–raisonneur et les autres types d'agents (reconstructeur de requête, compositeur ou manager) selon les besoins.
- Le module de raisonnement est le module central de l'agent. Il définit le comportement de l'agent en dirigeant ses interactions (cf. Section 5)

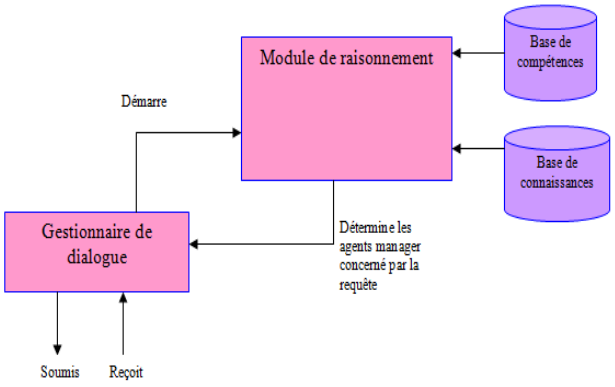


Figure 4.4. Structure de l'agent manager-raisonneur [106]

L'agent manager-raisonneur est initialisé avec la description sémantique (fichier OWL-S) du service web demandé auprès de l'utilisateur. Cette description est envoyée à cet agent depuis l'agent reconstructeur de requêtes.

3.2.3. Structure de l'agent compositeur

L'agent compositeur doit être capable de raisonner sur la tâche qu'il doit accomplir et doit être capable de communiquer avec les autres agents (manager). Il est initialisé avec la description sémantique du but utilisateur à réaliser et le groupe des agents managers obtenus auprès de l'agent manager-raisonneur, ces données constituent les connaissances et le domaine de compétences de l'agent compositeur.

L'agent compositeur [104] est composé de trois modules principaux (cf. Figure 4.5) :
- Le module de raisonnement.
- Le tableau de stockage qui sert de support des conjectures (cf. Chapitre 5) reçues auprès des agents managers.
- Le gestionnaire de dialogue qui permet le dialogue avec les autres agents.

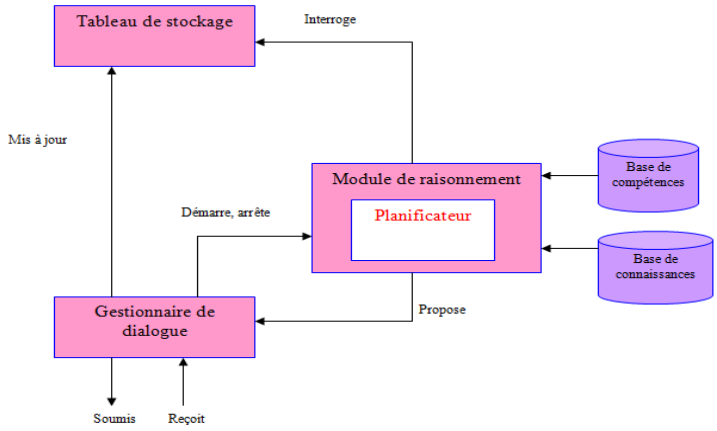


Figure 4.5. Structure de l'agent compositeur [104]

- Le module de raisonnement est le module central de l'agent. Il définit le comportement de l'agent en dirigeant ses interactions. Il s'appuie sur i) le tableau de stockage pour sélectionner les conjectures à raffiner et pour sélectionner le sous-ensemble des agents manager avec lesquels il continuera le processus de raffinement et sur ii) le gestionnaire de dialogue pour soumettre les propositions aux agents.
- Le tableau de stockage sert de support au dialogue en enregistrant les sous-conjectures proposées par les différents agents managers.

- Le gestionnaire de dialogue est le module qui permet d'échanger les propositions entre l'agent compositeur et les agents managers.

3.2.4. Structure de l'agent manager

Pour implémenter l'architecture de l'agent manager représentant un service web, nous reprenons la recommandation du W3C [19] définissant l'architecture d'un service web comme suit :

> "Un service web est vu comme une notion abstraite qui doit être implémentée par un agent concret. L'agent est une entité concrète (logicielle) qui envoie et reçoit des messages, alors que le service est l'ensemble de fonctionnalités offertes".

Dans cette définition, un service est vu comme partie d'un agent. De même, notre architecture dispose d'agents offrant des services. Il s'agit des agents manager.

Un agent manager est initialisé avec la description sémantique du service Web qu'il représente et avec un ensemble de données complémentaires. La description sémantique, au travers du processus de traduction, permet d'élaborer sa base de compétences, i.e., le domaine de planification. Les données complémentaires à la description OWL-S qui sont fournies, permettent d'obtenir la base de connaissances de l'agent manager, i.e., un ensemble de prédicats formant l'état initial du planificateur.
Comme le montre la figure 4.6, un agent manager est composé de deux modules principaux :
- Le module de raisonnement qui contient un planificateur.
- Le module gestionnaire de dialogue ou de communication permet le transport de message.

Les agents peuvent offrir un ensemble de fonctionnalités ou services, via leur base de connaissance, sur lesquels ils peuvent raisonner.

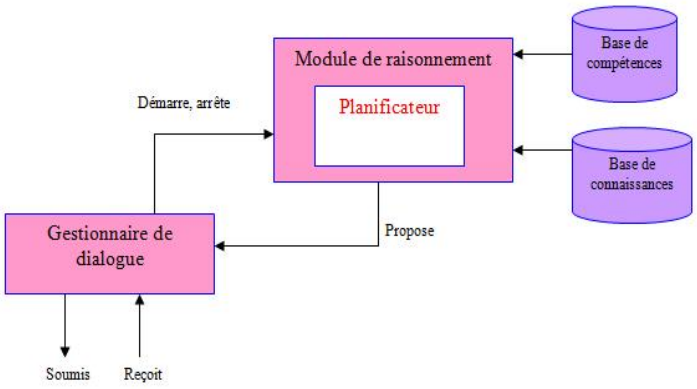


Figure 4.6. Structure d'un agent manager [104]

- Le module de raisonnement est le module central de l'agent. Il définit le comportement de l'agent en dirigeant ses interactions. Il s'appuie sur les messages reçus auprès de l'agent compositeur et le tableau de démonstration de ce dernier pour sélectionner les conjectures (cf. Chapitre 5) à raffiner et sur le gestionnaire de dialogue pour soumettre les propositions à l'agent compositeur et aux autres agents manager impliqués dans l'opération de composition de services web.
- Le module de gestion de dialogue permet d'échanger les propositions entre les agents managers et l'agent compositeur.

3.2.5. Structure des ontologies incluses dans notre architecture

OWL-S est très riche et n'impose que très peu de contraintes sur la manière d'exprimer la sémantique des services web. Puisque notre travail est inspiré de celui de [69] et [28], nous avons fait un certain nombre d'hypothèses et de restrictions sur ce dernier afin de faciliter le passage d'une représentation sémantique vers un plan:

- Les processus simples (Simple Processes) ne sont pas traités. En effet, ils correspondent uniquement à des abstractions de processus atomiques ou composites ;
- OWL-S permet de tenir compte de l'indéterminisme dû à l'exécution d'un service, c'est-à-dire qu'il laisse la possibilité de définir le résultat d'un service Web en fonction de son comportement (succès, erreur ou absence de réponse) via l'utilisation des classes ConditionalEffect et ConditionalOutput d'OWL-S qui permettent de définir la condition

sous laquelle un résultat est produit. Comme la composition aura lieu avant l'exécution du service, nous n'autorisons la définition que d'un seul résultat possible : le résultat correspondant au succès de l'exécution du service ;

Comme dans OWL-S, nous considérons un service comme un processus qui consomme des entrées et produit des sorties. Il est principalement décrit par son type de service global (par exemple, DisplayService) et ses types d'entrées/sorties (TextMessage). Ces types sont des concepts sémantiques définis dans l'ontologie globale de notre architecture, et ne correspondent pas à des types de données classiques en informatique. A l'inverse de simples mots clés, des types sémantiques permettent de raisonner sur les fonctionnalités des services en tenant compte d'ontologies abstraites. Un autre aspect d'OWL-S que nous utilisons est la possibilité d'enrichir une description de services de pré-conditions (qui doivent être vérifiées pour l'exécution du service) et d'effets (produits par l'exécution du service).

OWL-S fournit un cadre important de description des services web. Les principales informations utilisées dans une description de service sont :

- **serviceType** définit le type d'un service.
- **hasInput/ hasOutput** définissent les entrées/sorties (E/S) d'un service, et **parameterType** décrit leurs types respectifs.
- **hasLocal** définit des paramètres supplémentaires. Dans notre modèle, ces paramètres désignent les éléments du monde réel qui sont pertinent dans l'utilisation du service, par exemple dans le cas d'un service composé cette information peut prendre comme valeurs les noms des agents impliqués dans le processus de composition. Comme pour les entrées/sorties, ces paramètres ont un type. D'autre part, leur valeur peut être fixée soit dans la description (avec **parameterValue**) soit définis au moment de la composition.
- **hasInputProperty/ hasOutputProperty** définit les propriétés des entrées/sorties. Une propriété d'une sortie est un effet qui met en jeu un paramètre de sortie, et une propriété d'une entrée est une pré-condition qui met en jeu un paramètre d'entrée. Lors de la composition de services, une relation de dépendance entre services peut être établie par une entrée de l'un et une sortie de l'autre. Les propriétés des E/S, ainsi que leurs types, sont utilisés pour vérifier la cohérence de ces liens.

L'architecture proposée dispose, également, d'un ensemble de sous ontologies. Chaque sous-ontologie regroupe l'ensemble des concepts d'un certain type de services Web. Par exemple, une sous-ontologie *i* qui représente les concepts reliés à des services de réservation de tout type de transports, une autre qui regroupe les concepts de services Web de réservation d'hôtels, etc. Nous avons pensé à cette décomposition pour améliorer le temps de réponse aux requêtes.

4. Rôles et comportement des différents agents

4.1. Rôles des agents

Quatre principaux rôles interviennent dans le modèle de composition que nous voulons concevoir :

1. L'agent manager s'assure que la description d'un service importé est consistante par rapport à la sous ontologie en contrôlant, en particulier, ses opérations, ses entrées et ses sorties qui doivent être des concepts de la sous-ontologie correspondante. Il stocke l'ensemble des descriptions OWL-S des services et les localisations de ses fournisseurs. Au moment de la composition, il correspond à un service Web. Il est initialisé avec la description sémantique du service et avec un ensemble de données lui permettant de raisonner. Par exemple, un agent représentant un service de réservation de transport ferroviaire dispose de la liste des trajets existants. Ces données forment la base de connaissances de l'agent manager. A partir de la description sémantique, un agent manager va créer son domaine de planification qui regroupe, sous la forme de méthodes et d'opérateurs extraits des processus de la description OWL-S, les actions réalisables par l'agent, c'est-à-dire, sa base de compétences.

2. L'agent reconstructeur de requêtes a pour tâche principale de reconstruire la « requête utilisateur » à partir des descriptions enregistrées au niveau de l'ontologie globale des services Web. Celle-ci supporte en même temps les services atomiques et composés. Le résultat de cette tâche est un fichier OWL-S.

3. L'agent manager-raisonneur a pour rôle principal de vérifier si la requête représente la description d'un service composé. Dans ce cas, l'agent manager-raisonneur invoque l'agent compositeur, qui à son tour invoque les agents concernés par la requête.

Par contre, si la requête utilisateur concerne un service atomique, l'agent manager-raisonneur invoque directement l'agent manager concerné puisqu'il a une vue globale du système.

4. L'agent compositeur a pour rôle de coordonner et d'assembler les agents manager afin d'exécuter les opérations requises. Il initialise également le processus de composition en soumettant aux agents managers le but à réaliser sous la forme d'une première conjecture qui représente le plan initial.

4.2. Comportement des agents

Le comportement des différents agents, illustré sur la figure 4.7, peut être synthétisé comme suit :

- L'agent reconstructeur de requêtes reçoit l'ensemble des requêtes et contraintes que l'utilisateur a formulées, ensuite il essaie de transformer les requêtes qui étaient sous format XML en un fichier OWL-S. Enfin, il mémorise ses contraintes et encapsule ses requêtes dans un message, devenant un message initiateur (*m0* sur la figure 4.7), qu'il diffuse à l'agent manager-raisonneur.
- Lorsque l'agent manager-raisonneur reçoit le message, il le traite pour déduire s'il s'agit d'un service simple pouvant être traité par un seul agent manager (il lui envoie un message *m1* sur la figure 4.7) ou bien s'il s'agit d'un service composite et dans ce cas il délèguera directement la tâche à l'agent compositeur après avoir lui envoyer une liste initiale des agents manager (concernés par la requête) encapsulé dans un message *m2* sur la figure 4.7.
- Une fois que l'agent compositeur a reçu le message *m2* auprès de l'agent manager-raisonneur, la phase de planification, donc de composition, commence. L'agent compositeur décompose le but global en des sous-buts et les attribue aux agents manager concernés par la requête utilisateur (messages *mb21*, *mb22*, *mb23*, etc.).
- Un agent manager peut recevoir un message *m1* auprès de l'agent manager-raisonneur et dans ce cas il essaie de réaliser le but qui lui a été assigné. Comme il peut recevoir un message *mbx* auprès de l'agent compositeur, dans ce cas il réalise la partie qui lui a été assignée. La composition des résultats individuels aboutit à la réalisation du but global.

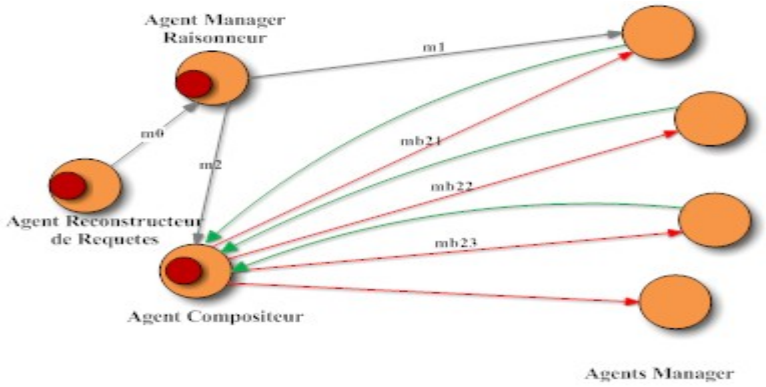


Figure 4.7. Comportement des différents agents

5. Communication inter-agents

Le module de communication (gestionnaire de dialogue) gère la réception et l'interprétation des messages provenant des autres agents. Cette gestion concerne le médium linguistique de la transmission des messages, c'est-à-dire l'expression et l'interprétation d'un message. Nous avons retenu, comme langage d'expression des messages le langage de communication ACL de la FIPA [50].

ACL présente sur KQML (Knowledge Query and Manipulation Language) [134] l'avantage d'une sémantique plus rigoureuse ainsi qu'un effort important de standardisation.

5.1. Envoi de messages

Un agent envoie un message dans deux cas de figure :

1. S'il a reçu un message nécessitant d'être traité, il construit alors un message pour répondre à celui qu'il a reçu et l'envoie à l'émetteur de ce dernier ;
2. S'il exécute une action dont les effets consistent en l'envoi d'un ou plusieurs messages, alors il construit ces messages et les envoie au(x) destinataire(s) spécifié(s).

5.2. Messages échangés

FIPA fournit des "performatifs" basés sur l'acte de discours et une syntaxe standard pour les messages. Ces messages sont basés sur la théorie des actes de discours, qui est le résultat de l'analyse linguistique de la communication humaine. La base de cette théorie est de produire une action à partir du langage. L'exemple dans la figure 4.8, présente la structure d'un message ACL:

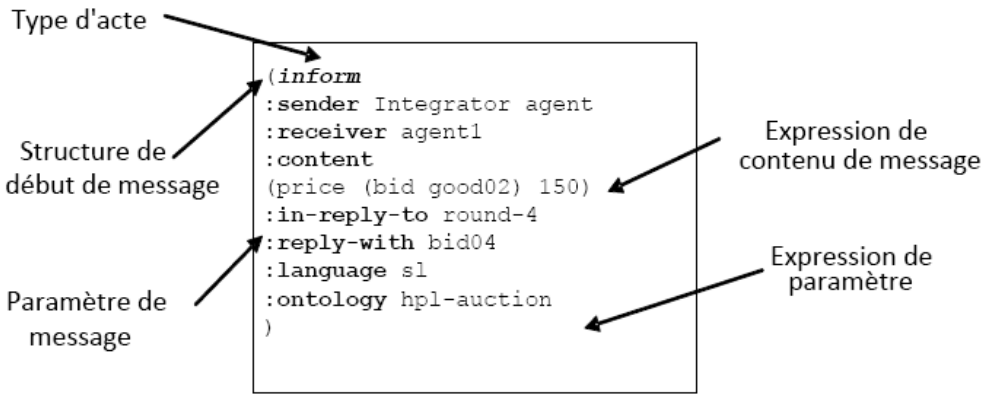


Figure 4.8. Structure d'un message ACL

La signification sémantique des paramètres d'un message est la suivante [50] :

Sender : Dénote l'identité de l'expéditeur du message ;

Receiver : Dénote l'identité du destinataire du message qui pourrait être le nom d'agent simple, ou un tuple de noms d'agent.

Content : Dénote le contenu du message ;

Reply-with : Présente une expression qui sera employée par l'agent répondant à ce message pour identifier le message original.

In-reply-to : Dénote une expression qui fait référence à une action précédente à laquelle ce message répond.

Language : Dénote le codage du contenu de l'action.

Ontology : Dénote l'ontologie employée pour donner la signification (le sens) des symboles dans l'expression.

Contenu de messages

Dans le langage FIPA-ACL, aucun langage spécifique pour la description des contenus des messages n'est imposé. Plusieurs langages peuvent être utilisés tels que le langage KIF (Knowledge Interchange Format), le langage sémantique (SL) et Le langage XML (Extensible Markup Language).

Nous utilisons le langage XML pour la description, la spécification et l'interprétation du contenu des messages (Content) échangés entre les différents agents. Donc, les messages échangés entre les agents sont décrits à l'aide de FIPA-ACL et XML.

Le contenu de messages envoyés par l'agent compositeur aux différents agents manager est la description du sous but annoncé. En conséquence, les messages envoyés par les différents agents manager à l'agent compositeur contiennent la description (en XML) de leurs réponses. L'utilisation d'XML pour le contenu des communications entre agents permet de faciliter l'intégration avec les applications Web existantes.

6. Conclusion

Nous avons donné une description globale de notre approche de composition de services web et brièvement introduit ses composants tout au long de ce chapitre. Nous avons notamment présenté l'architecture sur laquelle notre approche s'appuie. Celle-ci est fondée sur un système multi-agent constitué de quatre types d'agents offrant des services sur lesquels ils peuvent raisonner, nous avons décrit la structure de ces agents.

Le SMA sur lequel notre architecture se base permettra d'implanter la composition dynamique des services web à travers la coordination des agents du système. La particularité de notre architecture est, d'un côté, qu'elle s'appuie sur des agents capables d'examiner et de raisonner sur leurs propres actions, et d'un autre côté est qu'elle définit plusieurs niveaux de responsabilité dispersés entre les différents agents du système.

Dans le prochain chapitre, nous voyons plus amplement comment les différents agents exploitent leurs capacités pour collaborer afin d'aboutir à un plan solution (une composition possible de services web).

CHAPITRE 5

UN MODELE DE COMPOSITION DE SERVICES WEB PAR LA PLANIFICATION MULTI-AGENT

Pouvoir composer des services Web est un enjeu important pour le développement des activités des entreprises sur Internet. En effet, pour une entreprise cliente (voir la spécification fonctionnelle d'ebXML dans le chapitre précédent), l'utilisation d'un service Web d'une autre entreprise permet de réduire ses coûts de la même façon qu'elle fait sous-traiter certains aspects de sa production ou de ses tâches administratives. L'entreprise a donc besoin d'outils afin de pouvoir composer les différents services Web qu'elle utilise.

Les travaux existants sur la composition dynamique de services Web ont mis en avant le fait que la description syntaxique (WSDL) des services n'est pas suffisante et ont proposé des solutions basées sur une description sémantique (OWL-S).

Dans le chapitre portant sur la composition de services web et sur la synthèse de l'état de l'art, nous avons mis en valeur l'utilisation des techniques d'intelligence artificielle qui ont été appliquées à la composition de services. Nous avons constaté que la *planification classique* est une solution prometteuse du fait de la correspondance entre la notion de processus atomique et composite d'OWL-S et la notion d'opérateur et de méthode de la planification, ainsi que la possibilité de spécifier un service Web par ses pré-conditions et ses effets. Cependant, la composition par *planification classique* ne traite pas l'aspect distribué des services Web et a donc ses limites [69] :

– elle a lieu au niveau de l'utilisateur et ne permet donc pas d'être réutilisée pour résoudre des objectifs similaires d'autres utilisateurs ;

– la description des services Web doit être complète et totalement observable. Or les entreprises peuvent être réticentes à fournir une description détaillée du fonctionnement de leurs services ;
– la tolérance aux erreurs est inexistante, du fait que le processus de composition est centralisé, et nécessite l'ajout de mécanismes de contrôle.

Dans ce chapitre, nous proposons un modèle de composition semi distribuée de services web. Les questions de la découverte et de l'exécution des services n'entrent pas dans le champ de cette étude. Notre objectif est de comprendre comment des algorithmes de planification distribuée peuvent contribuer à la création d'un service composite vu comme un plan définissant des relations de précédence et de causalité entre services élémentaires. Nous considérons qu'au moment de la composition chaque service est un agent autonome capable de planifier sous contrôle de l'agent compositeur (*cf.* Architecture proposée dans le chapitre 4), c'est pour cette raison que notre modèle est dit *semi distribué*. Lorsque l'agent manager-raisonneur ne peut pas répondre à la requête d'un utilisateur (entreprise client), cette requête devient le but d'un processus semi-distribué de planification, orchestré par l'agent compositeur, dont le plan solution, s'il existe, est une représentation du service composite constitué des agents ayant pris part à sa construction.

1. Vue d'ensemble du modèle de composition proposé

Avant de présenter en détail notre modèle de composition de services Web basé sur la planification multi-agent, il convient d'introduire les grandes lignes de son fonctionnement. Le modèle proposé se place dans un contexte multi-agent centré utilisateur dans lequel les agents manager sont les représentants des services et sont chargés de proposer, de coordonner leurs compétences et de collaborer avec l'agent compositeur afin de réaliser une tâche soumise par l'utilisateur. Il s'appuie sur le modèle théorique de la synthèse dialectique de plans proposé par Pellier [29] : les agents échangent des propositions sous la forme de conjectures afin de co-construire un plan solution.

Donc le but de notre travail est de proposer un modèle dynamique et automatique de composition de services. Comme l'illustre la figure 5.1, notre approche se déroule en trois étapes pré-composition et une de composition proprement dite :

1. D'abord, un utilisateur humain (représentant de l'entreprise cliente de la spécification fonctionnelle d'ebXML) formule ses besoins en services à travers une interface graphique. Cette dernière propose des champs à remplir afin de faciliter la saisie de ces besoins. Les besoins de l'utilisateur incluent des requêtes en services, i.e. questions ou commandes, ainsi que des contraintes globales, par exemple sur un prix ou une durée totale, sur les services demandés. Dans notre travail, nous n'abordons pas les problèmes relatifs au traitement automatique de la langue naturelle.
2. L'agent reconstructeur de requêtes permet de transformer le fichier XML représentant de la requête utilisateur en un fichier OWL-S (*cf.* sous-section 3.1). Ce fichier sera le point de départ de tout un éventuel processus de composition. Ce fichier sera envoyé à l'agent manager-raisonneur
3. Après avoir reçu le fichier OWL-S auprès de l'agent reconstructeur de requêtes, l'agent manager-raisonneur décide d'invoquer soit un des agents manager ou bien l'agent compositeur. Cette décision dépend de la nature de la requête utilisateur qui indique s'il s'agit d'un service atomique ou s'il s'agit d'un service composite (*cf.* sous-section 3.2).
4. Le processus de composition (*cf.* sous-section 3.3) proprement dit est alors déclenché par l'agent compositeur, il l'initialise également en soumettant aux agents managers

impliqués le but à réaliser sous la forme d'une première conjecture qui représente le plan initial.

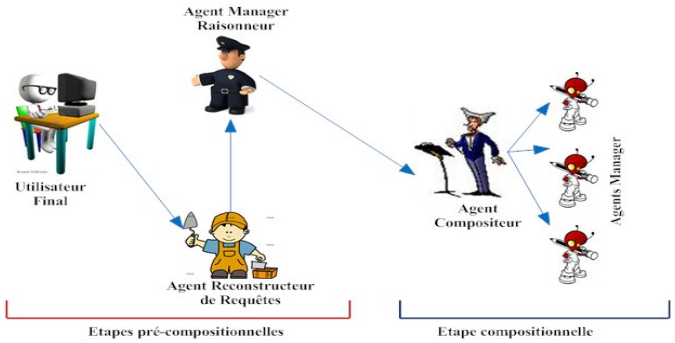


Figure 5.1. Vue d'ensemble de l'approche de composition

Ainsi, notre modèle prend en compte une spécification de besoins en services formulés par l'utilisateur et permet la découverte de services susceptibles de répondre à tout ou partie de ces besoins. Dans notre modèle, ces services interagissent entre eux de manière autonome et semi-décentralisée afin de couvrir les besoins exprimés. Aussi, nous émettons l'hypothèse que l'environnement, et notamment les services disponibles, ne changent pas durant la planification et l'exécution de la composition.

2. Définitions préliminaires

Dans cette section, nous définissons les notions préliminaires nécessaires à la formalisation de notre modèle de planification semi-distribuée pour la composition de services web.

Définition.2.1 (Opérateur)

Les opérateurs de planification sont définis comme des fonctions de transition au sens classique [84] : actions instantanées, statiques, déterministes et observabilité totale.

Un opérateur peut être défini par le quadruplet :

$$o = (name(o), precond(o), add(o), del(o))$$

– *name(o)*, le nom de l'opérateur, est défini par une expression de la forme $n(x_1, \ldots, x_k)$ où n est un symbole d'opérateur et $x_1, \ldots, x_k$ représentent les paramètres de l'opérateur.

– *precond(o)* représentent les préconditions de l'opérateur o, c'est-à-dire, les propriétés du monde nécessaires à son exécution.

– *add(o)* et *del(o)* définissent deux ensembles de propriétés décrivant respectivement les faits à ajouter et les faits à supprimer de l'état du monde après l'exécution de *o*.

Définition.2.2 (Action)

Une action est une instance d'un opérateur. Si *a* est une action et s_i un état tel que $precond^+(a) \in s_i$ et $precond^-(a) \cap s_i = \varnothing$ alors *a* est applicable à s_i, et le résultat de cette application est l'état :

$$s_{i+1} = \gamma\ (s_i, a) = (s_i - effets^-(a)) \cup effets+(a)$$

Les pré-conditions positives ($precond^+$) expriment les propriétés qui doivent être vérifiées pour qu'une action puisse être appliquée. Les pré-conditions négatives($precond^-$) expriment celles qui doivent être absentes de l'état pour que l'action soit appliquée.

Les effets d'un opérateur spécifient les propriétés, du monde, modifiées par l'exécution d'une action. D'un point de vue formel, si une action est définie par un opérateur qui transforme un état s_i en un état s_{i+1}, les effets d'une action sont représentés par les propriétés ajoutées ($effets^+$) et les propriétés enlevées ($effets^-$) à l'état s_i pour obtenir l'état s_{i+1}.

Définition.2.3 (Domaine de planification)

En planification, un domaine définit l'ensemble des opérateurs qui peuvent s'appliquer sur le monde. Un problème doit spécifier l'état initial ainsi que le but à atteindre.

Un domaine *D* de *L* est un système de transition d'états restreint S= (*S*, *A*, γ) tel que :

- $S = 2^{\{\text{les atomes instanciés de } L\}}$
- A = {l'ensemble des opérateurs instanciés de *O*} où *O* est l'ensemble des opérateurs
- $\gamma\ (s, a) = (s - effets^-(a)) \cup effets^+(a)$ si $a \in A$ et *a* est applicable à $s \in S$.

Il reste maintenant à définir un problème de planification. Un problème doit spécifier les états initiaux des croyances des agents, les opérateurs et méthodes qu'ils peuvent appliquer ainsi que le but qu'ils doivent réaliser. Le but est représenté par un ensemble de propositions décrivant les propriétés du monde qui doivent être vérifiées.

Définition.2.4 (Problème de planification)

Un problème *P* pour un domaine *D* est un triplet P = (*O*, s_0, *g*) où :

- s_0, l'état initial, est un état quelconque de *S* ;
- *g*, le but, définit un ensemble cohérent de prédicats instanciés, c'est-à-dire, les propriétés du monde devant être atteints ;

- *O* est l'ensemble des opérateurs applicables.

Nous faisons l'hypothèse restrictive que l'union des croyances des agents d'un problème de planification est cohérente (cas classique de la planification mono-agent), c'est-à-dire, pour deux agents *A* et *B*, si une proposition $p \in belief(A)$ alors $\neg p \in belief(B)$. Cependant, aucune hypothèse n'est faite sur le possible partage de croyances entre les agents managers en termes de faits ou d'opérateurs.

Définition.2.5 (Plan solution)

Un plan solution se définit comme un chemin dans un espace d'états. Le passage d'un état à l'autre s'effectue par l'application d'une action, c'est-à-dire, un opérateur complètement instancié. Par conséquent, un plan solution pour un problème de planification $P = (O, s_0, g)$ est une séquence d'actions décrivant un chemin d'un état initial s_0 à un état final s_n. Tel que le but *g* soit inclus dans s_n. Autrement dit, un plan π est une solution pour le problème *P* si $g(s_0,\pi)$ satisfait *g*.

Définition.2.6 (Conjecture)

Une conjecture est un tuple $\chi=(A, <, I, C)$ tel que :

- $A = \{a_0,\ldots, a_n\}$ est un ensemble d'actions.
- $<$ est un ensemble de contraintes d'ordre sur les actions *A* de la forme $a_i<a_j$, *c'est-à-dire.*, a_i précède a_j.
- *I* est un ensemble de contraintes d'instanciation portant sur les variables des actions *A* de la forme $x=y$, $x\neq y$ ou $x=cst$ tel que $cst \in D_x$ et D_x est le domaine de *x*.
- *C* est un ensemble de liens causaux de la forme $a_i \xrightarrow{p} a_j$ tels que a_i et a_j sont deux actions de *A*, la contrainte d'ordre $a_i<a_j$ existe dans $<$, la propriété *p* est un effet de a_i et une précondition de a_j et finalement les contraintes d'instanciation qui lient les variables de a_i et de a_j portant sur la propriété *p* sont contenues dans *I*.

Définition.2.7 (Méthode)

Une méthode est un triplet *m = (name(m), precond(m), reduction(m))*

– *name(m)* est une expression de la forme $n(x_1, ..., x_k)$ où *n* représente le nom de la méthode et $x_1, ..., x_k$ ses paramètres.

– *precond(m)* représente les pré-conditions (c.à.d. un ensemble de prédicats) devant être vérifiées dans l'état courant pour que *m* soit appliquée.

– *reduction(m)* définit la séquence d'opérateurs ou de méthodes pour réaliser *m*.

Définition.2.8 (Hypothèse)

Soit une conjecture $\chi=(A, <, I, C)$. Une hypothèse formulée par χ est définie comme une précondition p d'une action $a_j \in A$ telle que pour toutes actions $a_i \in A$, le lien causal $a_i \xrightarrow{p} a_j \notin C$.

3. Etapes pré-compositionnelles du modèle proposé

3.1. Reconstruction de requêtes

Cette étape est réalisée par l'agent reconstructeur de requêtes (cf. chapitre 4). Lorsqu'un utilisateur envoie sa demande en remplissant un formulaire graphique, cette demande sera directement transformée en un fichier XML. Ce dernier est considéré comme étant l'entrée de l'agent reconstructeur de requêtes.

Cette étape est divisée en quatre sous tâches (cf. Figure 5.2): *parcours du fichier XML*, *génération d'instances*, *extraction d'une sous-ontologie spécifique* et *production de la description du service correspondant à la requête utilisateur.*

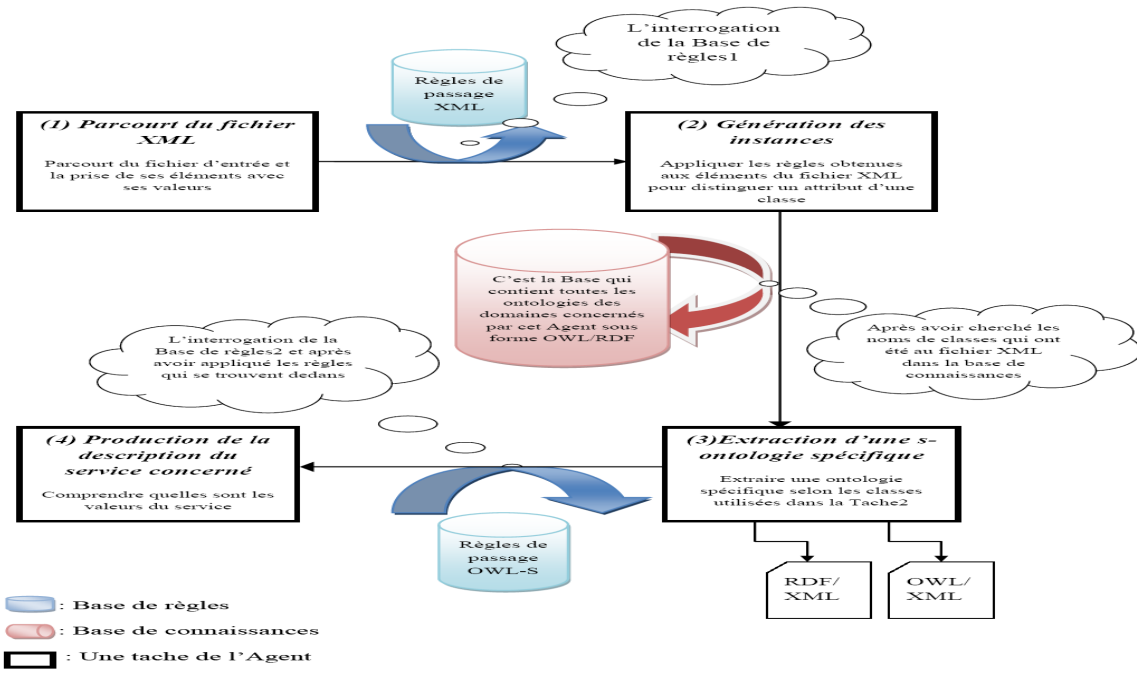


Figure 5.2. Fonctionnement de l'agent reconstructeur de requêtes [106]

3.1.1. Parcours du fichier XML et génération d'instances

L'agent reconstructeur de requêtes parcourt le fichier XML pour extraire son arborescence et connaitre tous ses constituants et ses balises puis il applique la première règle de sa base de règle, pour connaitre quelles sont celles qui représentent les classes et celles qui représentent les attributs [106].

Règle1 : « Si une balise possède des sous balises alors c'est une classe, sinon c'est un attribut»

Alors en appliquant cette règle (cf. Figure 5.3), on obtient les noms des classes avec les noms de ses attributs ainsi que leurs valeurs (instances).

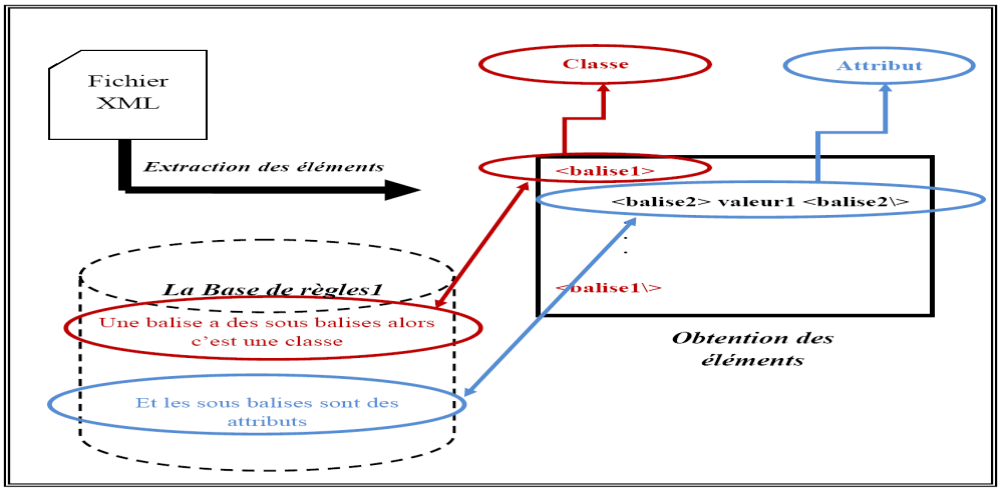


Figure 5.3. Parcours du fichier XML et génération d'instance [106]

3.1.2. Extraction d'une sous-ontologie spécifique

La base de connaissance de cet agent est un ensemble d'ontologies qu'il a pu enrichir avec le temps, elle contient aussi des traces des différentes requêtes que les clients envoient à chaque fois. Elle nous informe des classes obtenues du fichier XML pour connaitre les relations et les classes qui ont des relations directes avec les classes produites de la phase précédente, on obtient un fichier qui contient une structure plus complète des classes. Toute en ajoutant de nouvelles relations et classes pour enrichir la base de connaissance de l'agent.

Donc, dans cette phase, l'agent extraits une partie de l'ontologie spécifique selon les classes utilisées dans la tâche 2 (Fichier OWL / XML). Afin de produire un fichier plus lisible et conviviale fichier OWL / XML, nous avons constaté qu'il est recommandé de faire une petite transformation de OWL / XML pour passer au format RDF / XML. Pour cela, nous avons défini deux règles principales (cf. Figure 5.4 et Figure 5.5).

Règle 2 : « « Si on trouve des classes dans le fichier XML obtenu à l'entrée et qu'elles sont dispersées dans différentes ontologies, alors le type du service produit à la sortie de l'Agent sera un service composé »

Autrement dit, un service atomique est un service dont toutes les classes appartiennent à une même ontologie.

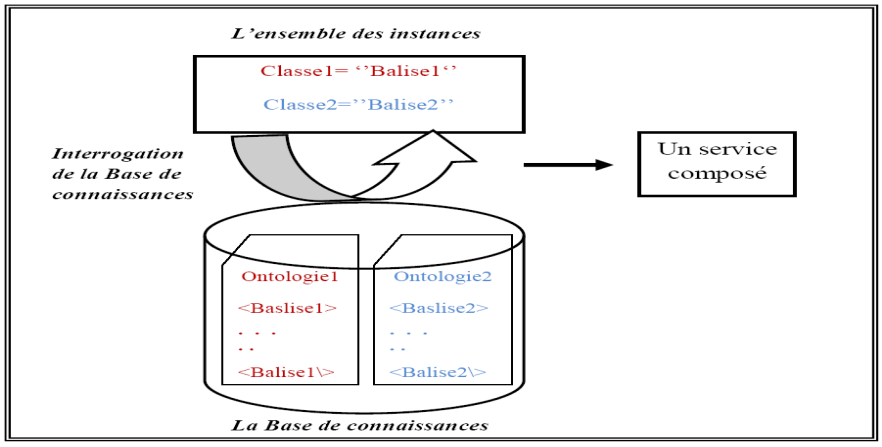


Figure 5.4. Illustration de la règle 2

Règle 3 : «« Les valeurs des attributs des instances obtenues sont les inputs du service, et les attributs vides (sans valeurs) représentent ses outputs. »

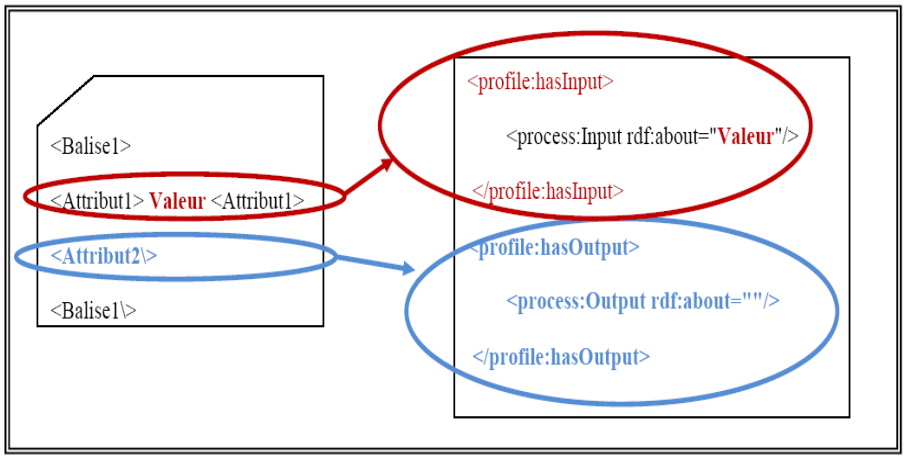


Figure 5.5. Illustration de la règle 3

3.1.3. Production de la description du service web

Après avoir appliqué les règles 2 et 3, nous pouvons comprendre et décider à propos du type du service. Nous pouvons également produire la description du service correspondant à la demande de l'utilisateur (en affectant les valeurs du fichier XML). Ainsi, le résultat est un fichier OWL-S à envoyer à l'agent manager-raisonneur.

Cette étape peut être récapitulée par le diagramme d'activité illustré dans la figure 5.6 ci-dessous :

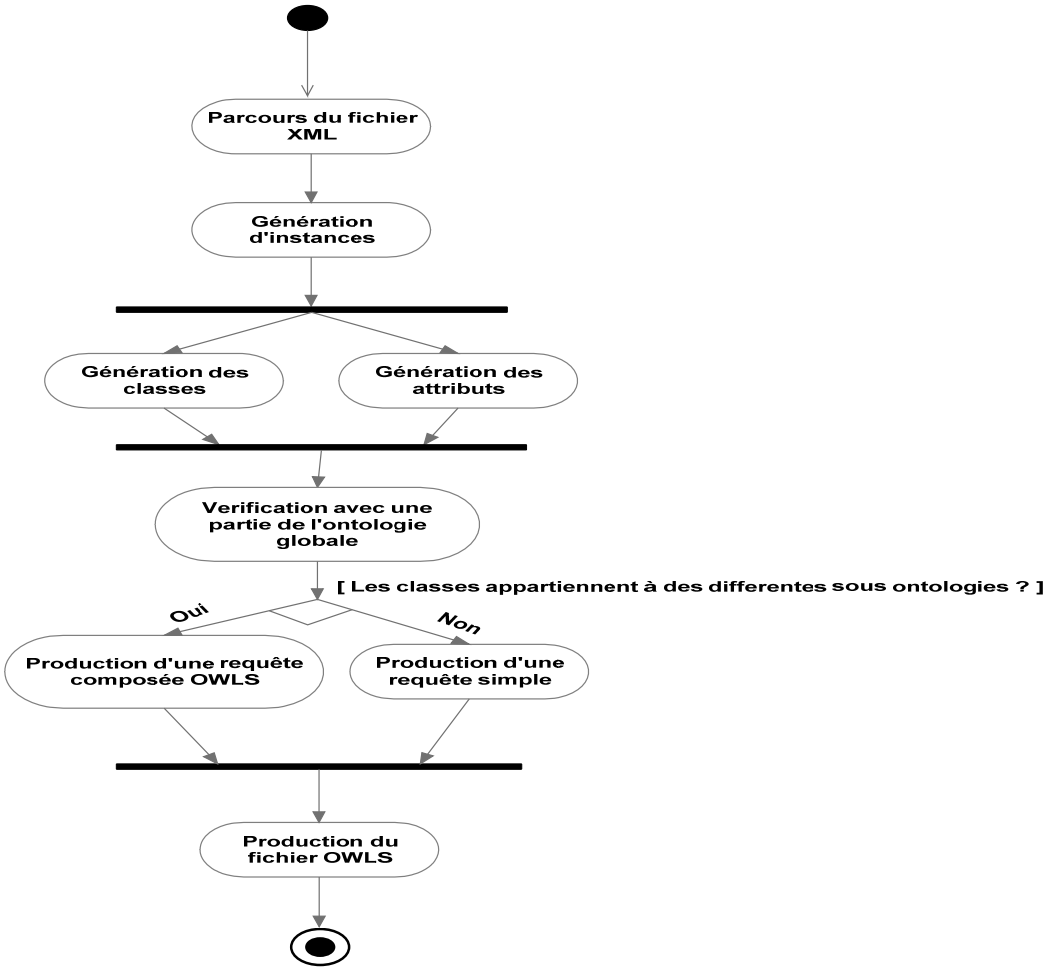


Figure 5.6. Diagramme d'activité de l'étape « reconstruction de requêtes »

3.2. Prise de décision d'une éventuelle composition

L'agent manager-raisonneur parcours, dans un premier temps, le fichier OWL-S, qu'il a reçu auprès de l'agent reconstructeur de requêtes, afin de décider si le service demandé est un service atomique ou composé.

3.2.1. Cas d'un service atomique

S'il ne trouve pas la balise qui indique que le service est composé<*process :CompositeProcess*>, alors il en déduira que le service est atomiques (dans ce cas il trouvera <process: AtomicProcess ….>).

Exemple d'un service atomique :

```
<process:AtomicProcess rdf:ID="LogIn">
   <process:hasInput rdf:resource="#AcctName_In"/>
   <process:hasInput rdf:resource="#Password_In"/>
</process:AtomicProcess>
<process:Input rdf:ID="AcctName_In">
   <process:parameterType rdf:resource="&concepts;#AcctName">
</process:Input>
<process:Input rdf:ID="Password_In">
   <process:parameterType rdf:resource="&concepts;#Password">
</process:Input>
```

Dans ce cas, l'agent manager-raisonneur choisira l'agent manager impliqué dans la satisfaction de la requête; ce choix est effectué à partir d'un tableau de sélection dont la structure est la suivante :

Groupe d'agents	**Service fournis**	**Agents disponibles classés selon les performances**
1	Transport par avion	a1, a12, a14
2	Réservation d'hôtels	a24
3	Service de payement	a3, a101

Tableau 5.1. Structure du tableau de sélection

Après avoir choisir l'agent manager adéquat, l'agent manager-raisonneur lui envoie un message (le fichier OWL-S) indiquant le service demandé avec les paramètres et contraintes fixés par l'utilisateur.

3.2.2. Cas d'un service composé

Dans ce cas, l'agent manager-raisonneur détecte la balise <*process:CompositeProcess*> indiquant que la demande de l'utilisateur nécessite la composition de deux ou plusieurs services web indiqué entre les deux balises <*process:composedOf*> et <*/processus: composedOf*>. Ainsi, l'agent manager-raisonneur déduira les agents manager impliqués dans le processus de composition, car il a une vision globale du système. Donc, il délègue la tâche de composition à l'agent compositeur en lui envoyant le groupe d'agents manager impliqués ainsi que la liste des

agents manager disponibles à l'heure actuelle (nous ne gérons pas la disponibilité et la non disponibilité dynamique des différents agents manager à cause de la complexité du problème).

Exemple de services composés

```
<process:CompositeProcess rdf:ID="BravoAir_Process">
   <rdfs:label>This is the top level process for BravoAir</rdfs:label>
     <process:composedOf>
       <process:Sequence>
         <process:components rdf:parseType="Collection">
            <process:AtomicProcess rdf:about="#GetDesiredFlightDetails"/>
            <process:AtomicProcess rdf:about="#SelectAvailableFlight"/>
            <process:CompositeProcess rdf:about="#BookFlight"/>
         </process:components>
       </process:Sequence>
     </process:composedOf>
</process:CompositeProcess>

<process:CompositeProcess rdf:ID="BookFlight">
   <process:composedOf>
     <process:Sequence>
       <process:components rdf:parseType="Collection">
         <process:AtomicProcess rdf:about="#Login"/>
         <process:AtomicProcess rdf:about="#ConfirmReservation"/>
       </process:components>
     </process:Sequence>
   </process:composedOf>
</process:CompositeProcess>
```

Le diagramme d'activité de l'étape de prise de décision, effectuée par l'agent manager-raisonneur, est illustré par la figure 5.7.

4. Processus de composition

Au moment du processus de composition, chaque agent manager représente un service retenu appartenant à la sous-ontologie qu'il gère. Au moment de la composition, son but est de simuler l'exécution du service. Cet agent est une entité autonome qui contient un planificateur et qui est capable de communiquer avec l'agent compositeur, dans le but de co-construire un plan d'exécution des services Web. Les agents managers sont initialisés avec la description sémantique du service et avec une base de connaissances. La description OWL-S permet de définir le domaine de planification de l'agent tandis que la base de connaissances apporte les connaissances qui seront nécessaires à l'agent pour raisonner.

Comme nous avons mentionné auparavant, notre modèle s'appuie sur la synthèse dialectique de plans avec centralisation de prise de décision (agent compositeur) : les agents managers échangent les propositions, avec l'agent compositeur, sous forme de sous-conjectures afin de

construire un plan solution. Ce modèle est constitué de trois phases : la création du domaine de planification de chaque agent manager au moment de leur initialisation, le raffinement des conjectures proposées et envoyées à l'agent compositeur et la communication entre l'agent compositeur et les agents managers, c'est-à-dire, la soumission d'une nouvelle conjecture aux autres agents (par l'intermédiaire de l'agent compositeur) suite au raffinement d'une conjecture.

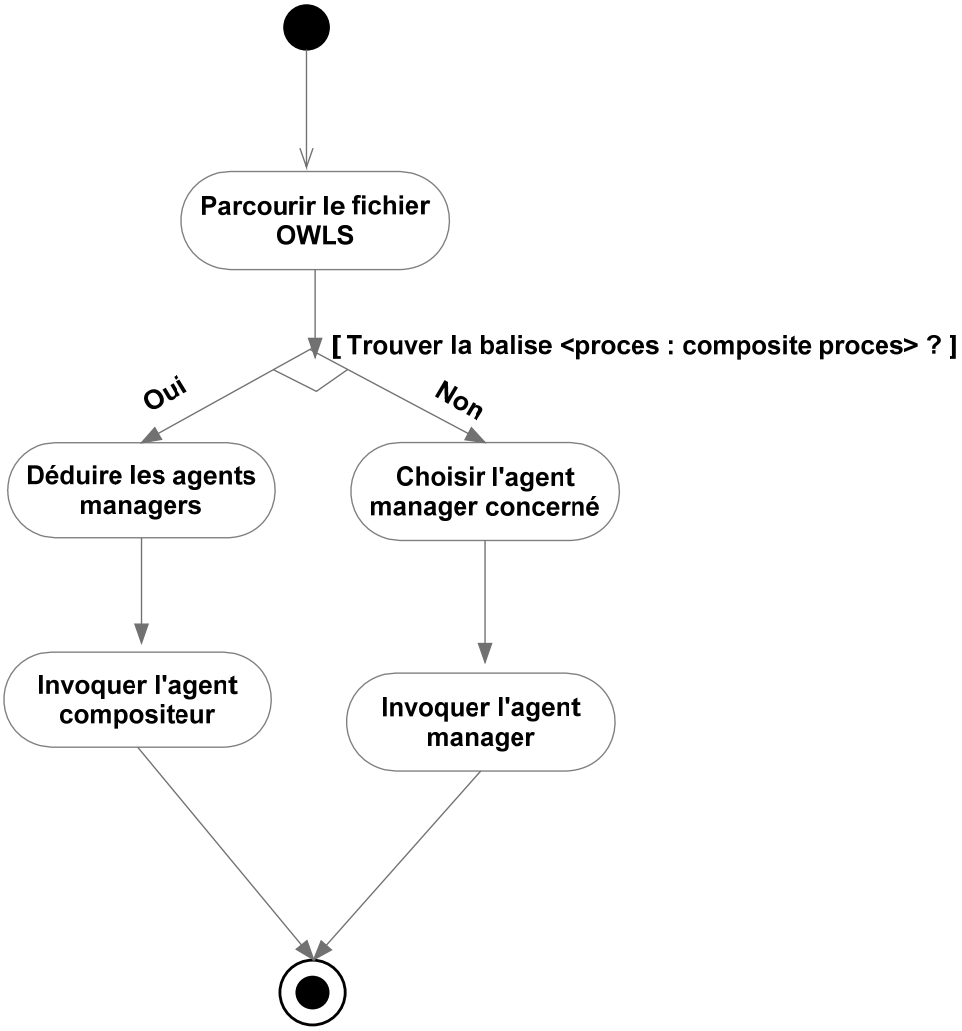


Figure 5.7. Diagramme d'activité de l'étape « prise de décision d'une éventuelle composition »

4.1. Initialisation des agents managers et de l'agent compositeur

Chaque agent manager correspond à un service Web retenu. Il est initialisé avec la description sémantique du service et avec un ensemble de données permettant à l'agent manager de raisonner. Par exemple, un agent représentant un service de réservation de transport ferroviaire dispose de la liste des trajets existants. Ces données forment la base de connaissances de l'agent manager. L'agent compositeur est initialisé avec la description sémantique de la requête utilisateur récupérée auprès de l'agent manager-raisonneur, et un ensemble de données (ex. Liste d'agents manager impliqués dans le processus) lui permettant de raisonner. Ce dernier dispose d'un mécanisme de génération de plan permettant de générer un plan abstrait formé de tâches élémentaires à effectuer. Ces tâches expriment les fonctionnalités de services (agents manager) requises pour répondre à la requête utilisateur.

4.2 Création de la base de compétences

A partir de la description sémantique, l'agent compositeur crée son domaine de planification qui regroupe, sous la forme de méthodes et d'opérateurs extraits des processus de la description OWL-S, les actions réalisables par l'agent, c'est-à-dire, sa base de compétences.

4.2.1. Création du domaine de planification à partir de la description sémantique d'un service Web

Le modèle de processus OWL-S décrit de manière déclarative les propriétés et le comportement d'un service Web. La traduction de la description sémantique d'un service vers un domaine de planification consiste à représenter les processus OWL-S sous forme d'opérateurs et de méthodes. L'algorithme de traduction que nous utilisons est celui de [69] .Cet algorithme prend en entrée une description OWL-S. La propriété « *DescribedBy* », qui signifie qu'un service est décrit par un processus, permet d'obtenir « le point d'entrée » de la description, c'est-à-dire, le premier processus à traduire.

L'application de cet algorithme nécessite d'imposer un certain nombre d'hypothèses et de restrictions :

- les processus simples (*SimpleProcess*) ne sont pas trait´es. En effet, ils correspondent uniquement à des abstractions de processus atomiques ou composites ;
- les structures de contrôle qui régissent l'enchaînement des sous-processus dans un processus composite sont variés (*Sequence*, *Concurrent*, *Split*, *Split+Join* , *Unordred*,

Choice, *If-Then-Else*, *Repeat-Until*, *Repeat-While*), mais seule la structure de contrôle *Sequence* est prise en compte ;

- OWL-S permet de tenir compte de l'indéterminisme dû à l'exécution d'un service, c'est-à-dire qu'il laisse la possibilité de définir le résultat d'un service Web en fonction de son comportement (succès, erreur ou encore absence de réponse) via l'utilisation des classes *ConditionalEffect* et *ConditionalOutput* qui permet de définir la condition sous laquelle un résultat est produit. Comme la composition a lieu avant l'exécution du service, nous n'autorisons la définition que d'un seul résultat possible : le résultat correspondant au succès de l'exécution du service ;
- nous émettons l'hypothèse que le premier processus de la description sémantique d'un service web (la requête utilisateur) est un processus composite dont les pré-conditions contiennent le but que ce service permet d'atteindre.

L'algorithme de traduction réutilisé (cf. Algorithme 5.1) est un algorithme récursif. Son principe est de transférer le premier processus (correspond à la requête initiale de l'utilisateur) en un opérateur si ce dernier est un processus atomique, sinon il traduit récursivement les processus intervenants dans les structures de contrôle de ce processus composite, puis crée la méthode correspondante. Il permet d'ajouter les pré-conditions et les effets du processus atomique à l'opérateur correspondant.

Cet algorithme règle aussi les problèmes de gestion des pré-conditions, des effets et d'ajout de buts à atteindre.

Algorithme 1 : Traduction de la sémantique vers la planification

```
Algo Traduction_Processus(Processus p)
   si p est atomique
   alors
      créer nouvel opérateur o
      o.nom = p.nom
      o.paramètres = p.inputs
      o.préconditions = p.préconditions + prédicats_typage(p.inputs)
      o.effets = p.effets + prédicats_typage(p.outputs)
   sinon // p est composite
      composition = Traduction_structure_contrôle(p.structure_contrôle)
      pour chaque processus p2 de composition
         Traduction_Processus(p2)
      fin_pour
      créer nouvelle méthode m
      m.nom = p.nom
      m.paramètres = p.inputs
      m.préconditions = p.préconditions + prédicats_typage(p.inputs)
      m.décomposition = décomposition_processus(composition)
   fin_si
fin_Algo
```

Selon cet algorithme nous avons deux cas de figure : cas d'un service web atomique et celui d'un service web composite.

A. Traduction des processus atomiques. Si le processus à traduire est un processus atomique, alors la correspondance est directe. Un opérateur de planification est créé, avec, comme identifiant, l'identifiant de ce processus, c'est-à-dire le nom de l'opération du service correspondant. Et les pré-conditions et les effets du processus atomique sont ajoutés à l'opérateur correspondant. Ce cas est ignoré dans notre cas, car nous avons supposé que l'agent compositeur n'est sollicité que s'il s'agit d'un service composite.

B. Traduction des processus composites. Un service composite ne correspond pas à une opération d'un service Web, mais permet de définir l'ordre et les conditions d'exécution des différentes opérations du service. Ainsi le cœur du processus composite est la structure de contrôle qui définit cet enchaînement. La première étape de la traduction d'un processus composite consiste donc à traduire la structure de contrôle. Puis la méthode correspondante est

créée. Les pré-conditions permettant son déclenchement sont ajoutées ainsi que la décomposition, i.e., la séquence d'opérateurs correspondant aux processus invoqués dans la structure de contrôle.

L'algorithme de traitement des structures de contrôle est récursif car les éléments d'une séquence peuvent être des processus mais également d'autres structures de contrôle. Pour chaque composant (représenté par la classe *perform*), l'algorithme de traduction de processus est invoqué. Celui-ci maintient une liste des processus qui interviennent dans la structure de contrôle en liant les variables de ces processus avec le processus appelant, mais également en liant les variables entre eux dans le cas où l'un consomme une donnée produite par l'autre.

C. Gestion des pré-conditions. Les pré-conditions d'un processus ont une correspondance directe avec les pré-conditions d'un opérateur ou d'une méthode du fait de l'hypothèse formulée sur l'écriture des pré-conditions et effets dans la description OWL-S. Les pré-conditions peuvent être hypothétiques, i.e., leur vérification n'est pas imposée pour déclencher l'opérateur ou la méthode.

D. Gestion des effets. Tout comme les pré-conditions, les effets d'un processus atomique sont traduits directement. Mais ils doivent être distingués en effets positifs et en effets négatifs au moment de la création de l'opérateur. Nous choisissons de représenter les effets négatifs, dans la description OWL-S par des prédicats commençant par *not*.

4.3 Raffinement des conjectures

Un agent manager va raisonner, suite à la conjecture reçue auprès de l'agent compositeur, à partir de ses compétences (c'est-à-dire, les actions qu'il est capable de planifier) et de ses connaissances pour résoudre les buts contenus dans cette conjecture : il va la raffiner en y ajoutant une sous-conjecture, c'est-à-dire, une suite d'actions, ou en ajoutant des liens causaux. Un agent manager peut raffiner une conjecture en émettant des hypothèses sur les propriétés qu'il ne connaît pas. Ces hypothèses forment de nouveaux buts à résoudre pour les autres agents manager au travers de l'agent manager qui va formuler des nouvelles conjectures.

Le processus de composition peut être représenté par le diagramme d'activité ci-dessous :

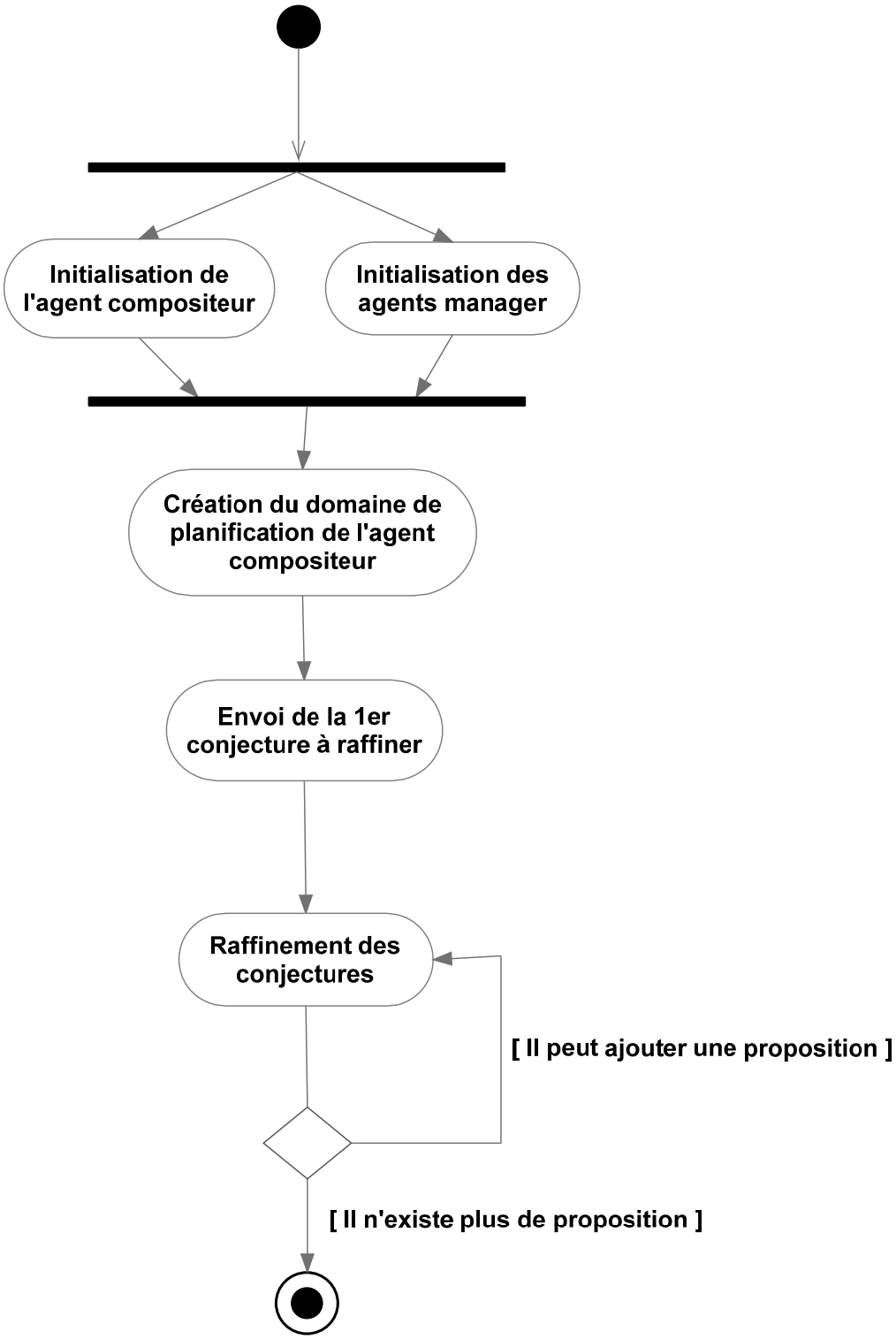


Figure 5.8. Diagramme d'activité du processus de composition

5. Co-construction de la composition

Le dialogue est initié par l'agent compositeur qui soumet à l'ensemble des agents managers concernés une première conjecture. Cette conjecture constitue le but utilisateur à réaliser. A la réception de cette première conjecture, les agents manager commencent à la raffiner. Lorsqu'un agent manager calcule une sous-conjecture, il la soumet à l'agent compositeur.

A la réception d'un raffinement, l'agent compositeur met à jour son tableau de stockage. Puis il essaie de renvoyer la nouvelle sous-conjecture à raffiner aux agents manager qu'il retient pour le prochain cycle de raffinement. Ces derniers essaient de proposer des raffinements pour chaque nouvelle hypothèse en tentant d'ajouter des liens causaux, puis en cherchant une sous-conjecture. Si la sous-conjecture permet de vérifier les dernières hypothèses du tableau de Stockage de l'agent compositeur sans en émettre de nouvelles, ou au contraire si l'agent compositeur n'est plus capable de faire avancer la synthèse dialectique, i.e., de raffiner les hypothèses restantes, alors il initie la phase de sortie du dialogue. Sinon il poursuit son raisonnement en envoyant des nouvelles conjectures aux agents manager qui, à leur tours, calculent et proposent des raffinements pour les hypothèses restantes.

Le tableau de stockage de l'agent compositeur (cf. Figure 5.6) sert de support au raisonnement mais également au dialogue en enregistrant les sous-conjectures proposées par les autres agents. C'est un graphe ET/OU dont les nœuds ET représentent les conjectures et les nœuds OU représentent les hypothèses contenues dans le nœud « conjecture » père [80].

La sortie du dialogue peut se produire sur proposition de l'agent compositeur. Suite au calcul d'une sous-conjecture, si le tableau de stockage de l'agent compositeur est terminal alors il propose de sortir sur succès en émettant un message de succès aux agents manager impliqués et en même temps il attend les acquittements de ces derniers. De la même façon, si les agents manager n'ont plus capable de proposer de raffinements, alors l'agent compositeur soumet une proposition d'échec à l'ensemble des agents manager et le dialogue se termine avec un état d'échec en attendant les acquittements.

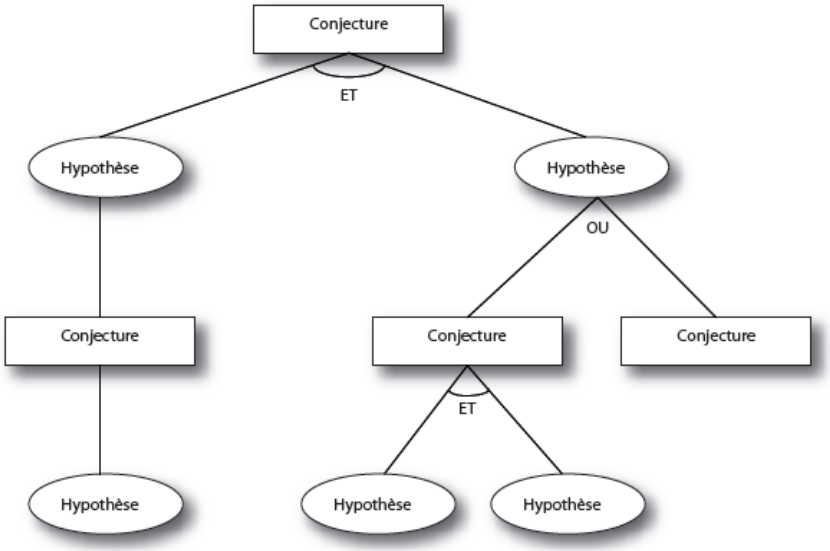


Figure 5.9. Architecture du tableau de stockage de l'agent compositeur

6. Conclusion

Dans ce chapitre, nous avons exposé notre modèle de composition de services web par synthèse dialectique de plans sous hypothèses dans lequel les agents manager, représentant les services web, échangent des propositions afin de co-construire un plan solution. A partir de la conjecture initiale émise par l'agent compositeur, qui représente le but initial de l'utilisateur, les agents manager « service » cherchent à raffiner successivement les hypothèses émises par les autres agents jusqu'à obtenir une conjecture solution, i.e., un enchaînement possible des actions, correspondant aux opérations des services web, qui permet de résoudre le but soumis par l'utilisateur et qui est exempt de toute hypothèse.

Dans le chapitre suivant, nous illustrons l'application de notre modèle avec une étude de cas sur laquelle nous décrivons l'utilisation des différentes étapes du modèle proposé.

CHAPITRE 6

ETUDE DE CAS ET IMPLEMENTATION

Nous venons de spécifier au long des précédents chapitres les algorithmes et structures de données permettant de concevoir un modèle dynamique pour la composition des services web via une architecture à base d'agents. Dans le présent chapitre, nous présentons quelques résultats obtenus à l'implémentation.

Dans une première partie de ce chapitre, nous introduisons un scénario permettant d'illustrer ce que nous attendons par composition automatique et dynamique et nous décrivons comment le modèle de composition proposé s'articule autour de notre architecture multi-agent. Nous présentons notamment les différentes phases de co-construction de la composition des services web.

1. Présentation de l'étude de cas

Le scénario suivant permet d'illustrer ce que nous attendons par composition automatique et dynamique: un enseignant X habite Lyon et doit se rendre à Tokyo pour assister à une conférence. Il décide d'organiser son voyage par internet en faisant appel à deux services web (cf. Figure. Chaque service est représenté par un agent manager : un agent *Airways* offrant un service de réservation de billets d'avion et un agent *Bank* (représentant la banque de l'enseignant X) qui est en charge de payer les différentes réservations que X sera amené à réaliser. Le problème que l'utilisateur soumet à l'interface utilisateur peut être résumé de la façon suivante :

– état initial : l'utilisateur X est à Lyon.
– état final : l'utilisateur X est à Tokyo.

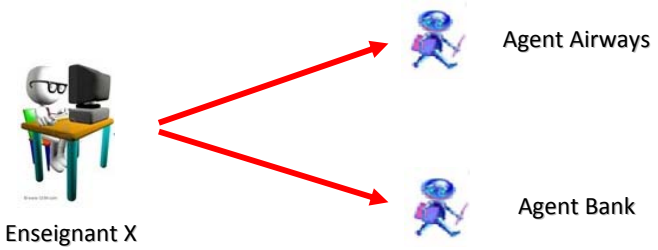


Figure 6.1. Invocation des deux agents Airways et Bank via Internet

Chaque service est décrit sémantiquement par un fichier OWL-S et dispose d'une base de connaissances sous forme de prédicats, propre à ses besoins. Par exemple, la base de connaissances du service Airways est la suivante :

(ExistVol Lyon Paris) (Prix Lyon Paris 60)
(ExistVol Paris Tokyo) (Prix Paris Tokyo 160)
...

Un plan possible résultant de la composition de ces deux services web peut s'exprimer, de façon informelle, ainsi:

1. Réserver l'avion de Lyon à Paris.
2. Payer le billet d'avion Lyon - Paris.
3. Réserver l'avion de Paris à Tokyo.
4. Payer le billet d'avion Paris -Tokyo.

Imaginons maintenant le dialogue que les différents agents composant notre architecture pourraient construire pour que X puisse se rendre à sa conférence :

X : « Je suis à Lyon et je dois me rendre à Tokyo. Pouvez-vous m'aider ? »

Agent manager-raisonneur : « Je ne peux pas te répondre immédiatement, je vais voir avec l'agent compositeur.»

Agent compositeur : « En ce qui me concerne, je vais invoquer les deux agents manager Airways et Bank.»

Airways : « En ce qui me concerne, je peux emmener l'utilisateur X à Tokyo à condition qu'il soit capable de se rendre à Paris et bon je peux l'emmener, de Paris vers Tokyo à condition qu'il me paye la somme de 150$ pour se rendre à Paris et la somme de 645$ pour se rendre à Tokyo»

Bank : «Parfait, je crois que nous tenons la solution. Je peux payer la somme de 795$, le compte de X est créditeur. »

Le plan solution est donc : « Prendre l'avion de Lyon à Paris puis un vol de Paris à Tokyo. » Sa construction repose sur une planification centralisée de l'agent compositeur avec la coopération de plusieurs agents planificateurs (agents manager).

2. Déroulement du processus de composition sur l'étude de cas

Initialisation. L'agent manager-raisonneur envoie la première conjecture à l'agent compositeur après avoir réalisé que la requête utilisateur ne peut être exécuté par un seul agent manager. Cette conjecture indique le but à résoudre, X est à *Tokyo*, ainsi que l'état initial, X est à *Lyon*.

Premier raffinement. Après avoir être invoqué par l'agent compositeur, les agents manager (service) tentent de résoudre le but représenté par l'hypothèse X est à *Tokyo*. Seul l'agent *Airways* est capable de raffiner cette première hypothèse mais la solution qu'il propose formule deux nouvelles hypothèses qui sont :

– X est à *Paris* ;

– X doit payer 645$ pour réserver une place sur ce vol.

La production de ce premier raffinement peut être représentée par le dialogue informel suivant :

X : « Je suis à *Lyon* et je dois me rendre à *Tokyo*. Pouvez-vous m'aider ? »

Agent manager-raisonneur : « Je ne peux pas vous répondre immédiatement, je vais voir avec l'agent compositeur.»

Agent compositeur : « En ce qui me concerne, je vais invoquer les deux agents manager *Airways* et *Bank*.»

Airways : « En ce qui me concerne, je peux emmener l'utilisateur X à *Tokyo* à condition qu'il soit à *Paris* et qu'il me paye la somme de 645$!»

Raffinements suivants. Les agents ont maintenant deux hypothèses à raffiner. L'agent ***Airways*** va proposer un raffinement pour l'hypothèse X est à *Paris* en formulant à nouveau une hypothèse et l'agent ***Bank*** va raffiner l'hypothèse X paye 645$ selon le dialogue suivant :

Bank : « Je peux payer 645$ que je débite du compte de X ».

Airways : « Je peux emmener X à *Lyon* à condition que je reçoive 150$ ». Ainsi une nouvelle hypothèse est émise : X paie 150$.

Bank : « Je peux payer 150$»

Raffinement solution. En raffinant l'hypothèse X paie 150$, l'agent ***Bank*** propose une solution pour la dernière hypothèse restante. Il soumet donc son raffinement à l'agent compositeur, qui gère les différents raffinements enregistrés au niveau du tableau de stockage, c'est-à-dire il propose une sortie sur succès. L'agent compositeur vérifie que son tableau de stockage est terminal et acquitte la proposition de sortie sur succès.

Le tableau de stockage de l'agent compositeur est illustré par la figure 6.2 ci-dessous.

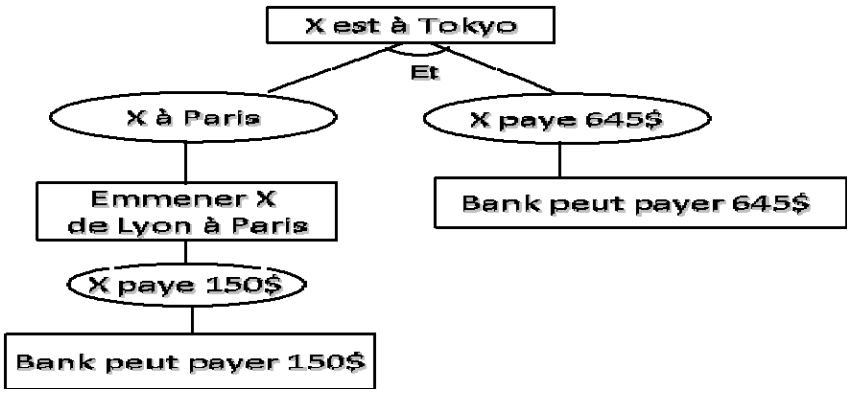


Figure 6.2. Contenu du tableau de stockage de l'agent compositeur

3. Expérimentations

Pour implémenter notre modèle de composition, et évaluer ses performances, nous devions disposer d'un SMA composé des différents agents impliqués (*cf.* chapitre 4).

3.1. Implémentation de l'agent reconstructeur de requêtes

Pour mettre en œuvre ce dernier, nous avons utilisé le *NetBeens* qui est un environnement de développement intégré (*IDE*) pour Java, placé en open source par Sun en juin 2000 sous licence *CDDL* (Common Development and Distribution License). En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, *C*, *C++*, *XML* et *HTML*. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages web). NetBeans est lui-même développé en Java, ce qui peut le rendre assez lent et gourmand en ressources mémoires. On a choisi le langage java car il support des packages externes à la réutilisation parmi ces packages ceux qui aident à la manipulation des fichiers *XML*, *OWL* et *OWL-S* ce qui répond à nos besoins.

Les packages externes. On a téléchargé les packages suivants qui nous ont aidé pour pouvoir manipuler les fichiers *XML*, *OWL*, *OWL-S* :

- ***JDOM :*** pour la manipulation des fichiers *XML* (la requête utilisateur).
- ***OWL_API :*** pour la manipulation des fichiers *OWL* (l'ontologie de la base de connaissances).
- ***OWL-S_API :*** pour la manipulation des fichiers *OWL-S* (la sortie qui est la description sémantique du service correspondant à la requête).

L'agent reconstructeur de requêtes contient cinq (05) modules et manipule deux fichiers : *requete.xml*, *owl_xml.xml*.

3.1.1. Fichier en entrée. Le fichier *requette.xml* représente la requête de l'utilisateur (entreprise cliente) qui a besoin de faire, par exemple, une réservation du vol devant un portail spécialisé dans le domaine de voyages par avion. L'utilisateur désire, par exemple, partir en France depuis l'Algérie alors la requête va correspondre à (cf.Figure 6.1):

3.1.2. Base de connaissances (ontologie impliquée). L'ontologie du domaine *(reservation vol) e*lle est écrite sous forme OWL/XML. C'est un fichier *« OWL_XML.OWL »* (cf. Figure 6.2).

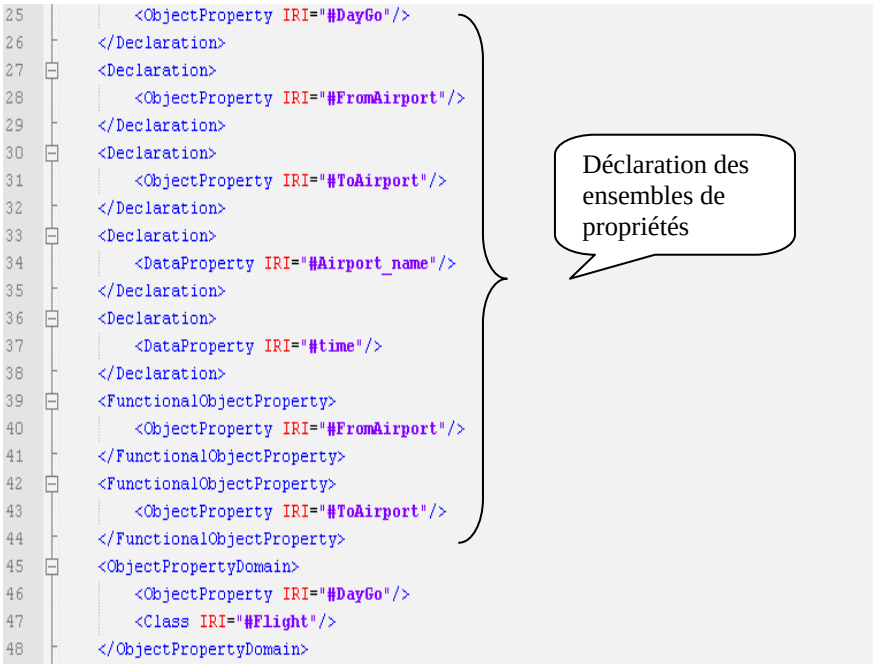
```xml
<?xml version="1.0" encoding="UTF-8"?>


<Root>

    <Flight>
        <FromAirport>Algeria</FromAirport>
        <ToAirport>France</ToAirport>
    </Flight>

    <Date>
      <Time> 10/04/2015 </Time>
    </Date>

</Root>
```

Figure 6.3. Le fichier XML qui représente la requête de départ

```xml
<?xml version="1.0"?>
<Ontology xmlns="http://www.w3.org/2002 07/owl#"
     xml:base="http://www.owl-ontologies.com/Ontology1270929847.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/ 2/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="http://www.owl-ontologies.com/Ontology1270929847.owl">
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="" IRI="http://www.owl-ontologies.com/Ontology1270929847.owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Declaration>
        <Class IRI="#Airport"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Date"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Flight"/>
    </Declaration>
    <Declaration>
```

Espaces de nommage

Déclaration des classes



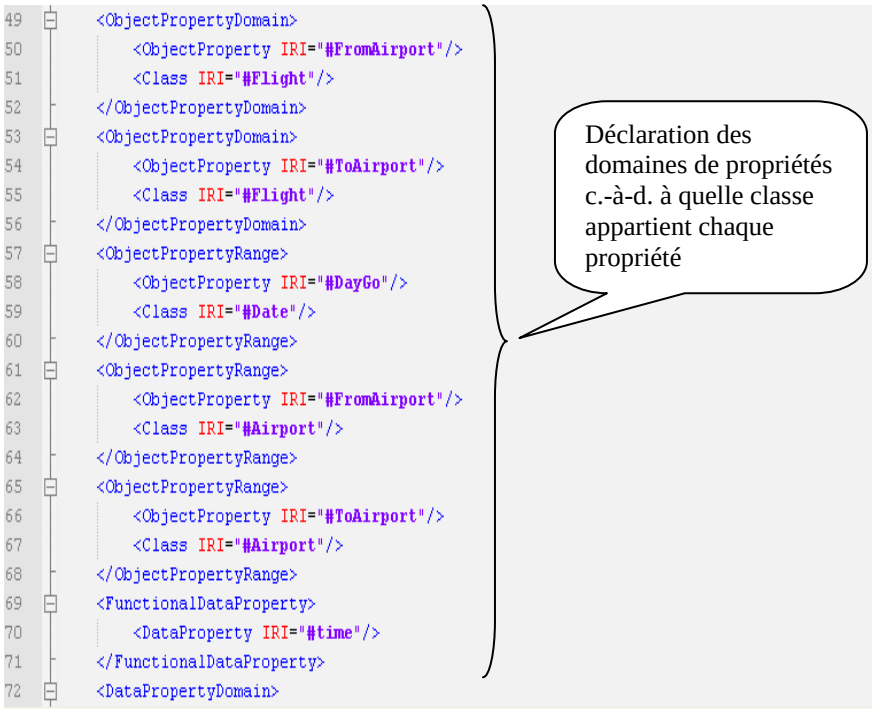
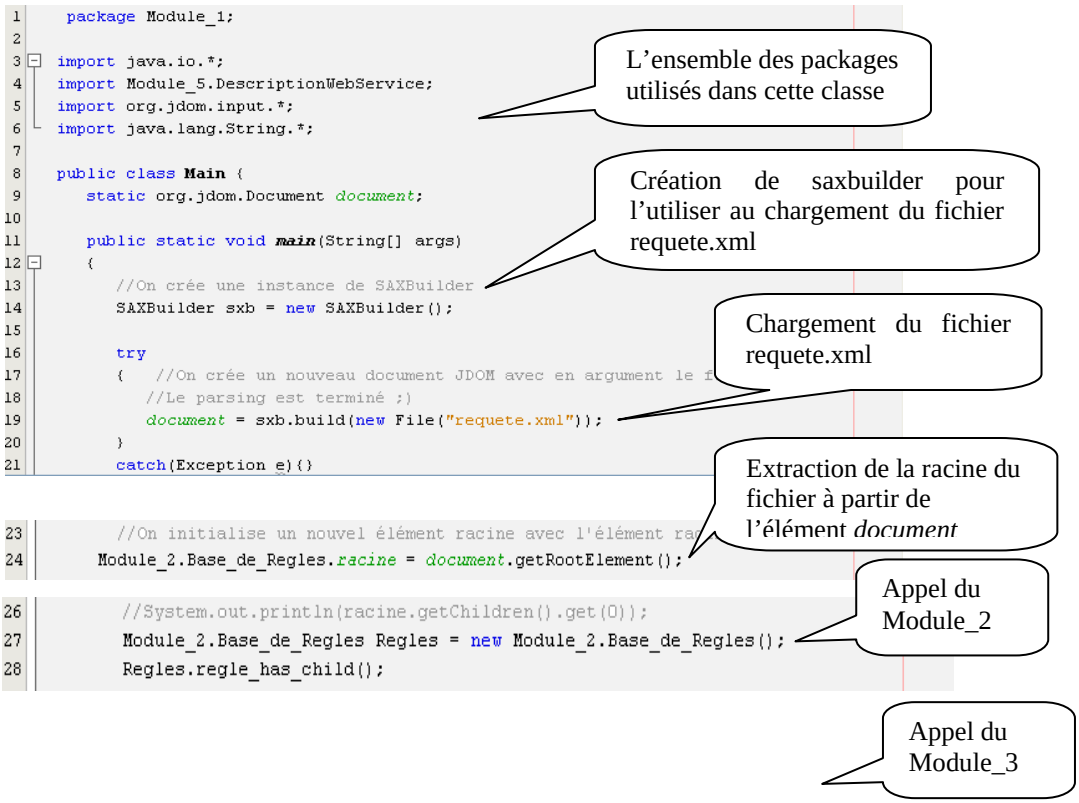


Figure 6.4. Le fichier *owl_xml.xml*

3.1.3. Moteur d'inférence. Le moteur d'inférence représente l'ensemble des traitements que l'Agent doit appliquer pour achever son travail. Il est représenté par la classe *Main.java*, cette classe permet de réaliser cinq tâches principales (cf. Figure 6.3) :

- Chargement du fichier *requete.xml* en utilisant deux classes du package JDOM (*saxbuilder* et *Document*).
- Faire appel au constructeur de la classe « Base_de_Regles.java» du Module_2 pour créer une instance qui s'appelle «Regle » puis la classe fait appel à sa methode «regle_has_child()».
- Faire appel au Module_3, après avoir créé les instances correspondantes à la requête, pour pouvoir changer le format de la base de connaissances au format rdf/xml .
- Faire appel au Module_5 qui produit un fichier « Service.owl » c'est le résultat de cet Agent (la description du service avec les instances produite à partir du Module_2).ces instances sont envoyées dès l'appel de la fonction du Module_5.
- Affichage de quelques informations concernant l'application.



```
30    //deuxième module qui change le format de l'ontologie de notre base
31    //de connaissances d' RDF/XML à OWL/XML
32  Module_3.Changer_Format_Ontologie module3=new Module_3.Changer_Format_Ontologie();
33  module3.changer();
```

```
38
39  System.out.println("it's ok this is the last instruction \n"+
40                     "if u want to see the objectif of this \n"+
41                     "intelligent agent go to window files and \n"+
42                     "select (requete.xml) which is the request \n"+
43                     "query of the user client and (Service.owl) \n"+
44                     "which represent the service corresponded to \n"+
45                     "the userquery.and all that will be after the \n"+
46                     "execution of course Thank you so much \n\n\n");
47  System.out.println("Group= 1: Soumatia Fatima Zohra \n"+
48                     "       2: SAIDI Halima");
49  }
```

Affichage d'informations d'application

Figure 6.5. Les composants du Module_1

3.1.4. Base de règles. Elle consiste en le Module_2, elle est composée de deux classes : *«Base_de_Regles.java, classe_xml.java »*. Le rôle de la première classe est d'extraire les balises et ses attributs à partir du fichier *requete.xml* (cf. Figure 6.4). Selon l'exemple démonstratif, on obtient deux instances : *Flight* et *Date ;*

- *« Flight »* avec les deux attributs : *« FromAirport », « ToAirport » et* comme valeurs *« Algeria », « France »* respectivement.
- *« Date »*avec l'attribut : *« Time »* a comme valeur *« 10/04/2015 »*.

Module 2

```
1   package Module_2;
2
3   import java.util.ArrayList;
4   import org.jdom.*;
5   import java.lang.String.*;
6
7
8   public class Base_de_Regles {
9
10
11    public static Element racine, element1, element2, element11, element12, element21;
12    String Flight, Date, FromAirportN, FromAirportV;
13    String ToAirportN, ToAirportV, TimeN, TimeV;
14
15  public classe_xml ch1=new classe_xml();
16  public classe_xml ch2=new classe_xml();
```

```
19
20    public void regle_has_child(){
21
22
23
24  // the name of classes Flight and Date
25          element1=racine.getChild("Flight");
26          element2=racine.getChild("Date");
27
28
29          Flight=element1.getName();
30          Date=element2.getName();
31
32  //the ArrayList of the two classes
33
34        ArrayList First_Names = new ArrayList();
35        ArrayList First_Values = new ArrayList();
36        ArrayList Seconde_Names = new ArrayList();
37        ArrayList Seconde_Values = new ArrayList();
38
39  // if the tags Flight and Date have other indertags so
```

Extraction des éléments qui se trouve après la racine

Appliquer la règle au premier élément

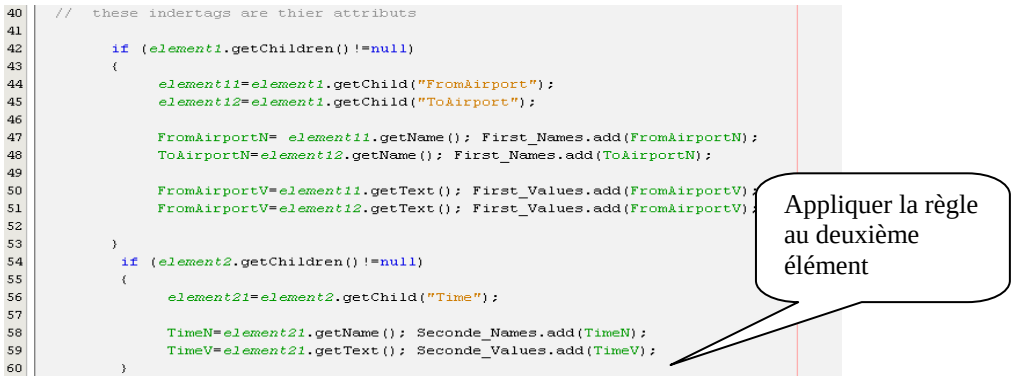


Figure 6.6. La classe Base_de_Règles et la représentation de ses règles

Après avoir extrait les informations nécessaires du fichier requete.xml qui sont : Le nom de la classe, le nom et la valeur de chacun de ses attributs, elle produit des *instances avec des attributs, en créant une instance au début de la classe « classe_xml »* qui s'appelle *« Cl »* puis elle fait appel à la fonction *« addchildclass() »* qui retourne une instance de même type *classe_xml*. Le rôle de cette classe est illustré dans la figure 6.5 et figure 6.6.

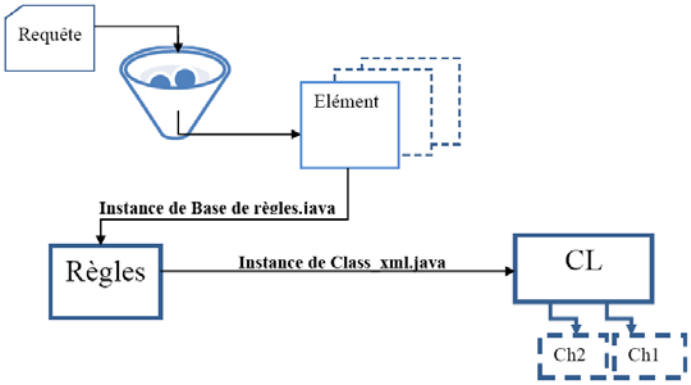


Figure 6.7. Production des instances

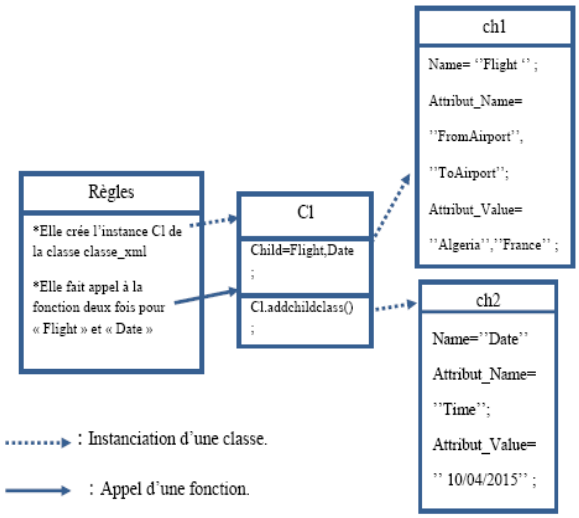


Figure 6.8. Instances produites en détail

Classe Changer_Format_Ontologie. Cette classe (cf. Figure 6.7) convertit le fichier en entrée *« OWL_XML.owl »*, qui est en format owl/xml, au format RDF/xml *«RDF_XML.owl »*.

Le résultat « RDF_XML.owl » est illustré dans la figure 6.8 ci-dessous.

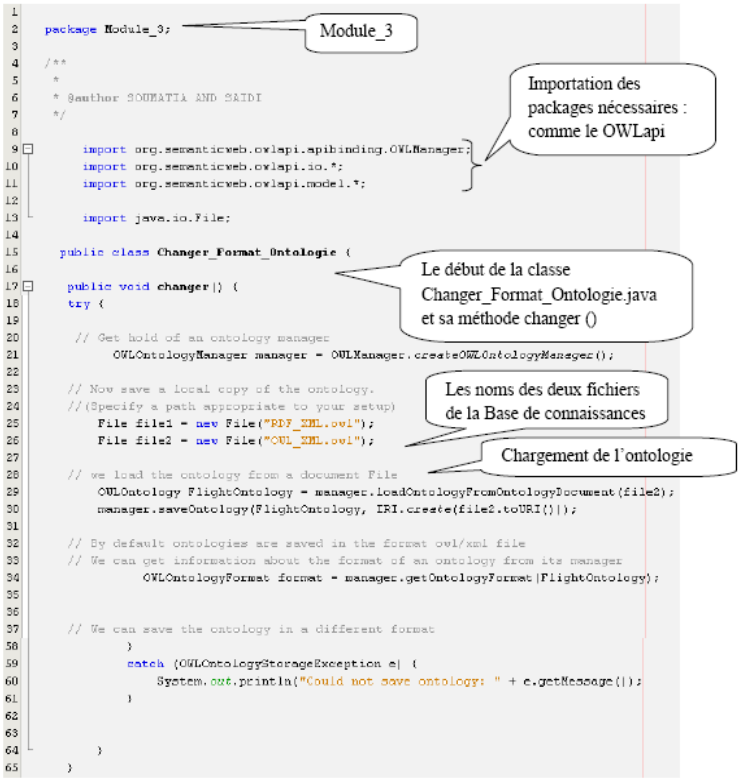


Figure 6.9. La classe Changer_Format_Ontologie.java

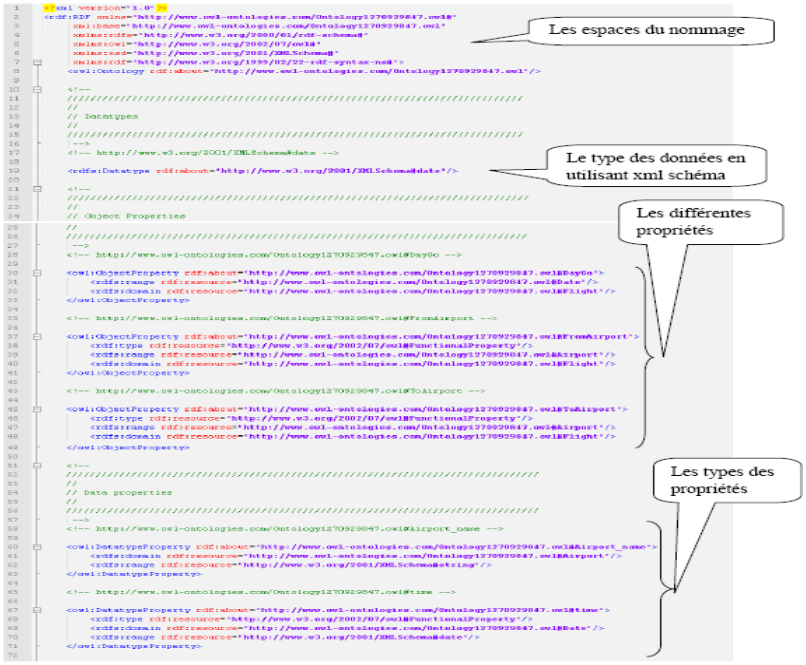


Figure 6.10. Le fichier RDF_OWL.owl

Le cinquième module de l'agent reconstructeur de requêtes, se compose de quatre classes *« DescriptionWebService.java,CreateServiceProperties.java,CreatePropertiesAtomicProcess.java, Afficher.java »*, son rôle est de produire un fichier OWL-S« squelette » qui représente la description du service. Il possède des méthodes de classes que l'on peut appeler pour ajouter des valeurs aux propriétés de chaque composant du service et ceci peut être fait par l'envoi des valeurs en utilisant ses paramètres (les paramètres des méthodes). Ces valeurs sont obtenues à partir des instances Flight et Date « le produit du Module_2 ».

DescriptionWebService.java. Cette classe crée une base d'ontologie et une ontologie vide des URIs, puis elle associe à cette ontologie les quatre composants nécessaires pour la description d'un service qui sont « Service, Profile, ProcessModel, Grounding », puis elle fait appel aux autres classes constituant ce module en envoyant les instances de la requête de l'utilisateur comme paramètres des fonctions pour qu'elle puisse produire le fichier « Service.owl » : la description du service correspondant à la requête.

CreatePropertiesProfile.java . Cette classe se charge de créer tous les attributs et les informations du profile et elle prend en compte toutes les données reçues afin de créer le profile avec ses valeurs. Par exemple, le nom du service et les inputs et les outputs du service…etc.

CreatePropertiesAtomicProcess.java. Cette classe ce charge de la génération de la section process du fichier OWL-S.

Afficher.java. La classe Afficher.java copie toutes les informations concernant la description du service correspondant à la requête dans un fichier nommé service.owl qui est le resultat de l'agent reconstructeur de requêtes (fichier de sortie) qui va être envoyé à l'agent manager-raisonneur.

3.1.5. Quelques captures d'écran

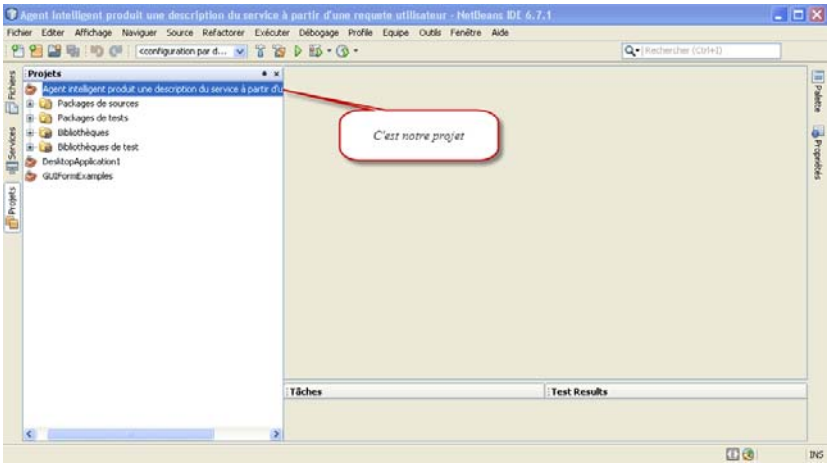


Figure 6.11. L'agent reconstructeur de requêtes sous l'environnement NetBeans

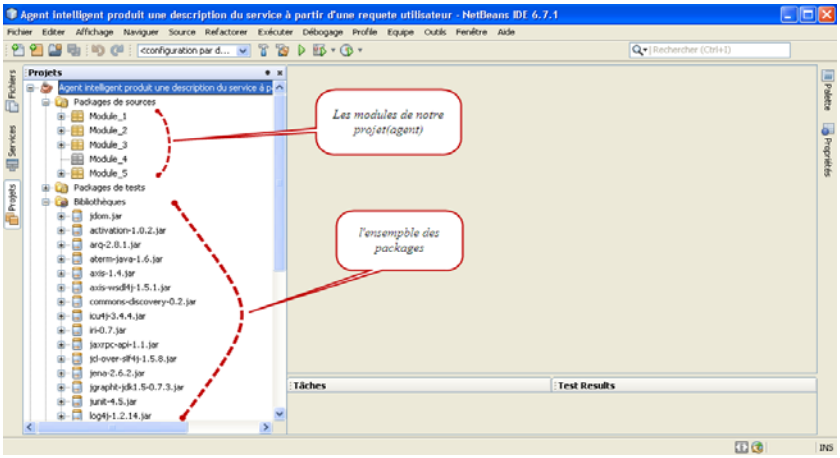


Figure 6.12. Ensemble de modules composant l'agent reconstructeur de requêtes

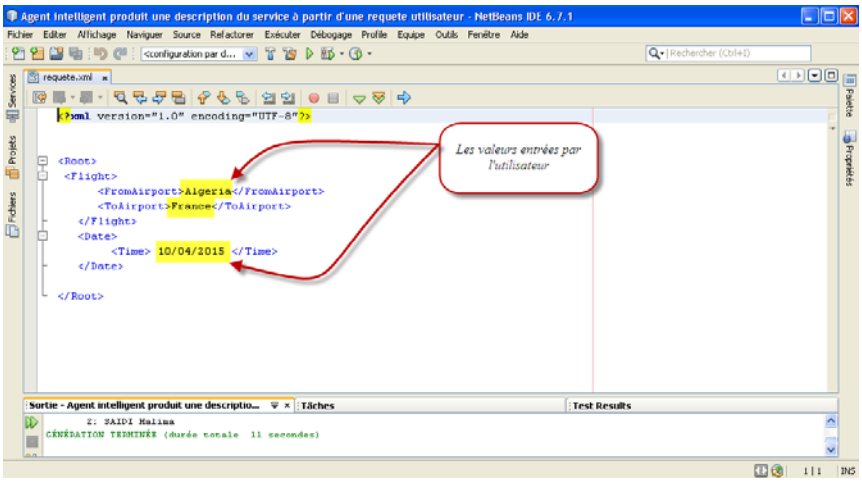


Figure 6.13. La requête utilisateur sous forme d'un fichier XML

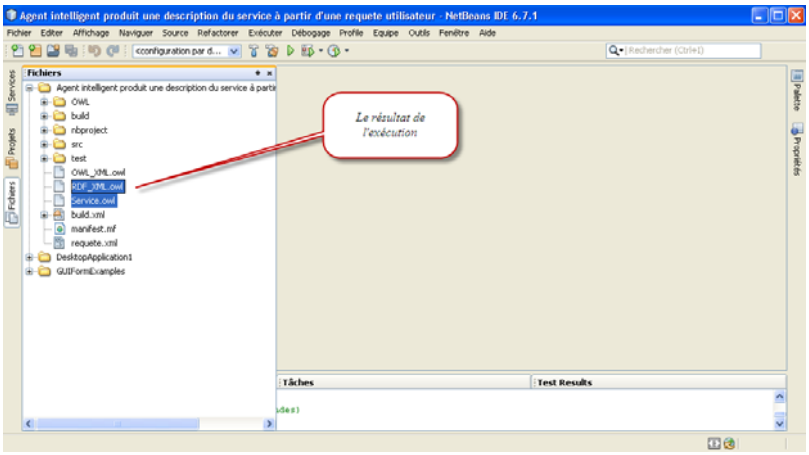


Figure 6.14. Résultat de l'exécution

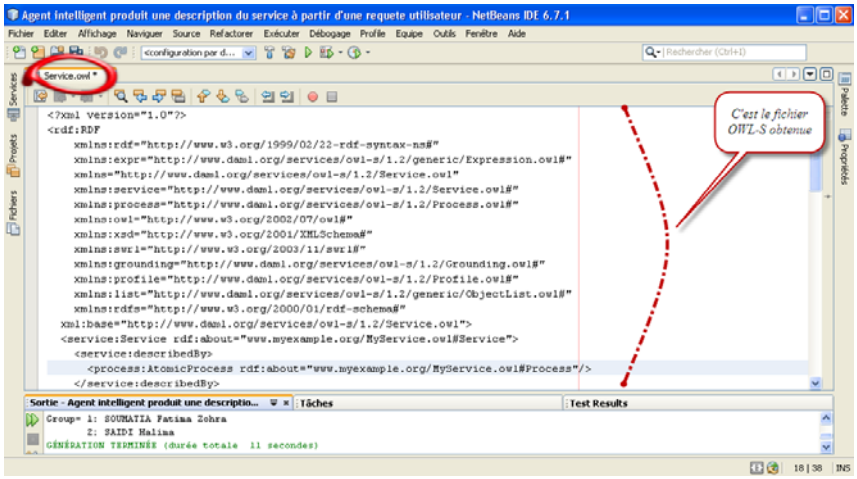


Figure 6.15. Le fichier OWL-S en sortie

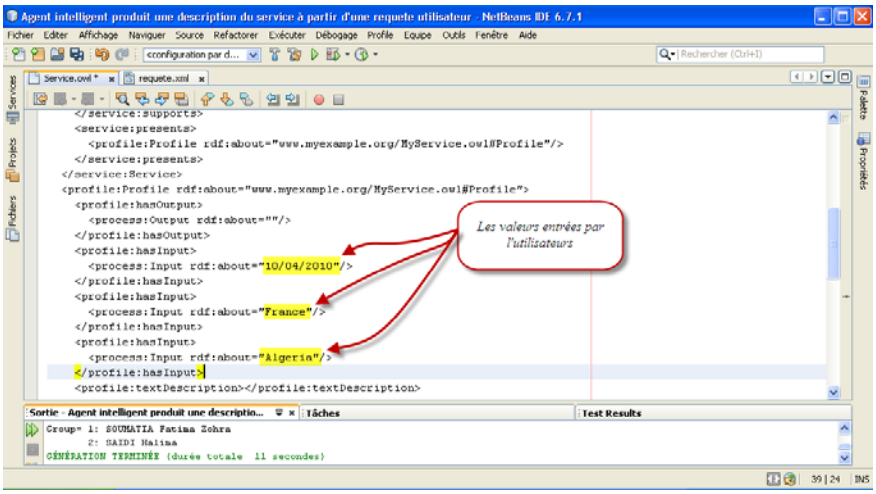


Figure 6.16. Valeurs des instances

4. Conclusion

Dans la première partie de ce chapitre, nous avons illustré notre modèle par une étude de cas qui porte sur la composition de services web aboutissant à l'organisation d'un voyage pour un enseignant désirant assister à une conférence à Tokyo. Cette étude a mis en avant le dialogue de trois agents dans le but de proposer un plan d'exécution des services qui répond favorablement à la requête de l'utilisateur.

Dans la deuxième partie, nous avons exposé quelques résultats donnés à l'implémentation de l'agent reconstructeur de requêtes, nous avons utilisé pour cela plusieurs outils tel que le NetBeans. Pour implémenter notre architecture, nous avons choisis la plateforme JADE. Cette plate-forme permet de simplifier le développement des systèmes multi-agent tout en fournissant un ensemble complet de services et d'agents conformes aux spécifications FIPA.

CONCLUSION GENERALE ET PERSPECTIVES

Des environnements tels que le web et l'informatique diffusée sont des environnements ouverts et distribués. Ils se caractérisent par des services très flexibles et dynamiques (où de nouvelles propriétés peuvent apparaître pour un service et de nouveaux services peuvent être disponibles sur une base quotidienne) et où les besoins des utilisateurs en services varient. Dans un tel contexte, le processus de composition de services web pour la satisfaction des besoins de l'utilisateur doit pouvoir s'adapter de manière dynamique aux besoins de ce dernier avec un minimum d'intervention de sa part.

C'est dans le cadre de cette problématique que notre travail de recherche s'est effectué. Après avoir présenté cette problématique et les verrous sous-tendus, ce manuscrit a décrit notre proposition. Celle-ci a consisté en l'étude et la conception d'un modèle dynamique de composition de services web sémantiques basée sur une architecture à base d'agents utilisant des mécanismes de planification multi-agent à savoir la planification sous hypothèse avec synthèse dialectique de plan. L'objectif est de faire intégrer cette architecture à la spécification fonctionnelle d'ebXML.

Il existe deux grands courants de composition de service web. Le premier, principalement utilisé par le monde industriel, permet de définir des procédés réutilisables sous la forme d'orchestration et de chorégraphie en utilisant des langages de gestion de procédés. Le second courant cherche à définir un enchaînement dynamique de services web permettant de résoudre un but spécifique fournit par l'utilisateur en tenant compte de ses préférences. Cette composition dynamique s'appuie principalement sur des techniques d'intelligence artificielle.

Après avoir étudié les travaux de ces deux courants, nous avons pu identifier les limites récurrentes des travaux existants de composition de services ainsi que les limites intrinsèques aux services web pour la conception d'une telle approche. C'est ensuite en étudiant les propriétés et modèles proposés dans la communauté agent et multi-agent que nous avons constaté que les systèmes multi-agent étaient un bon candidat pour la modélisation des problématiques impliquées par une approche dynamique de composition de services. Plus précisément, les modèles de planification multi-agent semblaient particulièrement bien adaptés pour modéliser les collaborations dynamiques entre services.

De ce fait, nous avons pu conclure que la solution consisterait en un modèle de composition car comme le requiert la solution à la problématique de la composition dynamique, ce modèle est applicable dans un contexte de résolution semi-distribuée de problèmes où un but global à satisfaire est présupposé (correspondant à un ensemble de besoins énoncés par l'utilisateur à satisfaire dans le problème de la composition de services).

Donc, la principale contribution de cette thèse consiste en la proposition à la fois d'un modèle de composition dynamique de services web et d'une architecture multi-agent qui l'a supporte. Aboutir à une telle proposition a requis la provision des contributions subalternes et résultats suivants :

– Une architecture générale combinant technologie services web et système multi-agent. L'implémentation de cette architecture est basée sur la plate-forme Jade. Cette plate-forme permet de simplifier le développement des systèmes multi-agent tout en fournissant un ensemble complet de services et d'agents conformes aux spécifications FIPA.

– Un modèle dynamique de composition de services web sémantiques.

Le modèle de composition que nous avons proposé est basé sur la planification semi-distribuée avec la synthèse dialectique de plans sous hypothèses. La particularité de notre modèle est que l'aboutissement à un plan solution n'est pas totalement distribué du fait de l'utilisation d'un agent compositeur qui collabore avec les différents agents manager (services Web retenus). L'idée est qu'il y a toujours un agent qui supervise l'état d'avancement du processus de composition et qui a une vision globale permettant de localiser le problème en cas d'échec. Les points qui font la force de ce modèle et qu'il nous semble important de souligner sont les suivants :

– l'utilisateur joue un rôle central dans la composition de services : c'est lui qui fixe l'objectif que l'agent compositeur avec les agents manager, représentant les services Web, doivent résoudre ;

– un agent manager peut apporter sa contribution à l'élaboration de la composition même s'il ne dispose pas de toutes les connaissances nécessaires. Il émet l'hypothèse qu'un autre agent sera capable de lui apporter les données manquantes.

Perspectives

Les expérimentations préliminaires de l'architecture proposée afin de supporter le modèle de composition proposé dans cette thèse, ont donné lieu à des propositions et à des réflexions à prendre en considération dans les futurs travaux, à savoir :

- Améliorer l'algorithme de traduction, utilisé, du langage OWL-S vers le formalisme de planification en permettant la traduction des structures de contrôles qui n'ont pas encore traitées, à savoir les structures de contrôle classique tels que *if-then-else* ou encore *repeat-until*, mais également les structures de contrôles qui n'ont pas de correspondance directe dans le planificateur que nous utilisons, comme *concurrent* (exécuter les services en concurrence), ou encore *Choice* (choisir un service parmi *n*).
- Prendre en considération le traitement des étapes en amont et en aval de la composition, i.e., la recherche de services web et l'invocation des services intervenant dans cette composition.
- Intégrer les techniques d'appariement sémantique dans le raisonnement des deux agents *manager-raisonneur* et *compositeur* afin de pouvoir sélectionner des services répondant à une plus haute qualité de services.
- Proposer un mécanisme de gestion de la disponibilité et de la non disponibilité dynamique des agents manager.

BIBLIOGRAPHIE

[1] A. Andrieux et al. Web Services Agreement Specification (WS-Agreement). Technical report, Globus Alliance, 2004.

[2] A. Berger and S. Pesty. Towards a Conversational Language for Artificial Agents in Mixed Community. In Proc. of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05), 2005, pp. 31–40.

[3] A. El Fallah-Seghrouchni and S. Haddad. A Coordination Algorithm for Multi-Agent Planning. j-LECT-NOTES-COMP-SCI, 1038.

[4] A. Gustavo, F. Casati, H. Kuno, and V. Machiraju. Web Services. Concepts, Architectures and Applications. Springer-Verlag Berlin Heidelberg, 2004.

[5] A. Rubinstein. Perfect Equilibrium in a Bargaining Model. Econometrica, 50(1), 1982, pp. 97–109.

[6] A.D. Preece, K. Hui, W.A. Gray, P. Marti, T.J.M. Bench-Capon, D.M. Jones, and Z. Cui. The KRAFT Architecture for Knowledge Fusion and Transformation. Knowledge Based Systems, 2000, 13(2-3), pp 113–120.

[7] A.H. Bond and L. Gasser. Readings in Distributed Artificial Intelligence. Morgan Kaufmann, 1988.

[8] B. Benatallah, M. Dumas, M.C. Fauvet, F.A. Rabhi, and Q.Z. Cheng. Towards Patterns of Web Services Composition.

[9] B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Semantic web enabled composition of web services. The VLDB Journal 12, 4, November 2003, pp. 333–351.

[10] C. Jonquet. Dynamic Service Generation : Agent interactions for Service Exchange on the Grid. PhD thesis, Université Montpellier 2, Montpellier, France.

[11] C. Parent and S. Spaccapietra. Issues and Approaches of Database Integration. Communications of the ACM, 41(5) , 1998, PP166–178.

[12] C. Peltz. Web Services Orchestration and Choreography. Computer, 26(10), 2003, pp46–52.

[13] C. Peltz. Web Services Orchestration and Choreography. IEEE Computer, vol.36, n°10, pp.46-52.

[14] C. Preist, A. Byde, C. Bartolini, and G. Piccinelli. Towards Agent-based Service Composition through Negotiation in Multiple Auctions. Technical Report hpl-2001-71, Hewlett Packard, 2001.

[15] C. Preist. Goals and Vision, Combining Web Services with Semantic Web Technology. **In:** R. Studer, S. Grimm, A.Abecker, Coord. Semantic Web Services – Concepts, Technologies, and Applications. Springer Berlin Heidelberg, 2007, pp.159-178.

[16] D. Austin, A. Barbir, E. Peters, and S. Ross-Talbot. Web Services Choreography Requirements W3C Working Draft. Technical report, World Wide Web Consortium (W3C), 2004.

[17] D. Austin,A. Barbir, E. Peters, and S. Ross-Talbot. Web Services Choreography Requirements. W3C Working Draft, 2004. http://www.w3.org/TR/ws-chor-reqs.

[18] D. Benmerzoug, Z. Boufaida, and M. Boufaida. From the Analysis of Cooperation Within Organizational Environments to the Design of Cooperative Information Systems : An Agent-Based Approach. In OTM Workshops, volume 3292 of LNCS, , Larnaca, Chypre, October 2004. pp. 495–506

[19] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. Technical report, W3C Working Group Note 11, 2004.

[20] D. Booth, M. Champion, C. Ferris, F. McCabe, E. Newcomer, and D. Orchard. Web Services Architecture. http://www.w3.org/TR/2003/ WD-ws-arch-20030514/, May 2003.

[21] D. Corkill. Hierarchical planning in a distributed environment. In Proceedings of the International Joint Conference on Artificial Intelligence,Tokyo, Japan, August 1979, Morgan Kaufmann Publishers, pp. 168–175.

[22] D. Fensel, C. Bussler, and A. Maedche. Semantic Web Enabled Web Services. Procs of the 1st International Semantic Web Conference (ISWC 2002), LNCS Springer 2002, Sardinia, Italy, pp.1-2.

[23] D. Fensel, C. Bussler, Y. Ding, and B. Omelayenko. The Web Service Modeling Framework WSMF. Electron Commerce Res, 2002, vol.1, n°2, pp.113–137.

[24] D. Martin et al. OWL-S : Semantic Markup for Web Services. Technical report, DAML Organization, 2004.

[25] D. Martin, M. Burstein, J. Hobbs, Paolucci, O. Lassila, D. McDermott, S. McIlraith, D. McGuinness, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. W3C Member Submission, 2004. http://www.w3.org/Submission/OWL-S

[26] D. McDermott. Estimated-regression planning for interactions with web services. In Proc of the 6th International Conference on AI Planning and Scheduling, Toulouse, France, 2002, AAAI Press.

[27] D. Pellier, and H. Fiorino. Dialectical theory for multi-agent assumption based planning. In Proc of CEEMAS'05 (2005), Springer Verlag Publishers, pp. 367–376.

[28] D. Pellier, and H. Fiorino. Un modèle de composition automatique et distribué de services Web par planification", RSTI - RIA - 23/2009. Intelligence artificielle et web intelligence, pp. 13-46.

[29] D. Pellier. Modèle dialectique pour la synthèse de plans. PhD thesis, UJF- Grenoble, 2005.

[30] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. Applied ontology, 2005, vol.1, pp.77-106.

[31] D. Vanderveken. On the Unification of Speech Act Theory and Formal Semantics. In P.R. Cohen, J. Morgan, and M.E. Pollack, editors, Intentions in Communication, MIT Press, Cambridge, MA, 1990. pp. 195–220.

[32] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In ISWC'03 (2003).

[33] D.L. McGuiness, and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, 2004. http://www.w3.org/TR/owl-features/

[34] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL).
http://www.w3.org/TR/wsdl, 2001.

[35] E. Durfee, and V. Lesser. Partial Global Planning : A coordination framework for distributed hypothesis formation. In IEEE Transactions on Systems, Man, and Cybernetics, September 1991, vol. 21, pp. 1167–1183.

[36] E. Durfee. Distributed problem solving and planning. In Lecture Notes in Computer Science. Springer Verlag Publishers, 2001, pp. 118–149.

[37] E. Oliveira, K. Fisher, and O. Stepankova. Multi-Agent Systems : Which Research for which Applications. Robotics and Autonomous Systems,1999, pp. 91– 106.

[38] E. Sacerdoti. A structure for plans and behavior. Elsevier Science Publishers, 1977.

[39] E. Sirin, and B. Parsia. Planning for semantic web services. In Proc of Semantic Web Services Workshop at 3rd International Semantic Web Conference, 2004.

[40] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. In Proc Web Services : Modeling, Architecture and Infrastructure. Workshop in Conjunction with ICEIS2003, Angers, France, 2002, ICEIS Press, pp. 17–24.

[41] E.H. Durfee and V. Lesser. Using Partial Global Plans to Coordinate Distributed Problem Solvers. In Proc. the 10th International Joint Conference on AI.

[42] ebXML, 2003. http://www.ebXML.org/specs/ ebTA.pdf

[43] F. Baader, C. Lutz, M. Milicic, U. Sattker, and F. Wolter. A Description Logic Based Approach to Reasoning about Web Services. Procs of the International Conference WWW, Workshop on Web Service Semantics: Towards Dynamic Business Integration (WSS2005), May 2005, Chiba Japan.

[44] F. Casati, M. Sayal, and M.C. Shan. Developping e-services for composing e-services. In Proc of 13th International Conference on Advanced Information Systems Engineering (CAiSE) (Interlaken, Switzerland, June 2001), S. Verlag, Ed.

[45] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.C. Shan. Adaptive and dynamic service composition in EFlow. In Proc of the 12th Conference on Advanced Information Systems Engineering (CAISE) (Stockholm, Sweden, June 2000), S. Verlag, Ed., pp. 13–31.

[46] F. Von Martial. Interactions among Autonomous Planning Agents. In Y. Demazeau and J.P. Müller, editors, Proc. of the 1st European Workshop on Modelling Autonomous Agents in Multi-Agent World, Cambridge, England.

[47] FIPA English Auction Interaction Protocol Specification. http://www.fipa.org/specs/fipa00031/index.html, 2001.

[48] FIPA Interaction Protocol Library Specification, 2001. http://www.fipa.org/ specs/fipa00025/XC00025E.pdf.

[49] FIPA Request Interaction Protocol Specification. http://www.fipa.org/specs/fipa00026/SC00026H.html, 2002.

[50] FIPA. Foundation for Intelligent Physical Agents : Communicative Act Library Specification, 1997.

[51] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web Services - Concepts, Architectures and Applications. Springer-Verlag, 2004.

[52] G. Klyne, and J.J Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 2004. http://www.w3.org/TR/rdf-concepts/

[53] G. Vauvert and A. El Fallah-Seghrouchni. Formal Specification of Opinions Applied to the Consensus Problem. In Proc. of the 8th Iberoamerican Conference on Artificial Intelligence. (IBERAMIA'02), 2002.

[54] G. Zlotkin and J.S. Rosenschein. Negotiation and Conflict Resolution in Non-Cooperative Domains. In Proc. of the AAAI'90, Boston, MA, 1990, pp. 100–105.

[55] Gartner. Hype Cycle for Web Services. http://www.gartner.com/, 2007.

[56] H. Baumeister. Customer Relationship Management for SME, 2002.

[57] H. Guyennet, D. Saint-Voirin, and I. Rasovska. State of the Art in Multiple Data Format Handling. Technical report, ITEA Proteus, 2003.

[58] H. Lauren, D. Roman, and U. Keller. Web Services Modeling Ontology-Standard, (WSMO-Standard), March 2004. http://wsmo.org/2004/d2/v0.2/.

[59] H.S. Thompson, D.Beech, M.Maloney, N.Mendelsohn. XML Schema Part 1: Structures Second Edition. W3C Recommendation **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/TR/xmlschema-1/>.

[60] I. Müller, R. Kowalczyk, and P. Braun. Towards Agent-based Coalition Formation for Service Composition. In Proc. of the International Conference on Intelligent Agent Technology (IAT'06), 2006, pp 73–80.

[61] I.Constantinescu, B. Faltings, and W. Binder. Type based service composition. In Proc of the 13th international World Wide Web Conference on Alternate Track, New York, USA, 2004, ACM Press, pp. 268–269.

[62] I.Muller, and R. Kowalczik. Service composition through agent-based coalition formation. In Proc of WWW Service composition with Semantic Web Services, Compiegne, France, Septembre 2005, pp. 34–43.

[63] IBM Alphaworks. BPWS4J. http://www.alphaWorks.ibm.com/tech/ bpws4j, 2002.

[64] IBM, Microsoft, SAP, Siebel Systems. Business Process Execution Language for Web Services Version 1.1. Technical report, 2003.

[65] J. Bentahar, Z. Maamar, D. Benslimane, and P. Thiran. Using Argumentative Agents to Manage Communities of Web Services. In Proc. of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07), Washington, DC, USA, 2007. IEEE Computer Society. pp. 588–593

[66] J. Cao, M. Li, S.S. Zhang, and Q. Deng. Composing Web Services based on Agent and Workflow. In Proc. of the 2nd International Workshop on Grid and Cooperative Computing (GCC), volume 3032, 2004, pp 948–955,.

[67] J. Farrell, and H. Lausen. Semantic Annotations for WSDL and XML Schema. W3C Recommendation, 2007. http://www.w3.org/TR/sawsdl/

[68] J. Ferber. Les systèmes multi-agents : Vers une intelligence collective. Inter-Editions, 1995.

[69] J. Guitton. Planification multi-agent pour la composition dynamique de services web. Rapport de stage, Master 2 Recherche, Grenoble 1, 2006.

[70] J. Moon, D. Lee, C. Park, and H. Cho. ebXML BP Modeling Toolkit. In Proc. of the 7th International Conference on Enterprise Distributed Object Computing (EDOC), page 296. IEEE Computer Society, 2003.

[71] J. Peer. A PDDL based tool for automatic web services composition. In Proc of the Second Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2004) at the 20th International Conference on Logic Programming, September 2004, Springer Verlag, pp. 149–163.

[72] J. Peer. Web service composition as planning - a survey. Tech. rep., Univ. Of St.Gallen, 2005.

[73] J. Pitt and A. Mamdani. Some Remarks on the Semantics of FIPA's Agent Communication Language. Autonomous Agents and Multi-Agent Systems, 2(4) ,

1999, pp. 333–356.

[74] J. Rao, and X. Su, X. A survey of automated web service composition methods. In Proc of Semantic Web Services and Web Process Composition, First International Workshop,San Diego, CA, July 2004.

[75] J. Rao, P. Kungas, and M. Matskin. Application of linear logic to web service composition. In Proc of the 1st International Conference on Web Services, Las Vegas, USA, June 2003.

[76] J.R. Searle. Speech Acts. Cambridge University Press, 1969.

[77] JACK. JACK Intelligent Agents Tutorials, Agent Oriented Software Pty. http://www.disi.unige.it/person/ZiniF/Didattica/Jack/jack_tutorial.pdf.

[78] JADE. Java Agent DEvelopement Framework. http://jade.tilab.com.

[79] K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan. Dynamic Discovery and Coordination of Agent-based Semantic Web Services. IEEE Internet Computing, 8(3) 2004, pp. 66–73.

[80] L. Amgoud, N. Maudet, and S. Parson. Arguments, dialogue and negotiation. In Proc of the European Conference on Artificial Intelligence, Berlin, August 2000, pp. 338–342, W. Horn, Ed., IOS Press,

[81] L.G. Gasser. The Social Dynamics of Routine Computer Use in Complex Organizations (Work, Management, Sociology). PhD thesis, 1984.

[82] M. Amor, L. Fuentes, and J.M. Troya. Putting Together Web Services and Compositional Software Agents. In Proc. of the International Conference on Web Engineering (ICWE'03), 2003, pp. 44–53.

[83] M. Carman, L. Serafini, and P. Traverso. Web service composition as planning. In Proc of ICAPS'03 Workshop on Planning for Web Services, Trento, Italy, June 2003.

[84] M. Ghallab, D. Nau, and P. Traverso. Automated Planning, Theory and Practice. *Morgan Kaufmann'04.2004*

[85] M. Laukkanen, and H. Helin. Composing workflows of semantic web services. In Proc of the Workshop on Web-Services and Agent-based Engineering (2003).

[86] M. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz. Reference Model for Service-Oriented Architecture 1.0. Technical Report wd-soa-rmcdl, OASIS.

[87] M. Matskin, P. Küngas, J. Rao, J. Sampson, and S. Abbas Petersen. Enabling Web Services Composition with Softward Agents. In IMSA , 2005 pp 93–98.

[88] M. P. Papazoglou. Agent-oriented technology in support of e-business. Commun. ACM, 44(4), 2001, pp. 71–77.

[89] M.P. Papazoglou and D Georgakopoulos. Service Oriented-Computing. Communications of the ACM, 46(10) :25–28, 2003.

[90] M.P. Singh and M.N. Huhns. Service-Oriented Computing, semantics, processes, agents. John Wiley and Sons, 2005.

[91] M. Sheshagiri, M. Desjardins, and T. Finin. A planner for composing services described in DAML-S. In Proc of the AAMAS Workshop on Web Services and Agent-based Engineering, Jun 2003.

[92] M. Singh. Agent communication languages : Rethinking the principles. IEEE Computer 31, 12 (December 1998), pp. 40–47.

[93] M. Wooldridge. An Introduction to Multi-Agent Systems. 2002.

[94] M. Wooldridge. Semantic Issues in the Verification of Agent Communication Languages. Autonomous Agents and Multi-Agent Systems, 3(1) , 2000, pp. 9–31.

[95] M. Wooldrige and N.R. Jennings. Intelligent Agents : Theory and Practice. The Knowledge Engineering Review, 10(2) 1995, pp. 115–152.

[96] M.N. Huhns and L.M. Stephens. Multiagent Systems and Societies of Agents, 1999, pp. 79–120.

[97] M.N. Huhns and M.P. Singh, editors. Readings in Agents. Morgan Kaufmann, San Francisco, CA, USA, 1998.

[98] M.N. Huhns. Agents asWeb Services. IEEE Internet Computing, 6(4) 2002 pp.93–95.

[99] M.N. Huhns. Software Agents : The Future of Web Services. In Agent Technologies, Infrastructures, Tools, and Applications for E-Services, 2002, pp. 1–18.

[100] N. Guarino. Semantic Matching : Formal Ontological Distinctions for Information Organisation, Extraction, and Integration. A multidisciplinary approach to an emerging information technology. International summer school , 1997, pp 139–170.

[101] N. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. In Proc. AAMAS'98, 1998.

[102] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Services Description Language (WS-CDL). Technical report, W3C, 2004.

[103] N.R. Jennings. An Agent-based Approach for Building Complex Software Systems. Communication of the ACM, 44(4) 2001, pp. 35–41.

[104] O. Hioual, and Z. Boufaida . Vers une architecture à base d'agents pour une composition sémantique et dynamique des services web dans un contexte d'ebXML", 8th ENIM/IFAC International Conference on Modelling and Simulation - MOSIM'10 - Hammamet – 2010 Tunisia, 10-12 May. pp 931-939

[105] O. Hioual, and Z. Boufaida. Towards a Semantic Composition of ebXML Business Processes, International Conference on Innovations in Information Technology. IIT 2008, Dubai, 16-18 Dec, pp.165-169.

[106] O. Hioual, Z. Boufaida. An Agent Based Architecture (Using Planning) for Dynamic

and Semantic Web Services Composition In an ebXML Context. International Journal of Database Management Systems (IJDMS) Volume 3, N°1, February 2011, pp. 110 - 131.

[107] O. Hioual, Z. Boufaida. Using Intelligent Agents in Conjunction with Heuristic Model for Negotiating ebXML CPP. In Information and Communication Technologies, ICCTA'06, pp. 268-273.

[108] O. Shehory and S. Kraus. Methods for Task Allocation via Agent Coalition Formation. Artificial Intelligence, 101(1–2) 1998, pp.165–200.

[109] O. V. Tschammer. An Agent-Based Platform for the Management of Dynamic Virtual Enterprises. PhD thesis, Berlin, 2001.

[110] P. Caillou, S. Aknine, and S. Pinson. Méthode de formation et de restructuration dynamique de coalitions d'agents fondée sur l'optimum de Pareto. In Proc. 9èmes Journées Francophones d'Intelligence Artificielles et Systèmes Multi-Agents (JFIAD), 2001.

[111] P. McBurney, S. Parsons, and M. Wooldridge. Desiderata for Agent Argumentation Protocols. 2002.

[112] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.T. Schmidt, A. Sheth, and K. Verma. Web Service Semantics – WSDL-S. W3C Member Submission, 2005. http://www.w3.org/Submission/WSDL-S

[113] R. Alami, F. Ingrand, and S. Suzuki. A paradigm for plan-merging and its use for multi-robot cooperation. In Proc of IEEE International Conference on Systems, Man, and Cybernetics, San Antonio, USA, 1994, pp. 612–617.

[114] R. Alami, F. Robert, F Ingrand, and S. Suzuki. Multi-Robot Cooperation through Incremental Plan-Merging. In Proc. of the international Conference on Robotics and Automation,. IEEE Computer Society Press, 1995, pp. 2573–2579.

[115] R. Finke, and N. Nilsson. STRIPS : A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 3-4 , 1971, pp. 189–208.

[116] R. Khalaf, N. Mukhi, and S. Weerawarana. Service-oriented composition in BPEL4WS. In Proc of the 12th International World Wide Web Conference (2003).

[117] R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda. Task ontology for reuse of problem solving knowledge. In Proc. 2nd international conference on building and sharing of very large-scale knowledge bases, 1995.

[118] R. Waltinger. Web agents cooperating deductively. In Proc of FAABS'00, April 2000, Greenbelt, Ed., vol. 1871 of Lecture Notes in Computer Sciences, Springer-Verlag, pp. 250–262.

[119] R.G. Smith. The Contract Net Protocol : High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers, C-29(12) , 1981, pp. 1104–1113.

[120] S. Aknine. Contribution à l'étude des méthodes de coordination dans les systèmes multi-agents, 2006.

[121] S. Aknine. Modèles et méthodes de coordination dans les systèmes multiagents. PhD thesis, Thèse Doctorat. Université Paris IX Dauphine, 2000.

[122] S. Ambroszkiewicz. Agent-based Approach to Service Description and Composition. In Proc. 1st International Workshop on Radical Agent Concepts (WRAC), 2003, pp 135–149.

[123] S. Dustdar and W. Schreiner. A Survey on Web services Composition. International Journal of Web and Grid Services, 1, 2005.

[124] S. Kraus, M. Nirkhe, and K. Sycara. Reaching Agreements through Argumentation : A Logical Model (preliminary report). In Proc. of the 12th International Workshop on Distributed Artificial Intelligence , Hidden Valley, Pennsylvania, 1993, pp. 233–247.

[125] S. McIlraith, and D. Martin. Bringing Semantics to Web Services. IEEE Intelligent Systems, 2003, vol.18, n°1, pp.90-93.

[126] S. McIlraith, and T. Son. Adapting Golog for composition of semantic web services. In Proc of the 8th International Conference on Knowledge Representation and Reasoning (KR '02), Toulouse, France, April 2002, Morgan Kaufmann Publishers, pp. 482–493.

[127] S. Narayanan, and S. McIlraith. Simulation, verification and automated composition of web services. In Proc of the 11th international conference on World Wide Web Honolulu, USA, July 2002, ACM.

[128] S. Paurobally, V. Tamma, and M. Wooldridge. Cooperation and Agreement between Semantic Web Services. In W3C Workshop on Frameworks for Semantics in Web Services. Innsbruck, Austria, June 2005.

[129] T. Finin, R. Fritzson, and D. McKay. An overview of KQML : A Knowledge Query and Manipulation Language. Technical report, University of Maryland Baltimore Country, 1992.

[130] T. Jarraya. Réutilisation des protocoles d'interaction et Démarche orientée modèles pour le développement multi-agents. PhD thesis, Université de Reims Champagne Ardenne, Décembre 2006.

[131] T. Melliti and S. Haddad. Synthesis of Agents for Web Services Interaction. In Proc. of Workshop on Semantic Web Services for Enterprise Application Integration and E-Commerce of the 5th International Conference on Electronic Commerce, Pittsburgh, USA, 2003.

[132] T. Osman, D. Thakker, and D. Al-Dabass. Bridging the Gap between Workflow and Semantic-based Web services Composition. In Proc. of the Web Service Composition Workshop WSCOMPS05, 2005.

[133] T. Osman, D. Thakker, and D. Al-Dabass. Bridging the gap between workflow and semantic-based web services composition. In Proceedings of WWW Service composition with Semantic Web Services (Compiegne, France, Septembre. 2005,

pp. 13–23.

[134] T. W. Finin, R. Fritzson, D. P. McKay, and R. McEntire. KQML As An Agent Communication Language. In Proc of the Third International Conference on Information and Knowledge Management (CIKM'*94*), ACM, 1994. pp. 456–463.

[135] T.R. Gruber. A Translation Approach to Portable Ontologies. Knowledge Acquisition, 1993, vol.5, n°2, pp.199-220.

[136] T.R. Payne, and O. Lassila. Semantic Web Services. IEEE Intelligent Systems, 2004, vol.19, n°4, pp.14-15.

[137] T.W. Malone and K. Crowston. The interdisciplinary study of coordination. ACM Comput. Surv., 26(1) 1994, pp. 87–119.

[138] UDDI Specifications Technical Committee Draft. http://uddi.org/pubs/ uddi_v3.htm, 2004.

[139] V. Ermolayev, N. Keberle, and S. Plaksin. Towards Agent-Based Rational Service Composition – RACING Approach. In M. Jeckle and L-J. Zang, editors, Proc. of the International Conference on Web Services Europe, volume LNCS 2853, Erfurt, Germany, September 2003. Springer-Verlag, pp. 167–182

[140] W3C. Simple object access protocol (soap). http://www.w3.org/TR/SOAP, 2000.

[141] Y. Charif and N. Sabouret. An Overview of Semantic Web Services Composition Approaches. In Proc. International Workshop on Context for Web Services (CWS'05), 2005.

[142] Z. Cheng, M. Singh, and M. Vouk. Composition constraints for semantic web services. In Proc of the WWW-2002 workshop 2002.

[143] Z. Guessoum and J-P. Briot. From Active Objects to Autonomous Agents. IEEE Concurrency, 7(3), 1999, pp. 68–76.

<u>عنوان الرسالة</u> : تركيب ذو دلالة لخدمات الويب في سياق ebXML

ظهور خدمات الويب باعتبارها التكنولوجيا الأساسية واستخدامها في عالم الصناعة أدت إلى ظهور مشاكل جديدة. في الواقع عندما لاتفي خدمات الويب المتوفرة باحتياجات المستخدمين فإن تركيبها حتى ولو كانت هذه العملية جد معقدة فيمكن أن تقدم حلولا ذات أهمية.

التخطيط الموزع يمكن استغلاله في التركيب الديناميكي لخدمات الويب. في هاته الرسالة نقترح بنية تسمح بتركيب خدمة الويب عند الضرورة وفقا لمواصفات العملية ebXMLهذه البنية تسمح بتعريف عدة مستويات من المسؤولية . فيما يخص البرمجة فقد استعملنا عدد من اللغات من اجل إنشاء البنية وطريقة التركيب المقترحة.

Titre : Composition sémantique des services web dans un contexte d'ebXML

Résumé : L'avènement des services Web comme une technologie incontournable et leur utilisation dans le monde industriel, posent de nouveaux problèmes. En effet, quand les services web existants ne répondent pas aux besoins des utilisateurs, leur composition, même si cette opération est très complexe, pourrait apporter des solutions intéressantes. La planification distribuée peut être exploitée pour composer dynamiquement ces services web. Dans ce travail de recherche, nous proposons une architecture multi-agent, permettant de composer des services web en cas de nécessité, conformément à la spécification fonctionnelle d'ebXML. Cette architecture définit plusieurs niveaux de responsabilité. La méthode de composition dynamique des services web que nous proposons s'appuie sur la synthèse dialectique de plans. Les agents managers (services web) proposent leurs compétences à l'agent compositeur afin de réaliser les objectifs fixés par l'utilisateur, en émettant, si nécessaire, des hypothèses sur les données manquantes.

Mots-clés: Système multi-agent, services web, composition sémantique, composition dynamique, ebXML, planification multi-agent

Title: Semantic web services Composition in an ebXML Context

Abstract: Web services advent as an inevitable technology pose new problems. Indeed, when existing web services do not meet the needs of users, their composition, even if this operation is very complex, could provide interesting solutions. Distributed planning can be exploited to dynamically compose these Web services. In this thesis, we propose a multi-agent architecture, for web services composing if necessary, in accordance with the functional specification of ebXML. This architecture defines several levels of responsibility. The proposed web services composition method is based on the dialectic plans synthesis. Manager agents (Web services) propose there competences to the composer agent in order to achieve the fixed user objectives by sending, if necessary, hypothesizes on incomplete data.

Keywords: Multi Agent System, Web Services, Semantic Composition, Dynamic composition, ebXML, Planning