N° d'ordre : 10/2007-M/IN

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie HOUARI BOUMEDIENNE

FACULTE D'ELECTRONIQUE ET D'INFORMATIQUE

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

EN: INFORMATIQUE

Spécialité : I.A.B.D.A.

Par : TEMGLIT Nacéra

Sujet :

Un modèle de composition des services web sémantiques

Soutenu le 15/07/2007, devant le jury composé de:

M^{r} A. KHELLADI	Professeur à l'U.S.T.H.B	Président de jury
M^{r} M. AHMED NACER	Professeur à l'U.S.T.H.B	Directeur de thèse
M^{me} Z. ALIMAZIGHI	Professeur à l'U.S.T.H.B	Examinateur
M^{r} N. BADDACHE	Professeur à l'U.S.T.H.B	Examinateur
Mme H. ALIANE	Chargée de recherche au C.E.R.I.S.T	Invitée

Remerciements

En premier lieu, je remercie Dieu, le tout puissant, de m'avoir donné le courage et la patience afin que je mène ce travail jusqu'à sa fin.

Je tiens à remercier Mr A.KHELLADI, directeur du CERIST, de m'avoir autorisé à préparer le magister.

Je remercie mon directeur de thèse Mr M. AHMED NACER de m'avoir fait confiance en me proposant ce sujet.

J'exprime mes vifs remerciements à mon encadreur Mme ALIANE pour son suivi attentif, sa compréhension et pour tous les moyens qu'elle a mis à ma disposition pour la finalité de ce travail.

Je remercie Mr A. KHELLADI pour avoir accepté de présider le jury, j'adresse mes remerciements à Mr N. BADDACHE et Mme Z. ALIMAZIGHI pour avoir accepté de faire part du jury et d'honorer par leur jugement notre travail.

Un grand merci aux membres de ma famille, à mes chers amis et à tous ceux qui m'ont encouragé et soutenu afin que ce travail arrive à sa fin.

Résumé

Les services web représentent un champ de recherche en plein développement. Ce sont des applications auto-descriptives et autonomes s'exécutant sur le web **[Duv04]**. La composition des services web constitue une évolution naturelle de la technologie des services web. Elle se réfère au processus de création d'un *service composite* offrant une nouvelle fonctionnalité, à partir de services web existants plus simples, par le processus de découverte dynamique, d'intégration et d'exécution de ces services dans un ordre bien défini afin de satisfaire un besoin bien déterminé **[Chak02]**.
En effet, la composition permet de mettre en place des composants services Web au profit de l'intégration d'applications sur le web afin d'atteindre de meilleures solutions.

Plusieurs travaux ont émergé, ces dernières années visant à construire des processus complexes par composition de services élémentaires et qui se veut, de plus en plus, une composition dynamique et adaptative aux préférences posées par les utilisateurs. Cependant, les travaux étudiés ne prennent pas suffisamment en compte le problème d'interaction avec l'utilisateur voulant exécuter un service composite. Les méthodes orientées Workflow supposent que la requête doit définir le modèle de processus décrivant la fonctionnalité complexe voulue. D'un autre coté, les approches sémantiques orientées planification, exigent une description formelle de l'état initial et du but à atteindre. Toutefois, cela semble irréaliste quand il s'agit d'un utilisateur du Web. Ce dernier n'est pas censé avoir des connaissances de bas niveau. Pour cela, nous avons essayé, dans ce travail, de proposer un modèle de composition qui permettra d'accéder graduellement aux fonctionnalités complexes d'un domaine en partant d'une requête simple. Ce modèle se base sur une représentation sémantique de l'ensemble des concepts manipulés par les services web du domaine d'application, à savoir, les opérations et les concepts statiques utilisés pour décrire les propriétés des services web. Différents niveaux d'abstraction sont donnés au concept opération pour permettre un accès progressif aux services concrets. Ainsi, deux plans de composition à granularités différentes (abstrait et concrets) sont générés. Ceci permettra de réutiliser des plans déjà construits pour répondre à des besoins similaires et même avec des préférences modifiées.

Mots clés: composition de services web, service web composite, requête utilisateur, plan de composition, matching sémantique, template de composition.

Table des matières

Chapitre 3: La composition des Services Web

Chapitre 4: Un modèle de composition des services web sémantiques

Introduction générale

La vision par laquelle on percevait le web a évolué. Ainsi, nous sommes passés d'une vision où il était constitué d'un ensemble de pages accessibles par mots clés à une vision du web où celui-ci devient un fournisseur de ressources accessibles par leurs contenus **[DAML-Sa].**

La majorité des ressources du web est constituée par des programmes (applications) et des contenus de documents accessibles en ligne. La distribution et l'hétérogénéité des applications disponibles sur le web constitue un grand défi concernant l'intégration d'application sur le web. Ainsi, la notion de « service web » est née. Un «service web » peut être considéré comme une ressource qui fournit non seulement de l'information mais qui autorise éventuellement d'effectuer une action ou un changement dans le monde.

Les « Services Web » représentent un défi informatique mais aussi économique. En effet, ils peuvent constituer un apport de rapidité et d'efficacité pour l'e-Business **[DAML-Sa]**. La notion de service web désigne essentiellement une application (un programme) mise à disposition sur Internet par un fournisseur de service, et accessible par des clients à travers des protocoles Internet standards**[Kel03]**. Leurs particularités par rapport aux autres technologies de l'informatique répartie résident dans le fait qu'ils offrent un modèle de composants (services web) à couplage faible en utilisant la technologie Internet comme infrastructure pour la communication, et ceci en mettant en place un cadre de travail basé sur un ensemble de standards.

La standardisation de cette technologie touche actuellement trois couches du modèle de fonctionnement global: Un protocole de communication (SOAP) **[SOAP]** permettant de structurer les messages échangés entre les composants "Service Web", une spécification de description des interfaces des services (WSDL) **[WSDL]** et enfin une spécification de publication et de localisation de services (UDDI) **[UDDI]**. Ce modèle présente les services web sous forme d'une collection d'unités de traitement (opérations) dont l'invocation n'excède pas un échange simple de messages (généralement requêtes /réponse). Cette infrastructure est suffisante pour mettre en place des composants interopérables et intégrables. Toutefois, certaines applications nécessitent l'exécution d'une interaction plus complexe en invoquant des opérations de services web de manière dépendante en obéissant à une logique métier spécifique en vue de parvenir à la fonctionnalité complexe souhaitée. Ceci résulte d'une nécessité conceptuelle de la mise en place d'une agrégation (composition) de plusieurs services simples en services plus complexes **[Mel03]**. Face à ce besoin, l'infrastructure de base des services web s'avère insuffisante et inadéquate pour rendre automatique et efficace la tâche de composition et aussi la tâche de localisation des services requis.

L'automatisation est un concept clé dans la technologie des services web et doit être présent à chaque étape du processus de conception et de mise en œuvre des services web à savoir leur découverte et leur composition. Elle est essentielle pour intégrer les facteurs suivants:

- Passage à l'échelle: il faut être capable de traiter un nombre important de services web
- Forte réactivité dans un environnement hautement dynamique.
- Réduction des coûts de développement et de maintenance des services web.
- Prise en compte des critères de qualité de services aussi bien du point de vue quantitatif que qualitatif.

En effet, le consensus sur la mécanique des interactions (*e.g.,*format des messages, types des données et protocoles d'échanges) n'est pas suffisant pour permettre aux services web d'interagir de manière claire et non ambiguë. Une simple description syntaxique via des interfaces WSDL limite les perspectives d'automatisation. Pour palier cette limitation, il y a clairement un besoin de langages de description qui permettent de conférer une signification explicite et non ambiguë aux descriptions des Services Web. Ceci rejoint les préoccupations à l'origine du Web sémantique, à savoir comment décrire formellement les connaissances de manière à les rendre exploitables par des agents logiciels. En conséquence, les technologies et les outils développés dans le cadre du Web sémantique peuvent certainement compléter la technologie des services web. Par exemple, la notion d'ontologie peut jouer un rôle prépondérant pour permettre d'expliciter la sémantique des services.

Plusieurs langages et formalismes issus des communautés industrielles et académiques ont vu le jour faisant l'objet d'une infrastructure étendue pour les services web. Cette extension touche deux aspects essentiels:

- La couche business-process (BPEL4WS **[BPEL4WS]**, WSFL**[WSFL]**, XLANG **[XLANG]**): la modélisation du processus métier constitue une évolution naturelle des services web.
- La couche sémantique (DAML-S **[DAML-Sa]**, OWL-S **[OWL-S]**).

Nous nous intéressons dans ce travail à la composition des services web. Cette dernière vise à créer un processus métier faisant appel à une combinaison de services web disponible afin de répondre à un besoin qui n'a pas pu être satisfait en invoquant un service unique. Le but de notre travail est de définir une approche progressive de composition de service web en partant d'une requête déclarative simple spécifiée par un utilisateur non-spécialiste. Ce dernier aura simplement à spécifier son besoin en terme de tâches à effectuer. La découverte des services participants implémentant les tâches spécifiées dans la requête, leur composition ainsi que leurs interactions doivent être effectuées de façon automatique et transparente par rapport à l'utilisateur.

Le nombre de travaux étudiant la problématique de la composition des services web est très important. Pratiquement, nous pouvons les classifier selon deux grandes catégories: les approches workflow **[Cas00]**et les approches utilisant les techniques de planification de l'IA **[McI02] [Nar02] [Gha98] [Pon02] [Med03] [Wu03] [Wal01] [Rao03] [Rao04]**. Les méthodes workflows supposent la définition préalable du modèle de processus. Uniquement la recherche des services concrets et leur liaison se fera de manière dynamique. Dans ce cas la requête de l'utilisateur doit définir un workflow abstrait de la tâche à effectuer. D'un autre coté, les méthodes orientées web sémantique

utilisent des techniques de planification afin d'automatiser entièrement le processus de composition. La requête de l'utilisateur doit définir formellement l'état initial et le but à atteindre et éventuellement un certain nombre de contraintes. Ensuite, un plan ou un processus est généré automatiquement par un prouveur de théorème ou à l'aide d'un planificateur IA sans faire appel à un workfolw prédéfini.
Cependant, toutes ces méthodes exigent de l'utilisateur d'avoir de bonnes connaissances concernant la spécification d'un workflow ou les langages formels (tel que la logique du premier ordre ou XML) pour spécifier son besoin. Or, sur le web, nous nous adressons très souvent à un utilisateur non-spécialiste, c'est à dire, un utilisateur qui n'est pas très qualifié pour exprimer de manière formelle et claire son besoin. De plus, très souvent il ne possède pas une idée complète et assez précise concernant le service qu'il cherche: s'il est simple ou il fait appel à une composition de service. Cela nécessite un modèle d'interaction à travers lequel l'utilisateur pourra accéder aux fonctionnalités complexes d'un domaine d'une manière aisée et progressive.

Nous essayons, à travers ce document, de répondre à cette problématique en proposant un modèle de composition des services web basé sur une représentation sémantique de l'espace des services web d'un domaine donné. Cette représentation distingue différents niveaux d'abstraction des tâches manipulées dans le domaine d'application (générique, abstraite et concrète) afin de faciliter l'interaction avec un utilisateur externe et accéder graduellement aux fonctionnalités complexes offertes. La résolution de la requête est faite en trois phases différentes pour aboutir à la dernière phase à une description détaillée du service composite construit en utilisant un langage de spécification de flux.

La première phase du processus de résolution consiste à générer un plan initial de composition en essayant de mettre en correspondance la requête de l'utilisateur avec un processus générique appelé "template générique". Ces templates donnent une vue générique d'un processus métier très souvent sollicité dans le domaine d'application. Nous proposons en conséquence, un mécanisme de matching (mise en correspondance) sémantique flexible dans le sens qu'il permettra d'accepter différents niveaux de correspondance qui peuvent être pertinents vis-à-vis du besoin de l'utilisateur. Ce schéma est utilisé dans toutes les étapes du processus de résolution de la requête. En effet, il peut être utilisé dans deux contextes différents: pour enchaîner deux opérations ayant des fonctionnalités complémentaires ou pour faire substituer deux opérations ayant des fonctionnalités similaires.

Cette thèse est organisée en deux parties présentant respectivement l'état de l'art et notre proposition. L'état de l'art regroupe trois chapitres différents. Dans, le premier chapitre, nous présentons le web sémantique et les différentes technologies permettant la réalisation de la vision du web sémantique.

Le deuxième chapitre étudie le concept de service web. Nous revenons sur la genèse des services web et les idées fondatrices que l'on trouve dans les architectures antérieures telles que CORBA, DCOM et J2EE. Nous présenterons ensuite, les standards permettant de mettre en œuvre le modèle des services web et les services web sémantique.

Dans le troisième chapitre, nous expliquerons la problématique de la composition des services web et nous présenterons quelques travaux étudiés dans ce cadre. Nous ferons, à la fin de ce chapitre, une synthèse sous forme d'un tableau comparatif qui permettra de mettre en relief la relation entre les études menées dans la littérature et l'objectif de notre travail.

Le quatrième chapitre est consacré à notre proposition qui est de définir un modèle de composition des services web sémantiques organisés en quatre parties: le support sémantique, le schéma de mise en correspondance, les différents niveaux de traitement de la requête et le processus de résolution de requête. A la fin de ce chapitre, nous reprenons le tableau comparatif défini dans le chapitre 3 pour faire une évaluation approximative de notre approche par rapport aux approches étudiées.

Chapitre 1 : Le Web sémantique

1. Introduction

En une dizaine d'années, le World Wide Web est devenu un moyen de partage d'information tellement populaire que la quantité de données disponible dépasse de loin ce qu'un utilisateur peut gérer seul. Par conséquent, le Web souffre aujourd'hui de son succès, ses utilisateurs s'en plaignent tous : on ne trouve pas ce que l'on cherche sur Internet... En effet, malgré la sophistication des moteurs de recherche actuels, les résultats manquent de pertinence, voire sont complètement hors sujet. Ces problèmes sont dûs, bien sûr à des ambiguïtés linguistiques, mais surtout à des incohérences sémantiques profondes **[Cham02].**

D'une autre part, les nouvelles perspectives d'application comme le commerce électronique, l'enseignement et la formation à distance, les services web et la vidéo Internet posent de nouveaux défis pour la recherche d'informations sur le Web.
Tim Berners-Lee **[Ber01]** suggère que la prochaine étape du web, qu'il appelle « Web sémantique », devra apporter des réponses à tous ces problèmes.

2. La vision du web sémantique

Dans le Web actuel, les pages Web représentent une énorme masse de connaissances qui est difficilement exploitable et sans une assistance automatique. Cette masse augmente sans cesse ainsi que le nombre d'utilisateurs qui veulent pouvoir trouver facilement les informations qu'ils y recherchent. L'éventail des thèmes traités dans les différentes pages Web est tel qu'une recherche par mot-clés retourne quasi systématiquement des pages qui ne portent pas toutes sur le même domaine de connaissance.

L'exploitation efficace des ressources du Web suppose donc que les moteurs de recherche puissent accéder à la thématique de chaque page, et non uniquement à des mots-clés qui ne permettent pas toujours de bien discriminer les sujets traités. De plus, la variété des sources d'information sur le Web (textes, images, etc) plaide pour un traitement de l'information qui soit indépendant des formes sous lesquelles elle est stockée, c'est-à-dire pour un traitement au niveau conceptuel. Une partie des manipulations d'informations actuellement assurées par les utilisateurs pourra ainsi être prise en charge par les machines.

L'idée du web sémantique est d'ajouter progressivement des descriptions sémantiques de contenu sur le Web actuel afin de faire émerger petit à petit un Internet plus intelligent.

C'est dans cette optique que Tim Berners-Lee a proposé de bâtir le Web Sémantique, couche de connaissances venant apporter au Web une dimension conceptuelle complétant la simple dimension textuelle : *« The vision of the Semantic Web is to provide computer interpretable markup of the Web's content and capability, thus enabling automation of many tasks currently performed by human beings* **[Ber01].**

Impulsé donc par Tim Berners-Lee dès 1998, développé dans le cadre du W3C (Consortium World Wide Web) depuis 2001, le web sémantique se veut une extension du web actuel, visant à rendre les contenus, non plus uniquement accessibles et affichables, mais aussi exploitables et interprétables par des machines.

Voici la définition donnée par Tim Berners-Lee , James Hendler et Ora Lassila **[Ber01]** : *"The semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation"*

Le Web sémantique fait alors référence à la vision du Web de demain comme un vaste espace d'échange de ressources entre êtres humains et machines permettant une exploitation, qualitativement supérieure, de grands volumes d'informations et de services variés. Espace virtuel, il devrait voir, à la différence de celui que nous connaissons aujourd'hui, les utilisateurs déchargés d'une bonne partie de leurs tâches de recherche, de construction et de combinaison des résultats, grâce aux capacités accrues des machines à accéder aux *contenus* des ressources et à effectuer des *raisonnements* sur ceux-ci **[Lau02]**.

Le travail du web sémantique consiste, d'une part, à proposer des *modèles* pour *formaliser* la sémantique c'est à dire, représenter le sens afin qu'il puisse être manipulable par la machine, et, d'une autre part, à doter les machines par des programmes capables d'exploiter cette sémantique lors des traitements des données du web (faire des inférences) **[Ouz03].**

Les recherches autour du web sémantique s'appuient sur un existant riche venant, par exemple, des recherches en représentation ou en ingéniérie des connaissances. Mais leur utilisation et leur acceptation à l'échelle du Web posent de nouveaux problèmes et défis: *changement d'échelle* dû au contexte de déploiement, nécessité d'un niveau élevé d'interopérabilité, ouverture, standardisation, diversités des usages, distribution bien sûr et aussi impossibilité d'assurer une cohérence globale. Comme l'écrit, en substance, Tim Berners-Lee*, le Web sémantique est ce que nous obtiendrons si nous réalisons le même processus de globalisation sur la représentation des connaissances que celui que le Web fit initialement sur l'hypertexte* **[Lau02].**

3. Les composantes essentielles du Web sémantique

Concrètement, le Web sémantique est une *infrastructure* pour permettre l'utilisation de connaissances *formalisées* en plus du contenu informel actuel du Web, même si aucun consensus n'existe sur jusqu'où cette formalisation doit aller. Cette infrastructure doit permettre d'abord de localiser, d'identifier et de transformer des ressources de manière

robuste et saine tout en renforçant l'esprit d'ouverture du Web avec sa diversité d'utilisateurs **[Lau02]**.
L'infrastructure du web sémantique s'articule autour de trois composantes essentielles: Méta-données ou annotations sémantiques, Ontologies, Systèmes de raisonnement. Ajoutant à cela, l'infrastructure du web sémantique fait appel à une autre notion essentielle qui est : URI (Uniform Ressouce Identifier).

3.1 Ressource, URI et URL:

- Tout objet (chose) ayant une identité peut être considéré comme une **ressource**. Parmi les ressources les plus largement connues, on trouve les documents électroniques, les images, les différentes applications sur le Web tel que les services web. Mais les êtres humains, les livres d'une bibliothèque ou encore les entreprises peuvent aussi être considérés comme des ressources.

- L'**URI** (Uniform Ressource Identifier) en français c'est identificateur uniforme de ressources défini par l'IETF. C'est une chaîne de caractère formatée qui identifie avec certitude et de façon unique une ressource par un nom, une localisation ou une autre caractéristique.
  Un URI doit permettre d'identifier une ressource de manière permanente (même si cette ressource est déplacée, supprimée ou cesse d'exister) **[URI]**.

- Les **URLs** (Uniform Resource Locator) sont un sous-ensemble des URIs qui permet d'identifier une ressource (propre d'un URI) tout en spécifiant son emplacement sur Internet et la méthode permettant d'y accéder.

3.2 Méta-données et annotations

Le Web sémantique repose sur une infrastructure dont l'objectif est de se donner la possibilité d'enrichir le Web actuel à l'aide d'informations dites « sémantiques », utilisables par des machines, qui faciliteront la recherche et l'usage de ressources Web (pages Web, images, services, etc.). Il s'agit donc d'associer à ces dernières des informations structurées et descriptives sous la forme de méta-données (ou annotations). Les méta-données (ou annotations) constituent alors un des principes de base du web sémantique. Associer par exemple une notice bibliographique comprenant des champs : *Auteur*, *Date de création*, *Date de modification*, *Mots-clés*, à une page Web permet de considérer celle ci non plus seulement comme comprenant du texte qui ne pourra qu'être traité statistiquement par un robot indexeur, mais également des informations structurées à la sémantique connue et utilisable comme telle par un agent logiciel.

Associer une information exploitable à une ressource signifie que cette information doit d'une manière ou d'une autre être structurée et descriptive aussi bien de la ressource que de son utilisation afin de faciliter et d'en améliorer l'accès dans le cas d'une ressource directement visualisée par un utilisateur (par exemple en permettant une recherche d'information plus efficace et plus ciblée), mais aussi l'exploitation dans le cas d'une ressource exploitée dans le cadre d'un service à l'utilisateur (l'utilisateur n'est alors pas forcément conscient de l'utilisation de la ressource).

Remarquons que deux termes principaux sont utilisés dans la littérature afin de décrire les informations associées à des ressources : *méta-données* et *annotations*
Si nous revenons sur ces notions, et de façon générale:

- Une **méta-donnée** est «une donnée sur une donnée» **[Ber97]** elle fournit une information descriptive sur une donnée (une ressource) et doit permettre le traitement automatique de la données sur laquelle elle porte.
- Une **annotation** est à la base une note critique ou explicative accompagnant un texte, et par extension, une quelconque marque de lecture portée sur un document, que celui-ci soit textuel ou image **[Yan03].**

On le voit, les termes de méta-donnée ou d'annotation prennent bien en compte cette notion d'ajout d'information à une ressource, et on pourra *a priori* les utiliser indifféremment pour décrire ces informations que le Web sémantique doit ajouter au Web pour le rendre plus utilisable par des machines.
Pour autant, si ces deux termes existent, c'est qu'ils n'ont pas le même sens. La communauté anglophone du Web sémantique, circonspecte sur la différenciation, considère que les annotations de pages Web *deviennent* des méta-données dès qu'elles sont stockées dans une base sur un serveur. On différencie alors l'information en tant qu'elle est attachée à (et présentée avec) une ressource et l'information manipulable et exploitable de façon plus indépendante de celle-ci **[Yan03]**.

Remarquons que pour le Web sémantique, on parlera le plus souvent de méta-données ou d'annotations *sémantiques* : « sémantique » au sens de « sémantique formelle ».
Par exemple, l'un des schémas de méta-données le plus connu et le plus simple dans le Web d'aujourd'hui est le *Dublin Core Schema* (DC). La version simple du schéma consiste en en un ensemble de 15 éléments indépendants, incluant par exemple, le titre, l'identificateur, le langage et la description.

Les méta-données s'exprime habituellement avec des formalismes logiques inspirés de la logique de description (DL) **[DL]** tel que RDF (Ressource description Framework) **[RDF]**. RDF est le standard le plus utilisé depuis 1999 pour la description des ressources du web sémantique en utilisant des méta-données, et ce, quelque soit la structure de cette ressource (Document HTML, XML, base de données…)**[Ouz03].** Citons aussi les cartes topiques ou Topic Map, un standard ISO, dont le but initial était d'annoter les documents Hypermédia (nous revenons sur ça un peu plus loin).

Dans le contexte du Web Sémantique, les méta-données constituent un module fondamental et permettent notamment de faciliter la recherche d'information : décrire le contenu et les relations entre les ressources (fichiers d'un site) et mieux référencer un site ou une page sur le web. Elles garantissent l'interopérabilité en assurant le partage et l'échange d'information rendant son contenu lisible et compréhensible par les machines.

3.3 Ontologie:

Il a été bien reconnu dans la communauté du web sémantique que les ontologies jouent un rôle clé en facilitant le partage d'information entre les communautés des humains et des agents logiciels.

Les ontologies sont apparues au début des années 90 dans la communauté Ingénierie des connaissances dans le cadre des démarches d'acquisition des connaissances pour les systèmes à base de connaissance (SBC). Faisant suite aux systèmes experts qui séparaient une base de connaissances «déclarative» et un moteur d'inférence «procédural», les SBC proposaient alors de spécifier, d'un côté, des connaissances du domaine modélisé et, de l'autre, des connaissances de raisonnement décrivant les règles heuristiques d'utilisation de ces connaissances du domaine. L'idée de cette séparation modulaire était de construire mieux et plus rapidement des SBC en réutilisant le plus possible des composants génériques, que ce soit au niveau du raisonnement ou des connaissances du domaine. Les connaissances du domaine précisent tout ce qui a trait au domaine. Dans ce contexte, les chercheurs ont proposé de fonder ces connaissances sur la spécification d'une *ontologie*, ensemble structuré par différentes relations, principalement l'hyperonymie des objets du domaine.
Le terme « Ontologie» est souvent utilisé aujourd'hui et possède de nombreuses définitions.

Définition (1) : *Ensemble des objets reconnus comme existant dans le domaine. Construire une ontologie c'est aussi décider de la manière d'être et d'exister des objets*.

Dans cette définition, les objets ne sont pas pris dans un sens informatique mais comme objets du monde réel que le système modélise. Pour poursuivre vers une définition de l'ontologie, il nous semble indispensable de rappeler que les travaux sur les ontologies sont développés dans un *contexte informatique* ou plus spécifiquement ici le contexte du *Web sémantique* où le but final est de spécifier un artefact informatique. Dans ce contexte, l'ontologie devient alors un *modèle des objets existants* qui y fait référence à travers des concepts, les concepts du domaine **[Char03].**

La définition la plus citée est celle de Thomas Gruber qui définit une ontologie de la façon suivante **[Gru93]**:

Définition (2): *"An ontology is an explicit specification of a conceptualisation"*. Cela se traduit par *« Une ontologie est une spécification explicite d'une conceptualisation »*

En reprenant les spécifications de Gruber **[Gru93]** et M. Uschold *et al* **[Ush96]** qui ont appliqué les résultats des réflexions dans le contexte applicatif d'une ontologie d'entreprise, *the enterprise ontology*, on peut alors accepter une autre définition de ce qu'est une ontologie.

Définition (3) *: Une ontologie implique ou comprend une certaine vue du monde par rapport à un domaine donné. Cette vue est souvent conçue comme un ensemble de concepts – e.g. entités, attributs, processus –, leurs définitions et leurs interrelations. On appelle cela une conceptualisation* **[Char03].**

Pour le web sémantique, une ontologie permet donc une représentation formelle (exploitable par la machine) et explicite d'une conceptualisation partagée par les acteurs du web sémantique. Elle sert à centraliser le sens de ce qui peut être manipulé par la machine **[Ouz03].** Elle constitue dans le cadre du web sémantique, une connexion entre

des agents logiciels (sources d'information, serveurs d'information, hétérogènes – *sémantique formelle*) et entre des utilisateurs humains (*sémantique interprétative)* et des agents logiciels. La langue et les différences culturelles ne devraient ainsi plus être un frein aux interactions automatiques entre machines et entre machines et humains.

Ontologie, thésaurus et taxonomie :

Utiliser un vocabulaire particulier pour décrire un domaine permet d'organiser, structurer et hiérarchiser la connaissance de ce domaine et aussi de partager l'information entre les membres d'un groupe utilisant ce domaine.

Vocabulaire contrôlé:

> C'est un ensemble de termes définis par un groupe (une communauté de pratiques) afin de pouvoir labelliser des contenus, écrire un document. La signification des termes n'est pas forcément définie et il n'y a pas nécessairement d'organisations logiques des termes entre eux. Par exemple, le glossaire d'un livre ou encore des catégories dans un système de carnets Web partagés entre différents auteurs.

Taxonomie:

> Dans une taxonomie, le vocabulaire contrôlé est organisé sous forme hiérarchique simple. Cette hiérarchisation correspond souvent à une spécialisation. Il existe donc un lien précis entre un terme du vocabulaire et ses enfants. Ce lien donne un sens supplémentaire, une signification. D'un vocabulaire contrôlé, on passe à un vocabulaire organisé. Par exemple, dans une classification animale, nous aurons les vertébrés, invertébrés et puis sous les vertébrés nous auront les mammifères, les ovipares, etc. Tous ces termes nous permettront de classifier les animaux. On pourra donc dire que Les mammifères sont une **sous-catégorie** (sous-classe) des vertébrés.

Thésaurus:

> Un thésaurus est une taxonomie qui fonctionne dans les deux sens. La taxonomie permettait d'obtenir une spécialisation des termes employés. Le thésaurus donnera de l'information sur les sujets connexes également. Le champ de connaissance fourni donc par un thesaurus est plus large que celui fourni par une taxonomie. Cet élargissement se fait grâce à d'autres liens autres que le lien de spécialisation. Par exemple les lien **« relatif à »** ou **« voir également »…** (élargissement). Par exemple, imaginons une taxonomie qui organise l'information à propos des différentes races de vaches et chacune des sous-branches de ces races. Une personne utilisant ce thésaurus voudra peut-être voir pendant sa recherche, explorer les types de fromages faits de lait de vache. Fromage ne fait pas partie de la taxonomie des vaches mais dans un thésaurus, celui-ci peut avoir un intérêt car le fromage est un des produits dérivés fait à partir du lait de vache. On a donc élargi le champ d'étude.

Ontologie:

> En fait un thésaurus ou même une taxonomie sont des formes d'ontologie dont la grammaire n'a pas été formalisée. Lorsque l'on établit une catégorie et une hiérarchisation de cette catégorisation, on établit des dépendances entre ces termes. Cette hiérarchisation ont un sens en dehors du vocabulaire lui-même. Par exemple, quand je dis ce terme est une sous-catégorie de cet autre terme Je viens de donner un sens à cette relation, je viens de dessiner une flèche entre les deux et j'ai qualifié la flèche en affirmant quel type de relation cela signifiait.
>
> Une ontologie correspond donc à un vocabulaire contrôlé et organisé et à la *formalisation explicite* des relations créées entre les différents termes du vocabulaire **[Gon04].**
>
> D'un autre coté, les ontologies prennent des formes plus complexes permettant la représentation de liens sémantiques plus spécifiques par exemple, "fait partie de," "est localisé dans," "est possédé par" ou "est associé à "...etc. Mais surtout les ontologies les plus abouties permettent également l'intégration de propriétés particulières, de règles d'utilisation et de contraintes.
> Comparant à la recherche basée sur des thésaurus, dans le cas des ontologies, les connaissances peuvent être plus complexes et rendre compte d'un domaine plus technique, un métier, ou une organisation.

Les ontologies s'expriment habituellement avec un langage logique, de façon à ce que l'on puisse faire des distinctions détaillées, précises, cohérentes, logiques et significatives entre les classes, les propriétés et les relations. Certains outils ontologiques peuvent effectuer des raisonnements automatisés au moyen des ontologies et ainsi apporter des services évolués aux applications intelligentes.

Dans le cadre du web sémantique, les moteurs d'inférence doivent permettre de rechercher des ressources (représentées par des URI) qui ont été préalablement annotées (décrites) au moyen d'énoncés reposant sur une ontologie (vocabulaire conceptuel). La recherche se fait en exploitant les connaissances représentées dans l'ontologie et les annotations. L'intérêt de reposer sur des connaissances modélisées est d'éliminer les ambiguïtés et d'exploiter ces connaissances pour mieux répondre aux requêtes de recherche d'information.

3.4 Le moteur de raisonnement:

Le raisonnement sur la connaissance est un point important du Web Sémantique. Les méta-données et les ontologies permettent de formaliser le sens des données qui doivent être interprétées et exploitées par un moteur d'inférence pour produire de nouvelles connaissance plus précises, plus adéquates et moins redondantes. Ainsi, même des données non explicites mais déductibles de données existantes seront générées automatiquement, sans intervention humaine.

Par exemple, si un patient présente un symptôme de catégorie A et un autre de catégorie B, alors il est considéré comme porteur de la maladie M (identifiée par *uri01*). L'acteur1 de l'hôpital 1 affirme que Dupont (identifié par *uri12*) présente le symptôme x et l'acteur 2 affirme que Dupont (identifié par *uri12*) présente le symptôme y et nous avons dans la base de connaissance les assertions : le symptôme x fait partie de la catégorie A et le symptôme y fait partie de la catégorie B. Ces connaissances peuvent être distribuées et représentées différemment par leurs acteurs, le rôle du web sémantique est de pouvoir inférer que Dupont (identifié par uri12) est porteur de la maladie M (identifiée par uri01). Pour cela, de nombreux moteurs d'inférences (Racer, Pellet, FaCT, FaCT++, Flora, Jena, Jess,...) ont été crées pour raisonner sur des logiques de description. La plupart acceptent en entrée des fichiers OWL. Par exemple, **RACER [RACER]** le moteur d'inférence le plus connu, fondé en 2004 par des chercheurs qui travaillaient à l'université de Hambourg, travaille sur les ontologies modélisées par son langage, mais accepte des ontologies décrites en RDF ou OWL, ces dernières étant traduites vers le langage utilisé par Racer. Ce moteur d'inférence possède également son propre langage de requête nRQL (new Racerpro query Language) pour interroger les ontologies.

4. Architecture et langages du Web sémantique

4.1. Les langages du Web sémantique :

Le Web sémantique doit pouvoir être manipulé par les machines. Il est alors nécessaire de disposer de langages pour exprimer les données et les méta-données, exprimer les ontologies et décrire les services.
Certes, il existe déjà des langages développés pour ces activités indépendamment du Web sémantique. Ils ne sont utilisés tels quel dans le Web sémantique car il est nécessaire de leur permettre d'accepter les caractères propres au Web à savoir sa distribution et son ouverture.

XML est le langage de base, il a l'avantage d'être fait pour la communication en réseau et de disposer de nombreux outils. Mais il a surtout la propriété d'être un méta-langage permettant de définir d'autres langages (une description de type de document, DTD, permet de décrire la grammaire des documents admissibles). Il est donc naturellement utilisé pour encoder les langages du web sémantique ainsi que les données. Cela a permis de créer une compatibilité entre les différents langages décrits en XML et de les considérer tous comme des documents XML.
Mais XML est limité car il ne dispose pas d'une sémantique (au sens logique de sémantique dénotationnelle d'un langage). Rien ne justifie donc les raisonnements ou manipulations appliquées à des documents XML. C'est pourquoi il est nécessaire de faire appel à d'autres sources d'inspirations.
La seconde source d'inspiration est celle de représentation des connaissances et notamment les langages de représentation de connaissances que sont les logiques de descriptions (DL) **[DL]** et les réseaux sémantiques ou bien encore les graphes conceptuels. Ces langages ont certaines limitations dans la prise en compte de la

nécessaire ouverture du web (relations entre objets distribués, ajout de connaissance incontrôlé). Ils sont donc reconsidérés afin de leur permettre de passer à l'échelle du Web. Enfin, dans le cadre de la description de services, les dernières sources d'inspiration sont les langages de description de plans et en particulier les langages de description de « Workflow ».

Différents consortiums et organismes mettent les acteurs autours d'une table pour définir les langages à utiliser dans le Web sémantique selon une approche *standardisante*. L'intérêt d'une telle approche est d'assurer des traitements uniformes sur l'ensemble des documents écrits dans ces langages.
Principalement, le Web sémantique s'appuie sur les langages proposés par le W3C qui a réussi à faire interagir un grand nombre d'acteurs tant académique qu'industriels. Nous décrirons ici trois sortes de langages :

- Des langages d'assertions et d'annotations sémantiques (RDF, RDFS et Carte topiques).
- Un langage de définition d'ontologies (OWL)
- Différents langages de description et de composition de services (UDDI et autres).

4.1.1 Des langages d'assertions et d'annotations sémantiques

a) RDF et RDFS :

RDF (Ressource description Framework) **[Las99] [Kly03]** est un langage formel qui permet d'affirmer des relations entre des ressources. C'est un standard émergeant depuis 1999 pour l'annotation des ressources du Web sémantique en utilisant des méta-données, et ce, quelque soit la structure de ses ressources (documents HTML, XML, texte, base de données….) . RDF est muni d'une syntaxe et d'une sémantique. Aucun mécanisme d'inférence n'est cependant proposé dans la recommandation. Une base de connaissances RDF est un ensemble de déclarations sous forme de triplets <*sujet, prédicat, objet*> où *sujet* est la ressource décrite, *prédicat* est la propriété attribuée à la ressource et *objet* est la valeur de cette propriété. On peut interpréter aussi un énoncé RDF comme <*ressource-attribut-valeur*> (la ressource est une valeur ou chaîne de caractère) ou chaque ressource est identifié par une URI (Uniform Ressource Description) **[Lau02]**.
Le W3C a défini une sérialisation d'une base RDF en utilisant la syntaxe XML. Soit l'exemple suivant :

```
<rdf : RDF>
    <rdf :Desription about= "http://www.hopitalX.fr/patient/Dupont">
         <nsh:nom> Béatrice Dupont</nsh:nom>
<nsh:soignéPar ressource=
"http://www.hopitalX.fr/médecins/Durand">
</rdf :Desription>
</rdf : RDF>
```

Cette déclaration RDF signifie que la ressource (patient) représentée par l'URI " *http://www.hopitalX.fr/patient/Dupont"* a la valeur Béatrice Dupont pour propriété nom (définie dans le namespace de l'hopital, *nsh*) et a comme valeur la ressource *"http://www.hopitalX.fr/médecins/Durand">* pour la propriété "*soignéPar*" (voir la figure 1.1).

Comme tous les modèles de représentation des connaissances, RDF est aussi un modèle de graphes. Les ressources et les objets peuvent être représentés par des nœuds. Ces nœuds sont connectés par des liens étiquetés par le nom de leurs propriétés. La représentation en graphe de l'exemple précédent est donnée comme suit **[Bag03]**:

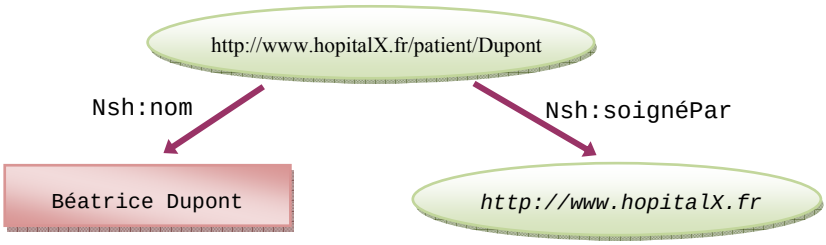


Fig 1.1 : Représentation en graphe d'un document RDF

Les schémas d'un graphe RDF sont spécifiés par le langage de spécification de schémas RDFS (RDF Schema) **[RDFS]**. RDF Schéma permet de spécifier le vocabulaire et la hiérarchie utilisés dans RDF ainsi que les propriétés de ces ressources de ces classes. Les bases de données traditionnelles définissent un schéma en terme de classe. Une classe est définie par ses attributs (ou propriétés). Par contre, RDF Schéma définit un schéma en terme de propriétés qui associent des classes de propriétés. Considérons l'exemple suivant:

```
Schéma Objet                          RDF Schema
     define class livre                    rdfs : property auteur
      { auteur : personne }                auteur rdfs : domain livre
                                           auteur rdfs : range person
```

Le schéma objet définit la classe (ressource en RDF) *livre* avec un seul attribut auteur de type classe *personne*. Ce schéma est défini en RDF Schéma par la propriété *auteur* qui associe les deux classes livre (comme domaine de définition) et personne (comme domaines des valeurs).

L'avantage d'une telle conception de schémas est que RDF Schema donne une souplesse quant à la définition et à l'évolution de schémas. En effet, pour ajouter une propriété dans un schéma objet, on doit redéfinir la structure de sa classe. Par contre, dans un schéma RDF il suffit d'ajouter les déclarations :

```
rdfs : property nom_de_la_propriété
nom_de_la_propriété rdfs : domain classe_du_domaine
nom_de_la_propriété rdfs : range classe_des_valeurs
```

Certains auteurs ont présenté une méthode de représentation d'un graphe RDF en utilisant les graphes conceptuels et la logique du premier ordre **[Bol00]** et **[Cor00]**. Cette traduction permet d'exploiter les mécanismes de raisonnement développés pour les graphes conceptuels et la logique du premier ordre.
On peut noter que RDF Schema n'intègre pas en tant que tel de capacités de raisonnement **[Lau02]**. Par contre, il est utilisé par le W3C comme syntaxe de définition des schémas des ontologies pour les langages DAML+OIL/OWL, il offre les primitives de représentation de structure. Les classes et leurs propriétés sont définies en RDF Schema (en utilisant les constructeurs *class, property, range, domain, sub/super-class,* exemple : `class personne sub-class-of animal`) et les instances de l'ontologie par RDF (`john is- instance-of personne).`

b) Les cartes topiques (Topic Maps) :

Une proposition concurrente à RDF(S) pour représenter les méta-données, par exemple pour les ressources web, est celle des Topic (Navigation) Maps dont un des buts originaux était de gérer et de fusionner des index et des glossaires provenant de plusieurs documents électronique. Une première standardisation, fondée sur une DTD SGML, en a été donnée par l'International Organization for Standardization (ISO). Issu donc de SGML, il s'est vu récemment attribué une syntaxe XML (XTM), c'est-à-dire, une sérialisation sous forme de document XML afin de pouvoir échanger des Topics Maps . Par ailleurs, un groupe de l'ISO s'occupe de définir un langage de requête pour les cartes topiques (TMQL) **[Lau02]**.

- Structure d'une Topic Map [Ouz03] :
Une Topic Map est un graphe de *topics* qui représentent des sujets, liés par des *associations* sémantiques. Au *topics* sont associées des ressources appelées *occurrences.* Une Topic Map est composée de deux partie : la partie des *topics* (dite partie abstraite) et la partie des *occurrences* (dite partie concrète).

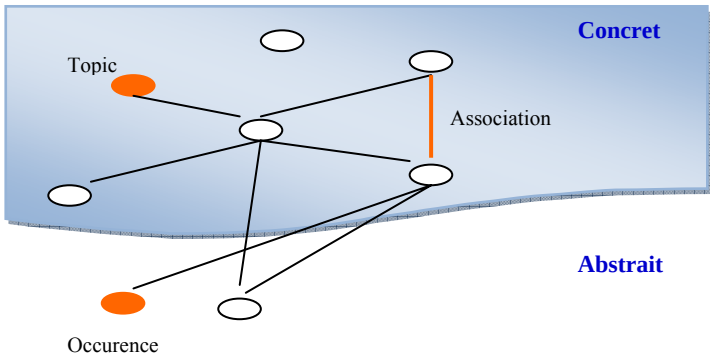


Fig 1.2: Structure de base d'une Topic Map

- **Topics** : Un *topic* peut être n'importe quoi, une idée, un concept, une entité, une personne, une situation, un objet réel …etc, peu importe s'il existe ou pas et quelque soit les caractéristiques et les relations qu'il peut avoir **[ISO99]**. Un *topic* est l'objet qui représente un sujet sur une Topics Maps : *Patient, examen, Dupont, est_dans, A_examiner, Symptôme,* etc sont des *topics*.

Un *topic* peut être une instance de un ou plusieurs *topics*. Par exemple, le *topic* Dupont peut être défini comme étant une instance du *topic Patient* ou *Personne.* Les *topics* peuvent donc avoir des types (on dit A est instance de B ou B est un type de A). ces types sont représentés par des *topics* en Topic Map.

- **Occurrences :** Une occurrence est une ressource associée à un ou plusieurs *topic.* Une occurrence d'un *topic* est toute information considérée comme pertinente à ce *topic* dans un contexte donné **[ISO99]**. Par exemple une occurrence du *topic* Dupont peut être sa photographie d'identité, sa page personnelle, son livret de famille, etc. Ces occurrences qui sont en fait des ressources web (images, vidéo, document texte, programme…) sont indépendantes des Topic Map sont donc généralement adressés par des lien XLink.
- **Association :** L'aspect le plus important dans ce formalisme de représentation des connaissances est la nature des relations qui lient les *topics* les uns aux autres. Dans une Topic Map, un *topic* est régi par les associations qui le lient aux autres *topics* du Topic Map. Par exemple,
  - Dupont *est un* patient
  - Le médecin John *a soigné* le patient Dupont.
  - John *travaille avec* Arnaud et Peter.
  - Paris est *la capitale* de France

sont des associations. Par exemple l'association '*est la capitale de'* est de type '*Capitale de'*, l'association '*travaille avec'* est de type '*Collègue'*. Une association peut être liée à plusieurs *topics.* Chaque *topic* (membre de l'association) joue un rôle dans chaque association à laquelle il participe.

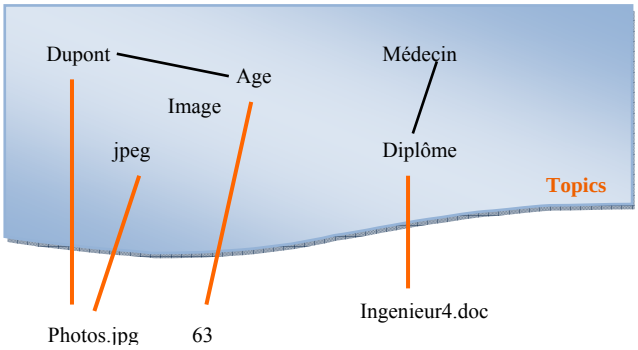


Fig 1.3: Topics, occurrences et associations

- **Scope :** Un *topic* est défini par quatre caractéristiques : son nom, son type, ses occurrences et ses associations ainsi que les rôles qu'il joue dans ces associations. Ces associations assignées à un *topic* ne sont considérées que dans certains contextes. La validité d'une caractéristique est exprimée par la notion de *scope*, définie par le standard **[ISO99].**
  Par exemple la personne Dupont peut être un patient (contexte *ctx1*) et un médecin (contexte *ctx2*) dans une Topic Map représentant un hôpital. Selon le contexte de traitement du *topic* Dupont, on ne considère que les caractéristiques valides à ce contexte. Par exemple, parmi les noms que peut avoir ce *topic*, Mr.

Dupont et Dr. Dupont, on assigne chacun d'entre eux au *topic* selon les deux contextes. C'est-à-dire le *topic* Dupont s'appelle Mr. Dupont dans le contexte *ctx1* et Dr. Dupont dans le contexte *ctx2*.

- **Facettes :** Une facette n'est autre qu'une paire (propriété, valeur) associée à une occurrence. Ces propriétés sont des méta-données associées au Topic Map pour décrire les occurrences. Faisant suite à l'exemple précédent, on associe la facette (langue, français) à l'occurrence *pagePersoFr* et (langue, anglais) à l'occurrence *pagePersoUk* représentant respectivement une version en Français et en Anglais de page personnelle de Dupont **[Ouz03]**.

Une des particularités des Topics Maps de l'ISO par rapport à RDF de W3C est la notion de « scope » qui permet de définir des contextes différents dans lesquels les éléments nommés identiquement peuvent avoir des significations différentes. Par contre, la notion de hiérarchie de classes n'existe pas dans les *topics*. Quoiqu'il en soit, les Topic Maps, de même que RDF(S) permettent aussi bien l'expression de méta-données que l'exploitation des relations entre éléments pour différentes tâches, par exemple l'aide à la navigation sur le Web sémantique **[Lau02]**.

4.1.2 Langage de définition d'ontologie (OWL)

RDF, langage dédié à l'expression d'assertions sur les relations entre objets, s'est heurté à la nécessité de définir les propriétés des classes dont ces objets sont instances. Cependant, l'extension à RDFS ne fournit que des mécanismes très basiques pour spécifier ces classes. Le langage OWL **[OWL]**, quant à lui, est dédié aux définitions de classes et de types de propriétés, et donc à la définition des ontologies. Inspiré des logiques de descriptions et successeur de DAML+OIL, il fournit un grand nombre de constructeurs permettant d'exprimer de façon très fine les propriétés des classes définies : propriétés de classe équivalente, de propriété équivalente, d'identité de deux ressources, de différences de deux ressources, de contraire, de symétrie, de transitivité, de cardinalité, …etc, permettant de définir des rapports complexes entre des ressources. La rançon de cette expressivité est l'indécidabilité du langage obtenu en considérant l'ensemble de ces constructeurs. C'est pour cela que OWL a été fractionné en trois sous langages distincts : OWL LITE, OWL DL, OWL FULL **[Bag03]**.

OWL LITE convient aux utilisateurs qui ont principalement besoin d'une hiérarchie de classification et de contraintes simples. Par exemple, alors qu'il supporte les contraintes de cardinalité, il autorise seulement des valeurs de cardinalité 0 ou 1. OWL fournit un chemin rapide pour la migration pour les thesaurus et d'autres taxonomies.

OWL DL **:** convient aux utilisateurs qui veulent le maximum d'expressivité tout en maintenant la complétude (toutes les conclusions sont calculées) et la décidabilité (tous les calculs s'effectuent dans un temps fini). OWL DL inclut tous les constructeurs du langage OWL, mais ils peuvent être utilisés seulement sous certaines restrictions (par exemple, lorsqu'une classe est sous classe de plusieurs classes, elle ne peut pas être instance d'une autre classe). OWL DL est appelé ainsi en raison de sa correspondance avec les logiques de description.

OWL Full convient aux utilisateurs qui veulent le maximum d'expressivité et la liberté syntaxique de RDF sans aucune garantie sur la conclusion. Par exemple en OWL Full, une classe peut être traitée simultanément comme une collection d'individus et comme un individu "in its own right". OWL Full permet à une ontologie d'augmenter la signification de son vocabulaire (RDF ou OWL) prédéfini **[Bag03]**.

Chacun de ces sous langages est une extension de son prédécesseur, à la fois dans ce qui peut être légalement exprimé et dans ce qui peut être bien conclu. Dans la suite, nous décrirons d'abord les caractéristiques de OWL Lite, suivi d'une description des caractéristiques qui sont ajoutées dans OWL DL et OWL Full. OWL DL et OWL Full contiennent les mêmes caractéristiques mais OWL Full est plus souple sur la façon dont ces caractéristiques peuvent être combinées.

Les préfixes `rdf:` ou `rdfs:` sont employés dans un document OWL lorsque les termes sont déjà présents dans RDF ou RDF Schéma. Sinon les termes sont introduits par OWL. Ainsi le terme `rdfs:subPropertyOf` indique que `subPropertyOf` est déjà dans le vocabulaire de RDF-S. En outre, le terme `Class`, plus précisément énoncé comme `owl:class` est un terme introduit par OWL.

OWL LITE :

- Reprend tous les constructeurs de RDF (c'est-à-dire fournit des mécanismes permettant de définir un individu comme instance d'une classe, et de mettre des individus en relation).
- Utilise les mots-clés de RDFS (`rdfs:subClassOf`, `rdfs:Property`, `rdfs:subPropertyOf`, `rdfs:range`, `rdfs:domain`), avec la même sémantique.
- Permet de définir une nouvelle classe (`owl:Class`) comme étant plus spécifique ou équivalente à une intersection d'autres classes.
- `owl:sameIndividualAs` et `owl:differentIndividualFrom` permettent d'affirmer que deux individus sont égaux ou différents.
- Des mots-clés permettent d'exprimer les caractéristiques des propriétés : `owl:inverseOf` sert à affirmer qu'une propriété *p* est l'inverse de *p'* (dans ce cas, le triplet <*s p o*> a pour conséquence <*o p' s*>), d'autres caractéristiques sont par exemple la transitivité (`owl:TransitiveProperty`), la symétrie (`owl:SymmetricProperty`).
- `owl:allValuesFrom` associe une classe *C* à une propriété *P*. Ceci définit la classe des objets *x* tels que si <*x P y*> est une relation, alors la classe de *y* est *C* (quantification universelle de rôle en logique de descriptions). `owl:someValuesFrom` encode la quantification existentielle de rôle.
- `owl:minCardinality` (resp. `owl:maxCardinality`) associe une classe *C*, une propriété *P*, et un nombre entier *n*. Ceci définit la classe des objets *x* tels qu'il existe au moins (resp. au plus) *n* instances différentes *y* de *C* avec <*x P y*>.
  Pour des raisons d'efficacité algorithmique, OWL LITE ne permet d'utiliser que des entiers égaux à 0 ou 1. Cette restriction est levée dans OWL DL **[Bag03]**.

OWL DL

- Reprend tous les constructeurs d'OWL LITE.
- Permet tout entier positif dans les contraintes de cardinalité.
- `owl:oneOf` permet de décrire une classe en extension par la liste de ses instances.
- `owl:hasValue` affirme qu'une propriété doit avoir comme objet un certain individu.
- `owl:disjointWith` permet d'affirmer que deux classes n'ont aucune instance commune.
- `owl:unionOf et owl:complementOf` permettent de définir une classe comme l'union de deux classes, ou le complémentaire d'une autre classe.

OWL FULL

- reprend tous les constructeurs d'OWL DL.
- reprend tout RDF Schema.
- permet d'utiliser une classe en position d'individu dans les constructeurs.

Nous n'avons pas cité ici certains constructeurs, qui peuvent être trivialement implémentés grâce à ceux que nous avons évoqués, par exemple `owl:sameClassAs`, servant à affirmer que deux classes sont identiques, peut être écrit grâce à deux `rdfs:subClassOf`. Il serait intéressant d'identifier quels sont les constructeurs primitifs nécessaires pour ces langages, et ceux qui ne sont que des macros **[Bag03]**.

4.1.3 Les langages pour les services Web

Nous allons présenter dans cette partie les objectifs et les fonctionnalités des principaux langages consacrés aux services sur le WEB. La notion de services correspond à une approche spécifique des ressources disponibles sur le Web qui met l'accent sur les fonctionnalités offertes par tel ou tel logiciel en terme d'un processus métier. Suivant la définition usuelle, un service permet à un utilisateur, non seulement d'obtenir de l'information, mais aussi d'effectuer des changements sur l'état du monde. Le commerce électronique est, bien sûr, un exemple privilégié de ces approches.

De nombreuses techniques et plusieurs langages sont proposés comme des standards actuels ou futurs pour découvrir ou localiser les services Web (typiquement à travers un annuaire des services), pour invoquer ou activer ces services et les faire interopérer. De manière plus ambitieuse et moins aboutie aujourd'hui, il est proposé en plus de surveiller ou de suivre leur exécution (identifier les échecs, donner des traces) et surtout de les composer (sélection automatique, enchaînement et interopération) **[Bag03]**.

a. WSDL (Web Service Description Language):

WSDL **[WSDL]** est un langage basé sur XML servant à décrire les interfaces des services Web, c'est-à-dire en représentant de manière abstraite les opérations que les services peuvent réaliser, et cela indépendamment de l'implémentation qui en a été faite.

Dans WSDL, les services sont définis à l'aide de "endpoints". Les "endpoints" sont des ensembles de ports, c'est-à-dire d'adresses sur le réseau associées à certains protocoles et formats de données. Cela va permettre de fournir un cadre abstrait et indépendant des implémentations pour les communications avec les services. Il y a quatre types d'opérations de base définies dans WSDL : «sens-unique», «double-sens» requête-réponse, «double-sens» sollicitation-réponse, et «sens-unique» de message de notification.

b. UDDI (Universal Description, Discovery and Integration):

Le protocole UDDI (Universal Description, Discovery and Integration) **[UDDI]** est une plate-forme destinée à stocker les descriptions des services Web disponibles, à la manière d'un annuaire de style « Pages Jaunes ». Des recherches sur les services peuvent être effectuées à l'aide d'un système de mots-clés fournis par les organismes proposant les services. UDDI propose également un système de « Pages Blanches » (adresses, numéros de téléphone, identifiants…) permettant d'obtenir les coordonnées de ces organismes. Un troisième service, les « Pages Vertes », permet d'obtenir des informations techniques détaillées à propos des services et permettent de décrire comment interagir avec les services en pointant par la suite vers un PIP RosettaNet ou une "service interface" WSDL. Le vocabulaire utilisé pour les descriptions obéit à une taxonomie bien précise afin de permettre une meilleure catégorisation des services et des organismes.

c. SOAP (Simple Object Access Protocol) :

SOAP **[SOAP]** est un protocole léger pour l'échange d'informations dans un environnement décentralisé et distribué. Il est de ce fait particulièrement adapté au Web. Le principal avantage de *SOAP* est qu'il repose sur 2 standards : *XML* (pour la structure des messages) et *HTTP* (pour le transport) bien qu'il soit utilisable avec d'autres protocoles. Il fait l'objet d'une normalisation auprès du W3C. Par rapport à tous les autres protocoles de *RPC*, celui-ci présente l'avantage de l'interopérabilité et implémente l'indépendance des plates-formes et des langages de programmation. Le second avantage réside dans l'utilisation de protocoles standards qui facilitent la communication Inter-sites **[Duv04]**.

d. DAML-S

DAML-S **[DAML-S]** est un langage de description de services basé sur XML utilisant le modèle des logiques de descriptions (et plus précisément DAML+OIL, voir plus haut). Son intérêt est qu'il est un langage de haut niveau pour la description et l'invocation des services Web dans lequel la sémantique est incluse, contrairement par exemple à UDDI. DAML-S est composé de trois parties principales :

- *Service Profile,* qui permet la description, la promotion et la découverte des services, en décrivant non seulement les services fournis, mais également des pré-conditions à la fourniture de ce service. Les recherches sur les services

peuvent se faire en prenant n'importe quel élément de Service Profile comme critère.

- *Service Model,* qui présente le fonctionnement du service en décrivant dans le détail et de manière relativement abstraite les opérations à effectuer pour y accéder. C'est le Service Model qui va permettre une composition des services si besoin. Il permet également d'effectuer un contrôle poussé du déroulement du service.
- *Service Grounding,* va présenter clairement et dans le détail la manière d'accéder à un service. Tout type abstrait déclaré dans le Service Model s'y verra attribuer une manière non ambiguë d'échanger l'information. C'est dans cette partie que le protocole et les formats des messages entre autres sont spécifiés.

Nous revenons sur ces notions dans le chapitre suivant consacré à la technologie des Services Web et les différents langages définis dans ce cadre.

4.2 Architecture du Web sémantique

Le Web sémantique est une architecture multi-couches s'appuyant sur une pyramide de langages dont seulement les couches les plus basses sont aujourd'hui relativement stabilisées. La figure si-dessous illustre une des versions de l'organisation en couches proposée par le W3C. Deux types de bénéfices peuvent être attendus de cette organisation :

1. Elle permet une approche graduelle dans les processus de standardisation et d'acceptation par les utilisateurs.
2. Par ailleurs, si elle est bien conçue, elle doit permettre de disposer du langage au bon niveau de complexité, celle-ci étant fonction de l'application à réaliser.

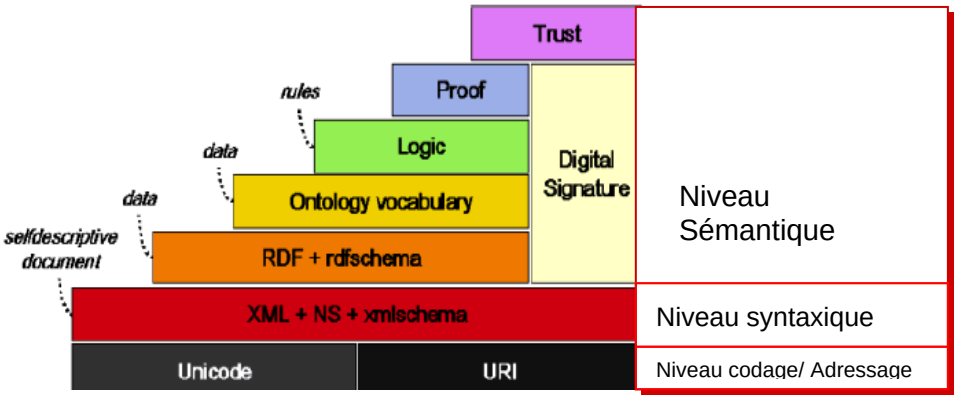


Fig 1.4: L'architecture du Web sémantique d'après Tim.Berner-Lee

Les trois niveaux les plus importants du Web sémantique sont : le niveau codage/adressage, le niveau syntaxique et le niveau sémantique **[Lau02]**.

4.2.1. Le niveau «Codage/Adressage »:

Le niveau le plus bas concerne le codage des données brutes utilisant l'Unicode et l'identification des ressources du Web **[Ber01]**. Le World Wide Web repose sur un concept très important qu'est l'URI (Uniform Ressource Identifier) pour l'adressage des ressources sur le Web. Tout ce qui est disponible sur Internet doit être identifié par un URI. Le point central des URIs est l'URL (Uniform Resource Locator) traditionnelle utilisée pour définir les liens du Web. Ces URLs sont utilisées pour référencer des fichiers Web à travers un protocole particulier, comme HTTP ou FTP. A partir du moment où toutes les ressources sont identifiées par un URI, tout le monde peut compléter la description d'une ressource sans avoir besoin de lui assigner un nouvel identifiant **[Lau02]**.

4.2.2. Le niveau syntaxique:

Le deuxième niveau concerne la structuration des documents manipulées dans le Web Sémantique. La spécification de la structure logique des documents Web repose sur XML. XML est considéré comme une première représentation significative des données puisque il permet de baliser les documents selon leurs contenus **[Ber01]**. Le but d'utiliser XML est de faciliter la diffusion et l'échange des informations sur Internet. Les espaces de noms (*namespace)* sont utilisés pour unifier le vocabulaire utilisé pour le balisage des données en XML et les schémas XML (tel DTD, XML Schema) permettent de décrire la structure des documents XML. Cependant, XML ne possède pas de sémantique formelle permettant l'interprétation par la machine. Il décrit uniquement la structure de l'information et sa syntaxe **[Lau02]**.

4.2.3. Le niveau Sémantique:

XML offre une solution à la structuration des données, mais la structure ne suffit pas à rendre des données complètement exploitables par un système. Il est nécessaire de rendre les données interprétables par une machine par l'association d'une sémantique formelle.

- **Ajout de méta-données (annotations) :** RDF **[RDF]** est le premier niveau de formalisation de la sémantique pour le Web sémantique. Il est actuellement le langage le plus utilisé et le plus répandu pour représenter des méta-données. Tandis que XML structure des données, RDF leur associe des méta-données et les met en relation. RDF Schema, comme son non l'indique, permet de typer les ressources et les propriétés utilisées par RDF.

- **Formalisation du sens (ontologies):** L'une des problématiques du Web sémantique concerne le partage et la réutilisation de l'information par tous : hommes et machines. Comme nous l'avons déjà mentionné, RDF permet d'ajouter de la connaissance (méta-données) aux ressources du Web sans aucune signification ni interprétation. La tâche d'interprétation est assignée à cette couche dans l'architecture du Web sémantique. Cette couche est représentée par

les ontologies. Les ontologies se trouve au cœur du Web sémantique puisqu'elle permet de fournir le sens de toutes les ressources **[Ber01]**.

En ce qui concerne les langages de définitions d'ontologies, le Web sémantique s'oriente vers le langage OWL **[OWL]**. OWL (Ontology Web Langage) est un langage pour le balisage sémantique utilisé pour la publication et le partage d'ontologies sur le Web. OWL est dérivé de DAML+OIL et construit sur le langage RDF/RDFS. DAML+OIL est un langage de définition d'ontologies résultant de la fusion de deux projets : DAML (DARPA Agent Markup Langage) issu de travaux américains et OIL (Ontology Inference Layer) issu des travaux européens.

- **Inférence logique, preuve et confiance :** Une ontologie définit les concepts d'un domaine particulier, mais n'explique pas comment il faut les utiliser. Par exemple, considérons un agent électronique qui doit être capable de déduire que si une commande est de 10 objets, que 4 ont été commandés à un premier fournisseur et que 6 ont été commandés à un second fournisseur, alors la commande est passée. Ce type de connaissance ne fait pas partie de l'ontologie mais concerne les actions, processus et déductions logiques sur les objets de l'ontologie. Le raisonnement sur la connaissance est donc un point important du Web Sémantique**[Ber01]**. La possibilité de faire des déductions plus complète pourra s'appuyer sur la standardisation d'un langage de règles, comme RuleML (Rule Markup Langage) qui est en cours de maturation. De manière plus ambitieuse encore, la capacité de produire des preuves des déductions faites pourra augmenter le niveau de confiance des utilisateurs dans ces déductions. Notons que ce problème ne se réduit évidemment pas à cette capacité. Elle doit reposer aussi sur des méthodes de qualification de l'origine de l'information, par exemple par des méta-données et des annotations, éventuellement certifiées par des signatures électroniques. La signature électronique (numérique) est l'une des techniques retenues pour contrôler la validité des connaissances. C'est une suite de bits associée à une connaissance pour identifier l'auteur de cette connaissance **[Lau02]**.

5. Applications des technologies du Web sémantique [Lég03]

Les technologies du Web sémantique sont de plus en plus appliquées à un large spectre d'applications au sein desquelles une connaissance de domaine est modélisée et formalisée (ontologie) afin de servir de support à des traitements très diversifiés (raisonnements) effectués par des machines. En outre, ces représentations peuvent être rendues compréhensibles par l'homme pour assurer un couplage optimal entre raisonnements humains (cognitifs) et mécaniques (sémantique formelle) confiant à l'homme et à la machine des tâches complémentaires.

Pour citer quelques-unes de ces applications : *Portails d'entreprises et Mémoire d'entreprises, E-Commerce*, *E-Work, Traitement Automatique des Langues et Traduction Automatique, Recherche d'Information, Intégration d'Entreprises et EWork,*

Communautés d'Intérêts, Data Mining, etc. D'un point de vue économique et sociétal, cette technologie doit pouvoir contribuer à la croissance économique, en permettant aux entreprises d'inter-fonctionner plus aisément et de trouver plus rapidement de nouvelles et meilleures opportunités de marchés, mais également contribuer à la société civile dans sa vie quotidienne au travail et pour ses loisirs.

5.1 E-Commerce (Commerce Electronique):

Le commerce électronique doit permettre un échange plus fluide d'information et de transactions entre tous les acteurs économiques, depuis l'offreur de produits ou services jusqu'aux clients finals.
Les applications du E-commerce essentiellement application B2B utilisent les échanges informatisés via des structures de messages et de protocoles très codifiées, pré-établies et normalisées (EDI – *Electronic Data Interchange* ou Échange de Données Informatisés) **[EDI]** récemment assouplies via des standards basés sur XML (*eXtensible Markup Language*). De nombreux acteurs du commerce électronique ont uni leurs efforts pour concevoir un nouveau standard pour le commerce électronique : ebXML (*electronic business in XML*). ebXML se positionne dans la complémentarité et dans la continuité de EDIFACT. Ce dernier est particulièrement adapté aux échanges de gros volumes avec des partenaires stables, alors que ebXML doit répondre, en plus, à la problématique des petits échanges entre partenaires épisodiques.

Actuellement, les systèmes à base d'ontologies apparaissent comme une technologie clé pour le développement de solutions d'E-Commerce efficaces, ouvertes et profitables. Cependant, par manque de normes de modèles de domaine et de processus métier dans les plus larges secteurs économiques, la variété des pratiques commerciales, la difficulté de la description des produits, de services et leurs interactions rendent l'adoption de standards de domaine et de transactions économiques difficiles à atteindre.

Malgré toutes ces difficultés, de réels bénéfices pourraient être tirés de l'usage d'ontologies dans les domaines suivants :

- Catégorisation de produits dans des catalogues.
- Catégorisation de services (dont les Services web).
- Pages Jaunes des sociétés de services.
- Identification des pays, régions et monnaies.
- Identification des organisations, de personnes et d'entités légales.
- Médiation entre différentes ontologies de E-commerce.

Citons l'exemple du système ONTOSEEK (1996-1998) a couplé une représentation des connaissances du domaine (langage à pouvoir d'expression très limité de la famille de graphe conceptuel GC) à une large ontologie linguistique multilingue (SENSUS basé sur WORDNET) pour une recherche de produits en langue naturelle multilingue. Les principaux choix d'architecture fonctionnelle d'ONTOSEEK:

- usage d'une *Ontologie linguistique généraliste* pour représenter finement les produits.

- grande flexibilité terminologique dans l'expression des requêtes, grâce à un mécanisme d'*intersection sémantique* entre les requêtes et la description des produits.
- Assistance interactive pour la formulation de la requête par généralisation et spécialisation.

5.2 Applications médicales :

La médecine est un des domaines d'applications privilégiés du Web sémantique comme elle l'a été, à une autre époque, des techniques de l'Intelligence Artificielle, en particulier les systèmes experts. C'est en effet un domaine complexe où les informations à partager sont nombreuses et où il n'y a pas ou peu de solutions algorithmiques à ce partage comme à l'usage des connaissances, en particulier cliniques. Ainsi, un des principaux mécanismes du Web sémantique qui est la description de ressources via des annotations est de la plus grande importance en bio-informatique, plus particulièrement autour des questions de partage des ressources génomiques. Dans le contexte, plus ancien, de la recherche d'information, la médecine a une longue tradition de développement de thésaurus comme le MeSH (*Medical Subject Heading*) ou UMLS (*Unified Medical Language System)* et les utilise maintenant dans le cadre des mécanismes du Web sémantique. Enfin, et plus récemment, les services Web proposent des solutions à la problématique récurrente et non résolue de l'interopérabilité en médecine, en particulier dans le contexte des SIS (Systèmes d'Information Hospitaliers).

5.3 Portail et Mémoire d'entreprise :

Depuis quelques années, la capitalisation des connaissances est vue comme un sujet stratégique pour les entreprises. C'est ainsi que se sont développées tant du point de vue méthodologique que technologique les activités de « Mémoire d'entreprise » ou de « gestion des connaissances de l'entreprise » (KM – *Knowledge Management*).
Très clairement le KM est interdisciplinaire et fait appel à la gestion des ressources humaines, à l'organisation et à la culture de l'entreprise, et enfin aux technologies NTIC qui peuvent y jouer un rôle très fort de mutation des usages. Dans **[Hei96]**, la "mémoire d'entreprise" est définie comme la "représentation explicite, persistante, et désincarnée, des connaissances et des informations dans une organisation". La construction d'une mémoire d'entreprise repose sur la volonté de "préserver, afin de les réutiliser plus tard ou le plus rapidement possible, les raisonnements, les comportements, les connaissances, même en leurs contradictions et dans toute leur variété" **[Pom96]**.
Le développement de portails des savoirs servant les besoins de l'entreprise ou de communautés est plus ou moins à ce jour une tâche essentiellement manuelle. Dans un contexte économique très versatile et opportuniste, *Ontologies et Outils d'inférence, TAL, devraient faciliter la maintenance évolutive* des portails qui doivent être à jour et de plus en plus pertinents. Les services classiques associés aux solutions de KM pour lesquelles les technologies du web sémantique seront fortement contributives sont :

- Accès des employés en situation de mobilité à la mémoire de l'entreprise (Mobile KM)
- Partage entre employés d'une même communauté (P2P – *Peer-to-Peer computing*) où la construction de la connaissance (Ontologie et annotations) s'opère de manière naturelle et consensuelle.
- Intégration des mémoires d'entreprises décentralisées et multinationales.
- Formation professionnelle continue (*e.Learning*) sur le portail de l'entreprise sur lequel l'employé se voit offrir des parcours de formation diversifiés et surtout personnalisés.

Le KM est évidemment un champ applicatif des technologies du Web sémantique très prometteur. Les technologies documentaires classiques ont clairement montré leurs limites (très faible capitalisation des savoirs). L'introduction de ces nouvelles technologies laissent entrevoir de réelles avancées de l'offre et des usages.

5.4 Traitement automatique des langues :

L'usage D'ontologies "Linguistiques" Dans Les Applications : Une ontologie linguistique sert de format (représentation) pivot entre applications ou entre interprétations possibles communes de différentes langues car elle considère les concepts pas les termes de la langue **[Zyl00].** Les ontologies linguistiques ont généralement pour objet de résoudre les questions suivantes : comment représenter les connaissances d'un univers donné et comment lier cette représentation à celles aujourd'hui classiques des grammaires et des lexiques ?
De nombreuses applications du web sémantique devraient à l'avenir faire appel aux outils traditionnels du TAL enrichis des représentations et des traitements sémantiques associés.

La Traduction Automatique : Une application de génération de langue naturelle fait traditionnellement appel à une représentation pivot à laquelle on relie les différents termes d'une base lexicale multilingue. Ces applications sont des systèmes de traduction à base de connaissances (KBMT – *Knowledge-Based Machine Translation*), traduisant via le sens (sémantique) un texte d'une langue vers d'autres langues. La représentation du sens est modélisée dans une ontologie indépendante des langues qui joue le rôle « d'interlingua ».
Les principaux bénéfices attendus sont : de fournir un fondement pour représenter le sens de texte dans un «interlingua» (Inter-Lingual-Index), pour permettre à des lexiques de différentes langues de partager un même modèle. Au travers de cet index, les langages sont interconnectés de telle sorte qu'il est possible de passer des mots d'une langue aux mots similaires d'une autre langue.
Voici quelques classes d'applications archétypes de l'usage immédiat et tangible des technologies du web sémantique. Il ne fait aucun doute que cette technologie du sens doive apporter un saut qualitatif indiscutable si ce n'est une réelle rupture technologique. Toutefois, la technologie est encore immature et de nombreuses questions scientifiques restent ouvertes telles :

- le passage à l'échelle du Web.
- la tenue en contexte de forte hétérogénéité (modélisations et langages).
- la tenue en milieu fortement évolutif.

6. Conclusion

Partage, Echange, Réutilisation : Ces trois mots définissent à eux seuls le Web. Le Web Sémantique constitue une sorte de révolution de fond du Web actuel. Il est le point de départ pour le développement de services Web intelligents. En effet, non seulement l'homme pourra partager, échanger et réutiliser la connaissance et l'information qui est disponible sur le Web mais en plus, il pourra le faire plus vite et avec l'aide des machines. HTML a été le langage du Web jusqu'à présent. Il permettait de présenter l'information aux humains. Désormais, il est nécessaire de présenter cette information de façon à ce qu'un programme puisse s'en servir. Les initiatives de standardisation sont nombreuses mais essentielles pour le Web, et certains standards tels que XML sont déjà bien implantés dans l'informatique d'aujourd'hui. XML est devenu le langage privilégié en ce qui concerne l'échange de données et leur stockage.
En ce qui concerne la gestion des ontologies, les extensions de RDF, RDFS et OWL sont devenues des incontournables. Avec le Web Sémantique, les ontologies vont devenir les points centraux des systèmes d'information de demain. Même si leur principal défaut aujourd'hui est d'être élaborées pour des applications particulières, néanmoins il existe déjà des banques d'ontologies sur le Web. Cette révolution technologique offre de nouvelles opportunités au développement de systèmes et de services intelligents et faciles d'utilisation fondés sur la connaissance des contenus du Web.

Nous avons voulu à travers ce chapitre souligner un certain nombre de points importants pour la réalisation de la vision du Web sémantique. Nous pouvons souligner aussi que les travaux visant la réalisation du Web sémantique se situent à des niveaux de complexité très différents. Les plus simples utilisent des jeux plus au moins réduits de méta-données. Dans ce cas, des langages de représentation simples sont suffisants. Dans les travaux plus complexes mettant en œuvre des architectures sophistiquées, pour permettre par exemple l'exploitation de ressources hétérogènes, des langages plus expressifs et plus formels sont nécessaires. L'architecture proposée par le W3C du Web sémantique doit permettre une approche graduelle dans le processus de standardisation et d'acceptation et aussi doit permettre de disposer du bon langage au niveau de complexité de l'application à réaliser.

La problématique des services Web enrichit d'une nouvelle dimension la perspective du Web sémantique. Elle correspond à une approche spécifique des ressources disponible sur le Web qui met l'accent sur les fonctionnalités offertes par tel ou tel logiciel en terme de processus métier. Les approches services Web et celles du Web sémantique partagent le but commun de rendre l'information sur le Web plus accessible aux

machines. Un certain nombre de recherches se proposent de les coupler. Le Web sémantique doit permettre de représenter et de manipuler des connaissances non seulement structurelles, mais aussi de nature *fonctionnelle*, c'est-à-dire de représenter formellement les connaissances liées au fonctionnement des composants actifs qui y agissent.

Dans le prochain chapitre, nous allons voir en détail ce que recouvre cette notion de « Services Web » et « services web sémantiques » ainsi que les technologies nécessaires à leur développement et à leur déploiement.

Chapitre 2: Les Services Web

1. Introduction

Avec l'essor du Web qui a eu dans les dernières années, il a surgi le besoin de permettre qu'une application cliente invoque et exécute un service d'une application serveur en utilisant Internet. Ce besoin a été l'origine de ce qui se connaît aujourd'hui comme *Services Web*. En tenant en compte que les services Web permettent de connecter des applications différentes s'exécutant sur des plates-formes différentes à travers Internet, l'utilité de cette technologie devient évidente et importante. Pour cette raison les activités de recherche et de développement autour du sujet Service Web ont un dynamisme très haut. Certains grands groupes du secteur de l'informatique présentent les *services Web* comme une révolution qui va modifier le modèle économique actuel de consommation.

L'idée au centre des Services web est de construire un monde où les systèmes logiciels de toute origine sont capables, dans un premier temps, de se découvrir automatiquement, et, dans un deuxième temps, de fonctionner ensemble dans un système de communication universel ".

La mise en œuvre des Services Web ouvre de nouvelles perspectives dans la construction d'architectures de systèmes d'information. En effet, elle masque la complexité des technologies de mise en œuvre des systèmes repartis telles que *CORBA*, *DCOM*, *JAVA/RMI*. On passe d'une technologie de composants à une architecture orientée services facilitant ainsi l'intégration d'applications.
Dans cette partie, nous revenons sur la genèse de cette technologie et nous allons essayer, par la suite, de décrire les solutions apportées par les Services Web pour repousser les limitations de l'informatique repartie.

2. Origine des Services web

Le Web sémantique s'intéresse principalement aux informations statiques disponibles sur le web et les moyens de les décrire de manière intelligible pour les machines. Les services web, quant à eux, ont été créés à l'origine pour avoir comme préoccupation première l'interopérabilité entre applications via le Web **[Kel03]**. Ils prennent leurs origines dans l'informatique repartie et dans l'avènement du Web. Ainsi, il parait important de présenter quelques approches d'interopérabilité des systèmes informatiques et un peu d'historique des architectures réparties qui permettent de connaître les ancêtres des services web et ainsi de mieux comprendre leur raison d'être actuelle.

2.1 L'informatique répartie

2.I.1 Genèse de l'informatique repartie :

La répartition des applications informatiques fait l'objet d'un fort développement depuis une vingtaine d'années. Cet essor s'est accentué avec les progrès technologiques des réseaux de télécommunication. En effet, les applications réparties doivent leur naissance en tout premier lieu au mariage de l'informatique et des télécommunications.
Un point anecdotique concerne la terminologie. Doit-on parler de systèmes répartis ou de systèmes distribués. Les constructeurs d'ordinateurs ou de réseaux utilisent plutôt le terme "distribué". Les spécialistes du logiciel préfèrent le terme "réparti". Dans la littérature et sur le Web, le terme "Informatique répartie" et "informatique distribuée" sont largement confondus **[Duv04]**. Dans ce document les deux termes seront utilisés indifféremment.
Le but de l'informatique repartie est de permettre à une application sur une machine d'accéder à une fonction d'une autre application sur une machine distante et ce, de la même manière que l'appel d'une fonction locale, indépendamment des plates-formes et des langages utilisés.

Les années 1970, l'architecture mainframe (1 tier) :
Pour des raisons de coûts; chronologiquement, l'informatique s'est d'abord organisée de façon centralisée. Les premiers ordinateurs et leurs périphériques étaient très volumineux et très coûteux. L'architecture centralisée consiste en un noyau central autour duquel tous les périphériques sont regroupés (voir la figure 2.1). Ce noyau central prend en charge la plupart des fonctions. Les seules fonctions réparties étaient alors la saisie au clavier et la visualisation sur un simple terminal **[Duv04]**. L'avantage d'une telle architecture est la facilité d'administration et de déploiement.

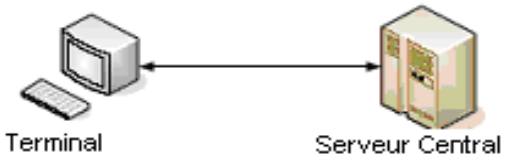


Fig 2.1: L'architecture mainframe

Les années 1980, l'architecture client/ serveur (2 tiers) :
Avec la baisse continue du coût des machines et la remise en question du centralisme informatique au tournant des années 70, puis l'affirmation du micro-ordinateur comme "terminal intelligent" dans les années 80 et l'avènement du modèle client-serveur, les architectures de réseau informatique se sont de plus en plus orientées vers une distribution des ressources et de la puissance informatique.
Une application bâtie selon un modèle client-serveur lorsqu'elle est composée de deux programmes, coopérant l'un avec l'autre à la réalisation d'un même traitement. La première partie, appelée module client, est installée sur le poste de travail alors que la seconde, appelée module serveur, est implantée sur l'ordinateur chargé de rendre le service (micro, mini ou grand système éventuellement situés dans des lieux

géographiques différents). Ce modèle permet d'offrir une plus grande disponibilité, y compris en cas de panne **[Duv04]**.

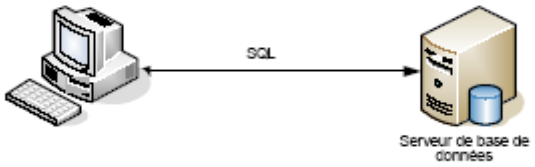


Fig 2.2: L'architecture client serveur 2 tiers

Fin des années 1980, l'architecture client/ serveur (3 tiers) :

L'architecture trois tiers permet de séparer clairement les couches présentation, métier et de données représentées respectivement par le client (le demandeur de ressources), le serveur d'application et le serveur de bases de données. Grâce à la possibilité d'exécuter les trois couches sur des machines dédiées, la disponibilité s'en retrouve ainsi améliorée et les performances de traitement augmentées **[Duv04]**.

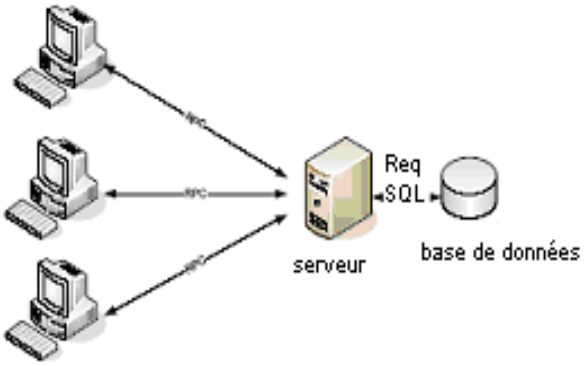


Fig 2.3: L'architecture client serveur (3 tiers)

Début des années 1990, l'architecture 3 tiers objet :

Dans un environnement procédural, seul les données peuvent transiter entre les différentes machines d'un système réparti. Dans un environnement objet, l'état d'un l'objet (l'instance) peut être transféré sur un réseau, par état il faut comprendre que les données et le comportement de ce dernier sont transférés. La possibilité d'utiliser des modèles objets a considérablement augmenté la qualité potentielle d'un développement d'application. En effet, le modèle objet permet, entre autre, une meilleure réutilisation et une meilleure évolutivité des applications **[Duv04]**.

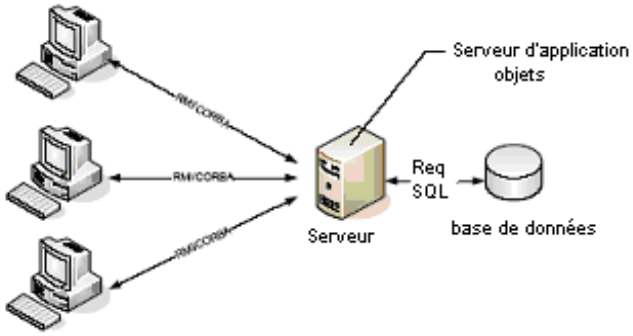


Fig 2.4: L'architecture 3 tiers objet

Les années 2000, l'architecture Internet (3 tiers) :
Internet est sans doute l'exemple le plus marquant d'un réseau à architecture distribuée puisqu'il ne possède aucun site central (voir la figure 2.5).

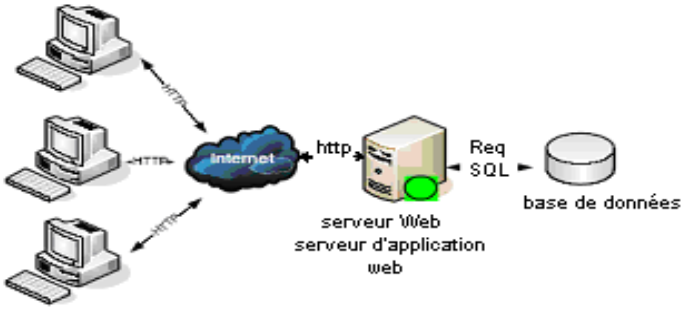


Fig 2.5 : L'architecture Internet (3 tiers)

2.1.2 Les technologies de mise en œuvre de l'informatique repartie:

Les technologies présentées ci-dessous permettent le développement d'applications réparties ou distribuées :

- RMI proposé par Sun.
- DCOM (Distributed Component Object Model) proposé par Microsoft.
- CORBA (Commom Object Request Broker Architecture) proposé par l'OMG (Object Managment Group).
- Services Web proposés par le W3C (World Wide Web Consortium).

a) La technologie CORBA : [Plo01]

CORBA, Common Object Request Broker Architecture, est un standard de l'OMG proposé dans les années 90. L'objectif de cette dernière est né du besoin de faire communiquer des applications entre elles dans un environnement hétérogène et distribué (systèmes différents et langages différents).
Les applications reparties sont composées d'un ensemble d'objets distincts pouvant être répartis sur différentes machines et différents sites. Il est alors nécessaire d'uniformiser leur représentation pour pouvoir accéder à leurs fonctionnalités. Cependant, il est également nécessaire de leur permettre de communiquer à travers un réseau. Ils doivent, en particulier, pouvoir s'échanger des données ou invoquer des méthodes distantes. De nombreux protocoles de communications existent. Cependant le protocole utilisé doit être indépendant des couches réseaux et des architectures matérielles. CORBA propose deux technologies principales pour résoudre ce type de problèmes :

a- Le langage IDL :
IDL, Interface Definition Language, est un langage d'interfaçage des applications hétérogènes. Ce langage a pour objectif de fournir une représentation uniforme des objets CORBA (applications connectées à l'environnement CORBA). Il permet ainsi d'exprimer des interfaces objet à partir d'applications pouvant être codées dans différents langages orientés objet (Java, C++, Smalltalk,...) ou procéduraux (C, Cobol, …). De cette façon, *CORBA* parvient à faire communiquer des clients *Java* avec des

logiciels *C++* par exemple, sans que ni l'un ni l'autre n'ait besoin de savoir dans quel langage les objets sont véritablement implémentés. En effet, seules les interfaces *IDL* sont nécessaires aux clients pour connaître les méthodes qu'ils peuvent invoquer sur les serveurs.

Les appels de méthodes distantes dans l'architecture CORBA sont réalisés par les applications clientes. Ces applications émettent des requêtes à partir de leurs interfaces IDL et en utilisant les méthodes définies au niveau des interfaces IDL des applications distantes. Ces applications ne peuvent communiquer que si elles sont reliées à un bus CORBA appelé le *« bus logiciel de CORBA »* ou *«ORB (Object Request Broker)»*. Ce dernier assure le dialogue entre les objets CORBA connectés et permet ainsi l'exécution de méthodes distantes. (voir la figure ci-dessous).

b- Le protocole IIOP :

IIOP, Internet Inter-ORB Protocol, utilise un protocole de communication entre applications CORBA distantes. Ce protocole de communication a pour principaux avantages d'être indépendant des architectures matérielles et de s'appuyer sur un réseau très étendu qui est Internet.
Avec le protocole IIOP, l'architecture CORBA permet facilement de mettre en oeuvre des systèmes informatiques utilisant des composants dispersés sur le réseau Internet.

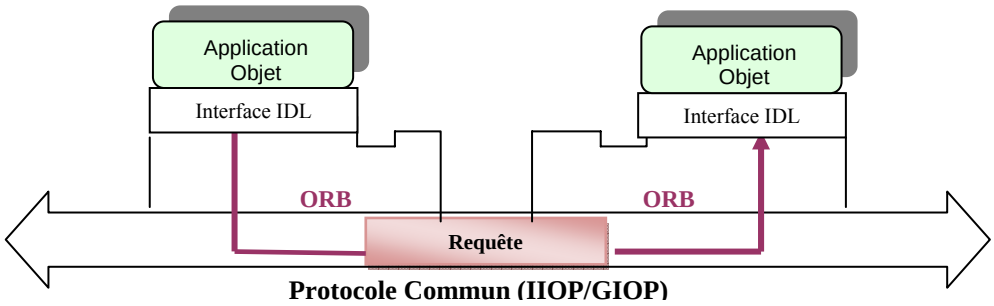


Fig 2.6: Intégration d'appications hétérogènes et distantes à l'aide de CORBA

IIOP est un protocole de communication de type client/serveur. Cependant, une même application peut ouvrir plusieurs connexions clientes ou serveurs.
Ainsi, les applications distantes peuvent accéder aux fonctionnalités disponibles sur le réseau Internet et proposer des services aux applications distantes. On remarque alors que le bon fonctionnement de l'architecture CORBA nécessite que les applications soient connectées au bus de manière permanente.

b) La technologie RMI :

RMI (*Remote Method Invocation*) est une solution proposée par SUN pour adapter le principe des RPCs (Remote Procedure call) à la programmation orientée objet avec une sémantique JAVA. Cette dernière permet de manipuler des objets distants (objets instanciés par une autre machine virtuelle) : localiser, invoquer et transférer des objets en paramètre ou en retour. Dans un environnement objet, il faut comprendre que l'état d'un l'objet (l'instance) peut être transféré sur un réseau, c'est-à-dire, les données et le

comportement de ce dernier. L'encodage d'un objet sur un flux se fait via un mécanisme de *sérialisation Java*.

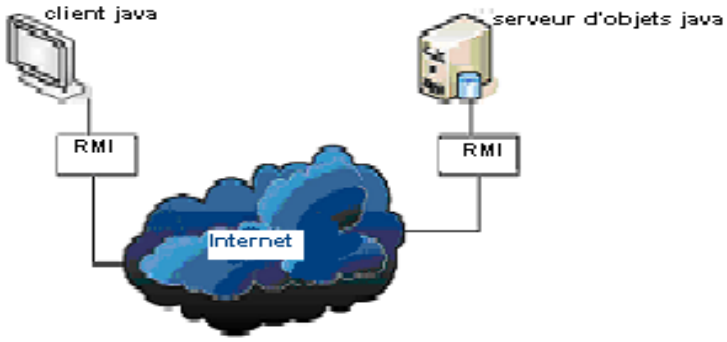


Fig 2.7: Schéma de fonctionnement de RMI

L'objet peut se trouver sur une autre machine physique, ainsi un client peut invoquer des méthodes d'un objet se trouvant sur un serveur d'application. Contrairement à la norme *CORBA*, *RMI* est une solution propriétaire *Java*. Toutefois, *CORBA* est plus compliquée à mettre en oeuvre, c'est la raison pour laquelle de nombreuses applications utilisent la technologique RMI **[Duv04]**.

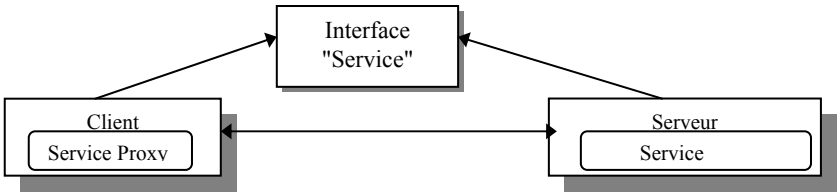


Fig 2.8: Modèle client /serveur RMI

La définition d'un service dans RMI est codée en utilisant une interface et son implémentation est codée à l'aide d'une classe située à distance. Dans RMI, on a deux classes qui implémentent la même interface. La première classe est une implémentation du service sur le serveur et la deuxième est un proxy pour le service distant, il tourne sur le client.

c) EJB (Entreprise Java Bean) **:**

La communauté Java a basé son environnement sur l'utilisation d'un unique langage de programmation. Cette caractéristique a permis de définir des parties de code réutilisables de plus en plus importantes sous forme d'une bibliothèque de composants.
Java définit de manière formelle un modèle de composant Java et une interface de programmation pour les serveurs d'applications. Cet outil, plus connu sous le nom d'EJB (Enterprise Java Beans), définit une architecture de composant pour faciliter le déploiement d'applications distribuées ou reparties.

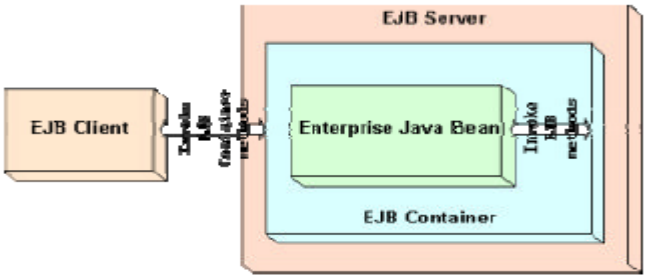


Fig 2.9: Une architecture EJB

Cet environnement est proche de celui défini par CORBA. Il se base sur le langage Java pour définir des composants homogènes en bénéficiant de la portabilité et de la puissance de Java. L'appel d'un composant *EJB* se fait par l'utilisation de l'API *RMI*.

L'avantage d'utiliser la plate-forme Java est l'aspect multiplate-forme du langage utilisé c'est à dire un même programme Java peut être exécuté sous différents systèmes d'exploitation sans qu'il soit nécessaire de modifier le code. Cette particularité permet d'utiliser localement des applications distantes échangées par des réseaux tels qu'Internet.

d) La technologie DCOM :

DCOM est la technologie de composants développée par *Microsoft* pour *Windows* en réponse à *CORBA*. La grande force de *DCOM* est le nombre de machines sur lesquelles on peut le trouver, simplement en raison des parts de marché de *Windows* dans le monde.
Contrairement à *CORBA*, *DCOM* spécifie les interfaces entre les composants au niveau binaire, et non au niveau source. Ceci signifie que cette technologie est indépendante des langages de programmation utilisés. Par contre, *DCOM* souffre d'une complexité non négligeable et de son aspect fermé vis à vis des autres systèmes à composants tels que *CORBA c*ar les composants DCOM ne peuvent être utilisés que sous les systèmes d'exploitation Windows et ces composants doivent être installés et configurés sur la machine à partir de laquelle ils sont utilisés.

2.2. Le besoin en interopérabilité des systèmes informatiques

L'interopérabilité résout de nombreux problèmes fréquemment rencontrés dans l'industrie informatique. Ces problèmes apparaissent lorsqu'on cherche à faire interagir (interopérer) un ensemble d'applications ou d'outils hétérogènes qui n'ont pas été conçus pour fonctionner ensemble. L'évolution de l'informatique est fortement liée à cette volonté de créer des systèmes homogènes. Les types d'incompatibilité sont multiples : utilisation de langages de programmation, de plate-formes, de protocoles de communication réseaux différents.

D'un coté, on peut affirmer aujourd'hui que l'évolution de l'industrie informatique n'a pas de fin et il devient nécessaire de rendre interopérable les technologies passées, présentes et futures.

D'un autre coté, la diversité des technologies proposées par les constructeurs ne sont pas toujours compatibles les uns avec les autres. Les différents constructeurs et vendeurs cherchent à imposer sur le marché leurs langages, leurs systèmes ou autres outils de développement. Pour ces raisons, il est nécessaire d'uniformiser la représentation des applications indépendamment de leur langage d'implémentation ou du système sur lequel ils sont exécutés.

De plus, les applications distribuées sont composées d'un ensemble d'objets pouvant être répartis sur différentes machines et différents sites. Comme nous venons de le voir, il est nécessaire d'uniformiser leur représentation. Cependant, il est également nécessaire de leur permettre de communiquer à travers un réseau en s'échangeant des données ou en invoquant des méthodes distantes. Pour ces raisons, il faut utiliser des formats de données standardisés et des protocoles de communications indépendants des couches réseaux et des architectures matérielles.

Les technologies vues précédemment représentent des plates-formes de développement et de déploiement d'applications réparties. Ces plates-formes ont initié la possibilité de partager des services entre applications distantes et ont proposé des solutions à certains aspects de l'interopérabilité. Ces solutions sont proches l'une de l'autre. Nous allons examiner dans ce qui suit l'apport de ces plate-formes du point de vu de l'interopérabilité.

CORBA :

Pour répondre au problème d'hétérogénéité des applications, l'OMG a définit au sein de l'architecture CORBA un langage commun d'interfaçage des objets logiciels (langage pivot) : ***IDL***, *Inteface Definition Language*. Il permet, comme déjà vu, d'interfacer des applications codées dans différents langages de programmation orientés objet (Java, C++, Smaltalk..), procéduraux (C, COBOL) ou autres. Ces interfaces décrivent les attributs et méthodes accessible par d'autres applications via le bus COBA.

Pour résoudre ces problèmes d'interopérabilité provoqués par les échanges d'information entre applications distantes, CORBA s'appuie sur des protocoles de communication standardisés : GIOP (General Inter-ORB Protocol) et son instanciation pour le protocole TCP/IP d'Internet, ***IIOP*** *(Internet Inter-ORB Protocol).* Ce dernier a pour principaux avantages d'être indépendant des architectures matérielles et de s'appuyer sur un réseau très étendu qui est Internet.

Les applications sont alors reliées à un bus CORBA (ORB) et communiquent par l'intermédiaire de leurs interfaces IDL en utilisant un protocole de communication indépendant des architectures matérielles IIOP.

Toutefois, l'inconvénient de la technologie CORBA est surtout la complexité de mise en œuvre d'une telle architecture. Ajoutant à cela, elle n'offre aucun moyen de découvrir dynamiquement des services offerts et de raisonner sur des compositions d'objets CORBA pour garantir des services plus élaborés.

JAVA, RMI et EJB :

La communauté Java a basé son environnement sur l'utilisation d'un langage unique de programmation pour fournir un cadre de travail homogène. Toute application développée dans ce langage peut interagir facilement avec l'ensemble des applications Java existantes déployées sous différents systèmes d'exploitation. La particularité de cette plate-forme est surtout l'aspect multi-plateforme de son langage qui permet de séparer la partie logiciel de la partie matérielle des systèmes informatiques.

Comme dans l'architecture CORBA, RMI permet de séparer le code qui définit une fonctionnalité et le code qui l'implémente, et de tourner sur différentes machines virtuelles. Cela correspond bien au besoin des systèmes distribués en matière d'interopérabilité où les clients sont concernés par la définition d'un service et les serveurs par leur fourniture. Ainsi, les clients des objets distant ne voient jamais directement les classes qui les implémentent. Ils ne voient que les interfaces distantes.

En effet, bien que RMI supporte les environnements hétérogènes et facile à mettre en œuvre, il n'est pas possible de faire interopérer des applications écrites en langage de programmation autre que Java. Le principe du langage unique est très souvent un inconvénient lorsqu'il s'agit de récupérer un système existant pour le faire évoluer vers une plate-forme Java.

L'architecture des Entreprise JavaBeans **EJB** est une architecture de composants pour le développement et le déploiement d'applications d'entreprise distribuées proposé par la communauté Java. Les applications écrites en utilisant l'architecture des EJBs peuvent être écrites une fois, puis déployées sur toute plate-forme serveur qui supporte la spécification des EJBs. Le point fort de cette architecture est la portabilité des composants EJB, leur persistance et bien sûr tous les avantages du développement à base de composant. Ajoutant à cela, la possibilité d'interagir avec l'architecture CORBA. Mais l'un des points faibles est que l'utilisation de la technologie EJB suppose de bonnes notions au niveau de certains concepts comme les objets distribués, les objets métier et les CTM (Component Transaction Monitors).

En effet, cette spécification des EJBs délimite un cadre de développement de composants "clients", à la fois dans "l'esprit" Java et pour les plate-formes Java. l'interaction avec les autre plates-forme de développement réduit en quelques sorte la portabilité du code.

La technologie COM :

Depuis quelques années, les solutions proposées par Microsoft pour répondre aux problèmes de l'interopérabilité reposent principalement sur les technologies COM ou COM+. Ces outils sont basés sur le principe de réutilisation de code.

Les composants COM sont des bibliothèques de types ou de méthodes disponibles dans un système donné. Cette technologie est très limitée du point de vue de l'interopérabilité. Les composants COM ne peuvent être utilisés que sous les systèmes d'exploitation Windows. Et ces composants doivent être installés sur la machine à partir de laquelle ils sont utilisés. Ils ne sont pas accessibles à travers un réseau comme Internet. Le principal avantage des composants COM est qu'ils peuvent être programmés dans les différents langages de programmation de Microsoft tels que Visual Basic, C++, Visual J++ et par partenaires tels que Microfocus et Fujitsu(Cobol), Borland (Delphi), …etc

Discussion :

Toutes les technologies présentées si-dessus sont des middlewares tentant d'apporter des solutions pour la problématique d'interopérabilité des applications hétérogènes et réparties. En général, on peut constater qu'ils apportent trois types de réponses :

- L'uniformisation de la représentation des systèmes informatiques : Corba utilisant IDL, l'environnement java utilise dans son environnement un langage unique de programmation.
- Utilisation de bibliothèques de composants : Réutilisation du code existant tel que EJB, CCM (CORBA Component Model).
- L'échange d'information entre applications distantes indépendamment de leurs langages et plates-formes d'exécution : par exemple le IIOP de CORBA **[Plo01].**

Cependant, ces middlewares connaissent plusieurs limitations communes:

- Ces solutions sont monolithiques : chacune propose sa propre infrastructure.
- Le partage d'applications difficiles entre les différentes technologies.
- La notion d'annuaire ou moteur de recherche d'applications réparties est inexistante.
- Pas de découverte automatique des fonctionnalités implémentées.
- Couplage fort entre le client et l'application : Implique des contraintes sur les clients et donc des configurations spéciales.
- Solutions souvent pas standards ou standard avec limitation **[Duv04].**
- Manque de sémantique dans la description des interfaces des applications.
- N'offre aucun support permettant de raisonner sur des compositions dynamiques d'objets afin de construire des objets composites implémentant des fonctionnalités plus élaborées.
- Ne favorise pas les échanges et la collaboration business to business sur Internet.

2.3 Naissance des Services web

Les *Services Web* ont pour objectif de repousser voire de supprimer toutes les limitations des technologies de mise en œuvre de l'informatique répartie exprimées au point précédent et surtout d'apporter une réponse au problème d'interopérabilité des systèmes informatiques. Ils s'inscrivent donc dans la continuité de ces initiatives (CORBA, EJB/RMI, DCOM,..) en apportant toutefois une réponse plus simple et s'appuyant sur des technologies et des standards reconnus et maintenant acceptés par tous **[Via04].**
Les Services Web sont nés aussi de cette volonté de définir une infrastructure technique favorisant l'intégration d'applications inter- et intra-entreprise s'appuyant sur les technologies issues du monde de l'Internet afin d'en tirer le meilleur profit.

La particularité des services Web réside dans le fait qu'elle utilise la technologie Internet comme infrastructure pour la communication entre les composants logiciels (les services Web) et ceci en mettant en place un cadre de travail basé sur un ensemble de standards. Par contre, la mise en oeuvre des architectures citées auparavant soulève des difficultés dans le cadre d'une infrastructure ouverte telle que Internet. En effet, ces architectures, bien qu'utilisant un modèle objet distribué, proposent chacune sa propre infrastructure. Ce qui impose une forte liaison entre les services offerts par les composants et leurs clients **[Mel04]**.
Cette technologie des Services Web, initiée par IBM et Microsoft, puis en partie normalisée par le W3C est maintenant acceptée par l'ensemble des acteurs de l'industrie informatique sans exception. C'est surtout ce point qui fait des Services Web une technologie révolutionnaire.

Rôle de XML dans l'infrastructure Services Web :

Ayant fait le choix des protocoles du Web comme support de communication entre les composants, le modèle des services Web fonde sur XML la syntaxe et la sémantique d'un service. XML est aujourd'hui un standard qui permet de décrire des documents structurés transportables sur les protocoles communs d'Internet. XML constitue la technologie de base des architectures services web, il est un facteur important pour contourner les barrières techniques. En effet, il apporte à l'architecture l'extensibilité et la neutralité vis à vis des plates-formes et des langages de développement. De plus, grâce à la structuration, XML permet la distinction entre les données des applications et les données des protocoles permettant ainsi une correspondance facile entre les différents protocoles. L'interopérabilité entre les systèmes hétérogènes demande des mécanismes puissants de correspondance et de gestion des types de données des messages entre les fournisseurs et les clients. C'est une tâche où les schémas de type de données XML s'avèrent bien adaptés. Aujourd'hui, la technologie des services Web est essentiellement basée sur XML ainsi que les différentes spécifications qui tournent autour, les espaces de nom, les schémas XML, et les schémas de Type (XSD) **[Mel04].**

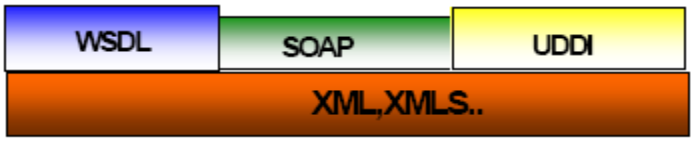


Fig 2.10: Rôle de XML dans le processus de standardisation

Le développement d'Internet par la démocratisation du haut débit, la structuration des données via XML et la recherche d'interopérabilité sont les trois points qui ont favorisé l'essor des services web.
Grâce à cette technologie des Services Web, nous passons d'un monde d'architectures fortement couplées à un monde d'architectures faiblement couplées où les applications et librairies communiquent par des protocoles, des schémas et des messages. De plus, l'architecture des Services web s'est imposée grâce à sa simplicité, sa lisibilité et ses fondations normalisées hérités du XML.

2.4. Architecture Orientée service (SOA) :

2.4.1. Principe de l'architecture :

L'une des dernières tendances en matière de logiciels est le concept d'Architecture Orientée Service (SOA). Les Architectures Orientées Service, comme leur nom l'indique, ne représentent pas une technologie mais une façon de concevoir et de déployer des applications. Plus précisément, il s'agit de structurer des projets selon une approche basée sur le principe de "services" et non plus, comme par le passé, sur la base d'applications. Plutôt que de privilégier une architecture applicative basées sur des contraintes techniques, l'architecture SOA propose de découper les fonctionnalités d'une application en services métier réutilisables dans d'autres applications **[Duv04]**.

Le concept de « service » est un concept clé de l'architecture orientée service. Il consiste en une fonction ou fonctionnalité bien définie. C'est aussi un composant autonome qui ne dépend d'aucun contexte ou service externe.
L'architecture des Services Web comme l'architecture du Web sont des instances d'architectures orientées services (SOA : Service Oriented Architecture). Elle propose une perspective globale sur le développement, la gestion et le fonctionnement des services Web **[Bar03]**.
L'architecture SOA est un modèle abstrait qui définit un système par un ensemble d'agents logiciels distribués qui fonctionnent de concert afin de réaliser une fonctionnalité globale préalablement établie **[Hea01]**. Les agents dans un système distribué n'opèrent pas dans un même environnement de traitement et sont obligés de communiquer par échange de messages afin de solliciter des services dans le but d'accomplir leur résultat souhaité. Le modèle d'architecture SOA représente un modèle d'architecture compatible avec la nature de la problématique et les objectifs visés par le projet Services Web **[Mel04]**.
Le choix d'une architecture SOA entre dans la perspective de transformer le Web en une énorme plate-forme de composants faiblement couplés et automatiquement intégrables. C'est un modèle qui promet interopérabilité, fléxibilité et réutilisabilité.

2.4.2 Architecture d'une application repartie selon SOA :

a) Architecture classique (3 couches) :

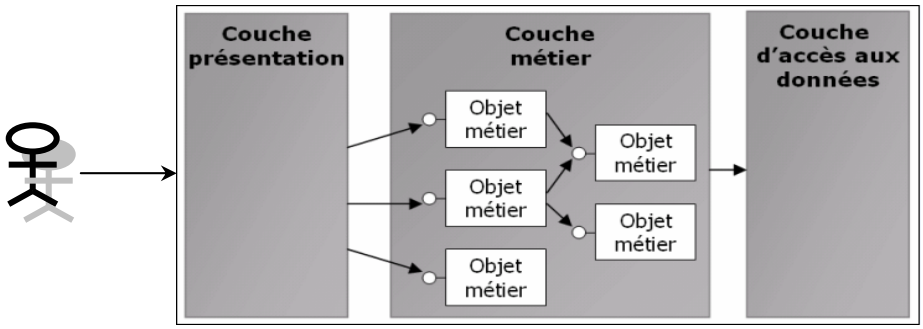


Fig 2.11: Architecture trois couches classiques

Couche présentation : Interface permettant l'interaction entre l'utilisateur et l'application, l'interface permet de manipuler et afficher les données en manipulant simplement les objets de la *couche métier*.
Couche métier : La *couche métier* contient toutes les règles de fonctionnement relatives à un métier. Par exemple pour une banque, la *couche métier* implémentera, entre autre, toutes les règles relatives à la gestion de comptes bancaires et de la clientèle.
La couche données : La *couche données* modifie les données, récupère les données et assure la sécurité et l'intégrité des données.

Le code client présent dans la couche présentation est très lié aux objets de la *couche métier*, ce qui a pour conséquence que la plupart des modifications effectuées dans la couche métier ont des répercussions dans la couche présentation et requiert un nombre d'appels important entre les deux couches. Ce nombre d'appels peut rapidement devenir problématique lors d'application répartie sur plusieurs sites distants.

b) Architecture orientée service :

Le schéma si-dessous montre que la notion de service permet d'élever l'abstraction par rapport à la logique métier et ainsi augmente la possibilité de réutilisation et d'évolutivité des objets métier car *la couche présentation* n'est plus directement liée à la logique métier. Dans une architecture orientée service, la *couche présentation* ne manipule plus directement les objets métiers, mais passe par des services. Les objets métiers se trouvent dans des bibliothèques de classes directement chargées dans le même processus que les services, le coût des appels aux objets métiers est alors très faible.

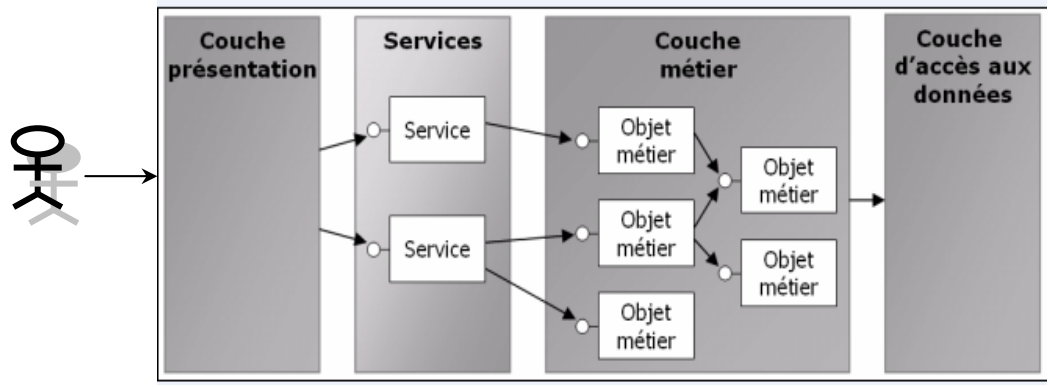


Fig 2.12: Architecture orientée services

Les services agissent comme des boites noires faisant abstraction de la complexité du modèle objet, présentant un ensemble de fonctionnalités restreints et permettant de réduire les échanges entre les couches.
En résumé: lors du développement d'applications sous forme de services, la logique métier et la capacité d'isoler la *couche métier* de la *couche présentation* sont une priorité **[Duv04]**.

2.4.3 Composants d'une SOA :

Une architecture de services est constituée de trois composants (voir fig 2.13). Le premier est le prestataire de services (le service réel). Vient ensuite le demandeur du service, autrement dit le composant qui accède au service. Enfin, l'agence de services fournit des services de découverte et d'enregistrement.

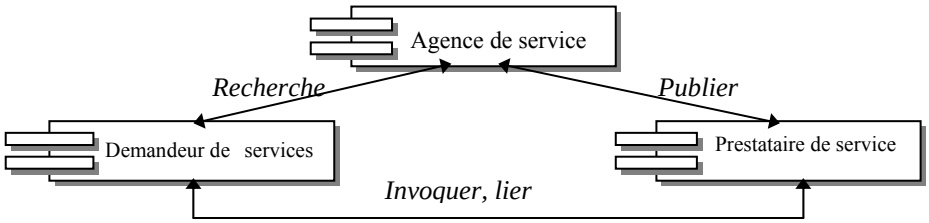


Fig 2.13: Les éléments d'un service Web.

Prestataire de services :
Un prestataire de services est la source de la fonctionnalité des services. Un prestataire publie un contrat d'interface utilisé par les demandeurs pour accéder au service. Le contrat définit ce que fait le service et comment y accéder.
Demandeur de services :
Le demandeur est le client d'un service. Il utilise l'agence pour découvrir quels services sont disponibles. Une fois un service localisé, le demandeur extrait le contrat d'interface correspondant de l'agence. Le client utilise le contrat d'interface pour comprendre comment (méthodes disponibles) et où (adresse) accéder au service.

Agence de services :
Une agence de services est un registre des services disponibles. Chaque prestataire publie son contrat d'interface à l'agence avec les informations à utiliser pour localiser le service. Le demandeur recherche les services appropriés dans l'agence et extrait le contrat d'interface correspondant.

La présentation des composants *SOA* ci-dessus nous permet d'affirmer que les *services Web* implémentent l'architecture orientée service. Le prestataire de services est le fournisseur du Service Web, le demandeur de service est le client du Service et l'agence de services est l'annuaire *UDDI*.

3. La technologie des Services Web :

Le paradigme des services Web repose sur une architecture par composants qui utilise des protocoles Internet comme infrastructure pour gérer la communication entre les composants. Elle offre un modèle à base de composants (les services Web) en ligne encapsulant une application logique derrière une interface interactive uniforme et standardisée. Le terme «services Web» est utilisé pour désigner aussi bien le paradigme que l'idée d'un composant accessible à partir du Web.

3.1. Définitions

La définition des Services Web divise clairement la vision académique et industrielle. Certains restreignent les composants Services Web à ceux qu'utilisent les standards de la technologie services Web tandis que d'autres proposent une définition plus générale. Pour cela, plusieurs définitions on été attribuées aux Services Web **[Mel04]**.

Définition 1 : "*Les services Web sont des applications auto-descriptives, modulaires et faiblement couplées qui fournissent un modèle simple de programmation et de déploiement d'applications, basé sur des normes, et s'exécutant au travers de l'infrastructure Internet. Les services Web réalisent des fonctions allant des simples requêtes aux processus métiers sophistiqués.*" **[Duv04]**

Définition 2 : *« Un Service Web est un composant implémenté dans n'importe quel langage, déployé sur n'importe quelle plate-forme et enveloppé dans une couche de standards dérivés du XML. Il doit pouvoir être découvert et invoqué dynamiquement par d'autres services».* **[Via04].**

Définition 3 : [CNW01] *« Un Service Web est une application accessible à partir du Web. Il utilise les protocoles Internet pour communiquer et utilise un langage standard pour décrire son interface».*
La définition des Services web ne serait pas complète si on évoquerait pas ses principaux standards : SOAP (Simple Object Access protocol) **[SOAP]**, WSDL (Web Service description Language) **[WSDL]** et UDDI (Universal Description, Discovery and Integration) **[UDDI]** mis en place par Microsoft, IBM, Sun…

Le XML (eXtensible Markup Language) est à la base de tous ces standards. Le fait que les Services web utilisent ce méta-langage de structuration de données leurs procurent l'avantage d'être non-propriétaire et réellement multi-plateforme. Il est donc recommandé de posséder un minimum de bases (XML, DTD, XML Schema, XSL) afin de mettre en place des Services web réellement optimisés. **[Via04].**

- **SOAP** (Simple Object Access protocol) : est un protocole d'échanges inter-application indépendant de toute plate-forme, basé sur le langage XML. Un appel de service SOAP est un flux ASCII encadré dans les balises XML et transporté par le protocole HTTP. La force de ce protocole réside dans son universalité et sa flexibilité.
- **WSDL** (Web Service description Language) : donne une description au format XML des Services web en précisant les méthodes pouvant être invoquées, leurs signature et leur point d'accès (URL, Port,…) . C'est en quelques sorte l'équivalent du langage IDL de la programmation distribuée CORBA.
- **UDDI** (Universal Description, Discovery and Integration): normalise une solution d'annuaire distribué des Services web, permettant à la fois la publication et l'exploration **[Via04]**.

3.2 Infrastructure de base des Services Web

L'architecture de référence des services web (voir la fig 2.14) s'articule autour des trois rôles suivants :

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

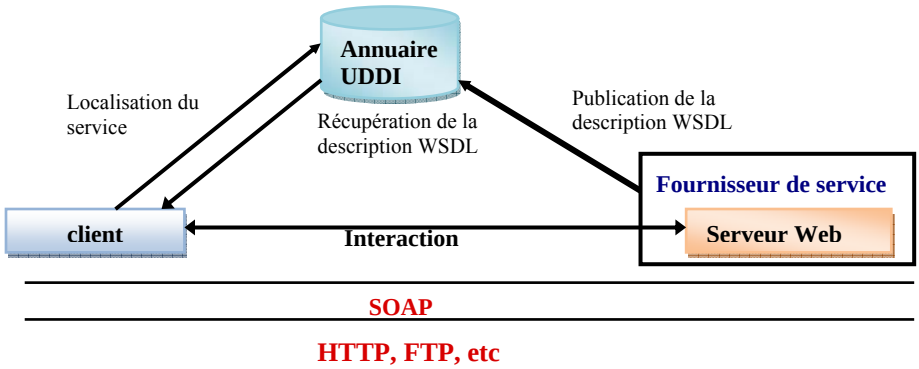


Fig 2.14: Les interactions de base des services Web

Les interactions de base entre ces trois rôles incluent les opérations de *publication*, de *recherche* et de *liens* (bind) d'opérations. Le fournisseur de services définit la

description de son service et la publie dans un annuaire de service. Le client utilise les facilités de recherche disponibles au niveau de l'annuaire pour retrouver et sélectionner un service donné. Il examine ensuite la description du service sélectionné pour récupérer les informations nécessaires lui permettant de se connecter au fournisseur du service et d'interagir avec l'implémentation du service considéré.

Pour garantir l'interopérabilité des trois opérations précédentes (publication, recherche et lien), des propositions de standards ont été élaborées pour chaque type d'interactions. Nous citons, notamment les standards émergents suivants que nous détaillons par la suite :

- SOAP définit un protocole de transmission de messages basé sur XML.
- WSDL introduit une grammaire commune pour la description des services.
- UDDI fournit l'infrastructure de base pour la publication et la découverte des services.

3.2.1 Communication : SOAP

La communication par message constitue un point crucial dans toute architecture SOA. Le standard actuel qui assure la messagerie est le protocole SOAP. Le protocole SOAP est une spécification XML qui définit un protocole léger d'échange de données structurées entre un réseau d'applications dans un environnement totalement distribué et hétérogène. Il est indépendant du contenu du message, à proprement parler, il laisse la responsabilité de l'interprétation aux couches de communication supérieures **[SOAP]**. Il se contente d'offrir la possibilité de structurer des messages destinés à des objectifs particuliers allant d'un simple échange de données jusqu'à l'appel de procédures à distance **[Mel04].**

Un message SOAP en cours de transit est composé de deux parties indépendantes :

- Une structure XML qui constitue le message SOAP : le message SOAP correspond à un document XML définie par une application cliente pour une application service suivant une structure particulière.
- Une entête du protocole de transport : le message SOAP est encapsulé dans un protocole de transport (HTTP sur la figure) en vue d'être livré à l'application destination. La figure 2.15 décrit la création, le transit et la consommation d'un message SOAP. La spécification SOAP est fondée sur trois parties : une syntaxe de structuration de message utilisant des techniques d'encodage, un modèle d'échange de messages et enfin un ensemble de modèles de transport basé sur les protocoles de transport applicatifs.

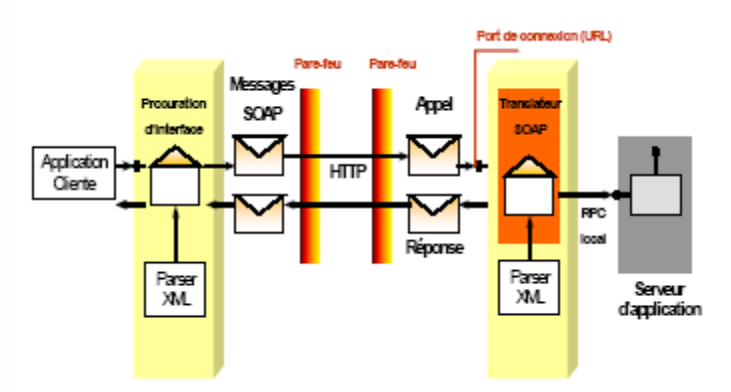


Fig 2.15: Un échange type de message SOAP

- Structure d'un message SOAP :

Un message SOAP est un document XML composé d'un élément *enveloppe*. L'élément *enveloppe* contient à son tour deux éléments fils, une entête (facultative) et un corps, appelés aussi respectivement extension et charge utile. L'entête du message obéit à un schéma spécifique et elle est destinée à contenir des informations nécessaires aux applications pour interpréter la charge utile. La charge utile est contenue dans un élément *body*. Elle obéit à une grammaire extensible. L'élément *body* peut contenir n'importe quelle structuration XML à condition qu'elle obéisse à une grammaire dont l'espace de noms est référencé **[Mel04].**

- **Le modèle d'échange de messages :**

Dans les systèmes distribués le choix du modèle d'échange de messages suscite toujours un débat opposant le modèle à sens unique au modèle requête/réponse. Le modèle à sens unique suppose l'absence de connexion et estime que chaque message doit comporter les informations sur son contexte (à savoir la session). Tandis que le modèle requête/réponse propose une approche orientée connexion permettant ainsi au message de se décharger des informations relatives au contexte applicatif. Étant conçue indépendamment du mécanisme du transport et du modèle d'échange, la spécification SOAP peut donc être utilisée par les deux modèles d'échange de messages. La spécification SOAP offre deux conventions pour codifier aussi bien le modèle de communication synchrone à travers le style *«RPC»* que le modèle asynchrone à travers le style *"document"*.

- **Le transport**

Une des mesures les plus importantes prises par les concepteurs du protocole SOAP est que la spécification du protocole ne comporte aucune indication sur le mécanisme de transport du message. En effet, SOAP fait une séparation totale entre le message (i.e. le document XML) et son moyen de transport. Il laisse aux développeurs le choix du moyen de transport. Ceci donne un certain nombre d'avantages :

1. La neutralité et l'extensibilité du protocole facilite la mise en place des spécifications pour plusieurs protocoles de transport existants ce qui assure une large portée pour le message SOAP.

2. SOAP est capable de s'adapter aux différents modèles d'échange de messages proposés par les protocoles de transport.
3. SOAP profite des services offerts par les protocole pour son transport tels que l'authentification, le chiffrement, le contrôle d'accès, la fiabilité, etc.

Actuellement, il existe des extensions de SOAP sur les protocoles de transport HTTP, SMTP ou encore FTP. Le transport sur HTTP c'est le protocole le plus utilisé pour le transport des messages SOAP. Le protocole HTTP constitue un excellent transport en raison de sa popularité sur le Web. Il existe deux règles à respecter pour pouvoir utiliser SOAP via HTTP :

1. Le mécanisme d'envoi doit utiliser la méthode HTTP POST standard.
2. La destination de la requête HTTP doit être adressée à une URI donnée sur le serveur Web.

Le bloc de données de la requête HTTP est composé du message SOAP lui-même (par conséquent le type MIME doit être *text/xml*) et d'un attribut *SOAPAction*. Ce dernier est destiné à indiquer au serveur que le message HTTP transporte un message SOAP. La partie du code suivante représente un message SOAP en cours de transit sur HTTP. Le message est une requête (formatée RPC) pour une opération d'addition de deux entiers.

```
POST /serveurMath.asp HTTP/1.1
Content-Type :text/xml
Content-Length : ###
SOAPAction : "urn :mathTest""
<soap          :Envelope          xmlns          :soap="          http
://schemas.xmlsoap.org/soap/envelope"
soap :encodinStyle=" http ://schemas.xmlsoap.org/soap/encoding">
<soap :body>
<math :add xmlns :math=" http ://exemple.com/math"/>
<math :op1> 25 </math :op1>
<math :op2>100</math :op2>
</math :add>
</soap :body>
</soap :Envelope>
```

Le protocole SOAP n'est pas concerné par la nature du programme cible ou le traitement des messages, il ne définit pas un environnement d'exécution mais plutôt un modèle de liaison qui assure l'interopérabilité au niveau applicatif entre des applications hétérogènes. La vision des services Web s'est concrétisée avec l'apparition de SOAP, en permettant à deux applications appartenant à des environnements technologiques hétérogènes d'échanger des données structurées.

3.2.2 La description de services WSDL :

Le protocole SOAP met à la disposition des services Web un moyen standard de structuration et d'échange de messages XML. Il ne fournit aucune indication sur la structure que le message doit respecter vis à vis du Service Web sollicité. C'est toujours dans le but de rendre les services Web faiblement couplés et autonomes, que la spécification WSDL a vu le jour **[WSDL].** Contrairement aux architectures monolithiques où la description des composants ainsi que les moyens de les invoquer

dépendent fortement de l'infrastructure utilisée, la spécification WSDL offre une grammaire qui décrit l'interface des composants services Web de telle façon qu'ils se suffisent à eux-mêmes.
Ariba, IBM et Microsoft sont à l'origine de la spécification du langage WSDL. Cette spécification présente les services comme des boites noires, présentant l'apparence d'un certain nombre de ports d'entrée et de sortie. Ces ports sont mis bout à bout, reliés par un protocole de transport (figure 2.16). La spécification du service est composée de deux parties : une définition abstraite des services en terme de messages échangés entre les différents types de port et la définition des mécanismes de liaison entre les définitions abstraites et un ensemble de techniques de déploiement (généralement des protocoles Internet).
Comme le montre la figure 2.16, une description WSDL est un document XML dont l'élément racine est une *définition*. Chaque document définit un service comme une collection de points finals ou *ports*. Chaque *port* est associé à une liaison spécifique qui définit la manière avec laquelle les messages seront échangés. Chaque liaison établit une correspondance entre un protocole et un *type de port* **[Mel04].**

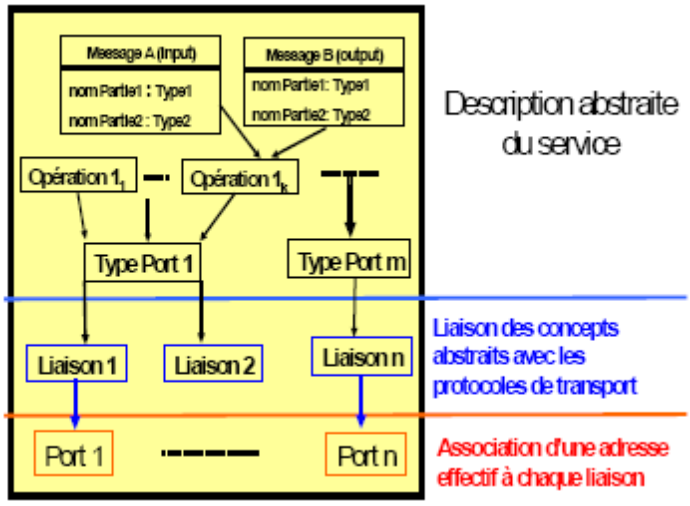


Fig 2.16: Les éléments constitutifs du langage WSDL

Un *type de port* se compose d'une ou plusieurs opérations qui représentent une définition abstraite des capacités fonctionnelles du service. Chaque opération est définie en fonction des messages échangés au cours de son invocation. La structure du message est définie par des éléments XML associés à un schéma de type spécifique.
La spécification WSDL joue un rôle important dans l'interopérabilité des composants services Web. Moyennant un schéma uniforme obéissant à une sémantique bien définie, elle permet aux composants de définir ce qui est nécessaire à leur invocation. La spécification WSDL est définie selon une sémantique totalement indépendante du modèle de programmation de l'application. Elle sépare clairement la définition abstraite du service (échange de messages) de ses mécanismes de liaison (définition des protocoles applicatifs). Cette dernière caractéristique permet au composant d'interagir même si l'application a été modifiée ce qui est un point important pour assurer l'interopérabilité des services. La complétude de la spécification WSDL permet

l'automatisation du processus d'invocation. En effet, elle contient toutes les informations nécessaires pour la mise au point d'un ensemble d'API qui génèrent automatiquement un client pour le service **[Mel04].**

2.2.3 Service recherche et publication UDDI :

Les deux standards exposés précédemment définissent ensemble l'aspect le plus basique de développement de l'infrastructure des services Web. Toutefois, dans un environnement ouvert comme Internet, le modèle de description des services Web n'est d'aucune utilité s'il n'existe pas un moyen de localiser aussi bien les services que leurs descriptions WSDL.
Un troisième standard appelé UDDI **[UDDI]** (semblable au registre CORBA) a été conçu pour réduire l'écart entre les applications clientes et les services Web et ainsi d'automatiser la communication entre prestataires et clients.
UDDI offre deux fonctionnalités au sein de l'architecture globale qui permettent aux fournisseurs de publier leurs services selon un modèle de description et au client l'interrogation des services. La spécification UDDI constitue une norme pour les annuaires des services Web. Les fournisseurs disposent d'un schéma de description permettant de publier des données concernant leurs activités, la liste des services qu'ils offrent et les détails techniques sur chaque service. Elle offre aussi une API aux applications clientes, pour consulter et extraire des données concernant un service et/ou son fournisseur.

Les registres UDDI sont des services Web qui offrent deux fonctionnalités de base : la publication des différents types d'information sur les services et leurs fournisseurs selon un schéma de description, et la consultation du contenu des registres. La mise en place d'un registre UDDI suit un processus uniforme imposé par la spécification. Chaque organisation qui veut mettre en place un registre UDDI doit suivre ce processus pour devenir un *opérateur UDDI*.

On peut voir UDDI comme un annuaire distribué des services qui stocke trois sortes de données : des données concernant les fournisseurs de services appelées pages blanches, des données concernant l'activité ou le service métier des fournisseurs appelées pages jaunes et les données techniques de chaque service publié qui constituent les pages vertes. La spécification UDDI offre un schéma qui structure d'une manière uniforme les différents types concernant les trois niveaux de description.

- **Structure du registre UDDI :**

En fait, le schéma sommaire de la structure du registre UDDI décrit la hiérarchie suivante (voir le schéma ci-dessous):

***businessEntity* :** Cette structure décrit l'entreprise, une ou plusieurs catégories auxquelles elle appartient (il existe plusieurs taxonomies), une description et l'adresse électronique des contacts qu'elle met à la disposition des clients pour assurer une assistance ou fournir des renseignements. Elle est en relation avec une ou plusieurs définitions des services ("businessService") fournis par l'entreprise. Les champs

"categoryBag" et "identifierBag" permettent de' distinguer les entreprises selon certains critères et permettent de faciliter la recherche dans le registre UDDI.

businessService : Cette structure est décrite au moyen d'une entrée "serviceKey" identifiant le service publié par une entité d'affaires "businessEntity". Elle fait référence à une catégorie de service pour permettre une recherche en fonction d'un type de service particulier. Enfin elle pointe vers une liste de liens ("bindingTemplate") décrivant les différents accès au service.

bindingTemplate : Une entreprise peut proposer ses services par différents moyens de communication ou protocoles. Pour chacune des méthodes, elle présente un enregistrements de type bindingTemplate. Ces derniers pointent vers une structures "tModel" et vers un points d'accès au service sur la web via http, FTP, SMTP…

tModel : Cette dernière structure permet de mettre en relation les trois précédentes avec des classifications préétablies. C'est un index utilisé pour classer les "businessEntity", les "businessService" et les "bindingTemplate". Cet index permet de rechercher les services en rapport avec des schémas de classification (taxonomies) et pointe vers un URL dans lequel se trouve une description ou des descriptions particulières concernant un service.

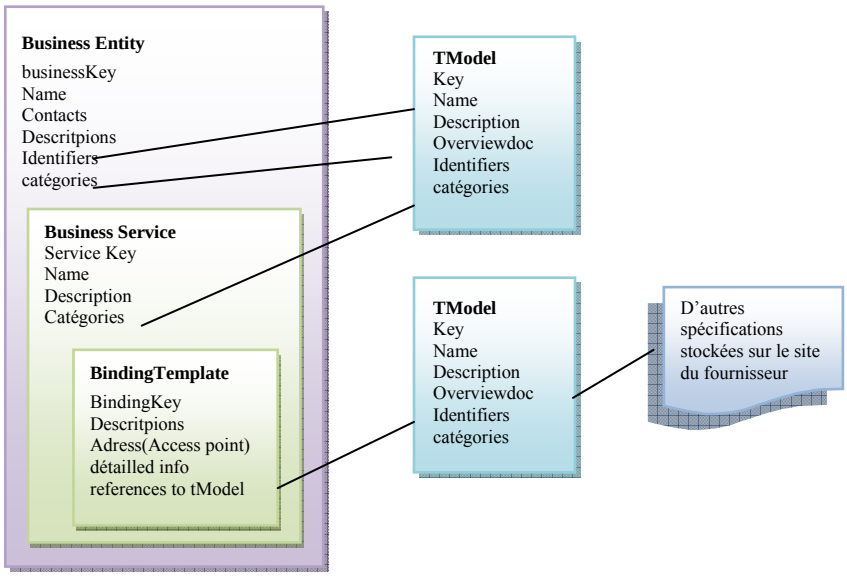


Fig 2.17: Structure générale du registre UDDI

Actuellement, SOAP, WSDL et UDDI sont les trois standards qui constituent l'architecture services Web. Ensemble, ils résolvent les problèmes de l'hétérogénéité des systèmes pour l'intégration d'applications en ligne **[Mel04]**. Toutefois, il existe d'autres aspects essentiels à mettre en œuvre, c'est ce que nous essayons d'expliquer dans la partie suivante.

3.3 Infrastructure étendue des Services web :

L'infrastructure de base autour des standards SOAP, UDDI et WSDL répond aux problèmes d'interopérabilité et d'intégration technique des applications. Cependant, cette infrastructure n'est pas suffisante pour permettre une utilisation effective des services web dans les domaines dont les exigences vont au-delà de la capacité

d'interactions simples via des protocoles standards. Par exemple, dans le domaine du e-business, cette utilisation est motivée par les possibilités de coopération et de coordination entre des entreprises telles qu'on peut les percevoir dans la mise en oeuvre de la gestion d'une chaîne logistique (eSCM) ou celle de la gestion des relations clients (eCRM). Le challenge est alors d'être capable de spécifier et de mettre en oeuvre des business processes intra ou inter entreprises. Ceci pose donc fondamentalement un problème d'intégration fonctionnelle des activités d'entreprises qui dépasse la simple capacité d'interactions via des protocoles standard. Il semble donc nécessaire de pouvoir organiser les processus métier inter et intra-entreprises. Il s'agit en d'autres termes d'organiser les Services web entre eux, de les composer, les orchestrer ou les coordonner. De nombreuses solutions de modélisation des processus métier s'appuient sur la notion de *workflow.*

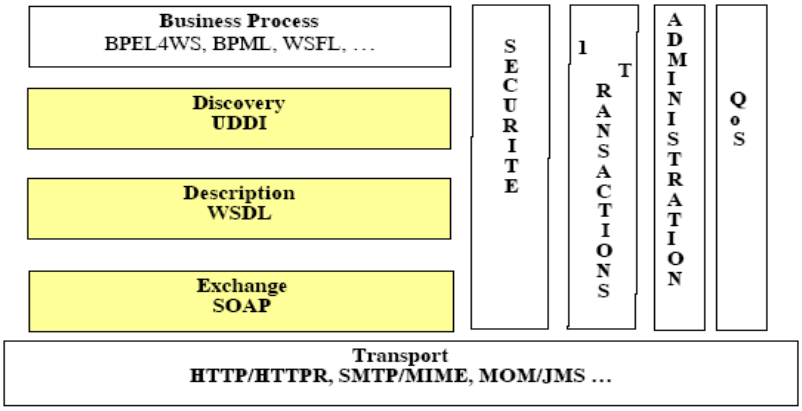


Fig 2.18: Pile des services web

Une architecture étendue est constituée de plusieurs couches se superposant les unes sur les autres, d'où le nom de *pile des services web*. La figure 2.18 décrit un exemple d'une telle pile. La pile est constituée de plusieurs couches, chaque couche s'appuyant sur un standard particulier. On retrouve, au-dessus de la couche de transport, les trois couches formant l'infrastructure de base décrite précédemment. Ces couches s'appuient sur les standards émergents SOAP, WSDL et UDDI.

Comme mentionné précédemment, l'infrastructure de base définit les fondements techniques permettant de rendre les business processes accessibles à l'intérieur d'une entreprise et au-delà même des frontières d'une entreprise. Dans ce contexte deux types de couches permettent de la compléter : *(i)* les couches dites transversales **[Got02]** (*e.g.*, sécurité, administration, transactions et qualité de services (QoS)) rendent viable l'utilisation effective des services web dans le monde industriel ; (ii) une couche Business process permet l'utilisation effective des services web dans le domaine du e-business. Dans la littérature, la couche Business Process relève les problèmes sous-jacents suivants:

- Comment les business processes peuvent-ils être représentés comme des services web ?
- Nécessité de décrire comment les services web sont utilisés pour implanter les activités d'un business process.

- Les problèmes de composition de service, c'est à dire, quel(s) partenaire(s) va (vont) exécuter quelle(s) partie(s) d'un business process ?

Différents auteurs de la communauté de recherche s'accordent sur la nécessité de spécifier le comportement *externe* de chaque partie impliquée dans le protocole d'intégration de processus (partie publique) métier sans pour autant révéler leurs implémentations *internes* (partie privée). Deux raisons justifient cette séparation :
1- Les entreprises ne tiennent pas forcément à révéler leurs prises de décisions internes et souhaitent préserver la confidentialité de leurs données.
2- La séparation publique-privé permet de modifier la partie privée indépendamment de la partie publique.

A cet effet, différents langages ont été proposés dans le but de décrire le processus public d'un service (*e.g.,* WSCL) ou la spécification, de manière procédurale, de la composition de services (*e.g.,* BPML **[BPML]**, BPEL4WS **[BPML4WS]**.

3.4 Quelques outils et plate-formes de développement des services web :

a) **Plates-formes commerciales :** les logiciels de déploiement des services web sont des produits d'éditeurs de logiciels reconnus : Microsoft, IBM et Sun. Nous avons restreint les éditeurs au plus importants car le concept de service web atteint aujourd'hui de nombreux acteurs du monde informatique.
   - Visual Studio .NET: de Microsoft
   - Web Sphere Studio Application Developper de IBM
   - Sun ONE developper Studio de Sun

b) **Plates-formes publiques :** l'utilisation des services web étant en plein expansion l'éditeur SUN et la communauté de logiciels Apache mettent à la disposition d'utilisateurs potentiels des outils publics dont les plus importants sont :
   - Java WSDP (Web Service developper Pack) de Sun
   - APACHE SOAP : de la famille APACHE
   - Axis APACHE Extensible intercation System

Tous ces produits permettant le déploiement l'intégralité des technologies de base des services web.

3.5 Les limites des services web

Malgré l'apport des services web quant au besoin de l'interopérabilité, ils présentent tout de même quelques inconvénients:

- Les normes de services Web dans les domaines de la sécurité et des transactions sont actuellement dans leur petite enfance comparée à des normes ouvertes plus mûres de l'informatique répartie telles que CORBA.
- Les services Web souffrent de performances faibles comparées à d'autres approches de l'informatique répartie telles que le RMI, CORBA, ou DCOM.

- Par l'utilisation du protocole HTTP, les services Web peuvent contourner les mesures de sécurité mises en place au travers des *firewalls*.
- Rien ne permet pour l'instant d'assurer la qualité d'exécution d'un Service Web. la qualité de service (QdS) n'est pas incluses dans l'architecture de base des services web **[Wik06].**
- La description purement syntaxique des services web en utilisant le langage XML n'est pas suffisant pour permettre une automatisation efficace de plusieurs tâches liées à leur cycle de vie tel que : leur découverte dynamique, leur composition en des services plus complexes, le suivi de leur exécution…Ce besoin constitue particulièrement le point de convergence des deux domaines du web : les services web et le web sémantique.

4. Les Services Web sémantiques [Kel03]

L'objectif ultime de l'approche services web est de transformer le Web en un dispositif distribué de calcul où les programmes (services) peuvent interagir de manière *intelligente* en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes. En d'autres termes, l'idée poursuivie avec les services web, est de mieux exploiter les technologies de l'Internet en substituant, autant que possible, les humains qui réalisent actuellement un certain nombre de tâches, par des machines en vue de permettre une découverte et/ou une composition automatique de services sur l'Internet. L'automatisation est donc un concept clé qui doit être présent à chaque étape du processus de conception et de mise en oeuvre des services web.

En effet, l'utilisation intensive des standards, d'une part, et celle du langage XML, d'autre part, constituent les deux caractéristiques fondamentales de la technologie des services web. Par conséquent, les différentes spécifications d'un service web peuvent être facilement traitées par une machine. Par exemple, une description WSDL peut être automatiquement analysée par un parseur XML pour générer un proxy client (*i.e.,* le code du stub client) facilitant ainsi la tâche du programmeur lors de l'implantation de son application. Cependant, le consensus sur la *mécanique des interactions* (*e.g.,* format des messages, types des données et protocoles d'échanges) n'est pas suffisant pour permettre aux services web d'interagir automatiquement de manière claire et non ambiguë. Par exemple, deux descriptions XML identiques peuvent avoir des significations différentes selon le contexte **[Pao02]**. L'absence d'une sémantique explicite limite les possibilités d'automatisation des services web. Pour pallier cette limitation, il y a clairement un besoin de langages de descriptions des services qui permettent de conférer une signification explicite et non ambiguë aux descriptions des services web. Il semble donc nécessaire de tendre vers des services intelligibles pour des machines : c'est le concept de « *Service web sémantique* ».
Une définition qui semble être la plus consensuelle est celle de Li et Horions : "Semantic Web Services aim to describe and implement Web services so has to make them more accessible to automated agents. Here, ontologies can be used to describe services so that agents (both human and automated) can advertise and discover services

according to a semantic specification of functionality (as well as other parameters such as cost, security, etc.)" **[Hub].**

La réalisation de l'approche service web sémantique rejoint les préoccupations à l'origine du Web sémantique, à savoir comment décrire formellement les connaissances de manière à les rendre exploitables par des machines. En conséquence, les technologies et les outils développés dans le contexte du Web sémantique peuvent compléter la technologie des services web en vue d'apporter des réponses crédibles au problème de l'automatisation. La notion d'ontologie peut jouer un rôle prépondérant pour permettre d'expliciter la sémantique des services facilitant ainsi les communications hommes-machines, d'une part, et les communications machines-machines, d'autre part.

WSDL est le langage de description des interfaces des Services web le plus émergent qui a été standardisé au sein du consortium W3C. Cependant, ce langage, se basant uniquement sur XML, permet une spécification des services web du point de vue syntaxique. Il ne possède pas une puissance d'expressivité suffisante pour permettre à un agent logiciel de comprendre la sémantique d'un service web en terme des ses capacités fonctionnelles (qu'est ce qu'il est capable de faire). Ce manque remet en cause la dynamicité des interactions ou des collaborations entre les différents services web et rend difficile la recherche précise de services appropriés.

Pour cela la communauté web sémantique a pensé de développer un langage ontologique pour les services web s'inspirant de OWL (ou DAML-OIL précedemment) appelé OWL-S ou DAML-S précedemment **[DAML-Sa] [OWL-S].**

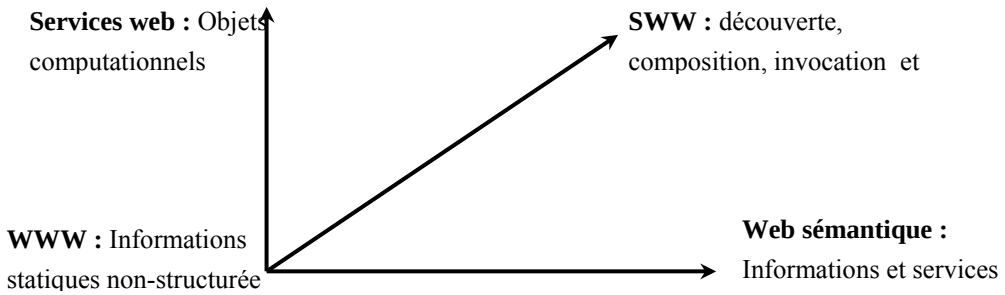


Fig 2.19 : La convergence du web sémantique et les services web [Dav04]

On peut constater que les deux principales influences ayant conduit au développement des services Web sémantiques sont d'une part, les services Web, qui peuvent être vus comme la partie transport et d'une autre part les langages du Web sémantique qui permettent de définir le monde réel au travers de formalismes logiques s'appuyant sur XML.

OWL-S (ou DAML-S) respecte une syntaxe XML et une sémantique de OWL (DAML-OIL respectivement) qui est lui même inspiré de la logique de description. Il permet de fournir à l'agent logiciel deux types de connaissances concernant un service web **[DAML-Sb]**.

1. Qu'est ce que le service a besoin de l'agent et qu'est ce qu'il peut lui offrir ? c'est la classe PROFILE qui permet de répondre à cette question en décrivant les capacités et les paramètres à fournir au services pour q'il puisse travailler.
2. Comment il travaille ? la réponse à cette question est donnée par la classe MODEL. Elle décrit le Workflow et les chemins d'exécution possible d'un service web.

Cette spécification de DAML-S (OWL-S) permet particulièrement d'automatiser les tâches suivantes :

- Découverte dynamique de services Web : c'est la possibilité de localiser automatiquement un service web qui répond à des besoins particuliers.
- Invocation de services Web :
- Composition et inter opération de services Web : la possibilité de combiner plusieurs services web afin de créer une nouvelle fonctionnalité.
- Moniteur d'exécution de services Web :

Pour l'instant, OWL-S (DAML-S) est un langage qui est encore en cours de spécification, mais dont les grandes lignes sont déjà tracées. Un moyen de l'interfacer avec WSDL a été proposé afin de pallier son absence de gestion d'échange de messages, ce qui permettra par exemple d'utiliser SOAP pour échanger des messages XML. OWL-S pourra alors être réservé à une description abstraite et sémantique des services, permettant également d'exprimer des contraintes sur les paramètres et d'utiliser des constructeurs (comme « si…alors…sinon… »).
De plus, l'utilisation de OWL-S (ou DAML-S) des logiques de descriptions pour modéliser les services permet une grande puissance d'expression, que ne possèdent pas les autres systèmes.
Nous revenons dans les parties qui suivent plus en détail sur DAML-S (OWL-S).

Il existe d'autres approches qui ont conduit le développement des cadres sémantiques des services Web, citons IRS-II (Internet Reasoning Service) **[IRS]** et WSMF(Web Service Modelling Framework) **[WSMF]**. Cependant, l'approche OWL-S (DAML-S) est celle qui est sur le point de standardisation par le W3C.

De manière générale, l'objectif visé par la notion de services web sémantiques est de créer un Web sémantique de services dont les propriétés, les capacités, les interfaces et les effets sont décrits de manière non ambiguë et exploitable par des machines et ce en utilisant les couches techniques sans pour autant en être conceptuellement dépendants. La sémantique ainsi exprimée permettra l'automatisation des fonctionnalités suivantes qui sont nécessaires pour une collaboration et une interaction inter-entreprises efficace:

- Processus de description et de publication des services.
- Découverte des services.
- Sélection des services.
- Composition des services.
- Fourniture et administration des services.
- Négociation des contrats.

5. Conclusion

Le paradigme émergeant des Services web représente une nouvelle avancée dans l'ère Internet. Ils s'inscrivent dans la continuité des technologies de mise en œuvre de l'informatique repartie. Leur atout majeur est de permettre l'interopérabilité entre les différentes applications en utilisant Internet comme infrastructure pour la communication. Par contre, les autres technologies telles que CORBA, J2EE, DCOM soulèvent des difficultés dans le cadre d'une infrastructure ouverte et dynamique telle que Internet. Ces approches ont produit des systèmes monolithiques et fortement couplés. Ils nécessitent de conclure des accords et de partager les contextes entre les différents systèmes des organisations qui interagissent. Ce type de collaboration statique n'est pas adapté aux environnements reposant sur le Web qui présente un caractère plus éphémère et dynamique.

L'architecture du Web est une instance d'architecture orientée services (SOA : Service Oriented Architecture). SOA propose une perspective globale sur le développement, la gestion et le fonctionnement des services Web. Grâce à ces derniers, nous passons d'un monde basé sur des architectures fortement couplées à un monde d'architectures faiblement couplées où les applications et librairies communiquent par des protocoles, des schémas et des messages standardisés. Les différents standards SOAP, WSDL et UDDI ont défini les interactions de bases entre les services web notamment l'échange de messages, la description des interfaces, la recherche et la publication des services. Ces trois standards reposant essentiellement sur le protocole HTTP et le langage XML ont permis d'apporter une réponse plus simple aux problèmes d'interopérabilité et d'intégration technique des applications. Cependant cette infrastructure s'est avérée insuffisante dans les domaines dont les exigences vont au-delà de la capacité d'interactions simples via des protocoles standards. D'autres couches faisant l'objet d'une architecture étendue ont été donc proposées concernant l'organisation et l'orchestration des services web au sein d'un processus complexe ainsi que la gestion de la sécurité, les transactions et la qualité de services.

Malgré tout ce qui a été cité, les services web présentent tout de même quelques inconvénients concernant leur maturité et leur performance comparée à d'autres approches de l'informatique répartie telles que le RMI, CORBA ou DCOM.

Mais l'handicap principal rencontré est d'un tout autre ordre. Il s'agit comment permettre aux services web d'interagir automatiquement de manière efficace et non ambiguë. L'absence d'une sémantique explicite dans la description des services web limite considérablement les possibilités de leur automatisation. C'est là où se situe la convergence du domaine du web sémantique avec la technologie des services web. DAML-S (plus récemment OWL-S) est le langage ontologique proposé donc par la communauté web sémantique pour la description explicite et non-ambiguë des fonctionnalités des services web. Ce langage devra permettre essentiellement

d'automatiser certaines tâches telles que : la découverte, la sélection et la composition des services web.

Dans le chapitre suivant, nous allons nous intéresser particulièrement à la problématique de composition des services web et les différents formalismes et approches proposées dans ce cadre. Nous allons énumérer aussi quelques difficultés rencontrées par les différentes communautés qui s'y intéressent pour l'automatisation efficace du processus de composition.

Chapitre 3 : La composition des services web

1. Introduction

La composition des services web constitue un axe de recherche qui suscite beaucoup d'intérêts de la part de chercheurs de communautés très variées telles que : Workflow, génie logiciel, représentation des connaissances, bases de données, multi-agent …etc
L'objectif de la composition de service est de créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services existants, composés ou non en vue d'apporter une valeur ajoutée **[Kel03]**.

Les services Web, tels qu'ils sont présentés, sont conceptuellement limités à des fonctionnalités relativement simples qui sont modélisées par une collection d'opérations. Toutefois, pour certains types d'application, il est nécessaire de combiner un ensemble de services Web (services Web basiques) en services Web plus élaborés (services composites) afin de répondre à des exigences plus complexes **[MEL04]** .

Particulièrement, s'il n'existe pas un service web qui pourra satisfaire la fonctionnalité requise par le demandeur, il doit y avoir une possibilité de combiner des services existants afin de répondre au besoin du demandeur. Avec la prolifération des services web, il est devenu très difficile de trouver le service spécifique pourrant effectuer la tâche voulue avec les préférences et les contraintes désirées. Il est devenu encore plus difficile lorsqu'il n'existe pas un service capable seul à satisfaire le but de l'utilisateur, mais plutôt une combinaison de services disponibles. Ajoutant à cela, le besoin de composer les services web existants en services plus complexes s'est accentué simplement parce que de nouvelles et de meilleures solutions peuvent être atteintes.

2. Problématique de la composition

2.1 Définition

La composition des services web se réfère au processus de création d'un *service composite* offrant une nouvelle fonctionnalité, à partir de services web existants plus simples, par le processus de découverte dynamique, d'intégration et d'exécution de ces services dans un ordre bien défini afin de satisfaire un besoin bien déterminé **[Chak02]**. En d'autre terme, étant donnée une spécification de haut niveau des objectifs d'une tâche particulière, la composition de services implique la capacité de découvrir, de sélectionner, de composer et de faire interopérer des services existants en vue de créer un service composite offrant une nouvelle fonctionnalité satisfaisant l'objectif visé **[Kel03]**.

Un service composite appelé aussi service agrégé **[MEL04]** est défini comme une conglomération de services externalisés (outsourced) travaillant en tandem pour offrir un service à valeur ajoutée. Les services atomiques externalisés sont appelés les *services*

participants ou *services partenaires* **[Med03].** Ces derniers peuvent appartenir à des organisations différentes, c'est pour cela, certains les appellent des services *externalisés (outsourced services).*

On distingue dès lors deux types de services Web : les services Web *basiques (atomiques)* et les services Web *composites*. Les services Web basiques désignent des services Web qui peuvent être modélisés par une collection d'opérations. L'invocation de ces opérations n'excède pas un simple échange synchrone ou asynchrone de deux messages. Dans ce type de service, il n'existe pas un ordre d'invocation des opérations, ils peuvent accepter la sollicitation de n'importe quelle opération selon l'ordre voulu par l'application cliente. Par contre, les services Web composites sont d'une nature plus complexe. Ils sont dotés d'un comportement défini par une structuration des séquences de messages échangés. Pour ce type de services, l'interaction n'a aucun sens si elle ne respecte pas l'ordre imposé par la logique de l'application **[MEL04]**. Par conséquent, La composition implique l'implémentation d'une logique métier définie par le client (utilisateur final ou application) ou dirigée par le but voulu à travers la combinaison de plusieurs services web.

Un exemple d'un service complexe est le service de planification de voyage. En effet, nous pouvons modéliser le service de planification de voyage (séjour) d'une agence de voyages comme la composition de plusieurs Services web atomiques (basiques) appartenant à des organisations différentes : réservation de vol, réservation d'hôtel et allocation de véhicule. La combinaison de ces trois services constituera le service composite «Organisation d'un voyage ».

Citons un autre exemple plus simple: supposant qu'on dispose de deux services web, un traducteur et un dictionnaire en ligne. Le premier traduit un texte entre différentes paires de langages et le deuxième donne la signification des mots en Anglais. L'utilisateur a besoin d'un dictionnaire français, or aucun des deux services disponibles n'est à la mesure de satisfaire ce besoin. Cependant, la combinaison des deux pourra le satisfaire de la façon suivante : l'entrée, qui est un mot en Français, sera d'abords traduit en un mot en Anglais en utilisant le traducteur, puis cette sortie (qui est un mot en Anglais) insérée comme une entrée au dictionnaire qui fournira la signification en Anglais et enfin cette signification est retournée vers le traducteur qui produira la signification en Français, voir le schéma suivant :

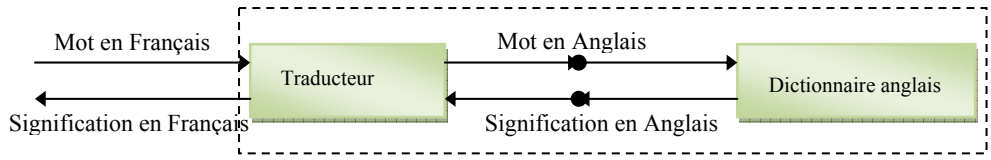


Fig 3.1 : Dictionnaire français comme service composite

Généralement, un système de composition comprend principalement trois types d'acteurs : le *fournisseur*, le *composeur* et le *demandeur* (consommateur). Le fournisseur est l'entité qui offre les services web simples pour l'utilisateur. Cette entité est responsable de leur description et leur publication sur le réseau. Le composeur est

l'entité responsable de créer le service composite. Une fois généré, sa description doit être publiée dans le registre UDDI **[UDDI]** ou dans un autre endroit de telle sorte a ce qu'il pourra être découvert de façon automatique. Le consommateur pourra être l'utilisateur final ou autre service web voulant invoquer le service composite **[Med03].**

2.2 Quelques sources de complexité

Le processus de création du service composite s'effectuait de manière manuelle et ad-hoc, ce qui était très dur à réaliser et demandant beaucoup de temps et des efforts considérables de programmation de bas niveau. Par conséquent, il est au-delà des capacités de l'être humain de traiter tout le processus de façon manuelle. Le besoin d'automatiser le processus de composition a déclenché un nombre considérable d'efforts de recherche et de développement aussi bien académiques qu'industriels. Mais malgré tous ces efforts d'automatisation, la composition des Services web reste une tâche hautement complexe et pose un certain nombre de défis. Sa complexité provient généralement des sources suivantes :

- Premièrement, l'augmentation dramatique du nombre des services web sur le web rend très difficile la recherche et la sélection des services web pouvant répondre à un besoin donné. Cette caractéristique pose particulièrement le problème de scalabilité.
- Les services web évoluent dans un environnement hautement dynamique et volatil, ils sont crées et mis à jour de façon irrégulière. Ainsi, une approche de composition doit tenir compte de ce comportement et doit détecter toutes les modifications survenues au moment de l'exécution et prendre des décisions avec toutes récentes informations. Cette caractéristique pose le problème de disponibilité des services participants au moment de l'exécution du service composite.
- Les services web sont d'habitude développés par différentes organisations qui utilisent différents modèles conceptuels pour décrire les caractéristiques des services web. Cela nécessite une technique d'interopérabilité sémantique pour vérifier leur compatibilité de composition et d'interaction en général **[Rao204]**.
- La modularité constitue une caractéristique importante des services Web, par conséquent, les services Web composites doivent être considérés récursivement comme des services Web et doivent garder les mêmes caractéristiques que les services Web basiques (services WSDL) à savoir auto-descriptifs, interopérables, et facilement intégrables **[MEL04]**.

2.3 Architecture générale d'un système de composition des Services web [Rao204]

Dans ce qui suit, nous allons proposer un cadre général pour la composition automatique des services web, sans considérer un langage, une plate-forme ou un algorithme particulier utilisé pour la génération du processus de composition. Notre but et de donner quelques bases pour pouvoir discuter les similarités et les différences entre les méthodes de composition proposées par les différentes communautés de recherches.

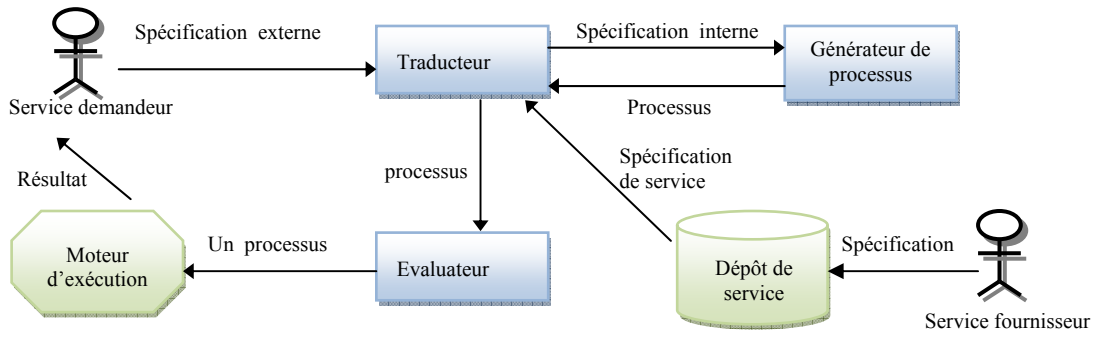


Fig 3.2: Architecture générale d'un système de composition de service

L'architecture générale d'un système de composition est illustrée par la figure si dessus. Elle comprend principalement trois types de participants (acteurs): le fournisseur, le consommateur et le composeur. Vu que la composition est une tâche très complexe, le composeur généralement se divise en d'autres sous rôles qui peuvent être les suivants : traducteur, générateur de processus et l'évaluateur, ajoutant à cela deux autres composantes au système de composition : le moteur d'exécution et le dépôt de services. Le traducteur traduit la description entre un langage externe utilisé par les participants et un langage interne utilisé par le générateur de processus. Ce dernier, pour chaque requête essaye de générer un plan combinant les services disponibles dans le dépôt de services qui pourront répondre au mieux aux besoins de la requête. Du moment que différents services web peuvent implémenter des fonctionnalités similaires, plusieurs plans de composition peuvent être ainsi générés. Ces plans seront soumis à l'évaluateur qui va sélectionner le meilleur plan pour l'exécution en se basant sur certaines contraintes et préférences du demandeur. Enfin, le moteur d'exécution, exécute le plan ainsi sélectionné et retourne le résultat au service demandeur.
Pour plus de précision, un processus de composition comprend les phases suivantes :

Présentation et publication des services atomiques: En tout premier lieu, les fournisseurs publient leurs services atomiques dans un endroit précis du web. Plusieurs langages sont disponibles pour la publication comme UDDI **[UDDI]** ou DAML-S ServiceProfile **[DAML-S]**. Les attributs essentiels pour décrire un service web comprend la signature, les états et les valeurs non-fonctionnelles.

- La signature fournit des informations concernant la transformation des données durant l'exécution du service web. Elle est représentée par les Entrées /Sorties (E/Ss)du service et les exceptions.
- Les états fournissent des informations concernant le changement d'état dans le monde dû à l'exécution du service web. Ils sont représentés par les Pré-conditions et les Post-conditions ou Effets (P/Es).
- Les valeurs non-fonctionnelles sont des attributs utilisés pour l'évaluation du service tels que le coût, la qualité de service, la sécurité...etc

Traduction entre le langage de description interne et externe : La majorité des systèmes de composition distingue entre les langages de spécification internes et externes des services web. Un langage externe est utilisé par les utilisateurs du service pour accroître leur accessibilité dans le sens où ils peuvent exprimer d'une manière

relativement simple qu'est ce qu'ils veulent ou qu'est ce qu'ils peuvent offrir. Par contre, un langage interne est utilisé par le générateur du processus de composition, parce qu'un générateur de processus nécessite un langage plus formel et plus précis, par exemple les langages de programmation logique.

Génération du modèle de processus de composition : A partir d'une spécification du besoin de l'utilisateur, le générateur de processus va donc tenter de résoudre sa requête en composant des services atomiques appropriés publiés par le service fournisseur. D'habitude, le générateur de processus prend comme Entrées les fonctionnalités des services et produit comme Sortie le modèle de processus décrivant le service composite. Ce modèle de processus contient un ensemble de services atomiques orchestrés à travers un flux de contrôle et de données.

Evaluation du service composite : Plusieurs services peuvent offrir les mêmes ou des fonctionnalités très similaires. Donc, il est possible qu'un planificateur génère plus d'un service composite pouvant satisfaire un besoin donné. Dans ce cas, chaque service composite généré sera évalué en utilisant des informations fournies par les attributs non-fonctionnel tel que le coût, la qualité, le temps de réponse... etc. Le demandeur du service doit spécifier un poids pour chaque attribut non-fonctionnel et le service composite qui correspond le mieux à cette spécification sera sélectionné pour être exécuté.

Exécution du service composite : Après qu'un service composite est sélectionné, il est près pour l'exécution. L'exécution d'un service composite peut être vue comme l'échange d'une séquence de messages entre les services participants selon le modèle de processus défini. Le flux de données (dataflow) du service composite peut être défini comme étant l'ensemble des actions qui consistent à transférer la donnée de sortie d'un service exécuté à l'entrée du service suivant.

3. Les différentes méthodes de composition des services web

Selon les travaux effectués dans le champ des services web, on peut classifier la composition selon différents points de vue : manuelle, semi-automatique et automatique ou bien statique et dynamique.
Dans le cas de la composition statique, le demandeur ou l'utilisateur doit définir à priori le modèle abstrait de processus décrivant le plan de composition, par exemple, sous forme d'un graphe **[Cas00].** Ce dernier est constitué d'un ensemble de tâches abstraites combinées selon un ordre bien précis. Chaque tâche contient une clause requête utilisée pour chercher le service web concret implémentant la tâche en question. Dans ce type d'approche, uniquement la sélection et la liaison aux services basiques se font de manière automatique par programme.

Par contre, la composition dynamique vise à créer le modèle de processus abstrait, sélectionner et lier (ou rattacher) les services atomiques aux tâches abstraites de manière automatique et à la demande **[Rao204].**
L'automatisation du processus de composition implique de trouver les méthodes automatiques pour générer le modèle de processus et des méthodes pouvant localiser et sélectionner les services participants. Ce type de composition soulève un défi pour la

communauté web sémantique qui essaye d'appliquer des techniques de planification de l'IA pour générer le plan de composition de façon automatique.

Pratiquement, deux communautés de recherche s'intéressent au problème de composition des services web: la communauté workflow (industrielle) et la communauté web sémantique (académique) qui utilise le principe de planification de l'IA.

Nous allons dans ce qui suit voir quelles sont les approches proposées par ces deux communautés, comment elles perçoivent et traitent le problème de composition des services web et quels sont les apports et les limites de chacune d'elles. Mais avant ça, nous commençons par voir comment se faisait la composition des services web par les premières initiatives (manuelles).

3.1 Les méthodes manuelles classiques

Les premières initiatives traitant la composition des services web étaient manuelles et ad-hoc. Le script workflow modélisant le service composite est généré par l'utilisateur graphiquement ou à travers un éditeur. Ce script workflow est soumis à un moteur d'exécution de workflow pour être exécuté **[Maj104]**.

Triana **[Maj204]** est l'un des systèmes utilisant cette méthode, il fournit une interface graphique à travers laquelle un utilisateur pourra rechercher et sélectionner le service requis à partir du registre UDDI en effectuant une recherche simple à base de mots clé. Il permet aussi de faire la composition des services (construire le graphe modélisant la composition) en utilisant des outils localement disponibles. Le graphe résultant est ensuite distribué à travers le réseau P2P ou Grid pour l'exécution.

BPWS4J **[BPWS4J]** est un autre exemple de système utilisant une méthode manuelle. Cet outil permet à un utilisateur de composer un graphe au niveau XML. Ce graphe accompagné par les documents WSDL correspondant aux services participants sera soumis au moteur d'exécution.

SELF-SERV**[Ben03]** est également un autre exemple qui permet à l'utilisateur de localiser les services participants en utilisant un service Builder. Ce dernier interagit avec le registre UDDI pour retrouver les services participants requis. Le graphe résultant est un diagramme d'état annoté qui sera ensuite exécuté selon un modèle P2P.

Chaque système possède ses propres caractéristiques. Par exemple Triana permet d'exécuter le service composite à travers un réseau P2P ou Grid. D'un autre coté, SELF-SERV permet d'agréger les services offrant la même fonctionnalité dans un même conteneur. Ceci facilitera la substitution de service au moment de l'exécution en cas de non-disponibilité du service en cours.

Toutefois, ils ont plusieurs limitations en commun, nous citons :

- La découverte et la sélection est in-scalable : la découverte et la sélection des services participants se basent sur une recherche classique à base de mots clés dans le registre UDDI. Vu le nombre très croissant des services web, cette méthode s'avère pratiquement inutile.
- La qualité de recherche des services est très limitée : les mécanismes disponibles de recherche des services via des annuaires UDDI sont relativement primitifs. En effet, ils sont basés sur des paires (clé, valeur) qui limitent fortement la qualité des

recherches et n'offre aucun support sémantique pour la découverte efficace et non-ambiguë des services web.

- Ils exigent à l'utilisateur d'avoir des connaissances de bas niveau pour la spécification du workflow représentant le modèle de processus, par exemple, dans BPWS4J, l'utilisateur est censé mettre en place un workflow au niveau XML ce qui est très contraignant à l'utilisateur. Malgrès que Triana fournit un outil (une interface graphique) pour sélectionner et composer un workflow, il est pratiquement impossible d'appliquer le même procédé pour les larges workflows.
- Excepté SELF-SERV, la majorité des systèmes ne prennent pas en considération le problème de disponibilité des services web. Il n' y a aucune garantie qu'un service web disponible au moment de la conception du plan de composition le serait toujours au moment de l'exécution.
- La majorité des étapes du processus de composition sont manuelles. Par conséquent, plus le nombre de services augmente plus le processus de composition est plus lent et pénible à réaliser.

Le seul avantage de la méthode manuelle est qu'elle est la plus adaptable aux besoins de l'utilisateur, car c'est lui qui va tout définir à son goût et selon ses préferences depuis le début.

3.2 Les méthodes Workflow à base de standards

Workflow est le terme utilisé pour définir la gestion des processus métier. Le workflow est un traitement automatisé qui organise et contrôle les tâches individuelles, les ressources et les règles nécessaires pour compléter un processus métier clairement défini. Les processus métiers sont un ensemble de procédures au sein d'une organisation, une séquence d'activités réalisées par différentes personnes (ou entités) vérifiant des *règles métiers* : les conditions d'enchaînement des processus élémentaires **[Lev02]**.

La communauté industrielle suppose que les services web peuvent être intégrés comme des processus Workflow. IBM et Microsoft développent le concept de « Workflow de service web » et propose chacun des outils logiciels supportant leurs nouveaux langages de gestion et d'orchestration des services web au sein d'un processus complexe.

Dans le contexte des services web, un workflow définit comment les services participants (partenaires) travaillent ensemble dans un même processus pour exécuter une tâche précise. Il est appelé aussi par certains auteurs de cette communauté « Orchestration et Chorégraphie». La spécification d'un Workflow doit comprendre la description explicite de deux aspects complémentaires : « flux de contrôle » et « flux de données » . Le flux de contrôle définit le séquencement des différentes activités dans un processus. Le flux de données définit comment les informations ou les données s'écoulent (circulent) entre les activités d'un même processus.

3.2.1 Langages et formalismes utilisés :

WSDL **[WSDL]** est le langage émergeant pour la description des interfaces de services web. Mais, il permet seulement de spécifier la syntaxe des messages qui entrent ou quittent un service. L'ordre dans lequel les messages doivent être échangés ne peut pas

être décrit grâce à la spécification WSDL. Ce besoin a donné naissance à une autre catégorie de langages pour la spécification de flux de composition.
La plupart des langages de flux proposés par la communauté industrielle sont basés sur XML, citant XLANG **[XLANG]** de Microsoft, WSFL **[WSFL]** de IBM, BPEL4WS **[BPEL4WS]** et XL **[XL]**. Ces langages se basent sur WSDL en proposant un ensemble d'opérateurs de structuration des opérations WSDL. Chaque langage propose ses propres constructeurs de combinaison des Services web pour bâtir un business process multi-partie. BPEL4WS combine les caractéristiques des deux langages WSFL et XLANG pour décrire un service composite **[Med03]**.
Nous allons décrire dans ce qui suit chacun de ces langages du point de vue conceptuel.

XLANG -XML business process language (de Microsoft) :
Le langage Xlang **[XLANG]** s'inspire des langages de processus ce qui nous permet de faire le rapprochement entre un service Web composite et un processus **[MEL04]**. Il définit un processus comme une orchestration de services web. La description complète d'un processus ne doit pas se contenter de décrire le comportement de chaque participant, mais également la manière dont ces participants interagissent pour produire le processus complet. XLANG se focalise sur la description du comportement visible sous la forme de messages échangés. La manière dont chaque participant implémente sa portion de processus n'est pas du ressort du protocole XLANG, c'est à dire, les services participants sont laissés à blanc.
Le but de XLANG est de spécifier tous les comportements et seulement les comportements que chaque participant veut exposer à ses partenaires afin que ces derniers puissent les incorporer dans leurs propres services.
Le processus complet ne s'exécute vraisemblablement pas sur un moteur unique, il est constitué de la collaboration entre plusieurs sous-processus éventuellement répartis sur des environnements d'exécution totalement hétérogènes.
XLANG permet de définir (entre autres) :

- Des flux de contrôle séquentiels et/ou parallèles
- Des transactions longues avec des actions de compensation
- Des corrélations entre les messages échangés
- Des gestions d'erreurs et d'exceptions flexibles
- Des descriptions de comportements modulaires

Microsoft a exploité le langage XLANG dans son environnement BizTalk **[BizTalk]**. BizTalk contient un environnement de développement : "BizTalk Orchestration Designer", permettant de générer les "schedules" XLANG (les fichiers XLANG sont appelés schedules). Le serveur BizTalk utilise ensuite ces "schedules" XLANG pour instancier les processus à exécuter **[Jam05]**.

WSFL - Web Services Flow Language (de IBM) :
Il définit une approche très similaire à celle de Microsoft. WSFL (*Web Services Flow Language*) **[WSFL]** est un langage XML permettant de décrire des orchestrations de Services Web. WSFL définit deux types de compositions de services Web :
Le premier type décrit les processus d'entreprise comme des chorégraphies de services web (cf. XLANG). Il explicite les flux de contrôle et de données entre les services web constituant le processus.

Le deuxième type définit les processus d'entreprise en explicitant les relations « producteur/consommateur de messages » entre les descriptions WSDL des différents services web constituant le processus global.

Le logiciel "MQ Series Workflow" de IBM, connu aujourd'hui sous le nom de "WebSphere Process Manager" **[WSMQ]**, a pris en charge la spécification WSFL, afin d'automatiser les flots de procédés métier **[Jam05]**.

BPEL4WS - Business Process Execution Language for Web Services (IBM, Microsoft et BEA) :

BPEL4WS **[BPEL4WS]** est le résultat de la fusion des idées développées par IBM dans WSFL**[WSFL]** (support pour les graphes orientés processus) et par Microsoft dans XLANG **[XLANG]** (structuration en bloc pour définir les processus). Sa première version (BPEL4WS 1.0) a été réalisée en Août 2002. Il est basé sur XML et propose un langage pour définir formellement les processus ainsi que les protocoles d'interactions métiers.

La vision de BPEL4WS consiste donc à voir un service web comme un processus avec des flots d'activités et de données qui interagit avec des processus partenaires. BPEL4WS emploie WSDL pour modéliser un processus et ses partenaires en tant qu'interfaces abstraites WSDL. De même, l'abstraction des messages définie par WSDL se voit employée dans BPEL4WS. Les activités de processus peuvent être définies à l'aide d'éléments orientés flots : des séquences ordonnées (*sequence*), des exécutions en parallèle (*flow*), des branchements (*switch*, *if*), repétitions (*while*) et des chemins alternatifs (*pick*).

BPEL4WS définit également des gestions pour les évènements (messages, alarmes), les exceptions (throw, catch) et les compensations (scope) qui permettent d'annuler une transaction dans son intégralité lorsque celle-ci échoue. Il constitue une couche supérieure au langage de description WSDL **[Pon04]**.

BPEL4WS permet la modélisation deux types de procédé (Workflow): le procédé exécutable (executable Business) et le procédé abstrait (business protocol).

* Exécutable Business : modélise le comportement des différents partenaires interagissant ensemble.
* Business protocol : établit une description du comportement visible de processus.

La structure générale d'une description BPEL4WS est la suivante :

```
<process>
   <partnerLinks>    ..    partenaires    utilisés    dans    le    processus..
</partnerLinks>
   <variables> ..données échangées au sein du processus.. </variables>
   <correlationSets>  ensemble de propriétés utilisé pour identifier une instance
     unique du processus </correlationSets>
   <faultHandlers> ..activités exécutées en réponse à des fautes précises..
</faultHandlers>
   <compensationHandlers>   ..code   exécuté   pour   «défaire»   une   action..
</compensationHandlers>
                    (activités) actions exécutées par le processus
</process>
```

- **Les activités élémentaires :**

```
<invoke   partnerLink=''…''
          portType=''….''
          operation=''….''
           inputVariable=''….''
           outputVariable=''….''/>

<receive  partnerLink=''…''
          portType=''….''
          operation=''….''
          variable=''….'' />

<reply    partnerLink=''…''
          portType=''….''
          operation=''….''
          variable=''….'' />
<assign>
   <copy>
        <from variable=''….''/> <to variable=''….''/>
   </ copy >
</ assign >

<throw faultName= "…"  faultVariable="…"/>
                    Pour la détection et la gestion d'erreur
<wait for=''…'' ?  Until=''…'' ? ''/>
                    arrête l'exécution du processus pour un temps spécifié
<terminate/>     termine le processus
<empty>            ne fait rien
```

- **Les activités de structure :**

```
<sequence>  Exécution des activités de manière séquentielle
<flow>  Exécution des activités en parallèle<while>  Répétition de l'activité
      jusqu'au non-respect de la condition
<pick>
<link>  Définition d'une dépendance d'exécution entre une activité source et
une cible
```

Un procédé métier décrit à l'aide de BPEL4WS peut être interprété et exécuté par un moteur d'orchestration par exemple le BPWS4J (Business Process Web Service for Java) **[BPWS4J]**.
BPEL4WS a eu beaucoup de succès, et beaucoup d'implémentations industrielles ont été créées pour prendre en charge cette spécification. IBM a même remplacé, dans le logiciel WebSphere, la spécification WSFL qu'il a pris en charge au début, par la spécification BPEL4WS, pour gérer les procédés métier **[Jam05]**.

3.2.2 Exemple de système de composition à base de Workflow:

eFlow : C'est une plate-forme de spécification et de gestion des services composites issue de HP (*Hewlett-Packard)* **[Cas00]**. eFlow utilise des méthodes statiques pour la génération du workflow modélisant le service composite et permet une liaison dynamique aux services concrets participants. Dans eFlow, un service composite est modélisé par un graphe qui définit l'ordre d'exécution des nœuds (représentant des tâches) dans le processus global. Ce graphe est créé statiquement mais peut être mis à jour dynamiquement. Les nœuds du graphe sont en trois types : les nœuds service, les

nœuds décision et les nœuds évènement. Un nœud service permet de représenter l'invocation d'un service atomique ou composite, un nœud décision spécifie les alternatives et règles de contrôle d'exécution du flux et le nœud évènement permet à un processus service d'envoyer et recevoir des évènements. Les arcs du graphe dénotent la dépendance d'exécution parmi les nœuds.
Bien que le workflow modélisant le service composite doive être spécifié de façon statique, EFlow permet la liaison dynamique des nœuds (service) aux services concrets. En effet, la définition de chaque nœud service contient une clause (recette) de recherche qui est déclenchée à chaque invocation pour retourner la référence du service concret spécifique. Cette caractéristique est très importante dans un environnement hautement dynamique comme le web où la disponibilité des services change très fréquemment.

3.2.3 Discussion

Pour la composition des services Web, les méthodes Workflow sont généralement utilisées dans le cas où la requête aurait déjà défini le modèle de processus. Cela suppose que l'utilisateur doit fournir le modèle de façon manuelle (à travers une interface) et statique, ce qui est particulièrement une limitation du point de vue automatisation et scalabilité.
En outre, les langages de spécification de flux (abstrait et exécutable) utilisés par ces méthodes tels que WSFL, XLANG et très fréquemment le BPEL4WS ne sont pas déclaratifs, c'est à dire, ils ne permettent pas d'encoder les workflows de manière à faciliter leur manipulation symbolique. Par conséquent, ils ne sont pas adaptés à plusieurs tâches de raisonnement automatique envisagées par les services web sémantiques.
Théoriquement, dans le BPEL4WS, la tâche de liaison des services partenaires aux ports physiques se fait au moment de l'exécution. Néanmoins, la description de ces services partenaires est en WSDL (portType WSDL), ce qui fait qu'une liaison dynamique effective ne peut se faire en se basant uniquement sur les matching des interfaces WSDL. Les services participants (partenaires) doivent être sélectionnés en se basant sur des informations sémantiques concernant leur comportement fonctionnel (fonctionnalités) et leurs attributs non-fonctionnels.
Ajoutant à cela, limiter la description des services web sur l'expressivité strictement syntaxique via des interfaces WSDL limite l'intégration (l'interoperation) des services participants. Car ces derniers opèrent sur des messages qui peuvent être syntaxiquement différents mais sémantiquement compatibles ou ayant des syntaxes identiques mais sémantiquement sont incompatibles. Par exemple, deux services partenaires ayant des messages différents mais décrits avec un même nom peuvent causer la liaison des services fonctionnellement incompatibles. Ou encore, deux messages possédants des noms différents peuvent ne pas être liés bien qu'ils soient sémantiquement compatibles.

En résumé, les perspectives d'automatisation du processus de composition des services web adoptés par cette communauté sont pour l'instant limitées par les formalismes de description employés. Certains auteurs les qualifient comme des méthodes syntaxiques à base de standards **[Tal04]**. De plus, elles sont des méthodes statiques et très limitées du point de vue automatisation et dynamicité. Nous pouvons surtout constater que leur inconvénient majeur est l'absence de sémantique explicite dans la description des services web par ces standards.

3.3 Les méthodes de planification [Rao204]

Construire un processus ou un plan pour atteindre un but donné, c'est à dire, pour avoir une certaine tâche effectuée est un problème complexe qui a été très largement étudié par les recherches du domaine de l'IA appelé aussi ″synthèse de comportement complexe″. Rusel et Norving caractérisent le problème de planification comme suit : « la planification peut être interprétée comme une sorte de résolution de problème où un agent utilise ses croyances à propos des actions disponibles et leurs conséquences dans le but d'identifier une solution à travers un ensemble de plans abstraits possible». Généralement, un problème de planification classique est décrit à travers le tuple (S, S0, G, A, Γ) où S est l'ensemble de tous les états possibles du monde, $S_0 \subset S$ dénote l'état initial du monde, $G \subset S$ dénote l'état but du monde que le système de planification tente d'atteindre, A est l'ensemble de toutes les actions que le planificateur peut effectuer afin de passer d'un état à un autre et finalement la relation $\Gamma \subseteq S \times A \times S$ qui définit les pré-conditions et les effets associés à chaque action.
De manière générale, un planificateur possède les entrées suivantes :

- Une description de l'état initial du monde (S_0), dénotée dans un langage formel.
- Une description de l'état but désiré (G), dénotée aussi dans un langage formel.
- Une description formelle de toutes les actions possibles qui peuvent être appliquées afin de progresser vers l'état but.

Au terme des services web, S_0 et G sont respectivement l'état initial et l'état but spécifié dans la requête du demandeur. A est l'ensemble des services web publiés (disponibles).
La majorité des méthodes sémantiques qui visent à générer de façon dynamique le service composite reposent sur les techniques de la planification de l'IA et la preuve de théorème.
L'hypothèse générale de ce genre de méthodes est que chaque service web peut être spécifié par ses pré-conditions et effets dans le contexte de la planification. On peut envisager cela de deux manières. Premièrement, si le service web prend des données en entrées et produit des données en sortie, dans ce cas, les pré-conditions et les effets sont respectivement ses entrées et ses sorties. D'un autre coté, un service web peut altérer l'état du monde après son exécution. L'état du monde requis pour que le service s'exécute est la pré-condition et le nouvel état généré après son exécution est l'effet du service sur le monde. Par exemple le service de *logging* à un site web possède comme information d'entrées le *username* et le *password* et l'information de sortie est un message de confirmation. Après l'exécution, l'état du monde change de *not_logged_in* à *logged_in*. L'état logged_in demeure jusqu'à ce que le service Log_Out soit invoqué. Donc, si l'utilisateur pourra spécifier toutes les pré-conditions et effets requis par le service composite demandé, un plan ou un processus est généré automatiquement par un prouveur de théorème ou à l'aide d'un planificateur IA sans faire appel à un workfolw prédéfini.

Dans ce qui suit, nous allons voir quelques méthodes de composition à base de planification que nous classifions en plusieurs catégories, notamment, le calcul de situation, PDDL (Planning Domain Definition Language), planification à base de règles,

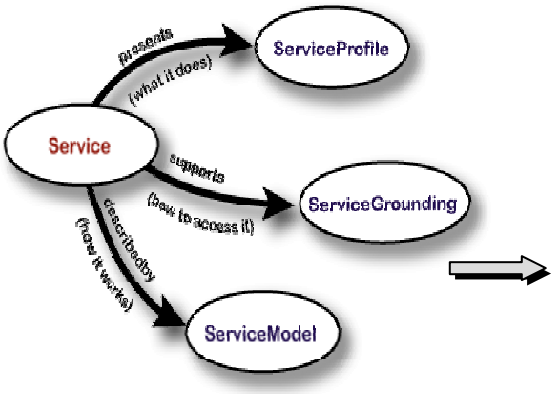
planification HTN (réseau de tâches hiérarchiques), le Model Checking et la preuve de théorème, et autres. Mais avant cela, nous allons d'abords voir quels sont les langages et formalismes utilisés par cette communauté pour la description des services web et le raisonnement sur leur composition.

3.3.1 Langages et formalismes utilisés

Les langages et formalismes logiques utilisés par la communauté de planification de l'IA pour la composition des services web sont très diverses. Certains d'entre eux sont de la logique classique comme la logique du premier ordre et la logique des propositions. Les autres sont très liés au domaine de la planification et la preuve de théorème. Vu leur diversité, nous les présenterons, en même temps, très brièvement avec les méthodes les utilisant. Nous allons plutôt mettre l'accent sur le langage OWL-S (appelé DAML-S dans les versions précédentes) **[OWL-S][DAML-S]** car c'est l'unique langage ontologique développé pour les services web qui annonce une liaison directe avec la planification de l'IA.
Le changement d'état produit par l'exécution d'un service est spécifié à l'aide des propriétés Pré-condition et Effet du ServiceProfile de OWL-S. Rappelons que, OWL-S est construit au-dessus de OWL qui lui même utilise la logique de description comme fondation logique. Par conséquent, il possède une puissance d'expressivité logique assez suffisante pour décrire propriétés conditionnelles des services web telles que les pré-conditions et les effets.
La majorité des méthodes de planification traitant la composition automatique des services web utilisent OWL-S (DAML-S) comme langage externe de description des services web et autres qui utilisent leurs propres langages logiques.

OWL-S (Ontology Web Language Service):

Le besoin de la sémantique dans la description des services web a été largement discuté par les auteurs de la communauté web sémantique **[Siv03]**. Ils supposent que l'annotation sémantique des services web fournit un moyen pour automatiser la découverte et la composition des services web. Le fruit de leurs efforts était la naissance d'un nouveau langage de méta-description de services qui utilise des ontologies OWL **[OWL]**.
OWL-S **[OWL-S]** est issu particulièrement, des travaux du programme du DARPA Agent Markup language (DAML) et prend la suite de DAML-S (DARPA Agent Markup language Service). Les intérêts liés à l'utilisation de OWL-S sont que ce langage inclut la sémantique et contient des fonctions indispensables à la mise en œuvre des services web notamment la description, la recherche, l'invocation et la composition. Il est composé de trois parties principales décrites par la *Fig 3.3:* le profil de service (Service Profile), le modèle de service (Service Model) et l'accès au service (Service Grounding). Les deux premières sont des représentations abstraites tandis que l'accès au service se situe au niveau concret de spécification **[Lop04].**




```
- <owl:Class rdf:ID="Service">
      <rdfs:label>Service</rdfs:label>
    </owl:Class>
  - <owl:Class rdf:ID="ServiceProfile">
      <rdfs:label>ServiceProfile</rdfs:label>
  - <owl:Class rdf:ID="ServiceModel">
      <rdfs:label>ServiceModel</rdfs:label>
  - <owl:Class rdf:ID="ServiceGrounding">
      <rdfs:label>ServiceGrounding</rdfs:label>

  - <owl:ObjectProperty rdf:ID="presents">
      <rdfs:domain rdf:resource="http://www.daml.org/services/owl-
          s/1.2/Service.owl#Service" />
      <rdfs:range rdf:resource="http://www.daml.org/services/owl-
          s/1.2/Service.owl#ServiceProfile" />
    </owl:ObjectProperty>

  - <owl:ObjectProperty rdf:ID="describedBy">
      <rdfs:domain rdf:resource="http://www.daml.org/services/owl-
          s/1.2/Service.owl#Service" />
      <rdfs:range rdf:resource="http://www.daml.org/services/owl-
          s/1.2/Service.owl#ServiceModel" />
    </owl:ObjectProperty>

  - <owl:ObjectProperty rdf:ID="supports">
      <rdfs:domain rdf:resource="http://www.daml.org/services/owl-
          s/1.2/Service.owl#Service" />
      <rdfs:range rdf:resource="http://www.daml.org/services/owl-
          s/1.2/Service.owl#ServiceGrounding" />
    </owl:ObjectProperty>
```

Fig 3.3:Structure générale de l'ontologie de service OWL-S

Service Profile: un OWL-S Service Profile décrit un service comme une fonction qui possède trois types d'informations : Une description textuelle du service et des informations sur son fournisseur, une description du comportement fonctionnel du service et un ensemble d'informations supplémentaires décrivant la qualité d'un service, sa catégorie (sa classification taxonomique), …etc

- Nom, contacts et description d'un service : ces données sont fournies à travers les attributs `serviceName`, `textDescription` et `contactInformation`. Text description fournit une brève description textuelle concernant le service. `ContactInformation` fournit des informations de contact de l'entité responsable du service (le fournisseur).
- Le comportement fonctionnel: décrit les fonctionnalités qu'offre le service en terme de transformation d'information dénotée par les Entrés/Sorties (I/O) et de changement d'état après l'exécution du service dénoté par les pré-conditions/effets (P/E). Les IOPEs publiés dans le Profile sont un sous ensemble de ceux publiés par Process Model du Service Model décrit ci-dessus. Par conséquent, la partie Process crée toutes les instances IOPEs et le profile pourra simplement pointer vers ces instances. L'ontologie Profile définit alors les propriétés suivantes qui pointent directement vers les instances IOPEs (ranges): hasInput, hasOutput, hasPrecondition, hasResult.
- D'autres attributs du profile : les IOPEs permettent de fournir une description fonctionnelle du service. Mais il existe d'autres aspects du service pour lesquels les utilisateurs doivent être conscients. Ces attributs additionnels comprennent la qualité que garantit un service par exemple le temps de réponse et le coût, une classification possible d'un service (serviceCategory) et éventuellement d'autres paramètres additionnels spécifiés dans la classe serviceParameter qui est une classe extensible .

Le modèle de service (Service Model) : Le modèle de service présente le fonctionnement du service et définit les différents processus de service web. La classe ProcessModel possède comme propriétés : les paramètres d'Entrées/Sorties (nombre,

type…), les participants au processus, les pré-conditions et les effets du service. Il existe trois types de processus :
- Le processus atomique : directement invoqué par l'intermédiaire d'un Grounding
- Le processus composé : décomposable en d'autres processus plus simples en utilisant les commandes de contrôle (par exemple : Si-Alors-Sinon, …)
- Le processus simple : non-invoquable, il fournit simplement une vue d'un processus atomique ou une représentation simplifié d'un processus composé.

Les composants principaux d'un modèle de processus sont l'ontologie de processus (Process Ontology) et l'ontologie de contrôle du processus (Process Control Ontology).
L'ontologie de processus, laquelle nous avons fait références dans la partie précédente, décrit un service en terme de IOPE. Cette ontologie peut être utilisée afin de supporter l'invocation et la composition automatique de services web.
L'ontologie de contrôle de processus décrit chaque processus comme étant un état en prenant en compte son activation, son exécution et sa terminaison.

Accès au service (Service Grounding) : décrit par l'intermédiaire du protocole à utiliser, les formats de messages et le type de transport. Le contenu abstrait des messages échangés est décrit en tant que propriétés d'un processus atomique. La spécification concertes des messages est effectuée par l'intermédiaire du langage WSDL.

Nous pouvons constater que le langage OWL-S s'adapte bien au besoin des méthodes de planification qui exigent une description de l'état du monde avant et après l'exécution d'un service donné. Cette description est explicitement déclarée à travers les pré/post-conditions dans le service Model en utilisant des termes à partir des ontologies pré-agrées.

3.3.2 Quelques méthodes et systèmes à base de planification

Calcul de situation:

Le calcul de situation de la logique des actions **[Pra90]** a été développé pour décrire les changements dynamiques dans le monde. Cette logique suppose que tous les changements dynamiques qui surviennent sur le monde sont dus à l'exécution de certaines actions. Chaque situation est définie par un historique du monde qui est une séquence d'actions. L'état du monde est décrit par des fonctions appelées *Fluents* relative à une situation.
Golog **[Lev97]**, est un langage de programmation logique de haut niveau pour la spécification et l'exécution d'actions complexes dans un domaine dynamique. Il est bâti au-dessus du calcul de situation et fournit des constructions extra-logiques pour assembler des actions primitives en des actions plus complexes. L'application de Golog et le calcul des situations dans le cadre des services web a été suggérée par S.McIlraith and T.Son dans **[McI02] [Nar02]**. Les auteurs adressent le problème de la composition des services web à travers l'adaptation de procédures génériques de haut niveau et la personnalisation de contraintes prédéfinies. La requête de l'utilisateur (procédure générique) et ses contraintes sont présentées dans le langage du premier ordre du calcul des situations. Ils considèrent qu'une action peut être primitive ou complexe. Une action

primitive peut être de type ″world-altering″ qui change l'état du monde ou bien de type ″information-gathering″ qui change l'état de l'agent de connaissance. Une action complexe est la composition de plusieurs actions primitives. L'agent de la base de connaissance fournit un encodage logique de l'ensemble des pré-conditions et effets des actions de services web dans
un langage de calcul des situations. Un service composite est un ensemble de services atomiques connectés de façon procédurale par des constructions du langage de programmation utilisé (if-then-else, while, …).

PDDL :
PDDL **[Gha98]** est un langage qui couvre différentes manières d'encoder les domaines et les problèmes de planification. Plusieurs planificateurs du domaine de l'IA acceptent des spécifications PDDL comme format de description des données d'entrée. De plus, DAML-S (plus récemment OWL-S) est fortement influencé par le langage PDDL, ce qui rend facile le passage d'une représentation PDDL à une représentation DAML-S. Dans le cadre de la composition, la description DAML-S des services web doit être traduite en représentation PDDL. Ainsi, plusieurs planificateurs peuvent être exploités pour générer le plan de manière automatique.
McDermott **[McD02]** a utilisé une telle méthode et il a introduit un autre type de connaissance appelée "*la valeur d'une action* " qui persiste et qui n'est pas traitée comme un littéral de vérité. Cette nouvelle connaissance permettra de distinguer entre la transformation d'information et le changement d'état du monde produit par l'exécution d'un service.

Planification HTN (Hierarchical Task Network):
Quelques autres techniques à base de planification ont été proposées pour la composition des services web. SHOP2 **[Wu03]** est un planificateur HTN (Hierarchical Task Network) **[Ero94]** appliqué à la composition automatique des services web qui sont fournis avec des descriptions DAML-S.
HTN est une méthode de planification par décomposition de tâches. Un système de planification à base d'HTN décompose la tâche complexe désirée en un ensemble de sous tâches, et ces tâches sont de nouveau décomposées en d'autres sous tâches et ainsi de suite jusqu'à l'obtention d'un sous ensemble de sous tâches atomiques, c'est à dire, qui peuvent être exécutées directement en invoquant des opérations atomiques. Durant chaque étape de décomposition, certaines conditions sont testées par exemple : ne pas excéder un certain montant financier des ressources...etc. La planification réussit si la tâche complexe aurait pu être décomposées en sous tâches primitives sans violer aucune des conditions imposées.
Les auteurs de **[Wu03]** suggèrent que le concept de décomposition de tâche par la planification HTN est similaire au concept de décomposition de service composite en services basiques dans DAML-S Process Ontology. Les auteurs affirment que la planification HTN est plus efficace que les autres langages de planification tel que Golog. Ils donnent aussi, dans leur document, une description détaillée du processus de traduction de la spécification DAML-S à la spécification SHOP2 des services disponibles.

Planification à base de règle : SWORD

SWORD **[Pon02]** est un toolkit de construction du service composite par la génération automatique du plan grâce à des descriptions à base de règles. SWORD n'utilise pas les standards émergeants tel que WSDL et DAML-S pour la description des services, mais plutôt, il utilise un modèle Entité/Relation comme modèle de spécification des services web. En effet, le modèle du monde consiste en un ensemble d'entité avec des relations entre eux.

Un service web est représenté sous forme d'une règle de Horn qui dénote les post-conditions à atteindre si telles pré-conditions sont vraies. Pour créer un service composite, le demandeur a besoin de spécifier uniquement l'état initial et final du service composite, ensuite la génération du plan peut être atteinte en utilisant un système expert à base de règles. En plus de la méthode de composition proposée, les auteurs donnent une discussion intéressante concernant le chaînage à base de règles. Ils disent que le chaînage à base de règle peut générer un résultat incertain si une pré-condition ne peut pas déterminer de façon unique une post-condition. Les auteurs argumentent qu'un résultat incertain peut être évité si les pré-conditions sont fonctionnellement dépendantes des post-conditions au sein d'un service. Ce problème est fréquemment rencontré par la majorité des méthodes de composition proposées mais les auteurs ne le déclarent pas explicitement.

Dans la même catégorie, on peut citer le travail fait par Medjahed **[Med03][Med4]** qui a utilisé des règles pour déterminer si deux services sont composables ou non. Ces règles sont appelées "règle de composabilité". Medjahed présente une technique de génération de service composite à partir d'une description déclarative de haut niveau de l'objectif visé. L'approche de composition proposée est en quatre phases (voir la *Fig 3.4*): la spécification, le Matchmaking, la sélection et la génération. Premièrement, une description de haut niveau de la composition désirée est générée en utilisant un langage XML appelé Composite Service Specification Language (CSSL) durant la phase de spécification. La deuxième phase qui est la phase de "Matchmaking" utilise les règles de composabilité pour générer le plan de composition à partir de la spécification générée à la première phase - qui soit conforme à la spécification du demandeur- . La troisième phase est la phase de sélection. Sachant que plusieurs plans de composition satisfaisant le besoin spécifié peuvent être générés, la sélection d'un plan sera basée sur certains paramètres de qualité (QdS) (le coût, le temps de réponse…etc) précisés par le demandeur dans la section profile de sa spécification. La dernière phase est la phase de génération. Une description détaillée du service composite sera générée automatiquement dans un langage de spécification de flux et ensuite présentée au service demandeur avant d'être soumis à l'exécution.

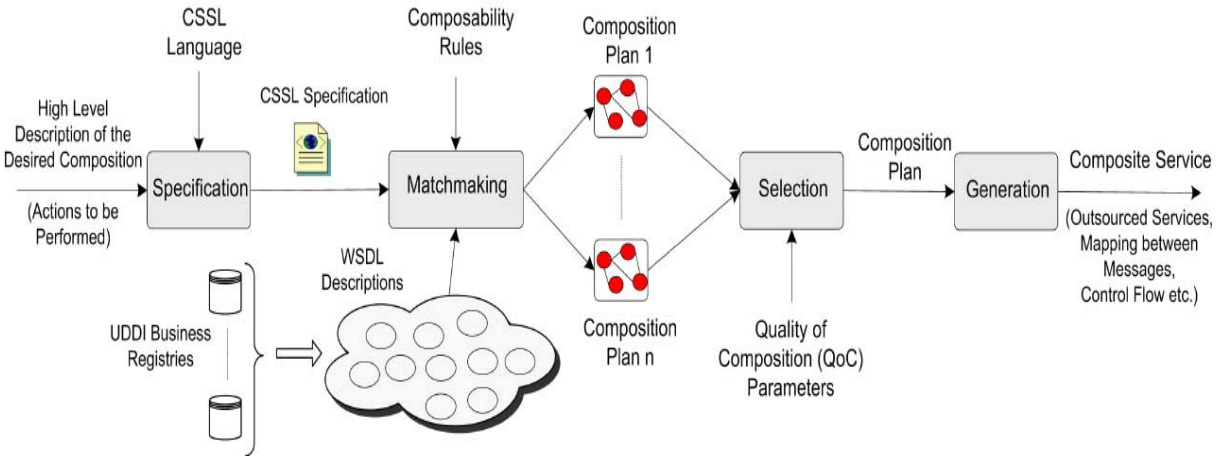


Fig 3.4: Schéma général du système de composition proposé par [Med03]

La plus intéressante contribution de ce travail est la définition des règles de composabilité qui guident la génération automatique du plan de composition en définissant tous les attributs possibles qui peuvent être pris en considération pour effectuer une composition cohérente.
En effet, les règles de composabilité considèrent les propriétés syntaxiques et sémantiques des services web. Les règles syntaxiques comprennent des règles sur les attributs représentant les modes d'opérations et les protocoles de liaison (binding) utilisés par deux services donnés. Les règles sémantiques comprennent le sous-ensemble suivant :

- La composabilité sémantique des messages : se sont des règles qui définissent que deux services sont composables si le message de sortie du premier service est composable avec le message d'entrée du deuxième.
- La composabilité sémantique des opérations : définit la compatibilité des domaines, des catégories et les buts métier de deux services.
- La composabilité qualitative : détermine si les préférences du demandeur en terme de qualité sont conformes au service composite.
- La validité de composition : détermine si la composition effectuée est valide (raisonnable) ou pas. Pour la calculer, l'auteur introduit la notion de Template de composition qui définit la dépendance entre les différents types de service.

La preuve de théorème :
Waldinger **[Wal01]** a élaboré une idée pour la synthèse de services en utilisant la preuve de théorème. L'approche est basée sur la déduction automatique et la synthèse de programme**[Man80]**. Initialement, les services disponibles et les besoins des utilisateurs sont décrits dans un langage du premier ordre de la logique classique, puis des preuves constructives sont générées par le prouveur de théorème SNARK. La description de la composition de service est ensuite extraite à partir des preuves particulières.
Lämmermann **[Läm02]** applique la synthèse structurelle de programme (SSP) pour la composition automatique des services. SSP est une approche déductive pour la synthèse des programmes. La spécification des services comprend uniquement les propriétés structurelles, c'est à dire les informations Input/Output. SSP utilise des variables propositionnelles pour identifier les paramètres Input/Output et utilise la logique

propositionnelle intuitionniste pour résoudre le problème de composition. En effet, le problème de composition se ramène à un problème de recherche de preuve. L'auteur utilise aussi les disjonctions de la logique classique pour décrire les exceptions qui peuvent survenir durant l'invocation des services.

Un autre travail utilisant la preuve de théorème est celui de Rao et .al. **[Rao03][Rao104]** qui a introduit une méthode automatique pour la composition des services web sémantiques en utilisant un prouveur de théorème de la logique linéaire. La méthode utilise comme langage externe pour la spécification des services web le langage sémantique DAML-S et des axiomes et des preuves extra-logiques de la logique linéaire comme spécification interne. La logique linéaire, comme ressource consciente, permet d'exprimer les propriétés fonctionnelles et non-fonctionnelles, qualitatives et quantitative d'un service. Ajoutant à cela, la logique linéaire est très proche du л-calcul qui est une fondation formelle de plusieurs langages de description des services composites. A la différence des autres méthodes qui utilisent des attributs non-fonctionnels seulement pour filtrer le plan généré, cette méthode considère ces attributs non-fonctionnels directement dans le processus de preuve de théorème.
Les attributs fonctionnels et non-fonctionnels sont exprimés sous forme de propositions dans des axiomes logiques (mais la différence est établie par les règles d'inférence de la logique linéaire).

3.3.3 Discussion

Les méthodes de planification apportent, sans doute, un très haut degré d'automatisation au processus de composition. Néanmoins, certains problèmes restent soulevés quant à l'efficacité de cette automatisation vis-à-vis du besoin de l'utilisateur et aussi l'adaptation des contextes des deux problématiques : la composition des services web et la planification telle qu'elle est appliquée dans le domaine de l'IA.
Certains auteurs **[Tal04]** affirment que les techniques classiques de la planification du domaine de l'IA ne peuvent pas être directement appliquées au problème de composition des services web pour les raisons suivantes :

- L'hypothèse du monde clos est très souvent faite dans les problèmes de planification classiques. Cela signifie que tous les objets doivent être disponibles à l'état initial et les actions ne font que changer l'état des objets. Par contre, les services web créent de nouveaux objets pendant l'exécution. En d'autres termes, si un littéral n'existe pas dans le monde courant, sa valeur de vérité est évaluée à faux. Le problème avec cette hypothèse du monde clos est que simplement avec les littéraux de vérité on ne peut pas exprimer cette nouvelle information produite par l'exécution d'un service. Par exemple, un service demandeur veut qu'après l'envoi d'un message à un service donné, un numéro identifiant ce message soit généré pour pouvoir l'utiliser ultérieurement pour d'autres communications **[Rao204]**.
- Types de données riches : Les objets messages manipulés par les services web possèdent une structure beaucoup plus riche que celle des objets utilisés actuellement dans les problèmes de planification de l'IA **[Tal04].** Les services web manipulent des objets messages qui peuvent contenir des parties ayant des descriptions complexes.

- Les langages courants de la planification n'ont pas encore atteint le degré d'expressivité logique requis par la composition des services.
- En utilisant les techniques de planification, la composition est traitée à un bas niveau. En particulier, la requête devrait définir l'objectif visé de manière rigoureuse (l'état initial et l'état but) et être exprimée dans un langage de bas niveau tel que la logique du premier ordre ou autre langage logique. Ce qui est contraignant pour un utilisateur non-spécialiste.

De plus, plusieurs méthodes utilisant les techniques de planification utilisent des ontologies de services et le langage OWL-S comme langage de spécification. Cependant, la construction de telles ontologies est très coûteux et demande beaucoup de compétences. Les outils courants comme protégé 2000 apporte une certaine assistance, mais son utilisation reste dans le cadre des projets prototypiques et au sein des groupes de recherche. En effet, il y a peu de motivations pour l'annotation des services web dans le cadre commercial **[Tal04]**.

3.4 Les méthodes de composition interactives

Pour remédier à quelques problèmes des méthodes manuelles et les méthodes automatiques, certains auteurs ont pensé aux approches semi-automatiques ou interactives en impliquant l'utilisateur dans les phases de décision par exemple, la sélection du meilleur service parmi un ensemble de services potentiels.
Le travail que nous pouvons citer dans ce cadre est celui de Sirin et al **[Sir03]** qui présentent une méthode de composition interactive. A chaque étape où l'utilisateur aurait à sélectionner un service participant, tous les services qui peuvent être composés avec le service courant lui sont présentés pour choisir celui qui semble le plus pertinent. Le choix des services candidats est basé sur les propriétés fonctionnelles et non-fonctionnelles. Les paramètres fonctionnels sont présentés par les classes OWL et un moteur de raisonnement est appliqué pour l'appariement (mise en correspondance) des services. Une mise en correspondance est définie entre deux services si le paramètre de sortie de l'un a la même classe OWL ou une sous-classe de celle du paramètre d'entrée de l'autre service. Le moteur d'inférence OWL peut ordonner les services résultat selon leur degré de mise en correspondance. Ce degré est calculé par la distance des classes des paramètres dans l'ontologie. Ensuite, ces services résultants passent par un autre filtre qui opère sur les attributs non-fonctionnels spécifiés par l'utilisateur comme des préférences. Uniquement ces services qui satisfassent les contraintes non-fonctionnelles seront présentés au service demandeur.

Discussion

L'idée de la composition semi-automatique ou interactive semble très intéressante parce qu'il est très difficile de capturer efficacement le comportement des services au moindres détails et les composer de manière entièrement automatique tout en maintenant une grande flexibilité vis-à-vis le besoin de l'utilisateur.

Cette méthode montre la possibilité d'un travail collaboratif entre un planificateur automatique et un être humains afin de générer un service composite répondant à un

besoin particulier. Par conséquent, l'utilisateur maintiendra un certain contrôle sur le processus envisagé mais il n'aura pas besoin de connaissances de programmation de bas niveau. Il pourra définir son propre processus en utilisant des outils graphiques pour modéliser et concevoir des services Web composites.

Néanmoins, bien que les méthodes semi-automatiques ont pu régler quelques problèmes de scalabilité rencontrées par les méthodes manuelles, elles restent toujours insuffisantes face à ce problème. En effet, le processus de filtrage des services peut aboutir à une liste large des services candidats, qu'il serait difficile de gérer par l'utilisateur pour faire une sélection.

On peut dire que les méthodes semi-automatiques peuvent être plus efficaces si le taux d'automatisation soit le plus élevé possible et l'implication de l'utilisateur durant le processus de composition soit le moindre possible.

4. Tableau comparatif des méthodes de composition étudiée:

Nous allons résumer notre étude concernant les différentes méthodes de composition des services en dressant un tableau comparatif (voir le tableau ci-dessous). Ce tableau permet une comparaison selon certains critères d'évaluation que nous avons pu déduire à travers notre étude.

- Niveau d'automatisation (manuelle, automatique, semi-automatique).
- Niveau de dynamicité (statique, dynamique).
- Niveau de Scalabilité (peut être étendu à un grand nombre de services).
- Spécification de la requête de l'utilisateur (utilisation d'un langage de bas niveau ou un langage proche de l'utilisateur).
- Niveau de granularité des plans de composition généré: La réutilisation des plans construits (abstrait et exécutable, ou bien seulement exécutable).
- Support sémantique et niveau d'abstraction (pas de support sémantique, DAML-S, OWL-S, autres).
- Recherche des services: Recherche à plusieurs niveaux ou à un seul niveau

	Méthodes manuelles	**Méthodes à base de workflow**	**Méthodes à base de planification**	**Méthodes interactives**
Niveau d'automatisation	Aucune automatisation du processus de composition (nul)	Uniquement la recherche et la liaison (binding) aux services concrets est automatique(moyen)	Pratiquement tout le processus se fait de manière automatique (élevé)	Semi-automatique
Niveau de dynamicité	Tout se fait statiquement	- Modèle de processus est généré statiquement - Liaison aux services concrets est dynamique	- la génération du plan de composition se fait de manière dynamique	- la génération du plan se fait de manière dynamique

Niveau de Scalabilité	Pas du tout	Moyen	élevé	Modeste.
Spécification de la requête de l'utilisateur	Très bas niveau, c'est l'utilisateur qui définit tout le plan de composition	Bas niveau : l'utilisateur doit mettre en place le modèle de processus en utilisant un langage de flux	Définit l'état initial, l'état final et certaines contraintes dans un langage de très bas niveau (les langages logiques)	Se fait interactivement avec le système (au fur et à mesure)
Niveau de granularité des plans de composition générés	Un seul plan généré (plan exécutable)	-Workflow abstrait (mais défini manuellement) -Et un workflow exécutable	Un seul plan (plan exécutable)	Uniquement le plan exécutable
Support sémantique et niveau d'abstraction	Pas de support sémantique (registre UDDI)	La majorité utilise des supports syntaxiques (WSDL, BPEL4WS, WSFL…)	- utilise les formalismes logiques, DAML-S ou OWL-S.	Support sémantique : DAML-S, OWL-S
Recherche des services	Primitif (dans les registres UDDI)	- la majorité utilise le moteur de recherche UDDI	- dans les profile DAML-S ou OWL-S	- dans les profiles DAML-S ou OWL-S

5. Conclusion

La composition des services Web s'impose comme l'application la plus naturelle qui permet de mettre à profit les composants Services Web au service de l'intégration d'applications sur le web. Elle constitue l'un des champs de recherche les plus actifs dans le domaine des services Web.

Nous avons voulu, à travers ce chapitre, expliquer le problème de composition des services web et donner un aperçu sur les diverses initiatives de composition conduites par les différentes communautés (industrielles et académique). Ces divers travaux ont montré la complexité de cette problématique et les divergences des points de vue sur la manière de percevoir le problème de composition. D'un coté, la communauté Workflow suppose que les services web peuvent être intégrés comme des processus Workflow. Elle a développé des outils et plates-formes qui supportent leurs nouveaux langages de flux pour les services web. Mais, les méthodes de composition proposées supposent la définition préalable du modèle de processus décrivant la fonctionnalité complexe voulue, puis la recherche des services concrets et leur liaison se fera de façon dynamique.

D'un autre coté, la communauté web sémantique compte beaucoup sur l'apport des méthodes de planification pour automatiser la génération du service composite. Elle considère que le problème de la composition peut être vu comme un problème de

planification, tel que, chaque service est considéré comme une action qui possède un contexte d'exécution représenté par les pré-conditions et les effets. La requête n'a qu'à définir l'état initial et l'état but que le système va tenter d'atteindre de manière automatique et éventuellement un ensemble de contraintes qui peuvent guider la génération du plan. Les techniques de planification visent à automatiser entièrement le processus de composition. Plusieurs méthodes proposées utilisent le DAML-S (ou OWL-S), développé par la communauté web sémantique comme langage de spécification externe des services web.

Les approches proposées par ces deux communautés possèdent différents niveaux d'automatisation du processus de composition. Cependant, nous ne pouvons pas affirmer que celle qui possède un très haut degré d'automatisation est la meilleure, car, comme nous avons pu le constater, l'environnement des services web est hautement complexe et il n'est pas toujours possible de tout générer de façon automatique et efficace.

Nous avons pu soulever d'autres types de problèmes concernant différents aspects de la composition que nous avons résumé dans le tableau précédent. En particulier, la première phase du processus de composition qui interpelle directement un modèle d'interaction entre un utilisateur externe et l'ensemble des services disponibles. L'utilisateur doit pouvoir exprimer ses besoins en matière de service de manière naturelle et accéder de manière automatique et transparente à leurs fonctionnalités. Les méthodes citées auparavant ne prennent pas suffisamment en compte le problème d'interaction avec un utilisateur voulant exécuter un service composite, sachant que les services web concernés sont à priori inconnus et dispersés à l'intérieur d'un vaste entrepôt de services.

Chapitre 4: Un modèle de composition des services web sémantiques

1. Introduction

Des applications complexes voulant accéder à des services web divers nécessitent des capacités avancées pour manipuler et délivrer des fonctionnalités complexes de services web. En général, les utilisateurs ont des besoins qui ne peuvent pas être satisfaits par la simple invocation d'un service web unique ou bien un ensemble de services web de façon indépendante. En effet, un tel besoin fait appel à un processus de composition de plusieurs services web pour délivrer la fonctionnalité complexe voulue.

La plupart des travaux s'intéressant au problème de la composition automatique des services web, ne permettent pas de fournir un moyen d'interroger de manière simple et plus ou moins naturelle des processus complexes faisant appel à une combinaison de services web. Généralement la requête de composition initiale décrivant l'objectif visé est formulée dans un langage de spécification de bas niveau. Or, un utilisateur ordinaire ne possède pas les capacités lui permettant de spécifier sa requête dans une forme directement traitable par un agent logiciel. Très souvent, il ne possède pas une idée complète et assez précise concernant le service qu'il cherche pour formuler une requête claire et non ambiguë décrivant son objectif. Surtout quand cela est couplé avec le fait que l'utilisateur n'a pas de connaissances concernant les langages de spécification des services web pour exprimer sa requête de composition. Cela nécessite un système à travers lequel l'utilisateur pourra accéder à des fonctionnalités complexes d'un domaine d'une manière aisée et naturelle.

Dans ce chapitre, nous proposons un modèle de composition des services web à travers lequel l'utilisateur aura un accès graduel aux fonctionnalités (simples ou complexes) d'un domaine d'application. En d'autres termes, en utilisant notre modèle, l'utilisateur spécifie de façon naturelle ce qui doit être fait en terme de tâches qui doivent être effectuées. Il ne sera obligé de connaître tous les détails techniques tels que les services participants et la manière dont ils seront composés et exécutés. De plus, l'utilisateur aura plus de liberté de donner ou non l'ordre d'exécution des tâches qu'il spécifie dans sa requête. La découverte des services participants, leur composition ainsi que leurs interactions doivent être effectuées de façon automatique et transparente par rapport à l'utilisateur.

Nous proposons une approche graduelle de résolution de la requête de composition, de sa simple spécification par l'utilisateur jusqu'à la génération du plan concret exécutable. La première étape du processus de résolution consiste à vérifier si le besoin de l'utilisateur correspond à un service simple ou bien à un processus complexe. Pour ce faire, nous avons défini un schéma de correspondance (Matching) sémantique qui intervient à différentes étapes du processus de composition. En particulier, la mise en

correspondance sera appliquée, à la première phase, entre la requête et un processus générique prédéfini dans le domaine d'application appelé *" template générique "*. Ce dernier, représente un plan initial de composition qui sera affiné jusqu'à retrouver les différents services participants pouvant accomplir ensemble la tâche en question. La mise en correspondance (Matching) est un mécanisme qui vise à retrouver les similarités sémantiques entre deux concepts syntaxiquement différents. Ceci nécessite une représentation sémantique de l'ensemble des concepts soumis au processus de Matching.
Avant d'expliquer notre approche, nous allons d'abords examiner quelques modèles de description et de découverte des services web sur lesquelles sont basées la majorité des méthodes de composition vues précédemment **[Maj204][Ben03][BPWS4J][Cas00], [Sir03][Arp04][meteor-S]**…etc. Nous allons découvrir, par la suite, leurs limites vis-à-vis de notre problématique.

2. Les modèles existants de description et de découverte des services web

La découverte dynamique et effective des services web est l'un des grands défis de la technologie des services web. C'est le mécanisme qui consiste à trouver (localiser), dans un vaste registre, le service qui convient au besoin du demandeur ainsi que toutes les informations permettant d'interagir avec ce service. Ces informations peuvent être la description WSDL du service web et les informations concernant son fournisseur.
La découverte est une sorte d'interaction avec l'utilisateur qui doit être supportées par un modèle de description. Ce dernier doit permettre d'accéder et d'interagir de façon automatique et correcte avec les services web.
Plusieurs initiatives ont traité la problématique de découverte des services web dont la plus répandue est celle combinant les deux standards UDDI/WSDL. Cette technique est supportée par la majorité des compagnies industrielles et est déjà implémentée dans plusieurs outils. Une autre initiative académique, issue de la communauté « web sémantique », utilise OWL-S comme support à la découverte **[Pil03]**.

2.1 Le modèle syntaxique basé sur UDDI/WSDL

UDDI est un service d'annuaire pour les services web disponible sur Internet. Il a été conçu pour héberger des informations sur les entreprises et leurs services de façon structurée. Il offre aussi, aux clients, des facilités de recherche des informations publiées concernant les services et leurs fournisseurs.
Typiquement, un service web est publié en enregistrant son document WSDL et des informations concernant son fournisseur dans le registre UDDI. Rappelons que, la structure du registre UDDI engendre quatre types de données définies à l'aide de XML Schema : BusinessEntity, BusinessService, BindingTemplate et TModel .
BusinessEntity : cette structure décrit l'entreprise : son nom, une description, les services offerts et les informations de contact qu'elle met à la disposition des clients pour leur assurer une assistance.
BusinessService : fournit plus de détails sur les services offerts identifiés par une entrée *"ServiceKey"* : le nom du service, sa description, une référence à sa catégorie et une liste de liens (BindingTemplate) décrivant différents points d'accès au service.

BindingTemplate : chaque service peut avoir plusieurs BindingTemplate decrivant chacune un protocole d'accès au service (eg. HTTP, FTP, SMTP, …)
TModel: permet d'associer un service à sa description WSDL et aussi de renseigner le clients sur l'existence d'autres spécifications ou standards que le service utilise.

UDDI fournit aussi des identifiants et des catégories pour classer les entreprises et les services à l'aide de taxonomies et classifications standards comme NAICS **[NAICS]**, UNSPSC…etc .

L'interface de programmation pour les applications (API) dialoguant avec UDDI comprend des méthodes pour interroger le registre ou pour le modifier. Pour l'interrogation, il existe des méthodes de type *"find_xxx"* pour rechercher des informations concernant les quatre structures du registre (business, service, bindingTemplate et TModel). Ces méthodes sont complétées par les *"get_xxxDétail"* qui fournissent une réponse détaillée des données du registre. L'accès au registre se fait grâce à des enveloppes SOAP encapsulées dans des messages HTTP.
La recherche dans le registre UDDI est en trois modes permettant de répondre aux questions suivantes: Qui fournit ce service ? Quel service propose l'organisation ? Comment accéder et utiliser les services web proposés ?

Page Blanche : pour chercher des informations sur une société donnée, son domaine d'activité. En résumé, elle contient toutes les informations utiles pour identifier une organisation.
Page Jaune : donne des informations sur les services web que la société procure par catégorie.
Page Verte : donne des informations sur les protocoles utilisés pour invoquer les services et sur la façon d les joindre.

Ces méthodes permettent d'avoir des détails à propos des entreprises et les services publiés mais nécessitent la connaissance préalable de renseignements techniques tel que l'identifiant (la clé) du service. Il existe heureusement d'autres moyens pour chercher les services sans posséder des identifiants ou d'autres informations techniques. Ces deux moyens sont la recherche à base de mots clés et l'exploration du registre UDDI au moyen des taxonomies.
La classification des services web selon ces taxonomies permettent d'explorer l'espace des services web selon leur type de produit, type d'industrie et région géographique. Malgré que ce moyen offre des facilités de recherche mais les résultats fournis sont grossiers et manquent de pertinence.

D'un autre côté, le registre UDDI offre un mécanisme simple de recherche à base de mots clés similaire à celui utilisé par les moteurs de recherche d'information sur Internet. Récemment, certains moteurs pour la recherche des services web dans le registre UDDI ont été mis en place **[Pil03]**. Ils se basent sur un simple Matching syntaxique entre les mots clés de la requête et les mots appartenant à la description textuelle d'un service donné. Si la description contient les mots clès de la requête, le service est donc retourné en résultat. Par exemple, le site "www.bindingpoint.com" offre un moyen de chercher les services web publiés dans un registre UDDI (voir les

figures 4.1, 4.2, 4.3). La recherche peut être effectuée selon le nom du service, sa clé, son fournisseur, le tModel…etc

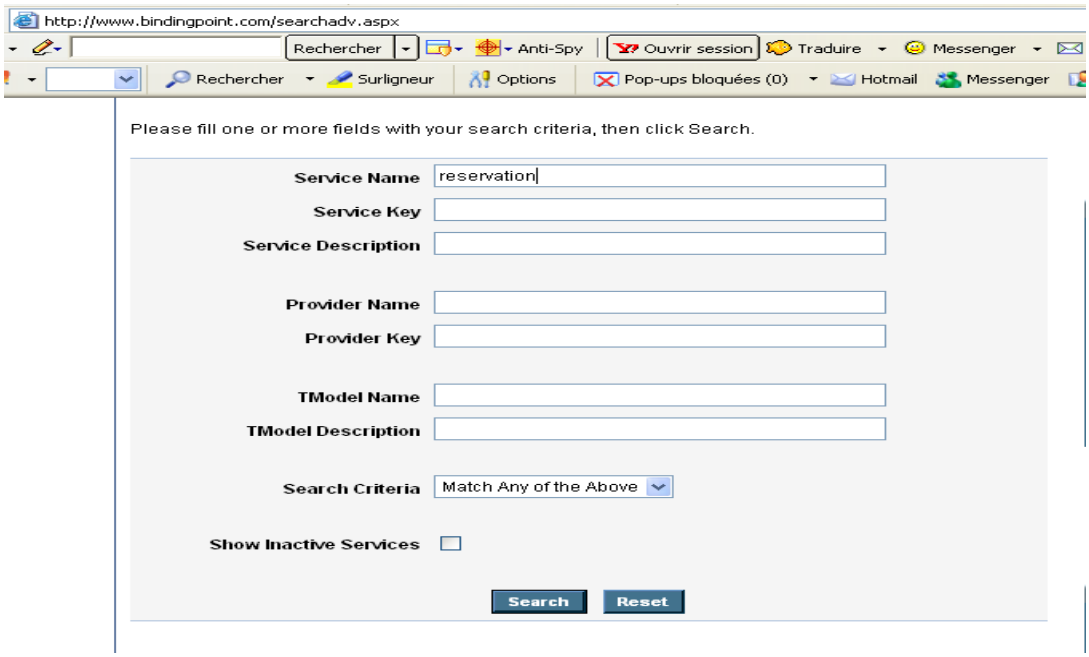


Fig 4.1 : Interface de recherche des services dans le registre UDDI

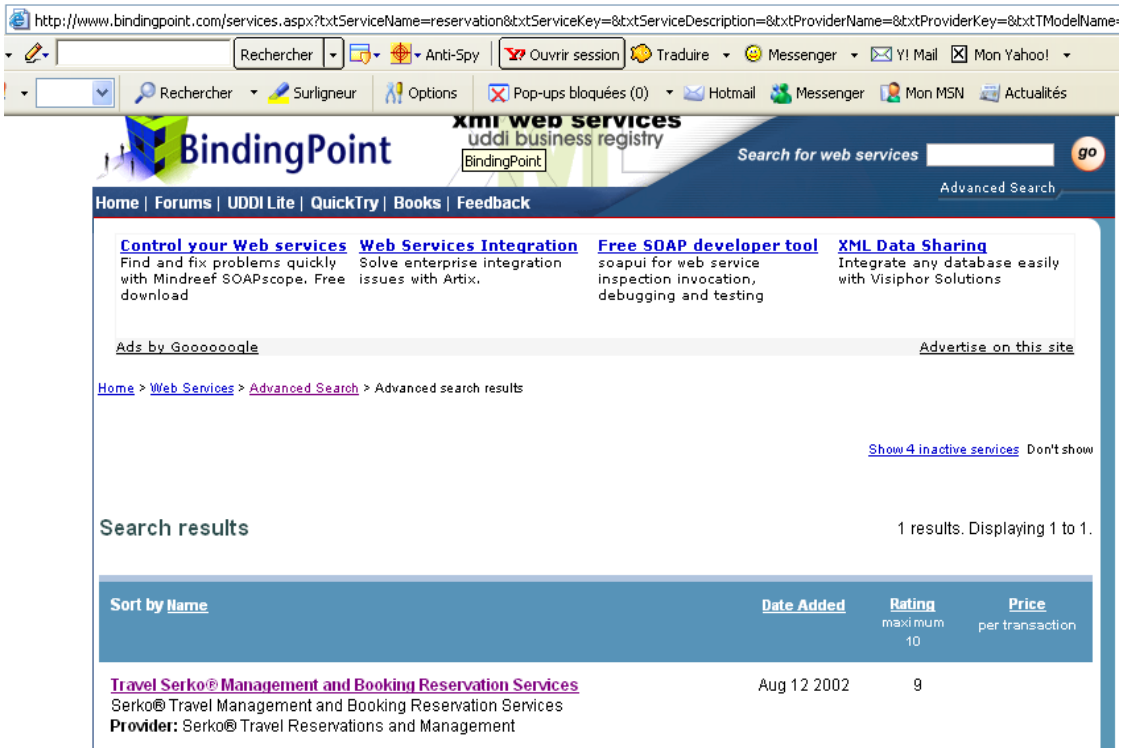


Fig 4.2 : La fenêtre résultat indiquant que le service recherché est trouvé

Fig 4.3 : Document WSDL du service trouvé

Discussion

- Les mots clés ne suffisent pas pour exprimer de façon fidèle le besoin de l'utilisateur en matière de service.
- Le simple Matching syntaxique fournit une manière très pauvre pour capturer les similarités entre deux concepts. Les résultats fournis manquent de précision et de pertinence. Plusieurs services peuvent contenir dans leurs descriptions les mots clés de la requête dont la plupart ne correspondent pas au profil du service désiré. D'un autre coté, de bons services candidats peuvent être rejetés.

D'une manière générale, le mécanisme de découverte supporté par le registre UDDI est relativement primitif. Il ne permet pas, par exemple, de chercher les services qui offrent telle ou telle fonctionnalité ou qui produisent tel ou tel résultat. De plus, UDDI et WSDL utilisent uniquement la spécification XML pour décrire les modèles de données. Le manque de sémantique du langage XML est un autre obstacle pour la description et la découverte effective des services web. Par exemple, deux descriptions XML identiques peuvent signifier deux choses complètement différentes selon le contexte ou elles seront utilisées. Pour cela, une autre initiative issue de la communauté web sémantique a tenté de baser la découverte des services web sur un modèle sémantique qui est le OWL-S.

2.2 Le modèle sémantique basé sur OWL-S (ou DAML-S précédemment)

Cette approche utilise comme formalisme de description des services web le langage OWL-S (DAML-S précédemment) **[OWL-S]**. OWL-S est un langage qui enrichit la description des services web avec des informations sémantiques grâce à des ontologies OWL (respectivement DAML-OIL). OWL-S décrit ce que le service est capable de faire et pas seulement comment il le fait. Comme nous avons vu dans les parties précédentes, une description d'un service web en utilisant OWL-S s'articule à travers trois classes d'informations : *ServiceProfile, ServiceModel* et le *ServiceGrounding.*

Le comportement d'un service peut être représenté en utilisant le *ServiceProfile,* et *ServiceModel*. Tous les deux décrivent les Inputs/Outputs, Pré-conditions et Effets d'un service mais de différentes perspectives. ServiceProfile pour la découverte et la sélection automatique d'un service web et ServiceModel pour contrôler l'interaction avec le service.
Le ServiceProfile spécifie les informations relatives aux capacités d'un service afin de permettre à un demandeur de voir si le service proposé lui convient. Dès lors, le "Service Profile " propose une vue du service web décomposée en trois grands aspects:

1. Une description du service et de son fournisseur. Ainsi, il présentera le nom du service et une petite description textuelle. De même, il décrira son fournisseur en termes de nom, d'adresse physique, d'adresse web, etc.
2. Une description du comportement fonctionnel du service. Deux types d'informations sont utilisés pour caractériser le comportement fonctionnel d'un service: les informations de transformation décrites par un ensemble d'Entrées du service, les "inputs" et d'un ensemble de sorties du service, les "outputs" et les informations capturant le changement d'état produit par l'exécution du service qui sont un ensemble de pré-conditions nécessaires au bon déroulement du service et un ensemble d'effets (post-conditions) de ce dernier sur le monde.
3. Une description des attributs non-fonctionnels utiles à la sélection automatique d'un "Service web". Ainsi, on pourra trouver comme attributs, le temps de réponse, le coût du service, etc.

Dans l'approche OWL-S, la section ServiceProfile est utilisée à la fois par les fournisseurs pour publier leurs services et par les clients pour spécifier leurs besoins. Par conséquent, elle constitue l'information utile à la découverte de services.

L'activité de découverte en utilisant OWL-S ServiceProfile consiste à partir d'un ServiceProfile hypothétique du service désiré par le demandeur, à fournir le service web du fournisseur qui propose un ServiceProfile qui correspond le plus à la description du demandeur. Cette activité est appelée Matching ou Matchmaking. Quelques algorithmes de Matching ont été proposés utilisant des politiques différentes **[DAML-Sb]**. Le plus connu est celui de Paolucci **[Pao02]**. Il utilise les paramètres Inputs/Outputs pour comparer sémantiquement le profile de la requête avec un profile d'un service publié. La recherche de similarité ne dépend pas uniquement des noms des paramètres qui sont seulement des noms symboliques attribués mais plutôt elle dépend essentiellement de la relation de subsomption reliant les concepts que représentent ces noms. En effet, cela

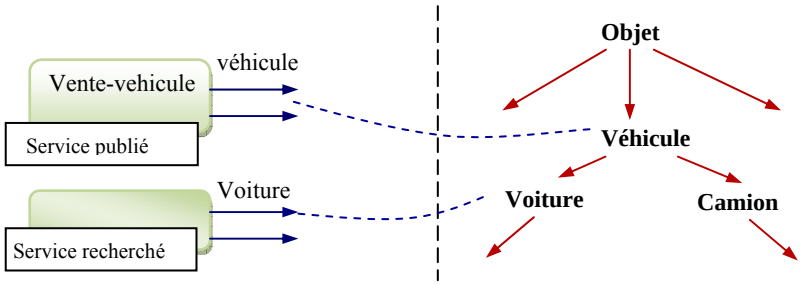
nécessite que les Inputs/Outputs soient accompagnés d'information sémantique permettant de trouver automatiquement le lien de subsomption entre eux. Cette sémantique est généralement supportée par des ontologies.
L'algorithme de mise en correspondance (Matching) reconnaît différents degrés de correspondance qui sont déterminés par la distance minimale entre les concepts dans l'ontologie où ils sont définis.

L'algorithme de Paolucci : L'algorithme évalue les similarités entre un service recherché R et un service publié P en calculant le lien de subsomption entre leurs paramètres d'E/S en utilisant l'ontologie où ils sont définis. L'algorithme reconnaît quatre degrés de correspondance :

- **Exact:** Le paramètre du service R et celui de P (In/OutP et In/OutR) font référence à la même classe dans l'ontologie.
- **Plug-in:** Le paramètre du service R est un sous concept de celui de P.
- **Subsumé:** Le paramètre de R est plus générique que celui de P.
- **Disjoint:** Aucun lien de subsomption entre les deux.

Les cas 'subsumé' est refusé car il satisfait partiellement le besoin et fournit donc un résultat incomplet. Par conséquent, uniquement le cas « Exact et Plug-in » sont retenus.

Exemple : Prenons l'exemple d'un service de vente de véhicule qui possède une sortie spécifiée comme « véhicule » et la requête cherche un service dont la sortie est spécifiée comme « voiture ». Malgré qu'il n'existe pas un Matching exact entre la sortie du service spécifié dans la requête et celle du service publié, en faisant référence au fragment de l'ontologie représentant le domaine (voir la figure 4.4), l'algorithme reconnaît un lien de subsomption de type "plug-in" entre *véhicule* et *voiture*. Donc, le service publié sera retenu.



Véhicule (concept publié) **subsume**
voiture (concept recherché)

Fig 4.4: Exemple de Matching de type subsumé

D'autres propositions ont été ajoutées à ce mécanisme de Matching pour le rendre plus efficace. Ces propositions consistent à prendre en considération dans l'algorithme de Matching d'autres paramètres fonctionnels caractérisant un service. Ces paramètres peuvent être les Pré-Conditions et les Post-Conditions (Effets) qui représentent le contexte d'exécution d'un service. Or, le Matching des conditions n'est pas trivial et nécessite un langage de règle qui permettra de faire des inférences pour pouvoir vérifier

des implications ou des équivalences sémantiques entre les pré-conditions et les post-conditions de deux services donnés.

Discussion :
Bien que le OWL-S se base sur un mécanisme plus intelligent pour la découverte des services web que celui des registres UDDI, Il reste insuffisant et inadapté quand il s'agit de découvrir une combinaison de services web et pas seulement un service unique. Ce besoin est particulièrement essentiel dans le cadre de la composition automatique. De plus, l'utilisateur doit fournir une description qui respecte la spécification OWL-S ServiceProfile de son service désiré, c'est à dire, il exprime sa requête dans le langage de description des services (OWL-S Service Profile) ce qui est contraignant pour un utilisateur non-spécialiste.

3. L'approche proposée

Notre problématique est alors de définir une approche progressive de composition de services web sémantiques en partant d'une requête déclarative simple spécifiée par un utilisateur non-spécialiste. Ce dernier aura simplement à spécifier son besoin en terme de tâches à effectuer, il ne sera pas contraint d'être conscient de tous les détails techniques tels que les services concrets participants et la manière dont ils seront composés et exécutés. La découverte des services participants, leur composition ainsi que leurs interactions doivent être effectuées de façon automatique et transparente par rapport à l'utilisateur. Cela interpelle la mise en place d'un modèle qui distingue différents niveaux d'abstraction (de granularité) des tâches manipulées dans le domaine d'application en se basant sur un support sémantique et un schéma de correspondance qui permettra de passer automatiquement d'un niveau à un autre. Le niveau le plus inférieur du modèle sera les tâches concrètes implémentées réellement par les services web publiés.
Ajoutant à cela, le modèle de composition proposé permettra aussi de générer des plans à deux niveaux de granularités : abstrait et concret. Cette distinction a pour but de favoriser la réutilisation des plans (workflows) déjà construits. En réutilisant les workflows abstraits, la même fonctionnalité peut être satisfaite en ayant comme option de pouvoir changer certains critère d'implémentation et de qualité répondant aux préférences variées de l'utilisateur.

3.1 Support sémantique du modèle proposé

Comme nous pouvons constater, l'organisation et la description sémantique de l'espace des services web est un besoin crucial pour permettre une interaction potentielle entre l'utilisateur et l'ensemble des services web disponibles. Les services web tel qu'ils sont définis et présentés sur le web ne permettent pas à leurs utilisateurs de découvrir les fonctionnalités complexes qu'ils peuvent offrir. Le modèle que nous proposons est basé sur un support ontologique qui distingue différents niveaux de description de l'espace des services dans un domaine d'application donné. Cette distinction a pour but de se rapprocher le plus possible de l'utilisateur en lui offrant une vue simplifiée et plus

naturelle sur les fonctionnalités offertes par l'espace des services web. Par conséquent, un utilisateur possédant des connaissances modestes concernant le domaine d'application et le modèle de recherche utilisé pourra avoir un accès aisé et graduel aux fonctionnalités complexes.
Notre support sémantique se base sur l'utilisation des ontologies. Une ontologie telle qu'elle a été définie par **[Gru93]** est une « Conceptualisation » ou une «abstraction » faite d'un domaine particulier pour identifier les concepts pertinents pour sa description de façon à pouvoir faire des inférences dessus. En effet, le domaine des services web manipule deux types de concepts de natures différentes : les concepts dynamiques représentés par les opérations des services web et les concepts statiques utilisés pour décrire les propriétés des services web. Pour cela, nous distinguons deux parties essentielles dans l'ontologie proposée: la première concerne la description des opérations du domaine et la deuxième concerne la description hiérarchique des concepts statiques pertinents du domaine d'application.

3.1.1 Les opérations

Un service web est considéré comme une interface décrivant un ensemble d'opérations ou de fonctionnalités reliées qui peuvent être accédées automatiquement à travers le web. Exécuter un service web consiste à invoquer ses opérations en envoyant et recevant des messages. L'opération constitue donc l'élément clé dans la structure d'un service web.
Notre représentation sémantique de l'ensemble des opérations distingue trois abstractions différentes des opérations de domaine: Opération générique, opération abstraite et opération concrète. Cette distinction a pour but d'élever le niveau d'abstraction de telle manière que l'utilisateur pourra spécifier sa requête en termes de tâches à accomplir et non pas en termes de services qu'il doit invoquer (voir la figure ci-dessous).

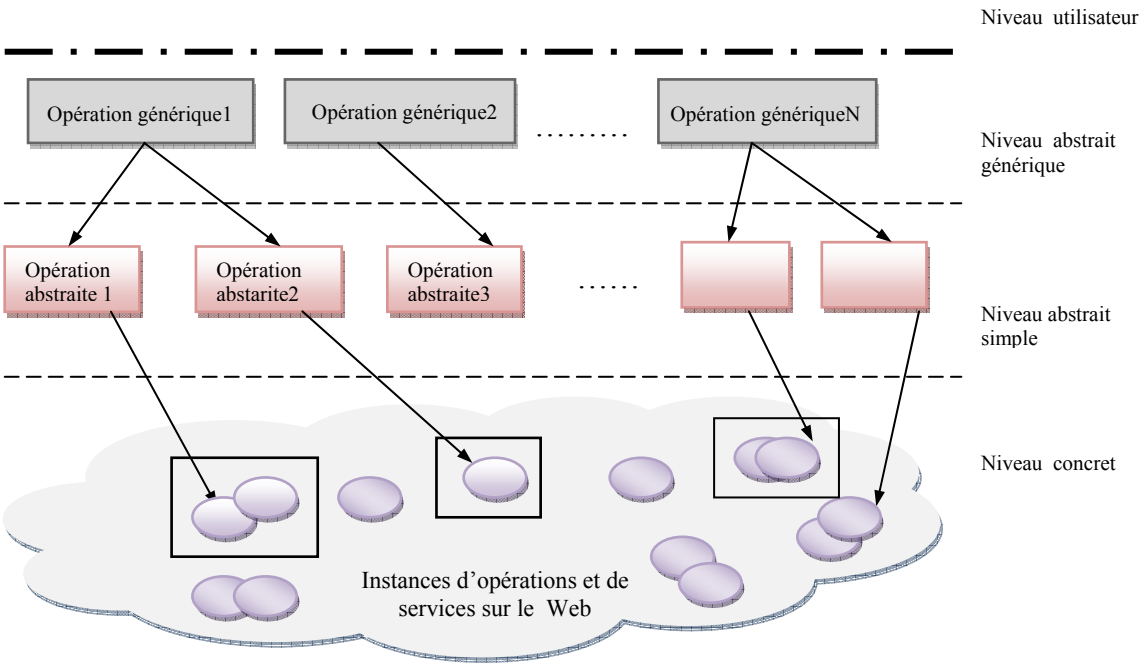


Fig 4.5: Les trois niveaux d'abstraction des opérations du domaine

a. Opération générique

Une opération générique est une opération ou tâche abstraite qui peut être décomposée ou non en opérations plus simples. Les opérations génériques représentent les traitements métiers les plus fréquemment sollicités dans un domaine d'application donné. Par conséquent, elles sont définies par les entités expertes du domaine dans le but d'offrir une vue générique d'un traitement métier qui peut faire appel à plusieurs fonctionnalités complémentaires. Le terme abstrait est utilisé ici pour signifier qu'aucune information d'implémentation n'est fournie dans la description de ces opérations et donc n'appartiennent à aucun service web existant.
Une opération générique est décrite formellement grâce aux attributs : *catégorie*, *fonction*, *description* et *paramètres d'entrée/sortie* (voir la figure ci-dessous).

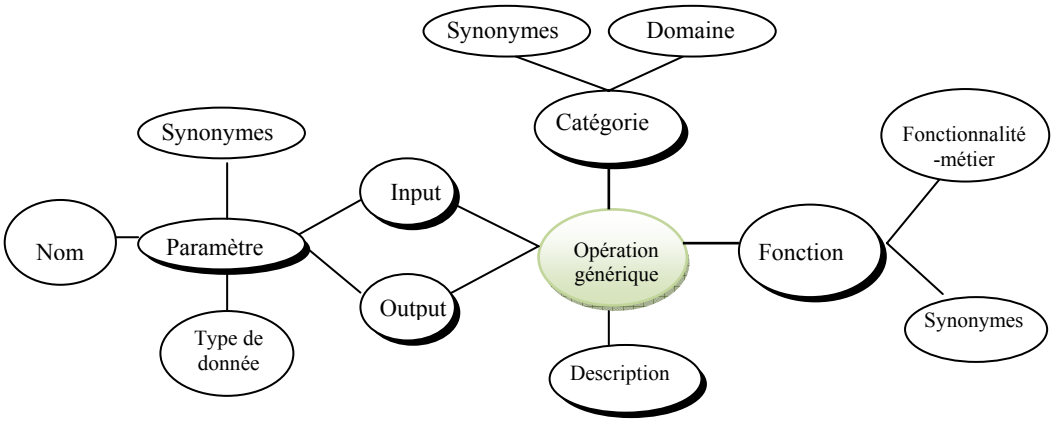


Fig 4.6: Description d'une opération générique

Catégorie : est définie par le tuple (*domaine, synonymes*), l'attribut domaine représente le centre d'intérêt de l'opération par exemple "santé", "vol". L'attribut *synonymes* donne tous les noms qui désigne le même domaine d'intérêts. Par exemple : "*voyage*", "*séjour*", "*tourisme*" peuvent signifier le même domaine d'intérêt dans une application particulière.
Fonction : est décrite par le tuple (*fonctionnalité, synonymes*). L'attribut *fonctionnalité* représente la fonctionnalité métier pour laquelle l'opération est dédiée, par exemple: "réservation", "allocation". L'attribut *synonymes* donne toutes les alternatives de noms désignant la même fonctionnalité métier, par exemple : "reservation" et "booking" en Anglais désigne la même fonctionnalité métier.

Plusieurs taxonomies peuvent être utilisées pour définir les attributs décrivant la catégorie et la fonction d'un processus métier. Un exemple de telles taxonomies comprend: NAICS **[NAICS]**, RosettaNet **[RosettaNet],** cXML(commerce XML) **[cXML]** et EDI (Electronic Data Interchange) **[EDI]**...etc. Chaque attribut est préfixé par l'espace de nom XML qui renvoie vers la taxonomie correspondante. Exemple: RosettaNet Business Dictionary fournit le vocabulaire qui peut être utilisé pour décrire des propriétés métier comme : le nom métier (business name), adresse, identifieur de taxe (tax identifier). RosettaNet Technical Dictionary contient les propriétés qui peuvent

être utilisées pour décrire les caractéristiques d'un produit (les composants d'un ordinateur par exemple) ou d'un service (ordre d'achat, ordre de réservation).

Description : est un texte en langage naturel résumant ce que fait l'opération. La description peut être utile pour fournir des informations à l'utilisateur humain à propos de l'opération.

Paramètres d'entrées/sorties : Très souvent, décrire une fonctionnalité en fournissant seulement son nom est très confus. Pour cela les paramètres d'Entrées/Sorties (E/Ss) permettent de compléter cette description de façon plus précise. Ils permettent de décrire une opération d'un service web comme étant une fonction qui possède des paramètres en Entrée et des paramètres en Sortie. Chaque paramètre d'Entrées/Sorties est décrit grâce au tuple (*nom-paramètre, synonymes, type*). L'attribut *type* décrit l'intervalle de valeur dans lequel le paramètre prendra ces valeurs. Les valeurs ne sont pas forcément de type primitif mais peuvent être des objets complexes appartenant à des ontologies prédéfinies.

Nous donnons comme exemple, l'opération de réservation de vol qui peut être décomposée en plusieurs sous opérations :

```
Catégorie:[flight-opération]
Fonction :[flight-booking, flight-reservation].
Inputs:   [departure-date], [return-date], [from,departure-city], [to,
            arrival-city].
Outputs:  [airline], [flight-Nr], [Seat-Nr].
```

b. Opération abstraite

Différents services web sont en concurrence pour offrir des fonctionnalités similaires, par exemple, plusieurs services de lignes aériennes peuvent offrir la même connexion entre deux villes, mais ils diffèrent dans la manière dont ils offrent cette fonctionnalité et aussi la manière dont ils l'implémentent: des contraintes sur certains paramètres, la qualité avec laquelle le service délivre la fonctionnalité et les protocoles utilisés pour l'accès effectif aux service (Binding). Pour cela, nous distinguons deux types d'opérations permettant de faire cette distinction : la fonctionnalité abstraite et la fonctionnalité concrète appartenant à tel ou tel service.

Une opération abstraite est donc une opération qui décrit de façon abstraite une fonctionnalité qui peut être implémentée par plusieurs opérations concrètes appartenant à des services différents. Elle possède une description similaire à celle d'une opération générique à savoir la *catégorie*, la *fonction*, une *description* et *les paramètres d'Entrées/Sortie.*

Contrairement à une opération générique, une opération abstraite n'est pas décomposable en d'autres sous-opérations mais elle est directement instanciée en opération(s) concrète(s). De plus, les opérations génériques servent à définir les Templates génériques que nous allons expliquer plus loin. Chaque opération générique sera considérée lors du processus de résolution de la requête comme une sous-requête de composition qui sera mise en correspondance avec une combinaison d'opérations abstraites. Sa résolution constituera donc un sous-plan du plan global.

Nous citons comme exemple, ces deux opérations dont la combinaison produit l'opération générique citée auparavant (flight-reservation):

Première opération:

```
Catégorie : [flight-opération]
Fonction :  [flight-lookup, serachFor-flight].
Inputs:     [departure-date], [arrival-date], [from, origin, departure-
            city], [to,  destination, arrival-city].
Outputs:    [airline], [flight-Nr].
```

Deuxième opération:

```
Catégorie : [flight-opération]
Fonction :  [buy-flightTicket].
Inputs:     [price], [credit-card-Nb], [flight-Nb].
Outputs:    [buyTicket-confirmation], [..].
```

c. Opération concrète

C'est une opération appartenant à un service web concret et disponible sur le web, donc elle possède une existence physique, c'est à dire, une implémentation réelle de sa fonctionnalité. Chaque opération concrète est une instance d'une certaine opération abstraite.
L'intérêt de séparer la définition d'une opération concrète de celle d'une opération abstraite est de permettre de générer des plans de composition à deux niveaux de granularité : un plan abstrait et un plan concret de composition.
Une opération concrète hérite la même description que celle d'une opération abstraite avec quelques extensions décrivant les détails d'implémentation et d'autres informations décrivant sa qualité. Les détails d'implémentation concernent les informations de *Binding (liaison)*.

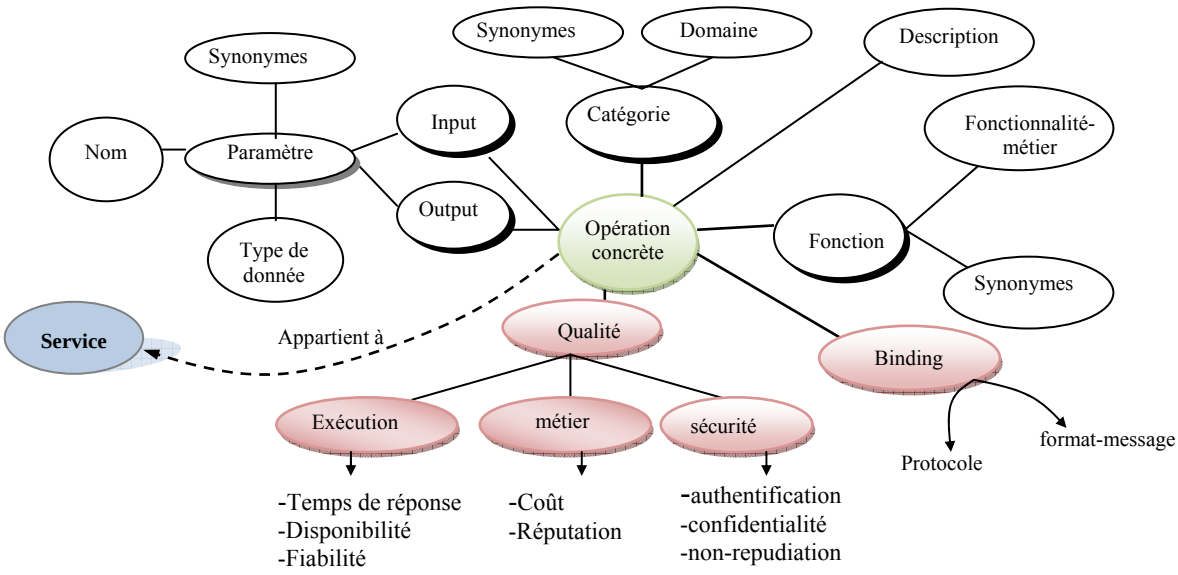


Fig 4.7: Description d'une opération concrète

L'attribut *binding* définit le format des messages et les protocoles réseaux utilisés pour interagir avec l'opération. Une opération concrète d'un service web peut être accessible à travers différent liaisons (Binding) comme SOAP/HTTP ou SOAP/MIME.

La qualité d'un service web ou d'une opération de service web définit la manière dont le service web ou l'opération délivre ses fonctionnalités aux utilisateurs. Plus concrètement, la qualité de service mesure les paramètres quantitatifs caractérisant les traitements offerts par un service (ou une opération de service). Les utilisateurs peuvent imposer leurs préférences concernant la qualité du service qu'ils cherchent. Les informations décrivant la qualité d'une opération (ou d'un service web) recouvrent trois aspects: l'exécution, la sécurité et la qualité métier (voir la tableau ci-dessous).

Groupe QdWS	Attribut QdWS	Définition
Exécution	Temps de réponse	Mesure la durée totale d'exécution grâce à la formule suivante : $T_{trait}(op)+T_{rés}(op)$ où T_{trait} est le temps nécessaire du traitement fait par l'opération et $T_{rés}$ est le temps nécessaire pour la transmission ou de réception des résultats.
	La fiabilité	l'aptitude d'une opération d'être exécutée avec succès. Elle est mesurée en se basant sur des données obtenues à partir d'un historique des invocations déjà réalisées de l'opération, en utilisant l'expression suivante : $Nb_{succès}(op)/Nb_{total}(op)$ où $Nb_{succès}$ représente le nombre de fois où l'opération a été invoquée avec succès et Nb_{total} représente le nombre total des invocations réalisées.
	Disponibilité	représente la probabilité que l'opération soit accessible. Elle est mesurée grâce à l'expression suivante : $T_{disp}(op)/T_{total}(op)$, où $Temps_{disp}$ l'intervalle de temps durant lequel l'opération a été disponible et $Temps_{total}$ est l'intervalle de temps total pris pour l'évaluation.
Sécurité	Authentification	un booléen égal à vrai si le consommateur du service (utilisateur ou autre service) doit être authentifié
	Non-répudiation	un booléen égal à vrai si les participants (client ou fournisseur) ne doivent pas renier l'utilisation ou la livraison de l'opération
	Confidentialité	donne la liste des paramètres d'E/Ss qui ne doivent pas être divulgués aux entités externes (autres que le fournisseur du service).
Métier	Coût	donne la somme d'argent (en dollar) à payer pour exécuter l'opération.
	Réputation	représente, comme son nom l'indique, la réputation de l'opération. Elle dépend des expériences des utilisateurs concernant l'invocation de l'opération. Chaque utilisateur donne une valeur dans un intervalle donné qui représente le taux de confiance qu'il donne à l'opération. La réputation correspond à la moyenne des taux collectés sur l'opération.

Tableau représentatif des différents paramètres de QdWS

La description des opérations concrètes est, en fait, une extension sémantique de la description WSDL des opérations des services web. Elle est attachée au registre UDDI via l'entrée OverviewDoc de la classe tModel qui permet de faire référence à d'autres descriptions, autre que la spécification WSDL, que le service utilise (voir le schéma ci-dessous).

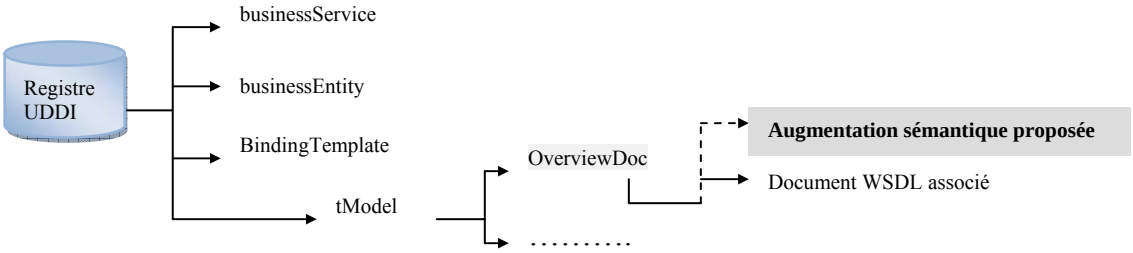


Fig 4.8: Attachement de l'extension sémantique proposée au registre UDDI

3.1.2 Les concepts statiques

Les ontologies sont des méthodes très intéressantes et efficaces pour décrire des domaines vastes d'une façon permettant de traiter automatiquement leurs informations.
L'ontologie des concepts que nous proposons est une description hiérarchique de l'ensemble des concepts statiques pertinents du domaine. La relation qui est essentielle dans le contexte de notre approche est la relation de subsomption. Cette relation permet de relier les concepts les plus spécifiques à leurs concepts génériques et de bâtir un mécanisme d'héritage entre eux. Les caractéristiques du concept générique sont héritées par leurs concepts spécifiques.

Plusieurs autres noms ont été attribués à la relation de subsomption : "is a", "subset of", "member of", "subconcept of", "superconcept", "subsume".
Les concepts statiques de notre modèle sont ceux utilisés pour décrire les propriétés (les paramètres) des services web, par exemple, les E/Ss d'une opération doivent avoir une représentation sémantique qui explicite leurs liens de subsomption afin de pouvoir vérifier si deux opérations ayant des E/Ss différents sont similaires pour être substituées ou compatibles pour être composées ensemble.

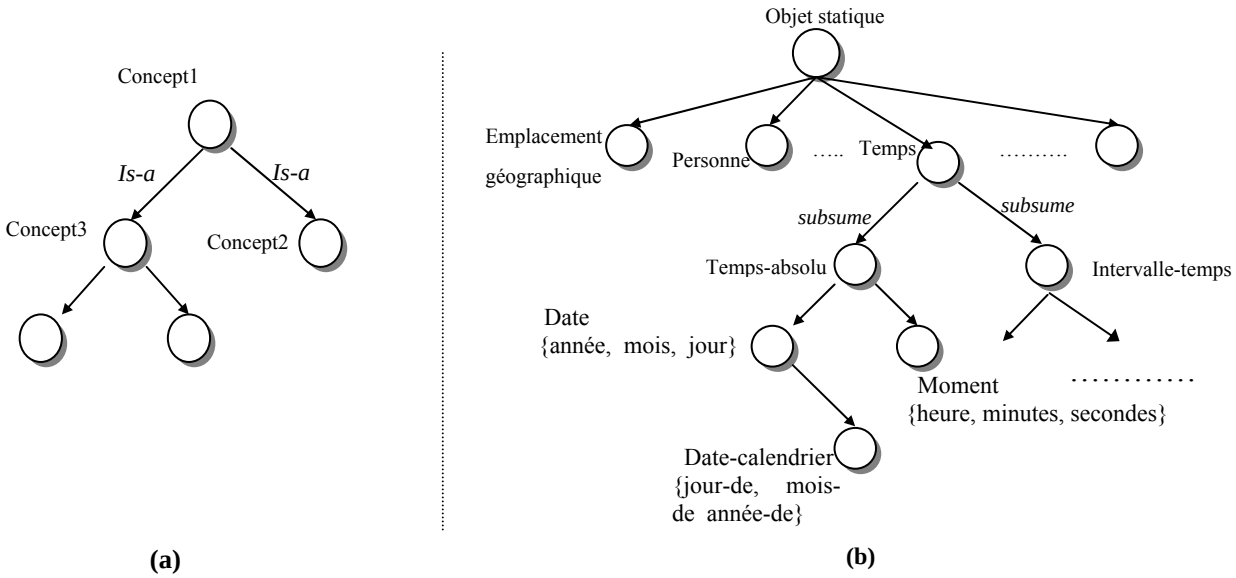


Fig 4.9 : La relation de subsomption dans une ontologie

On dit que le *concept1* ***subsume*** le *concept2* et le *concept3 est* ***subsumé*** par le *concept1,* c'est à dire, que la description de *concept1* est plus générique que la description de concept2 et concept3 (fig 4.9 - (a)) au sens où l'ensemble des individus représentés par Conept1 contient l'ensemble des individus représentés par Concept2 et concept3. On dit aussi que le concept3 et 2 sont disjoints c'est à dire qu'il y a aucune relation de subsomption entre eux.
L'exemple que montre la figure 4.9 - (b) décrit une portion d'une ontologie représentant l'entité « Temps ». On dit par exemple que 'temps absolu' subsume 'date (année, mois, jour)' et moment (heure, minute, seconde).

L'ontologie définie peut être construite à l'aide du langage ontologique OWL ou DAML-OIL qui sont bâtis au-dessus de RDF et RDF Schéma. Ils permettent de concevoir une ontologie en terme de classes de concepts, de propriétés propres aux concepts et de relations entre les classes et entre les propriétés.

3.2 Les différents niveaux de traitement de la requête de composition

Afin de permettre à un utilisateur d'accéder graduellement, à travers une requête simple, aux fonctionnalités complexes offertes par une combinaison de services web, nous proposons différents niveaux de modélisation du plan de composition Ces derniers constitueront les différentes phases de résolution de la requête de composition de sa simple spécification par l'utilisateur jusqu'à la génération du service composite associé.

3.2.1 Requête utilisateur

L'utilisateur peut ne pas avoir une idée complète et assez précise concernant le service qu'il cherche. Ainsi, la précision d'une formulation de requête peut varier d'un utilisateur à un autre. Nous allons traiter deux cas majeurs de requête qui peuvent amener à une composition de services:

- Premier cas : il spécifie son besoin en terme des sous-tâches avec leurs paramètres d'entrées/sorties (Voir la fig 4.10) qu'il veut exécuter de façon dépendante et suivant une certaine logique métier. Ces sous tâches feront très probablement parties d'un processus complexe.

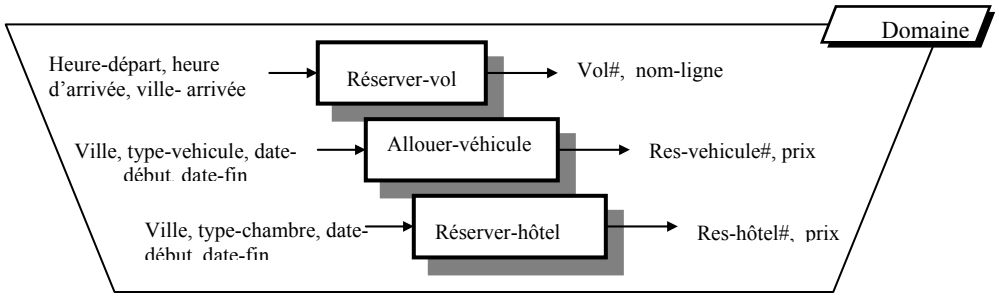


Fig 4.10 : Premier cas de requête

- Deuxième cas : il spécifie son besoin en terme d'une tâche unique. Cette tâche peut être satisfaite par un service unique comme elle peut faire appel à une composition de services simples. Or, l'utilisateur ne peut pas savoir si son besoin fait appel à un service complexe. C'est le système qui décide sur la nécessité de procéder à une composition si la demande n'a pas pu être satisfaite par un service simple.

L'utilisateur dans ce cas spécifie son besoin comme une boite noire qui accepte des entrées et fournit des sorties c'est à dire, il donne le nom de la tâche qu'il veut exécuter avec les données d'Entrée et les Sorties qu'il veut obtenir (Voir la fig ci-dessous).

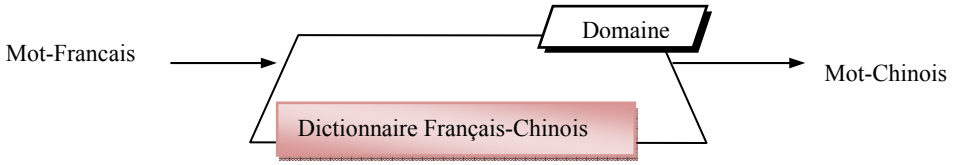


Fig 4.11: Deuxième cas de requête

Dans le cas général, nous considérons la requête comme étant composée de: Nom du domaine sur lequel il veut faire sa recherche, une ou plusieurs sous requêtes (sous-tâches) avec leurs noms de paramètres d'E/Ss, une valeur entière entre 0 et 1 représentant le seuil de correspondance qu'il précise lui même, un ensemble de conditions et un profile dans lequel l'utilisateur spécifie ses préférences.

$$Requête :- nomDomaine, \cup_{i=1}^{n} Sous\text{-}tâche_i (Nom, In, Out) , seuil, \cup_{j=1}^{k} Condition_j(In, Out), profile.$$

Condition: Représente des conditions sur les variables apparaissant dans la requête. Elles sont utilisées pour affecter des valeurs à ces variables lors de l'invocation des opérations appropriées. La forme générale de ces conditions est la suivante : X *op* C, où x est un paramètre d'entrée ou de sortie apparaissant dans la requête, C est une valeur constante et *op* est un opérateur $\in \{ =, \neq, >, \geq, <, \leq \}$. Cette partie sera utile lors de l'invocation effectives des opérations
Profile: exprime les préférences de l'utilisateur en terme de qualité de service qu'il veut avoir. Sa forme générale est la suivante : PQ= val, où PQ est le nom du paramètre de qualité et val est sa valeur.
La sections *Profile* et sera prise en considération à la dernière phase du processus de composition (voir plus loin).

3.2.2 Templates génériques

Les Templates génériques constituent la première représentation interne de la requête. Elles sont considérées comme des processus métiers connus dans un domaine d'application particulier. Elles sont conçues au-dessus des opérations génériques et sont supposées prédéfinies dans le domaine d'application.
Dans certains travaux **[Sir05],** les auteurs ont montré l'importance d'utiliser les templates workflow pour l'encodage des règles métiers d'une application B2B et pour la définition d'un modèle de processus dans lequel les activités sont définies en termes de descriptions abstraites qui seront instanciées et liées aux services concrets à l'exécution pour répondre à un besoin donné.
Pour les mêmes raisons, nous définissons les templates génériques mais à un niveau plus générique que ceux utilisés dans **[Sir05], [Cas00]**. Elles permettront, donc, d'offrir une vue générale et simplifiée d'un processus métiers complexes du domaine d'application et un haut niveau de réutilisabilité. En particulier, elles seront utilisées aussi pour définir un plan initial de composition.
Citons des exemples de processus très souvent sollicités qui peuvent être modélisés grâce à des templates génériques: organisation d'un voyage, achat de véhicule… etc

Définition : La combinaison de plusieurs opérations génériques au sein d'un même processus métier suivant un modèle d'orchestration constitue ce que nous appelons "une template générique". La spécification d'une template générique regroupe deux aspects majeurs : l'orchestration et la description.
L'orchestration : L'orchestration au niveau d'une Template générique donne l'ordre d'exécution des opérations génériques et les conditions sous lesquelles elles s'exécuteront. Les modèles d'orchestration de processus proposés dans la littérature sont nombreux comme les diagrammes d'activité d'UML, les réseaux de Petri, les diagrammes d'état (statechart), le diagramme de séquence et le π-calcul. Nous avons opté pour les diagrammes d'activités d'UML pour modéliser l'orchestration des opérations génériques dans une Template générique pour leur simplicité et leur facilité d'utilisation. De plus, plusieurs outils sont disponibles pour concevoir les processus métier utilisant les diagrammes d'activité d'UML.
Un diagramme d'activité ne modélise pas l'exact comportement interne d'un processus (comme le fait le diagramme des séquences) mais montre plutôt les traitements et les étapes généraux à un haut niveau d'abstraction. Par conséquent, Il offre une

représentation très lisible et naturelle d'un traitement complexe pour un utilisateur ordinaire.
Les diagrammes d'activité d'UML montrent le flux d'activité dans un processus complexe. Dans notre cas, les activités représentent les opérations génériques. Chaque opération générique constituera une sous-requête (ou sous-traitement) de la requête globale.
L'orchestration dans une template générique = orchestration inter-opération générique.
Exemple de la figure ci-dessous montre un diagramme d'activité simple décrivant la tâche de planification d'un voyage en terme de trois opérations génériques:

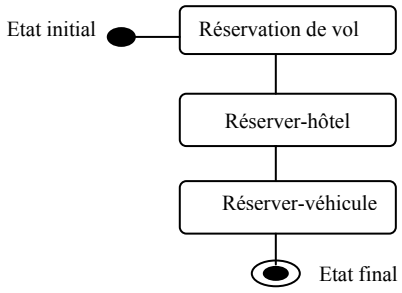


Fig 4.12: Orchestration d'une Template générique grâce à un diagramme d'activité

Au niveau d'une Template générique, nous précisons quels sont les paramètres de chaque activité et l'écoulement des données entre les différentes activités (opérations).

Description : Les diagrammes d'activité permettent la spécification de l'ordre d'exécution des opérations génériques à l'intérieur d'une Template générique. Mais ils ne fournissent aucun moyen pour décrire une Template générique du point de vue syntaxique ou sémantique. La description d'une Template générique se base sur le support ontologique défini pour les opérations génériques.
La description d'une Template générique produira un document XML (voir la figure) :

```
<Template>
  <Fonction> .......
    <nom>....</nom>
    <synonymes> ....... </synonymes>
  </Fonction>
  <description> ....... </description>
  <Paramètres>..... <Paramètres>
  <Op-générique1>
      <Catégorie>.......
          <domaine> ....... </domaine>
          <synonymes> ....... </synonymes>
      </Catégorie>
      <Fonction> .......
          <fonctionnalité-métier>....</fonctionnalité-métier>
          <synonymes> ....... </synonymes>
      </Fonction>
      <paramètres> .......
          <Input> .......
            <Nom> ....... </Nom>
            <type> ....... </type>
            <synonymes>.......</synonymes>
          </Input>
          <Output>.......
            <Nom> ....... </Nom>
```

```
            <type> ....... </type>
            <synonymes>.......</synonymes>
          </Output>
      </paramètres>
      <description> ....... </description>
  </Op-générique1 >
.......
</Template>
```

Fig 4.13: Description d'une Template de composition

Exemple : « Air Travel Planning Template »

```
Fonction :  [Air Travel planning, Air travel organisation].
Inputs:     [departure-date], [return-date], [from, origin, departure-
            city], [to, destination, arrival-city].
Outputs:    [airline], [flightNumber], [hotelName],
            [hotelReservationNumber], [carResevationNumber], [carType]
Opérations-génériques: [Flight Booking]{……}
                       [Hotel Reservation] {……}
                       [Car Rental] {……}
```

3.2.3 Plan abstrait de composition

Représente le deuxième niveau de traitement de la requête de composition. Un plan abstrait de composition est une agrégation d'un ensemble d'opérations abstraites suivant un modèle d'orchestration y compris le flux de données. Il peut être considéré aussi comme un Workflow non exécutable et il est généré de façon automatique à partir d'une Template générique. Le passage d'une Template générique au plan abstrait se fait grâce à un processus de mise en correspondance de chaque opération générique avec une opération abstraite ou une combinaison d'opérations abstraites et établir le flux de contrôle et de données entre elles.
Au niveau du plan abstrait, on distingue deux types d'orchestration : l'orchestration pré-établies dans la template générique (entre opérations générique) et l'orchestration générées automatiquement par le système lors de la génération des sous-plans. Cette dernière établit l'ordre d'exécution des opérations abstraites participantes appartenant aux sous-plans associés à chaque opération générique. Ce type d'orchestration est appelé orchestration intra-opération générique.
L'orchestration dans le plan abstrait= orchestration inter-opération générique + orchestration intra-opération générique.

Lors de la construction du plan abstrait, on ne considère que les attributs décrivant la sémantique de l'opération tels que la catégorie, la fonction et les attributs fonctionnels représentés par les Entrées /Sorties. Tous les attributs décrivant les détails d'implémentation et la qualité d'une opération (ou d'un service) seront considérés ultérieurement lors de la génération du plan concret.
La description d'un plan abstrait se fait grâce à un langage de spécification de workflow abstrait tel que BPEL4WS Abstract Process **[BPEL4WS]**. En réutilisant les workflows abstraits, la même fonctionnalité peut être satisfaite en ayant comme option de pouvoir changer l'implémentation des services participants et certaines préférences en terme de qualité du service composite résultant.

3.2.4 Plan concret de composition

Un plan concret de composition est une agrégation d'un ensemble d'opérations concrètes suivant un modèle d'orchestration définissant le flux de contrôle et le flux de données entre les opérations participantes. Il est généré automatiquement à partir du plan abstrait. Chaque opération abstraite du plan abstrait sera instanciée en une opération concrète grâce au processus de Matching (mise en correspondance). Plusieurs plans concrets peuvent être générés car plusieurs opérations concrètes peuvent implémenter la même fonctionnalité décrite dans une opération abstraite. Pour cela, la mise en correspondance entre une opération abstraite et une opération concrète considère d'autres informations concernant l'implémentation (ex : type de Binding) et la qualité d'une opération (ex: le coût, le temps de réponse). Ces informations seront nécessaires pour sélectionner parmi plusieurs services celui qui offre les meilleures performances de qualité et répondant au mieux aux préférences de l'utilisateur. Quelques valeurs de ces attributs peuvent être fournies dans le profile des préférences de l'utilisateur et d'autres attributs de qualité seront maximisés ou minimisés de façon automatique. Par exemple, le coût et le temps de réponse sont des attributs négatifs qui seront minimisés, par contre la disponibilité et la réputation sont des attributs positifs qui seront maximisée.

3.3 La résolution de la requête de composition

Suite à la prolifération rapide des services web, plusieurs fournisseurs offrent des fonctionnalités similaires de services web mais de manière différente. Les différences peuvent être sur les paramètres d'entrées nécessaires, les paramètres de sortie retournés, la qualité exhibée par le service, etc. Par conséquent, il n'est pas toujours possible de trouver une mise en correspondance exacte entre les besoins de l'utilisateur et les services web disponibles. Pour cela, nous avons basé notre modèle sur un schéma de mise en correspondance flexible qui accepte des résultats approximatifs mais qui restent pertinents vis-à-vis du besoin de l'utilisateur.

3.3.1 Schéma de correspondance sémantique (Matching sémantique) :

La mise en correspondance sémantique est un mécanisme qui vise à trouver les similarités sémantiques et éventuellement structurelles entre deux types d'informations : l'information recherchée et l'information fournie par le système (publiée) et ceci en utilisant un support sémantique. Notre schéma de correspondance se base sur l'algorithme de Paolucci **[Pao02]**. Cependant nous l'avons modifié pour qu'il devienne plus flexible dans le sens où il n'est pas nécessaire que toutes les E/Ss du service recherché soient mises en correspondance avec toutes les E/Ss du service publié. Nous avons besoin uniquement de faire correspondre toutes les sorties du service recherché à toutes ou quelques sorties du service publié (car un service qui produits des résultats en plus est retenu), et faire correspondre toutes ou quelques entrées du service recherché à toutes les entrées du service publié (car on suppose qu'un service donné ne peut s'exécuter si l'une de ses entrées n'est pas disponible). De plus, notre schéma utilise trois types d'information: les paramètres d'E/Ss, les noms des tâches et les noms de domaines.

La mise en correspondance survient à trois différents niveaux du processus de résolution de la requête (voir la fig 4.14):

- La mise en correspondance des tâches spécifiées par l'utilisateur à l'ensemble des opérations publiées (simple ou générique) ou à des Templates génériques prédéfinies.
- La mise en correspondance d'une opération générique avec une combinaison d'opérations abstraites appelée la *génération des sous plans abstraits*.
- La mise en correspondance d'une opération abstraite à une ou plusieurs opérations concrète appelée *l'instanciation des opérations abstraites* pour construire le plan de composition concret.

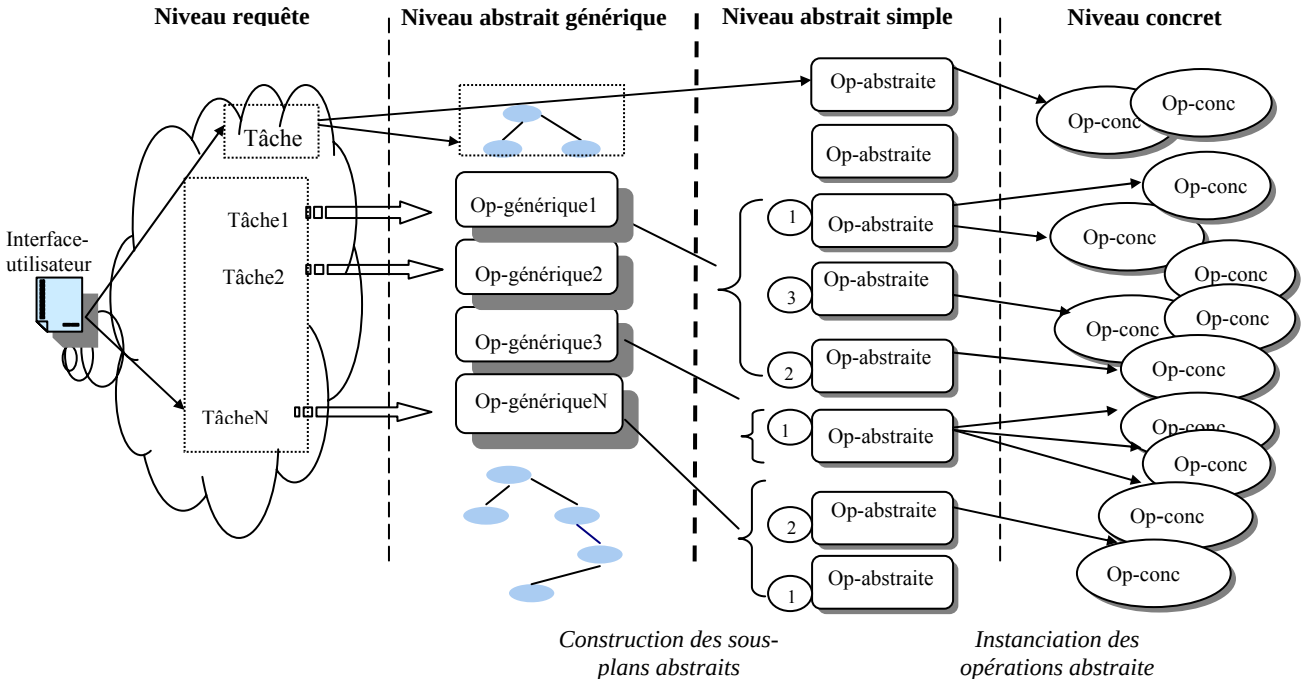


Fig 4.14: Les différents niveaux de mise en correspondance.

La dernière phase du processus consiste à générer la description détaillée du service composite dans un langage de spécification tel que le BPEL4WS (*Business Process Execution Language for Web Services*) , WSFL (Web Service Flow Language) ou alors XLANG **[Xlang]**. Cette spécification sera soumise directement à un moteur d'exécution de workflow tel que le BPWS4J **[BPWS4J]** ou le BPEL Orchestration Server **[BPEL]**. L'exécution effectue les invocations des services web partenaires appropriés. D'habitude, Les moteurs d'exécution de workflow fournissent les outils nécessaires pour contrôler l'exécution d'un workflow.

1) Modes de mise en correspondance:

a) Mise en correspondance horizontale : Ce type de mise en correspondance est nécessaire lorsqu'on a besoin de vérifier si deux opérations OP1 et OP2 peuvent être composées ensemble (enchaînées) afin d'obtenir une fonctionnalité plus complexe. Ceci consiste à faire correspondre les sorties de la première opération aux entrées de la

deuxième (voir la figure ci-après). Par exemple : enchaîner deux opérations abstraites lors de la construction des sous plans abstraits.

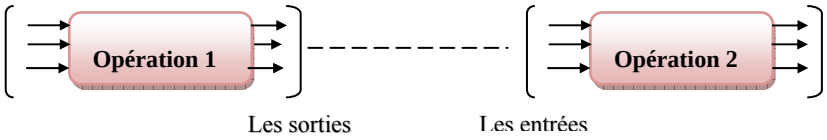


Fig 4.15: Mise en correspondance horizontale

b) Mise en correspondance verticale 1:1: Cette mise en correspondance est nécessaire lorsqu'on veut vérifier si deux opérations OP1 et OP2 ont des fonctionnalités similaires, c'est-à-dire, si l'opération OP1 peut être substituée par l'opération OP2. Par exemple, ce type de mise en correspondance est appliqué lorsqu'on veut substituer une opération recherchée par une seule opération publiée(opération abstraite à une opération concrète).

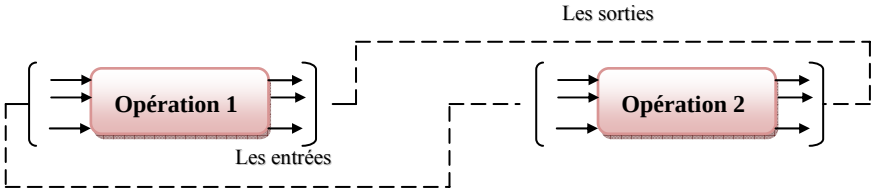


Fig 4.16: Mise en correspondance verticale 1:1

c) Mise en correspondance verticale 1:N : Cette mise en correspondance est nécessaire lorsqu'on veut vérifier si une opération OP1 peut être substituée par un ensemble d'opérations simples enchaînées tel que les entrées de l'opération recherchée (OP1) correspondent sémantiquement aux entrées de la première opération de la chaîne et ses sorties correspondent sémantiquement aux sorties de la dernière opération de la chaîne. Par exemple, mettre en correspondance une opération générique à une combinaison d'opérations abstraites.

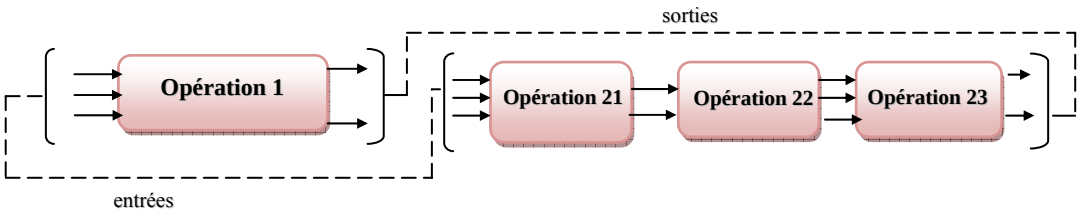


Fig 4.17: Mise en correspondance verticale 1:N

2) Règles de mise en correspondance:

En général, notre schéma de mise en correspondance sémantique considère trois types d'informations en se referant à la représentation ontologique déjà définie :

- Les domaines : cela est considéré comme un filtre initial qui délimite l'espace de recherche des opérations candidates.
- Les paramètres d'Entrées/Sorties de l'opération recherchée et ceux d'une opération publiée (simple ou générique)
- Les noms des tâches ou les noms des opérations publiées (simple ou générique).

Nous calculons, en même temps, le degré de correspondance qui représente le degré de similarité entre la requête ou une opération recherchée avec une Template générique ou une opération publiée (respectivement). Ce degré qui peut varier de *« exact »* à « *faible* » nous aidera à décider quel résultat accepter pour répondre au mieux à la demande de l'utilisateur. En effet, le résultat qui sera retenu est celui possédant un degré supérieur au seuil précisé par l'utilisateur et le plus élevé par rapport aux autres.

Nous définissons dans ce qui suit les règles de correspondance entre les paramètres d'entrées/sorties, les noms d'opération et les noms de domaine.

Mise en correspondance des domaines : On dit que le nom de domaine spécifié dans la requêtes (nomDomaine) correspond sémantiquement au domaine d'une opération publiée OP_p(simple ou générique) si nomDomaine=*Catégorie.nom.(*OP_p*) ou nom-domaine ∈ catégorie.synonymes* (OP_p).
Toutes les opérations publiées (simple ou générique) qui ne vérifient pas cette contrainte ne seront pas prises en considération dans les autres étapes du processus de résolution de la requête.

Les noms des opérations : si *nom(*OP_R*)=Fonct.fonctionnalité-metier(*OP_P*) ou nom(*OP_R*) ∈ Fonct.synonymes (*OP_P*)* on dit que le nom de l'opération recherchée OP_R correspond au nom de l'opération publiée OP_P et on note: nom(OP_R)==nom(OP_P) sinon on note nom(OP_R)!=nom(OP_P)

Mise en correspondance des paramètres d'Entrées /Sorties : cette règle est la plus importante car les paramètres d'E/Ss représentent les attributs fonctionnels d'une opération et définissent ainsi la transformation des données que l'opération effectue. Pour ce faire, nous définissons une fonction que nous appelons *"match"* qui vérifie si deux concepts syntaxiquement différents appartenant à une même ontologie sont sémantiquement similaires ou compatibles. Ces deux concepts peuvent être les attributs appartenant à deux opérations différentes. Dans notre cas, nous appliquons la fonction *"match(x,y)"* pour calculer la relation de subsomption entre deux paramètres x,y en utilisant l'ontologie du domaine dans laquelle ils sont définis.
Dans **[Pao02]**, x est le concept recherché et y est le concept publié. Dans le contexte de notre utilisation, nous considérons que x est un nom de paramètre d'une opération recherchée et y est soit égal à ″paramètre.nom″ ou il appartient à ″paramètre.synonymes″ d'une opération publiée. La fonction *"match"* retourne ainsi 4 valeurs :

- *match(x,y)= "exact "* si x, y font référence à la même classe dans l'ontologie.
- *match(x,y)= "plug-in "* si classe de x dans l'ontologie est une sous classe de y. On dit dans ce cas que le concept publié y *subsume* x c'est à dire que y est plus générale que le concept recherché x.
- *match(x,y)= "subsumé "* si classe de x dans l'ontologie est une super classe de y. Dans ce cas que le concept recherché x est plus générale que le concept publié y c'est à dire que y *est subsumé* par x. Ce qu'il fait que y satisfait partiellement x.

- *match(x,y)= "disjoint"* si aucune relation de subsumption lie x et y dans l'ontologie.

Les deux cas qui peuvent fournir un résultat pertinent sont le cas *"exact"* et le *"plug-in"*. Le cas "subsumé" n'est pas pris en considération car il fournit un matching non complet.

Soient OP_R et OP_P deux opérations recherchée et publiée (respectivement) et soient $In(OP_R)/Out(OP_R)$ l'ensemble des paramètres d'Entrées/Sorties de OP_R et $In(OP_p)/Out(OP_p)$ l'ensemble des paramètres d'Entrées/Sorties de OP_p. Nous définissons plusieurs cas de correspondance entre leurs paramètres d'Entrées/Sorties:

∗ Les sorties:

Rappelons que nous avons besoin de faire correspondre toutes les sorties du service recherché à toutes ou quelques sorties du service publié . Par conséquent, nous définissons les cas de correspondance comme suit :

a) Exact: $Out(OP_R) == Out(OP_P)$ si :

- $Out(OP_R)$ et $Out(OP_P)$ ont le même nombre de variables et
- $\forall x \in Out(OP_R)$ (resp. $Out(OP_P)$),$\exists y \in Out(OP_P)$ | *match(x,y)= "exact "* .

Ceci se traduit par: tous les paramètres de sortie (Output) de OP_R sont en correspondance exacte avec les paramètres de sortie OP_P deux à deux.

b) Plug-In: $Out(OP_R) =\approx Out(OP_P)$ si :

- $Out(OP_R)$ et $Out(OP_P)$ ont le même nombre de variables d'Entrées et
- $\forall x \in Out(OP_R)$,$\exists y \in Out(OP_P)$ | *match(x,y)= "exact "* ou *match(x,y)= "plug-in"* et $\exists z \in Out(OP_R)$,$\exists t \in Out(OP_P)$ | *match(z,t)= "plug-in"*

Ce cas se traduit par : il y a au moins un paramètre de sortie de OP_R qui est en correspondance Plug-In avec un paramètre de sortie de OP_p.

c) Inclusion exacte: $Out(OP_R) \subset Out(OP_P)$ si:

- $Out(OP_P)$ contient plus de variables que $Out(OP_R)$
- $\forall x \in Out(OP_R)$,$\exists y \in Out(OP_p)$ | *match(x,y)= "exact "* .

Tous les paramètres de sortie (Outputs) de OP_R sont en correspondance exacte avec les paramètres de sortie de OP_P mais $Out(OP_R)$ est un sous-ensemble de $Out(OP_P)$

d) Inclusion Plug-In: $Out(OP_R) \subset\approx Out(OP_P)$ si :

- $Out(OP_P)$ contient plus de variables que $Out(OP_R)$.
- $\forall x \in Out(OP_R)$,$\exists y \in Out(OP_P)$ | *match(x,y)= "exact "* ou *match(x,y)= "plug-in"* et $\exists z \in Out(OP_R)$,$\exists t \in Out(OP_P)$ | *match(z,t)= "plug-in"*

Ce cas se traduit par: $Out(OP_R)$ est un sous-ensemble de $Out(OP_P)$ et il y a au moins un paramètre de sortie de OP_R qui est en correspondance Plug-In avec un paramètre de sortie de OP_P.

∗ Les entrées:

Rappelons que, nous avons besion de faire correspondre toutes ou quelques entrées du service recherché à toutes les entrées du service. Par conséquent, nous définissons

$In(OP_R) == In(OP_P)$ (exact), $In(OP_R) =\approx In(OP_P)$ (plug-in) de façon similaire que $Out(OP_R) == Out(OP_P)$, $Out(OP_R) =\approx Out(OP_P)$ et les autres cas comme suit :

c) Inclusion exacte: $In(OP_R) \supset In(OP_P)$ si:

- $In(OP_R)$ contient plus de variables que $In(OP_P)$
- $\forall x \in In(OP_P), \exists y \in In(OP_R)$ | *match(y,x)= "exact "* .

Tous les paramètres d'entrée (Inputs) de OP_P sont en correspondance exacte avec les paramètres d'entrée de OP_R mais $In(OP_P)$ est un sous-ensemble de $In(OP_R)$

d) Inclusion Plug-In: $In(OP_R) \approx\supset In(OP_P)$ si :

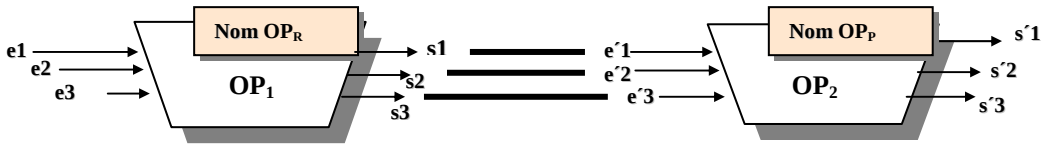
- $In(OP_R)$ contient plus de variables que $In(OP_P)$.
- $\forall x \in In(OP_P), \exists y \in In(OP_R)$ | *match(y,x)= "exact "* ou *match(y,x)= "plug-in"* et $\exists z \in Out(OP_R), \exists t \in In(OP_R)$ | *match(z,t)= "plug-in"*

Ce cas se traduit par: $In(OP_P)$ est un sous-ensemble de $In(OP_R)$ et il y a au moins un paramètre d'entrées de OP_R qui est en correspondance Plug-In avec un paramètre de d'entrée de OP_P.

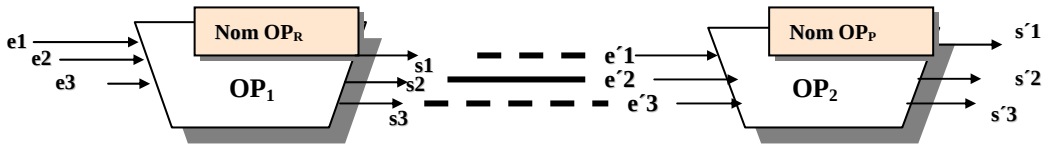
3) Les niveaux de mises en correspondance par mode:

- **Horizontale :** Dans ce cas, la mise en correspondance se fera uniquement sur les paramètres d'E/S, nous distinguons 4 cas de correspondance valables:

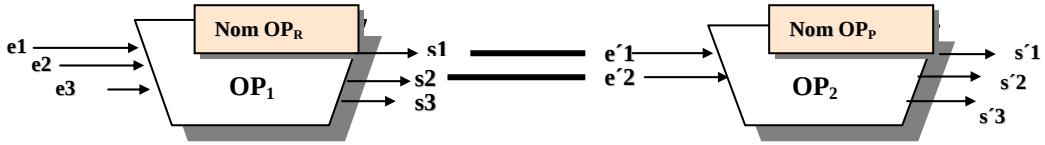
1) **Exacte:** $Out(OP_1) == In(OP_2)$



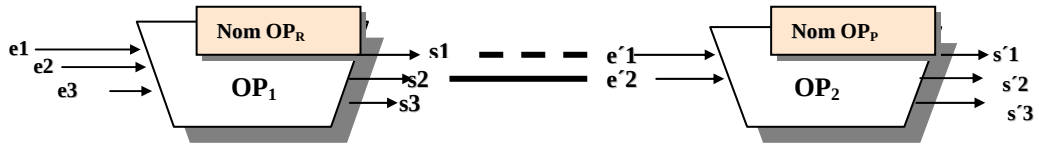
2) **Plug-In:** $Out(OP_1) =\approx In(OP_2)$



3) **Inclusion exacte:** $Out(OP_1) \supset In(OP_2)$



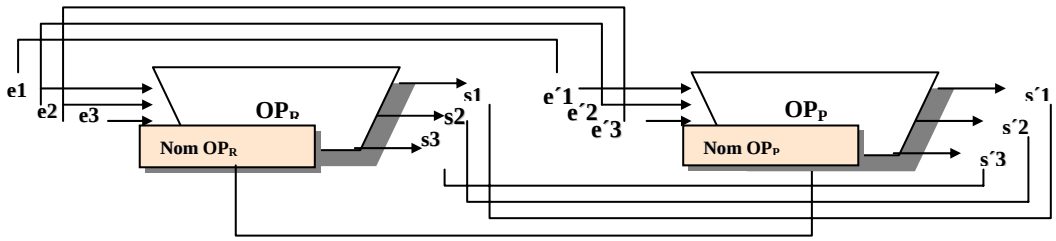
4) **Inclusion plug-in:** Out (OP_1) ≈⊃ In(OP_2)



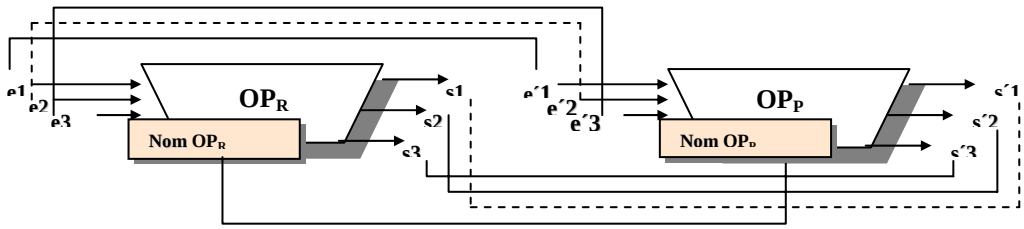
Nous affectons un degré de Matching pour chaque niveau qui quantifie le taux de correspondance sémantique entre OP_1 et OP_2 appelé *Degré-corresp*.
Degré-corresp(OP_1 :OP_2)={4/4=1 pour "exact", 2/4=1/2 pour"plug-in", ¾ pour"Inclusion exacte" et ¼ pour "Inclusion plug-in" }

- **Verticale 1:1**: La mise en correspondance verticale 1:1 se fera sur les paramètres d'E/S et les noms des opérations à mettre en correspondance (OP_R et OP_P). Nous distinguons les cas de correspondance suivants:

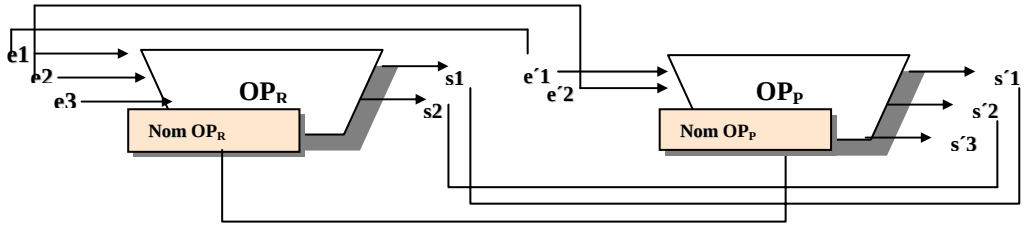
1) **Exact:** In(OP_R)==In(OP_P) et Out(OP_R) == Out(OP_P) et nom(OP_R)==nom(OP_P)



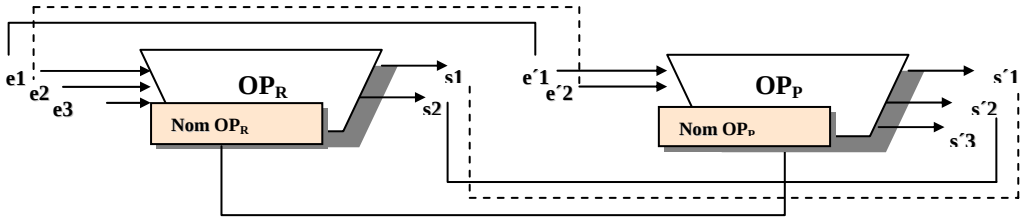
2) **Plug-in :** [In(OP_R) =≈ In(OP_P) et Out(OP_R) == Out(OP_P)] ou [Out(OP_R) =≈Out(OP_P) et In(OP_R)==In(OP_P)] ou [In(OP_R) =≈ In(OP_P) et Out(OP_R) =≈ Out(OP_P)] et nom(OP_R)==nom(OP_P)



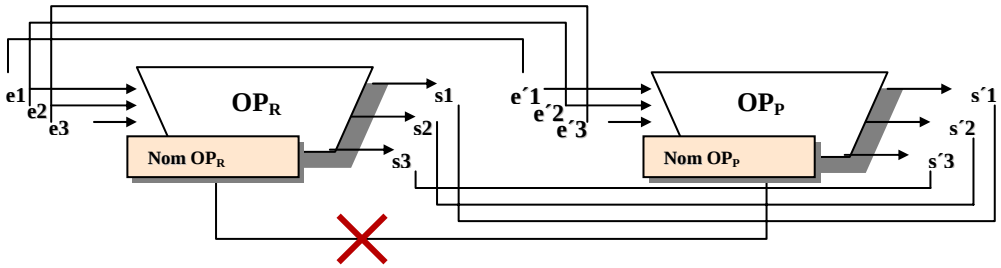
3) **Chevauché:** In(OP_R) ⊃ In(OP_P) ou Out(OP_R) ⊂ Out(OP_P) et nom(OP_R)==nom(OP_P)



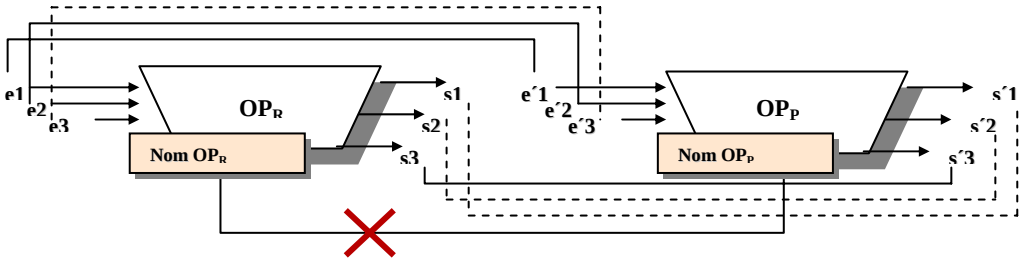
4) **Chevauché avec plug-in:** $In(OP_R) \approx\supset In(OP_P)$ ou $Out(OP_R) \subset\approx Out(OP_P)$ et $nom(OP_R)==nom(OP_P)$



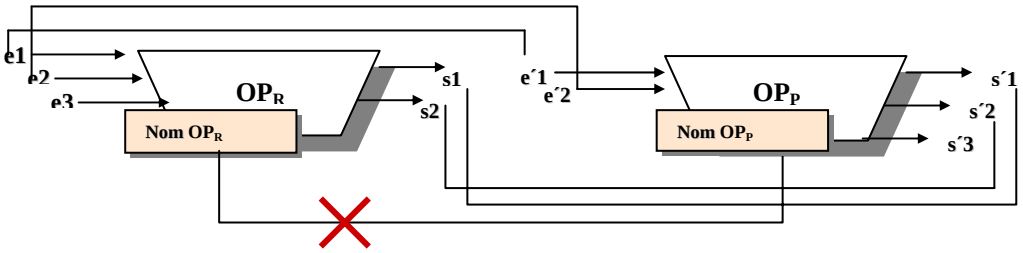
5) **Valide :** $In(OP_R)==In(OP_P)$ et $Out(OP_R) == Out(OP_P)$ et $nom(OP_R)!=nom(OP_P)$



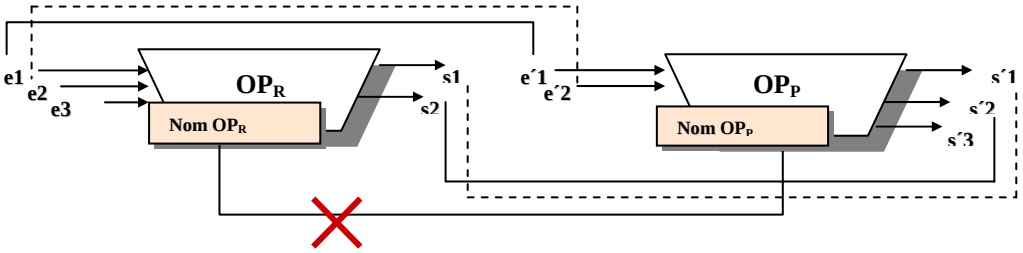
6) **Accepatable:** $[In(OP_R) =\approx In(OP_P)$ et $Out(OP_R) == Out(OP_P)]$ ou $[Out(OP_R) =\approx Out(OP_P)$ et $In(OP_R)== In(OP_P)]$ ou $[In(OP_R) =\approx In(OP_P)$ et $Out(OP_R) =\approx Out(OP_P)]$ et $nom(OP_R)!=nom(OP_P)$



7) **Faible:** $In(OP_R) \supset In(OP_P)$ ou $Out(OP_R) \subset Out(OP_P)$ et $nom(OP_R) !=nom(OP_P)$



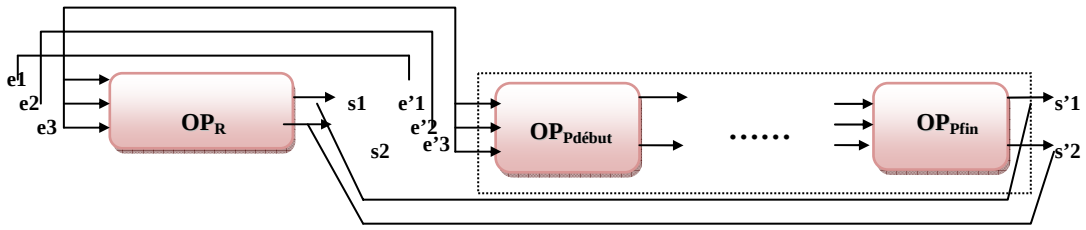
8) **Très faible:** $In(OP_R) \approx\supset In(OP_P)$ et $Out(OP_R) \subset\approx Out(OP_P)$ et $nom(OP_R)!=nom(OP_P)$



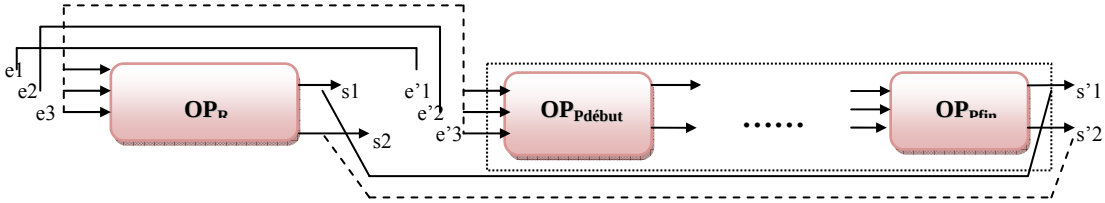
Nous quantifions chaque niveau de correspondance comme suit :
Degré-corresp(OP_R :OP_P)={ 8/8=1 pour "exact ", 7/8 pour "plug-in", 6/8=3/4 pour "chevauché", 5/8 pour "chevauché avec plug-In", 4/8=1/2 pour "valide", 3/8 pour "acceptable", 2/8=1/4 pour "faible", 1/8 pour "très faible"*}*

- **Verticale 1:N** : ce cas de mise en correspondance est une combinaison des deux cas précédents. En effet, les entrées de l'opération recherchée seront mises en correspondance avec les entrées de la première opération dans la chaîne des opérations liées horizontalement, et les sorties de l'opération recherchée avec les sorties de la dernière opération de la chaîne. Donc, une fois un ensemble d'opérations sont enchaînée (par une mise en correspondance horizontale) le principe devient le même que celui de la mise en correspondance verticale 1:1 mais en considérons uniquement les paramètres d'E/Ss. Nous distinguons 4 cas de correspondance valables :

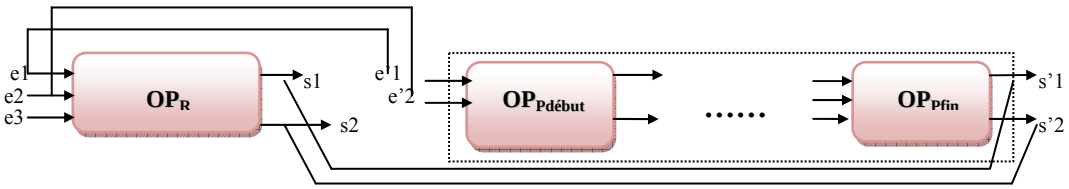
1) Exacte: $In(OP_R) == In(OP_{Pdebut})$ et $Out(OP_R) == Out(OP_{Pfin})$.



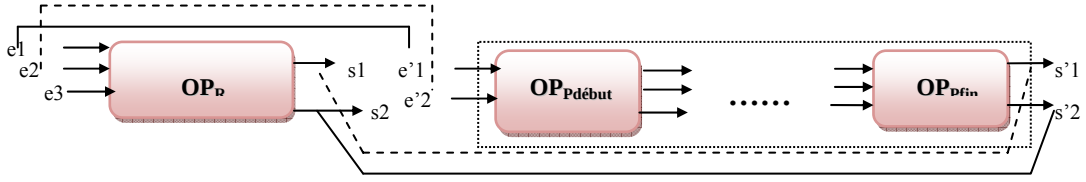
2) Plug-In: $[In(OP_R) =\approx In(OP_{Pdebut})$ et $Out(OP_R) == Out(OP_{Pfin})]$ ou $[Out(OP_R) =\approx Out(OP_{Pfin})$ et $In(OP_R) == In(OP_{Pdebut})]$ ou $[In(OP_R) =\approx In(OP_{Pdebut})$ et $Out(OP_R) =\approx Out(OP_{Pfin})]$



3) Chevauché: $In(OP_R) \supset In(OP_{Pdebut})$ ou $Out(OP_R) \subset Out(OP_{Pfin})$



4) Chevauché avec plug-in: $In(OP_R) \approx\supset In(OP_{Pdebut})$ et $Out(OP_R) \subset \approx Out(OP_{Pfin})$



Degré-corresp (OP_R : $OP_{Pdébut}$…OP_{Pfin})= { 4/4=1 pour *"exact"*, 3/4 pour *"Plug-in"*, 2/4=1/2 pour *"chevauché"et ¼* pour *"chevauché avec Plug-In"*}.

3.3.2 Processus de résolution de la requête :

La requête passe par plusieurs étapes intermédiaires avant d'arriver à la génération du plan de composition exécutable. La figure suivante schématise l'algorithme général du système de composition global proposé.

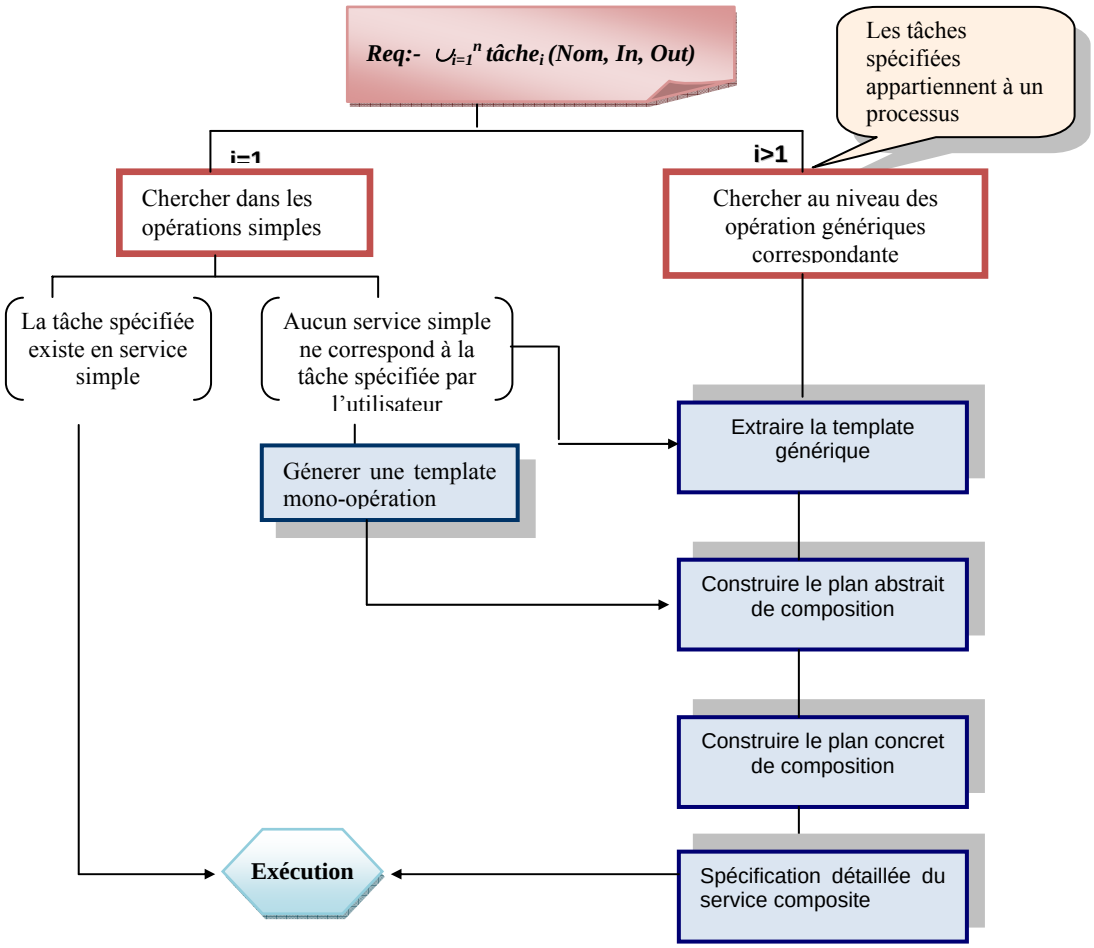


Fig 4.18: Procédure générale de résolution de la requête

1) Première étape du processus : Analyse de la requête et extraction d'un plan initial de composition

Nous distinguons selon le cas de la requête deux types de traitement :

- 1er cas de traitement : l'utilisateur spécifie dans sa requête plusieurs sous-tâches soit: `requête :- ∪i=1n Sous-tâchei (Nom,In,Out) tel que n !=0`.
La mise en correspondance dans ce cas se fera entre les sous-tâches et les opérations génériques. Ceci consiste à faire correspondre les noms des sous-tâches avec leurs paramètres d'E/S donnés par l'utilisateur aux opérations génériques appartenant à une même template générique avec leurs paramètres d'E/Ss.

On note ce cas de traitement par: *Requête:- nomDomaine,* $\cup_{i=1}^{n}$ *Sous-tâche*$_i$ *(Nom,In,Out)* $\Leftrightarrow$ *template-générique :* $\cup_{i=1}^{m}$*Operation-g*$_i$ *(Fonc Catégorie,In,Out)*

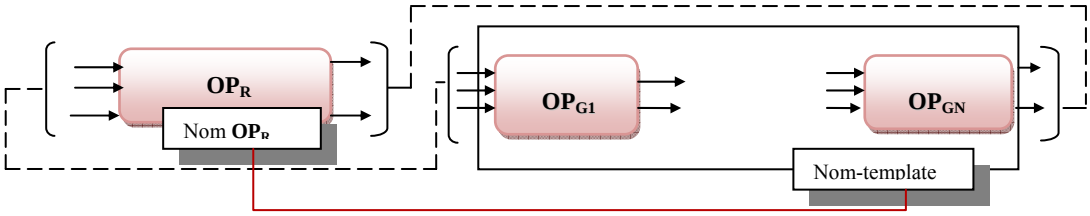
- Les sous-tâches : nous appliquons une mise en correspondance verticale 1:1 entre chaque sous-tâche (notée par OP_R) spécifiée dans la requête et chaque opération générique(OP_{PG}) (appartenant à une template générique). En remplaçons OP_P par OP_{PG}. Suivant les règles de correspondance déjà défini, nous obtenons, plusieurs niveaux de correspondance quantifiés par une valeur *degré* ($OP_{Ri} : OP_{PGi}$).
- La requête entière : Nous cherchons ensuite une template générique qui peut être **sémantiquement équivalente** à la requête (c'est à dire, le nombre des sous-tâches formant la requête est égale au nombre d'opérations génériques formant la template générique et chaque sous-requête correspond sémantiquement à une opération générique de la template) ou qui **couvre** le besoin de la requête (c'est à dire, le nombre des sous-requêtes dans la requête globale est inférieur au nombre d'opérations génériques formant la template générique et chaque sous-requête correspond sémantiquement à une opération générique). Uniquement ces deux types de résultats sont acceptables. Nous quantifions le degré de correspondance totale de la requête comme suit :

$$\textit{Degré-correspondance (Q)} = [\sum_{i=1}^{n} \textit{degré}\ (OP_R i : OP_{PG} i)/n] * \begin{cases} 1 & \textit{si}\ \ nb(\mathbf{OP_R}) = nb(\mathbf{OP_{PG}}) = n \\ ½ & \textit{si}\ \ nb(\mathbf{OP_R}) < nb(\mathbf{OP_{PG}}) \end{cases}$$

- 2eme cas de traitement :

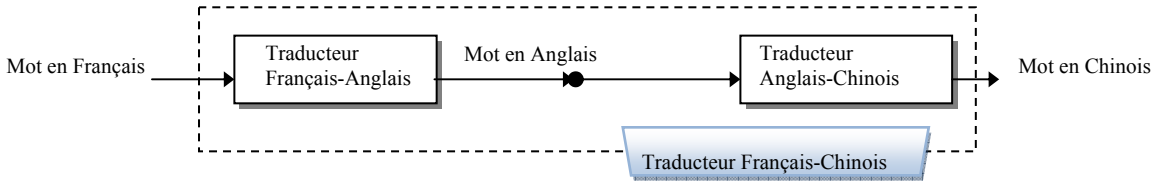
L'utilisateur spécifie une seule tâche et il n'a aucune idée si cette tâche peut être satisfaite par une opération d'un service unique ou bien elle fait appel à une composition de plusieurs opérations simples:

- Nous cherchons d'abords au niveau des opérations simples en essayant de trouver une équivalence sémantique entre la tâche recherchée et l'ensemble des opérations simples publiées et cela en appliquant une mise en correspondance verticale 1:1 déjà définie: *Requête:- service-complexe(Nom, In, Out)* $\Leftrightarrow$ *Opération-simple(Fonc, In, Out).*
  Le service qui répond le plus au besoin de l'utilisateur (un degré de mise en correspondance satisfaisant) sera sélectionné.
- Dans le cas où il n'y a aucune correspondance sémantique satisfaisante, la requête fait donc appel à un service complexe. Le système va tenter de trouver une Template qui pourra satisfaire la requête en appliquant une mise en correspondance verticale 1:N en considérons, de plus, le nom des opérations (voir la figure ci-dessous).



- Dans le cas ou il n'y a aucune template qui correspond à la requête, le système considère la tâche recherchée comme une template mono-opération et va procéder à la génération du plan abstrait par chaînage des opérations abstraites afin d'obtenir la fonctionnalité requise par la requête. Le chaînage des opérations fait appel à une correspondance horizontale entre les opérations. La première opération doit avoir des entrées équivalente (selon nos règles de correspondance) aux entrées de l'opération recherchée et la dernière opération de la chaîne doit avoir des sorties équivalentes aux sorties de l'opération recherchée (voir la partie suivante).

Exemple: Supposant, un utilisateur cherche un traducteur Français-Chinois qui n'existe pas en service unique ni comme processus complexe prédéfini. Par conséquent, le système va procéder au chaînage de deux opérations de services existants qui sont : traducteur Français-Anglais et traducteur Anglais-Chinois de la manière suivante :



Nous expliquons l'algorithme de chaînage utilisé dans la partie suivante.

2) Deuxième étape du processus : Construction du plan abstrait

A la fin de la première phase du processus de résolution, la requête sera décrite sous forme d'une template générique (pour le premier cas de requête). Cette dernière donne la forme générale initiale du plan de composition. La prochaine étape qui est la génération du plan abstrait de composition regroupe les deux étapes suivantes:

- Génération des sous plans abstraits correspondants à chaque opération générique dans une Template générique.
- Remplacer chaque opération générique par son sous-plan abstrait.

- **Génération des sous plans abstraits :**

Chaque opération générique est considérée comme une sous requête qui va être mise en correspondance verticalement avec une opération (1:1) ou une combinaison (1:N) d'opérations abstraites simples. Chacune retourne un sous-plan abstrait du plan global.
En effet, la construction d'un sous plan abstrait fait appel à un algorithme de chaînage des opérations abstraites qui utilise les règles de correspondance horizontale déjà définies pour enchaîner deux opérations abstraites. Cette algorithme de chaînage est un algorithme à retour arrière et procède comme suit:

- ✓ Pour chaque opération abstraite disponible, trouver une correspondance entre les sorties (output) de l'opération générique (opération recherchée) et les sorties d'une opération abstraite, soit cette opération OP_n.
- ✓ Prendre l'entrée de OP_n et trouver une opération abstraite qui peut générer une sortie qui correspond sémantiquement à l'entrée de OP_n. Soit cette opération OP_{n-1}.
- ✓ Répéter ces étapes jusqu'à trouver une opération OP_{n-i} possédant des entrées qui correspondent sémantiquement aux entrées de l'opération générique en question (opération recherchée en général).

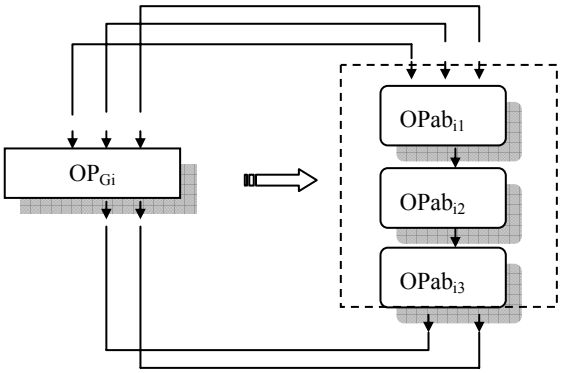


Fig 4.19: Application de l'algorithme de chaînage pour construire le sous plan abstrait

- **Remplacement des opérations génériques par leurs sous-plans abstraits:**

En utilisant la templates générique (TG) et les sous plans abstraits (SPab) générés automatiquement pour chaque opération générique, nous obtiendrons un plan abstrait de composition (Pab). Chaque opération générique sera substituée dans la template générique par le sous plan correspondant (voir la figure ci-après).

Si le sous plan a été obtenu grâce à une mise en correspondance 1:N, l'ordre d'exécution des opérations dans le PAb est similaire que leur ordre d'exécution dans le SPAb.

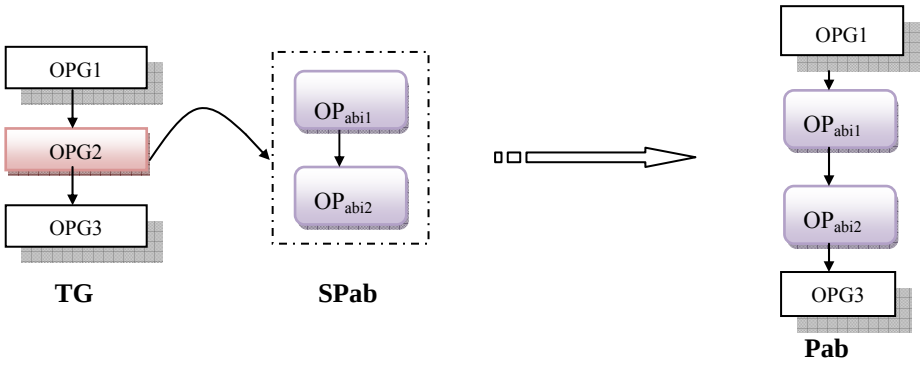


Fig 4.20: Remplacement d'une opération générique par son sous-plans abstraits

Dans le cas d'une seule opération (cas où la requête contient une seule tâche qui ne correspond ni à une opération simple ni à une template prédéfinie), le système procède directement à générer le plan abstrait correspondant à la tâche en question en appliquant l'algorithme de chaînage défini.

3) Troisième phase du processus : Construction du plan concret exécutable

La génération du plan concret consiste à faire correspondre chaque opération du plan abstrait (workflow abstrait) généré lors de la phase précédente à une opération concrète appartenant à une instance de service disponible. Cependant, plusieurs opérations concrètes peuvent implémenter une même opération abstraite. Par conséquent, plusieurs plans concrets peuvent être générés pour un même plan abstrait. Le problème qui se pose est le suivant : comment trouver un plan concret optimal (qui répond au mieux aux préférences de l'utilisateur) en un temps raisonnable? Ce problème a été particulièrement traité par **[Ouz04]**. Nous avons adapté les résultats de son travail à notre modèle. En effet **[Ouz04]** propose deux manières pour générer le plan concret exécutable :

Recherche exhaustive: cette stratégie consiste à générer tous les plans concrets possibles, les évaluer en terme de qualité de service puis sélectionner le meilleur pour être exécuté.
La génération des plans concrets, dans ce cas, consiste à faire correspondre chaque opération abstraite à toutes les opérations concrètes qui peuvent l'implémenter en utilisant le schéma de correspondance défini (dans notre cas c'est la correspondance verticale 1:1). Si on suppose que le nombre d'opérations abstraites dans le plan abstrait est N et chacune doit être mise en correspondance, au plus, à M opérations concrètes, le nombre total des plans concrets potentiels peut atteindre M^N.
L'évaluation des plans concrets en terme de qualité se fait en calculant la qualité de service (QdWS) de chaque plan concret généré de la manière suivante:
Si on suppose que la qualité de service d'un service concret quelconque est représentée par le vecteur suivant : QdWS(SCi)=(T-reponse(SCi), Dispo(SCi), Fiab(SCi), Coût(SCi), Réput(SCi), Auth(SCi), Confid(SCi), Répud(SCi)).Les paramètres QdWS du plan entier est calculé par la fonction d'agrégation de la manière suivante (voir le tableau ci-dessus):

QdWS	Fonction d'agrégation
Temps de réponse (T-reponse)	$\sum_{SWi \in \text{ long-path}}$ T-reponse(SCi)
Coût (Coût)	$\sum_{i=1}^{n}$ Coût(SCi)
Disponibilité (Dispo)	$\prod_{i=1}^{n}$ Dispo(SCi)
Réputation (Réput)	$1/n\sum_{i=1}^{n}$ Réput(SCi)
Authentification (Auth)	$1/n\sum_{i=1}^{n}$ Auth(SCi) ou $\prod_{i=1}^{n}$ Auth(SCi)
Confidentialité (Confid)	$\cup_{i=1}^{n}$ Confid(SCi)
Non-répudiation (Répud)	$1/n\sum_{i=1}^{n}$ Répud(SCi) ou $\prod_{i=1}^{n}$ Répud (SCi)
Fiabilité (Fiab)	$1/n\sum_{i=1}^{n}$ Fiab(SCi) ou $\prod_{i=1}^{n}$ Fiab (SCi)

Remarque: Le temps de réponse du plan entier est celui du plus long chemin dans le plan (long-path).
Le plan qui sera sélectionné pour l'exécution est donc celui qui possède la meilleure combinaison des valeurs obtenues par la fonction d'agrégation des différents paramètres de QdWS qui peuvent satisfaire au mieux les préférences de l'utilisateur.

L'avantage d'une telle stratégie est qu'elle garantit de trouver la meilleure solution (c'est à dire, le meilleur plan) en explorant toutes les solutions possibles.
Cependant, l'inconvénient de cette stratégie est évident: la complexité de calcul est d'ordre $\theta(M^N)$ ce qui est inapplicable si M (nombre maximum d'opération concrètes par opération abstraite)et N (le nombre d'opération abstraite dans le plan abstrait) sont très grands.

Sélection locale : Dans la stratégie de la sélection locale, la meilleure opération concrète est sélectionnée pour chaque opération abstraite au moment de la construction du plan concret. La mise en correspondance, l'évaluation et la sélection sont faites en une seule étape. Dans cette stratégie, l'évaluation est effectuée en utilisant une fonction appelée *fonction objective* qui permet de décider quelle est la meilleure opération concrète parmi les M opérations candidates qui va être choisie pour implémenter l'opération abstraite en cours. La *fonction objective* (F) de chaque opération (OP_c) est calculée de la manière suivante:

$$F(OP_c) = rating(OP_c) * md(OP_c) * (P_{neg} + P_{pos})$$
$$où \quad P_{neg} = \sum_{neg} (pQ_i^{max} - pQ_i / pQ_i^{max} - pQ_i^{min})$$
$$et \quad P_{pos} = \sum_{pos} (pQ_i - pQ_i^{max} / pQ_i^{max} - pQ_i^{min})$$

- md(*OPc*) définit le degré de Matching (de correspondance) calculé lors de la mise en correspondance de l'opération abstraite en cour à l'opération concrète OPc.
- Rating (*OPc*) représente le taux de fluctuation qui peut survenir durant le cycle de vie d'une opération concrète OPc, c'est à dire, il mesure la distance entre la qualité promise (*prQi)*et la qualité délivrée (*dQi*) par l'opération en calculant la formule suivante :

$$\text{rating } (OPc) = \sum_{QdWS \in pos} (dQi - prQi) + \sum_{QdWS \in neg} (1/dQi - 1/prQi)$$

- pQ_i^{max} est la valeur maximum du i^{eme} paramètres par rapport à toutes les opérations concrètes candidates (implémentant l'opération abstraite en cours) et le PQ_i^{min} est le minimum.
- *Neg et pos*: désignent respectivement les paramètres de qualité négatif et positif. Un paramètre négatif est un paramètre à minimiser tels que le coût, temps de réponse et un paramètre de qualité positif est un paramètre dont la valeur est à maximiser telles que la réputation, disponibilité…etc

La meilleure opération concrète qui sera choisie pour implémenter l'opération abstraite en cours est celle possédant la meilleure valeur de la fonction objective ($F(OP_c)$).
Cette stratégie est plus réaliste et moins coûteuse en temps. Cependant, elle peut rater la solution optimale.

4) Quatrième phase : Spécification détaillée et exécution du service composite:

Cette dernière phase de notre approche consiste à générer une description détaillée du service composite en utilisant un langage de spécification de flux tel que BPEL4WS ou alors WSFL ou autres. Cette description doit comprendre une définition des services participants, les messages échangés, leurs paramètres et le flux de contrôle et de données entre eux. Cette spécification sera soumise directement à un moteur d'exécution de workflow tel que le BPWS4J ou le BPEL Orchestration Server **[BPEL]**. Dans cette phase, nous pouvons faire appel au document WSDL pour récuperer des informations qui peuvent être utile à l'invocation effective des services web correspondants. L'exécution effectue alors les invocations nécessaires des services web partenaires appropriés suivant le modèle d'orchestration défini dans la spécification.

4. Tableau Comparatif de notre approche aux autres approches de composition étudiées:

Pour ce faire, nous reprenons le tableau déjà décrit dans le chapitre précédent pour faire une évaluation de notre approche selon les critères de comparaison déjà définis:

	Méthodes manuelles	**Méthodes à base de workflow**	**Méthodes à base de planification**	**Méthodes interactives**	**L'approche proposée**
Niveau d'automatisation	Aucune automatisation du processus de composition (nul)	Seule la recherche et la liaison (Binding) aux services concrets est automatique (moyen)	Pratiquement tout le processus se fait de manière automatique (élevé)	Semi-automatique	Assez élevé : l'utilisateur peut intervenir uniquement pour sélectionner un template
Niveau de dynamicité	Tout se fait statiquement	-Modèle de processus est généré statiquement -Liaison aux services concrets est dynamique	- la génération du plan de composition se fait de manière dynamique	- la génération du plan se fait de manière dynamique	Moyen : Les templates génériques sont statiquement construits. La génération du plan abstrait et concret est dynamique.
Niveau de Scalabilité	Pas du tout	Moyen	élevé	Modeste.	Assez élevé: pratiquement, toutes les étapes du processus de génération du plan se fait de manière automatique
Spécification de la requête par l'utilisateur	Très bas niveau, c'est l'utilisateur qui définit tout le plan de	Bas niveau : l'utilisateur doit mettre en place le modèle de processus en utilisant un	Définit l'état initial, l'état final et certaines contraintes avec un langage de très	Se fait interactivement avec le système (au fur et à mesure)	Langage simple et ne nécessite pas des détails techniques : spécifier les tâches qu'il veut

	composition	langage de flux	bas niveau (les langages logiques)		exécuter et ses préférences en utilisant les termes de son vocabulaire
Niveau de granularité des plans de composition générés	Un seul plan généré (plan exécutable)	-Workflow abstrait (mais défini manuellement) -Et un workflow exécutable	Un seul plan (plan exécutable)	Uniquement le plan exécutable	Deux niveaux de granularité : plan abstrait et plan concret implique un haut niveau de réutilisabilité
Support sémantique et niveau d'abstraction	Pas de support sémantique (registre UDDI)	La majorité utilise des descriptions syntaxiques (WSDL, BPEL4WS, WSFL…)	- utilise les formalismes logiques, DAML-S, OWL-S.	Support sémantique : DAML-S, OWL-S	Support sémantique qui permet un accès progressif aux fonctionnalités offertes.
Recherche des services	Primitif (dans les registres UDDI)	- la majorité utilise le moteur de recherche UDDI	- dans les profiles DAML-S ou OWL-S	- dans les profiles DAML-S ou OWL-S	Deux niveaux de recherche: - opérations simples - opération générique(template générique)

5. Conclusion

Nous avons proposé, dans ce chapitre, un modèle de composition des services web sémantiques. Ce modèle a été suggéré après l'étude des différentes approches de composition émergentes. En effet, notre modèle permet une résolution progressive de la requête de composition permettant à un utilisateur non-spécialiste d'accéder à des fonctionnalités complexes d'un domaine d'une manière aisée et naturelle. En d'autres termes, l'utilisateur spécifie simplement ce qui doit être fait en terme de tâches qui doivent être effectuées sans qu'il soit obligé de connaître tous les détails techniques tels que les services participants et la manière dont ils seront composés et exécutés. La découverte des services partenaires, leur composition ainsi que leurs interopérations sont effectuées de façon automatique et transparente par rapport à l'utilisateur. Ceci interpelle la mise en place d'un support sémantique qui permet une interaction potentielle entre l'utilisateur et les services publiés (simples ou complexes). Le support sémantique proposé distingue deux types de concepts qui peuvent être manipulés par les services web : les concepts statiques qui sont des concepts utilisés pour définir des propriétés statiques d'un service web, et les concepts dynamiques représentants les opérations des services web. De plus, le support sémantique permet de représenter les fonctionnalités d'un domaine donné selon différents niveaux d'abstraction :concret simple, abstrait simple et abstrait générique. Ceci a permis de générer des plans de composition à différents niveaux de granularité : abstrait et concret. Cette distinction a pour but de favoriser la réutilisation des plans (workflows) déjà construits pour répondre à des requêtes similaires. En particulier, en réutilisant les plans abstraits, le même besoin peut être satisfait en ayant comme option de pouvoir changer certaines préférences en terme de qualité du service composite résultant. De plus, l'idée du plan

abstrait s'adapte bien au problème de disponibilité rencontrés dans un environnement dynamique et volatil comme le web. En effet, si une opération concrète n'est plus disponible au moment de l'exécution, il ne serait pas nécessaire de tout refaire mais il suffit de chercher une autre opération concrète ayant la même fonctionnalité décrite par l'opération abstraite.

Toutes les étapes du processus de résolution utilisent un schéma de mise en correspondance pour vérifier si deux opérations différentes sont compatibles pour être composées ensembles ou possèdent des fonctionnalités similaires pour être substituées. Ce schéma définit différents niveaux de correspondance (exact, plug-in, inclusion, …) permettant d'accepter des résultats qui ne sont pas nécessairement identiques à ce qui est demandé mais qui restent pertinents vis-à-vis du besoin recherché. Ceci, rend le schéma de correspondance plus flexible et ayant un degré de réussite plus élevé.

Conclusion générale et perspectives

Aujourd'hui, les services web sémantiques constituent une voie prometteuse permettant de mieux exploiter les services web en automatisant, autant que possible, les différentes tâches liées au cycle de vie d'un service tel que leur découverte et leur composition en vue d'obtenir des services plus élaborés.

La composition des services web, en particulier, est devenue un axe de recherche incontournable du fait de l'augmentation des demandes complexes des utilisateurs en terme de processus orientés services. Comme nous avons pu le constater, le nombre de travaux dédiés à ce thème dénotent une vitalité réelle de ce domaine de recherche.

Le travail mené dans le cadre de cette thèse s'est concentré autour de la proposition d'un modèle de composition des services web sémantiques qui permet en premier lieu une interaction aisée et naturelle entre un utilisateur externe et un ensemble de services web formant un service composite. Ce modèle permet un accès progressif aux fonctionnalités complexes qui peuvent être offertes dans un domaine donné en combinant d'autres services plus simples.

La plupart des travaux s'intéressant à la composition des services **[Cas00][McI02] [Nar02] [Gha98] [Pon02] [Med03] [Wu03] [Wal01] [Rao03] [Rao04]** web considèrent que le besoin de l'utilisateur en terme de service complexe doit être spécifié en utilisant un langage de bas niveau. Or, un utilisateur du web, très souvent, possède des connaissances très limitées sur le domaine dans lequel il veut effectuer sa recherche. De plus il a souvent une idée imprécise et incomplète concernant son besoin. Pour cela, nous avons essayé de répondre à ce problème en proposant à l'utilisateur de spécifier son besoin en terme de tâches qui doivent être effectuées et il ne sera pas concerné d'être conscient de tous les détails techniques tels que les services participants et la manière dont ils seront découverts, composés et exécutés. Ces derniers sont effectués de façon automatique et transparente par rapport à l'utilisateur. Ainsi, le modèle proposé distingue différents niveaux d'abstraction (de granularité) des tâches manipulées dans le domaine d'application en se basant sur un support sémantique qui permettra de passer de manière progressive et automatique d'un niveau à un autre.

Les majeures contributions de cette thèse peuvent se résumer comme suit :

- Un support sémantique qui distingue différents niveaux d'abstraction des tâches manipulées dans le domaine d'application (générique, abstraite, concrète). Cette distinction a pour but de se rapprocher le plus possible de l'utilisateur en lui

offrant une vue réduite (générique) et simplifiée sur les fonctionnalités offertes par les services web.

- Une approche progressive pour la résolution de la requête allant d'une simple spécification en terme de tâches à effectuer jusqu'à la génération détaillée de la spécification du service composite résultant.
- Un mécanisme de Matching flexible qui accepte des résultats qui ne sont pas forcement identiques à la requête mais qui restent pertinents vis-à-vis du besoin de l'utilisateur.
- Notre modèle permet de générer des plans à deux niveaux de granularités : abstrait et concrets. Cette distinction à pour but de favoriser la réutilisation des plans (Workflows) déjà construits. En réutilisant les Workflows abstraits, la même fonctionnalité peut être satisfaite en ayant comme option de pouvoir changer certains critères d'implémentation et de qualité répondant aux préférences variées de l'utilisateur.

Vu la complexité du système de composition dans sa globalité, plusieurs extensions peuvent être envisagées comme continuation à ce travail notamment:

- Enrichir la description sémantique par de nouvelles informations caractérisant un service ou une opération qui sont les pré-conditions et les post-conditions et rendre ainsi plus intelligent le schéma de mise en correspondance en prenant en considération ces nouveaux paramètres (Matching des conditions).
- Prendre en considération les cas d'exception de la requête : le cas ou il n'a y a aucune correspondance entre les tâches spécifiées dans la requête et une Template générique.
- Implémentation de la partie du travail effectué.
- Développer plus finement les autres étapes du processus de résolution de la requête : la génération des sous plans abstraits, génération du plan concret et son optimisation en terme de QdS.
- Proposer l'architecture complète du système de composition proposé de la phase de publication des services jusqu'à la génération du plan exécutable.

Bibliographies

[Arp04] : I. B. Arpinar, B. Aleman-Meza, R. Zhang, and A. Maduko. *"Ontology-driven Web services composition platform"*. In *Proceedings of IEEE International Conference on Ecommerce Technology, CEC'04, San Diego, California, USA, July 6.9, 2004*, pages 146.152. IEEE Press, 2004.

[Bag03] : Jean-François Baget, Étienne Canaud, Jérôme Euzenat et Mohand Saïd-Hacid *«Les langages du Web sémantique »* dans le r*apport de l'action spécifique CNRS-STIC sur le Web sémantique* (J. Charlet, P. Laublet, C. Reynaud, Eds), déc. 2003

[Bar03] : Douglas K. Barry. *"Web Services and Service-Oriented Architectures":* The Savvy Manager's Guide. Morgan Kaufmann, 2003.

[Ben03] : Benatallah, B., Sheng, Q.Z., and Dumas, M.: *"The Self-Serv Environment for Web Services Composition"*, Jan/Feb, 2003, IEEE Internet Computing. Vol 7 No 1. pp 40-48.

[Ber97] : Berners-Lee T, "*Metadata Architecture"*. http://www.w3.org/DesignIssues/Metadata.html , Jan. 1997.

[Ber01] : Berners-Lee, T., Hendler, J., Lassila,O, *"The semantic Web"***.** Scientific American, May 2001, p.28 -37.

[BizTalk] : *"Microsoft BizTalk Server 2004"*, *"BizTalk Server 2004 Conceptual Overview"*, Microsoft, 2004. URL: http://www.microsoft.com/biztalk/evaluation/overview/biztalkserver.mspx

[Bol00] : Boley, H. *"Relationships between Logic Programming and RDF"*. In proceeding of PRIIA, 2000, p.201-218

[BPEL4WS] : BEA, IBM, Microsoft (2003) *"Business Process Execution Language for Web Services (BPEL4WS)"*. http://xml.coverpages.org/bpel4ws.html.

[BPWS4J] : IBM alphaWorks, *"BPWS4J A platform for creating and executing BPEL4WS processes"*, 9 Aout 2002 . http://www.alphaworks.ibm.com/tech/bpws4j

[BPML] : ***"****Business Process Modeling Language"*. URL: http://www.bpmi.org/

[Cas00] : F. Casati, S. Ilnicki, and L. Jin. *"Adaptive and dynamic service composition in Eflow"*. In Proceedings of 12th International Conference on Advanced Information Systems Engineering(CAiSE), Stockholm, Sweden, June 2000. Springer Verlag.

[Ces04] : *"What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta- model"*, CesarGon, January 21 2004, disponible sur : http://www.metamodel.com/

[Chak02] : D.Chakraborty, F. Perich, A. Joshi, T. Finin, and Y.Yesha. "*A reactive service composition architecture for pervasive computing environements*". In proc PWC, 2002

[Cham02] : Pierre-Antoine Champin, Yannick Prie et Alain Mille, **«** *Une approche fondée sur les usages pour l'assistance à l'utilisateur sur le WEB sémantique», in Actes de RFIA'2002, Angers, France, janvier 2002, pp 633--642, volume 2, 2002*

[Char03] : Jean Charlet, Bruno Bachimont, Raphaël Troncy *«Ontologie pour le web sémantique »*, research report ASWS, p. 43-63, 2003.

[Cor00] : Corby, O., Dieng, R., Hebert, C. *"Aconceptual graph model for w3c ressource*

description framework". Internatinal Conference on Conceptual Structures, INCS 2000, n°1867, p.468-482.

[CNW01] : Francisco Curbera, William A. Nagy, and Sanjiva Weerawarana. *"Web services : Why and how?*" OOPSLA 2001 Workshop on Object-Oriented Web Services, 2001.

[cXML] : cXML Consortium. *cXML*. http://www.cxml.org.

[DAML-Sa] : DAML-S Coalition: A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. "*DAMLS: Semantic markup for Web services". In Proc SWWS, pages 411-430, 2001.*

[DAML-Sb]: DAML-S Coalition, 2002: A. Ankolekar, M. Burstein1, J.R. Hobbs, O. Lassila,D.McDermott, D. Martin, S.A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara*:"DAML-S: Web Service Description for the Semantic Web". In the first Iinternational Semantic Web Conference* (ISWC), PAGES 348-363, 2002.

[Dav04] : N.J. Davies, D. Fencel, and M. Richardson. *"The future of web services".* BT Technology Journal, 22(1), Jan 2004.

[DL] : http://dl.kr.org/courses.html

[Duv04] : Patrick Duvanel, « *Les services Web :La problématique de la spécification* », rapport personnel, Août 2004.

[EDI] : XML/EDI Consortium. *XML/EDI*. http://www.xmledi-group.org.

[Ero94] : Erol, K., Hendler, J., and Nau, D. S.*"Semantics for hierarchical task network planning"*, 1994.

[Gha98] : Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D. *"PDDL: the planning domain definition language".* In AIPS-98 Planning Committee (1998).

[Got02] : K. D. GOTTSCHALK, S. GRAHAM, H. KREGER & J. SNELL (2002). *"Introduction to Web services architecture". IBM Systems Journal* 41(2): 170-177.

[Gru93] : Gruber, T. "*Toward principles for the design of ontologies used for knowledge sharing*". Guarino & Poli, Eds. In formal ontology in conceptual analysis and knowledge representation, Padova, Italy, 1993, 23p.

[Hea0] : K. Heather. *"Web services conceptual architecture (WSCA 1.0)",* may 2001. URL: http : //www-06.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[Hei96] : Van Heijst G., Van der Speek R., Kruizinga E. (1996) *"Organizing Corporate Memories", In B. Gaines, M. Musen end, Proceeding of the 10th Baff Knowledge. Acquisition forKnowledge-based Systems Workshop (KAW'96)*, Banff, Canada, Novembre 1996.

[IRS] : http://kmi.open.ac.uk/projects/irs/

[ISO99] : ISO/IEC 13250. Topics Maps, Dec. ISO/IEC FCD, 1999.

[Jam05] : Sonia JAMAL : *« Environnement de procédé extensible pour l'orchestration :Application aux services web ».* Thèse de doctorat, l'Université Joseph Fourier de Grenobl, Décembre 2005.

[Kel03] : Patrick Kellert et Farouk Toumani *:* "*les web services sémantiques*" Research Report LIMOS/RR 03-15, 11 Juillet 2003.

[Kly03] : KLYNE Graham & CARROLL Jeremy, Eds. (2003). *"Resource Description Framework (RDF): Concepts and Abstract Syntax"*. W3C Working Draft, 2003. URL: http://www.w3.org/TR/rdf-concepts/

[Koi01] : Koivunen M. et Miller E. (2001). "*W3c semantic web activity*"**.** Proceedings of the Semantic Web Kick-off Seminar. W3C
URL: http ://www.w3.org/2001/12/semweb-fin/w3csw.

[Läm02] : S. Lämmermann. "*Runtime Service Composition via Logic-Based Program Synthesis"*. PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, June 2002.

[Las99] : LASSILA Ora & SWICK Ralph, Eds. (1999). *"Resource Description Framework (RDF) Model and syntax specification"***.** Recommendation, W3C. URL : http://www.w3.org/TR/REC-rdf-syntax.

[Lau02] : Laublet P., Reynaud C., Charlet J, « *Sur Quelques Aspects du Web Sémantique »*, aux assises I3, Nancy, 2002.

[Lég03] : Alain Léger et Jean Charlet *« Applications du web sémantique»* dans le r*apport de l'action spécifique CNRS-STIC sur le Web sémantique* (J. Charlet, P. Laublet, C. Reynaud, Eds), déc. 2003.

[Lev02] : David Levy : « *Coordination de Web Services : « Langages de description et plate-formes d'exécution »*. Rapport de recherche XRCE Grenoble, septembre 2002.

[Lev97] : Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. GOLOG: *"A logic programming language for dynamic domains"*. Journal of Logic Programming, 1-3 (1997).

[Lop04] : Celine Lopez, 2004 « *service web et adaptabilité à l'utilisateur* » thèse de Master en mathématique Informatique, juin 2004, Université de Grenoble.

[Maj104] : Shalil Majithia, David W. Walker, W. A. Gray*: "A Framework for Automated Service Composition in Service-Oriented Architectures*". In Proceedings of DBLP conference ESWS (European Semantic Web Symposium) Heraklion, Crete, Greece 2004, p: 269-283.

[Maj204] : Shalil Majithia, Matthew S. Shields, Ian J. Taylor, Ian Wang: Triana: *"A Graphical Web Service Composition and Execution Toolkit"***.** In proc of ICWS conference 2004, p: 514-522.

[Maj03] : Taylor, I., Shields, M., Wang, I., and Philp, R: *"Grid Enabling Applications Using Triana, Workshop on Grid Applications and Programming Tools"*, June 25, 2003, Seattle. In conjunction with GGF8

[Man80] : Z. Manna and R. J. Waldinger. *"A deductive approach to program synthesis"***.** *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.

[McI02] : S. McIlraith and T. C. Son. "*Adapting Golog for composition of Semantic Web services"*. In proceedings of the 8th International Conference on Knowledge Representation and Reasoning(KR2002), Toulouse, France, April 2002.

[McD02] : D. McDermott. *"Estimated-regression planning for interactions with Web services"*. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*, Toulouse, France, 2002. AAAI Press.

[Med03] : B . Medjahed, A. Bouguettaya, and A. K. Elmagarmid. *"Composing Web services on the Semantic Web"*. *The VLDB Journal*, 12 (4), November 2003.

[Med04] : B . Medjahed. *"Semantic Web Enabled Composition of Web Services"*. PhD thesis in Computer Science and Applications, Virginia USA, January 2004.

[Mel04] : Tarek MELLITI, *« Interopérabilité des services Web complexes, Application aux systèmes multi-agents»*, thèse de Doctorat de l'Université Paris IX Dauphine, décembre 2004.

[Meteor-S] : *"METEOR-S: Semantic Web Services and Processes"*, URL: http://swp.semanticweb.org, (2002).

[NAICS] : *"North American Industry Classification System (NAICS)"*, http://www.census.gov/epcd/www/naics.html

[Nar02] : S. Narayanan and S. McIlraith. *"Simulation, verification and automated composition of Web service"*. In Proceedings of the 11th International World Wide WebConference, Honolulu, Hawaii, USA, May 2002. ACM. presentation available at http://www2002.org/presentations/narayanan.pdf.

[Ouz04] : Ouzzani, M., "*Efficient Delivery of Web Services"*, PhD Thesis, Virginia Polytechnic, (2004).

[Ouz03] : Mourad Ouziri, Partie Etat de l'art de *« Application des Topics Maps pour l'interrogation et le génération des documents virtuels»*, thèse de doctorat, Lyon, Décembre 2004.

[OWL] : http://www.w3.org/TR/owl-features/
http://www.ai.sri.com/daml/services/owl-s/1.2/

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

[Pao02] : Paolucci, M., Kawamura, T., Payne, T., Sycara, K., "*Semantic Matching of Web Services Capabilities*", in Proc. of Intl. Semantic Web Conference, Sardinia, Italy, pages 333–347. June 2002.

[Pil03] : Thomi Pilioura, Aphrodite Tsalgatidou , Alexandros Batsakis. *"Using WSDL/UDDI and DAML-S in Web Service Discovery"*. WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), Budapest, May 2003.

[Plo01] : Nicolas Ploquin, « *Etudes des différents modes d'interopérabilité en génie logiciel* », thèse de DEA Informatique de l'Université de Nantes, septembre 200.

[Pom96] : POMIAN J. (1996), *« Mémoire d'entreprise : techniques et outils de la gestion du savoir »*, Paris, Sapientia.

[Pon04] : Julien Ponge, 2004 *« Compatibilité et substitution dynamique des web services : Prise en compte de contraintes temporelles et de propriétés d'activation »*. Thèse de DEA IPIM, France

[Pon02] : S. R. Ponnekanti and A. Fox. *"SWORD: A developer toolkit for Web service composition"*. In Proceedings of the 11th World Wide Web Conference, Honolulu, HI, USA, 2002.

[Pra90] : Pratt, V. *"Action logic and pure induction"*. In Proc. Logics in AI: European Workshop JELIA '90 (1990), J. van Eijck, Ed., Springer-Verlag. Lecture Notes in Computer Science, pp. 97–120.

[Pré04] : *« Précis de répartition: Définition et problématique »*, 2004 disponible sur : http:www.enseeiht.fr/~padiou/3AI/CH1/main.pdf

[RACER] : http://www.racer-systems.com/

[Rao03] : Rao, P. K¨ungas, and M. Matskin. *"Application of Linear Logic to Web service composition"*. In Proceedings of the 1st International Conference on Web Services, Las Vegas, USA, June 2003.

[Rao104] : J. Rao, P. K¨ungas, and M. Matskin. *"Logic-based Web services composition: from service description to process model"*. In Proceedings of the 2004 International Conference on Web Services, San Diego, USA, July 2004. IEEE.

[Rao204] : Jinghai Rao and Xiaomeng Su, *"A Survey of Automated Web Service Composition*

Methods" Norwegian University of Science and Technology Department of Computer and Information Science N-7491, Trondheim, Norway, 2004

[RDF] : http://www.w3.org/RDF/

[RDFS] : MAGKANARAKI Aimilia et al, *"Benchmarking RDF Schemas for the Semantic Web"*, in Proceedings of the First International Semantic Web Conference, Sardinia, June 2002, LNCS 2342, Springer 2002. p.132-146.

[RosettaNet]: RosettaNet Consortium. *RosettaNet*. http://www.rosettanet.org.

[Siv03] : Sivashanmugam, K., K. Verma, A. Sheth and J. Miller, 2003. *"Adding Semantics to Web Services Standards"*. Proc. of the 1st Int. Conf. on Web Services (ICWS'03), Las Vegas, pp: 395-401.

[Sir03] : E. Sirin, J. Hendler, and B. Parsia. *"Semi-automatic composition of Web services using semantic descriptions"*. In Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS 2003.

[Sir05] : Sirin, E., et al, *"Template-based composition of semantic web services"*, AAAI Fall Symposium on Agents and the Semantic Web, Virginia, November 2005.

[SOAP] : W3C (2003), *"Simple Object Access Protocol (SOAP)"*. http://www.w3.org/TR/soap.

[Tal04] : Muhammad Adeel Talib, Yang Zongkai and Qazi Mudassar Ilyas: *"Modeling the Flow in Dynamic Web Services Composition", Information Technology Journal 3 (2): 184-187, ISSN 1682-6027, 2004, Republic of China.*

[UDDI] : W3C (2003), *"Universal description, discovery, and integration (UDDI)"*. http://www.uddi.org

[URI] : http://websemantique.org/URI

[Ush96] : *"Ontologies: Principles, methods and applications"*. *Knowledge Engineering Review*, 1996.

[Via04] : Maxime vialette, « *Web Service : communication inter- langage* », Sun Microsystem laboratoire SUPINFO des technologies, 2004.

[Wik06] : Wikipedia l'encyclopedie libre. Disponible sur l'URL: http://fr.wikipedia.org/wiki/Architecture_Orient%C3%A9e_Services

[Wal01] : R.Waldinger. *"Web agents cooperating deductively"*. In Proceedings of FAABS 2000, Greenbelt, MD, USA, April 5–7, 2000, volume 1871 of Lecture Notes in ComputerScience, pages 250–262. Springer-Verlag, 2001.

[WSDL] : W3C (2003), *"Web Services Description Language (WSDL)"*. http://www.w3.org/TR/wsdl.

[WSFL] : IBM (2003), *"Web Services Flow Language (WSFL)"* disponible sur l'URL: http://xml.coverpages.org/wsfl.html

[WSMF] : http://www.swsi.org/resources/wsmf-paper.pdf

[WSMQ] : "*WebSphere MQ*", IBM http://www-306.ibm.com/software/integration/wmq/

[Wu03] : D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. *"Automatic Web services composition using SHOP2"*. In *Workshop on Planning for Web Services*, Trento, Italy, June 2003.

[XL] : Florescu D, Gr¨unhagen A, Kossmann D (2002) *"XL: an XML Programming Language forWeb service specification and composition"*. In: Proceedings of theWWW 2002 conference, Honolulu, May 2002, pp 65–76.

[XLANG] : Microsoft (2003), *"Web Services for Business Process Design (XLANG)"*.

disponible sur: http://xml.coverpages.org/xlang.html.

[Yan03] : Yannick. Prié, Serge. Garlatti (2003) *"Annotations et métadonnées dans le Web sémantique "*, dans le r*apport de l'action spécifique CNRS-STIC sur le Web sémantique* (J. Charlet, P. Laublet, C. Reynaud, Eds), déc. 2003.

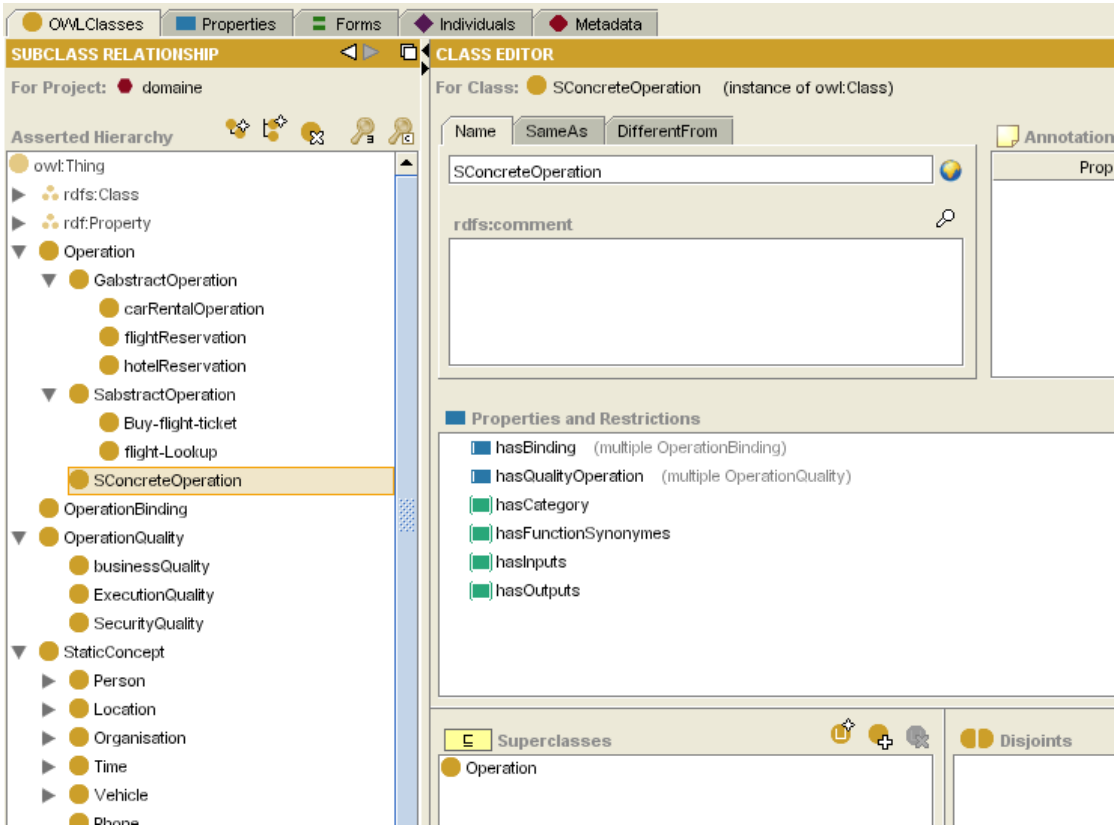
[ZYL 00] : ZYL J.& CORBETT D, *"A framework for Comparing the use of a Linguistic Ontology in an Application"*, *Workshop Applications of Ontologies and Problem-solving Methods,ECAI'2000,* Berlin Germany, August.

Table des figures

Annexe:

Création de l'ontologie proposée en OWL

OWL comme DAML-OIL adoptent une approche orientée objet pour décrire les ontologies, c'est à dire en terme de classes de concepts, de propriétés propres aux concepts et relations entre les classes et propriétés. Nous avons utilisé comme outil de création de l'ontologie Protégé2000[1]. Ce dernier constitue un environnement graphique qui autorise une liberté de conception des ontologies OWL et RDFS.



Aperçu général sur la structure de l'ontologie créée avec Protégé 2000

Le code source OWL de l'ontologie créée:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>

---------------Déclarations des opérations du domaine----------------
  <owl:Class rdf:ID="carRentalOperation">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="GabstractOperation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="hotelReservation">
    <rdfs:subClassOf>
    <owl:Class rdf:about="#GabstractOperation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Buy-flight-ticket">
    <rdfs:subClassOf rdf:resource="#SabstractOperation"/>
  </owl:Class>
  <owl:Class rdf:ID="flight-Lookup">
    <rdfs:subClassOf rdf:resource="#SabstractOperation"/>
  </owl:Class>
  <owl:Class rdf:ID="SabstractOperation">
    <rdfs:subClassOf>
    <owl:Class rdf:ID="Operation"/>
    </rdfs:subClassOf>
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >decrit une opération abstraite Simple</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="flightReservation">
    <rdfs:subClassOf>
    <owl:Class rdf:about="#GabstractOperation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#GabstractOperation">
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >decrit une opération abstraite simple</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Operation"/>
  </owl:Class>
  <owl:Class rdf:ID="SConcreteOperation">
    <rdfs:subClassOf rdf:resource="#Operation"/>
  </owl:Class>
  <owl:Class rdf:ID="OperationBinding"/>
  <owl:Class rdf:ID="ExecutionQuality">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="OperationQuality"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="businessQuality">
    <rdfs:subClassOf rdf:resource="#OperationQuality"/>
  </owl:Class>
```

```
  <owl:Class rdf:ID="SecurityQuality">
    <rdfs:subClassOf rdf:resource="#OperationQuality"/>
  </owl:Class>

------------------déclaration des concepts statiques---------------
  <owl:Class rdf:ID="Bus">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Vehicle"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Adresse">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Location"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Hotel">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Organisation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Area">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Location"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Duration">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Time"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Phone">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="StaticConcept"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Seat">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="AirCraft"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="moment">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Time"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ServiceSupplier">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Person"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Food">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:about="#Time">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:about="#Location">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:ID="Date">
    <rdfs:subClassOf rdf:resource="#Time"/>
  </owl:Class>
```

```
  <owl:Class rdf:ID="Country">
    <rdfs:subClassOf rdf:resource="#Area"/>
  </owl:Class>
  <owl:Class rdf:about="#Person">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:ID="CarRentalAgency">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Organisation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Beverage">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:about="#Vehicle">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:ID="Ticket">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:ID="Car">
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
  </owl:Class>
  <owl:Class rdf:ID="Airport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Organisation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#AirCraft">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:ID="City">
    <rdfs:subClassOf rdf:resource="#Country"/>
  </owl:Class>
  <owl:Class rdf:ID="Meal">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>
  <owl:Class rdf:ID="Client">
    <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>
  <owl:Class rdf:about="#Organisation">
    <rdfs:subClassOf rdf:resource="#StaticConcept"/>
  </owl:Class>

----------------déclaration des propriétés des classes-------------

  <owl:ObjectProperty rdf:ID="hasBeverage">
    <rdfs:domain rdf:resource="#Meal"/>
    <rdfs:range rdf:resource="#Beverage"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasManufacturer">
    <rdfs:domain rdf:resource="#Vehicle"/>
    <rdfs:range rdf:resource="#Organisation"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasQualityOperation">
    <rdfs:domain rdf:resource="#SConcreteOperation"/>
    <rdfs:range rdf:resource="#OperationQuality"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasPhone">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
```

```
          <owl:Class rdf:about="#Person"/>
          <owl:Class rdf:about="#Organisation"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasParticipantsOp">
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >toutes les operations participantes (sou-opération), il peut
être vide si on les connait pas encore</rdfs:comment>
    <rdfs:domain rdf:resource="#GabstractOperation"/>
    <rdfs:range rdf:resource="#SabstractOperation"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasAdress">
    <rdfs:range rdf:resource="#Adresse"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Class rdf:about="#Organisation"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasTimeZone">
    <rdfs:domain rdf:resource="#Area"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasBinding">
    <rdfs:range rdf:resource="#OperationBinding"/>
    <rdfs:domain rdf:resource="#SConcreteOperation"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasFood">
    <rdfs:range rdf:resource="#Food"/>
    <rdfs:domain rdf:resource="#Meal"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="Availability">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#ExecutionQuality"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Cost">
    <rdfs:domain rdf:resource="#businessQuality"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Reputation">
    <rdfs:domain rdf:resource="#businessQuality"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasDuration">
    <rdfs:domain rdf:resource="#Duration"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#duration"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasSynonymes">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasOutputs">
    <rdfs:domain rdf:resource="#Operation"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasArrivalCity">
    <rdfs:subPropertyOf>
      <owl:DatatypeProperty rdf:ID="hasInputs"/>
```

```
    </rdfs:subPropertyOf>
    <rdfs:domain rdf:resource="#flightReservation"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasDateTime">
    <rdfs:domain rdf:resource="#Date"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasDescription">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Bref    description    textuelle    de    ce    que    l'opération
fait</rdfs:comment>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasFunction">
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Quelle fonctionnalité métier assure l'opération</rdfs:comment>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="FlightNumber">
    <rdfs:domain rdf:resource="#Buy-flight-ticket"/>
    <rdfs:subPropertyOf>
      <owl:DatatypeProperty rdf:about="#hasInputs"/>
    </rdfs:subPropertyOf>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasTime"/>
  <owl:DatatypeProperty rdf:ID="hasFunctionSynonymes">
    <rdfs:domain rdf:resource="#Operation"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasDepartureDate">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
    <rdfs:domain rdf:resource="#flightReservation"/>
    <rdfs:subPropertyOf>
      <owl:DatatypeProperty rdf:about="#hasInputs"/>
    </rdfs:subPropertyOf>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Origine">
    <rdfs:domain rdf:resource="#flight-Lookup"/>
    <rdfs:subPropertyOf>
      <owl:DatatypeProperty rdf:about="#hasInputs"/>
    </rdfs:subPropertyOf>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="No-repudiation">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
    <rdfs:domain rdf:resource="#SecurityQuality"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasFName">
    <rdfs:subPropertyOf rdf:resource="#hasFunction"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasMessageFormat">
    <rdfs:domain rdf:resource="#OperationBinding"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasName">
```

```
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasCategory">
    <rdfs:domain rdf:resource="#Operation"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasTypeOpeartion">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasInputs">
    <rdfs:domain rdf:resource="#Operation"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasfare">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Ticket"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasProducer">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#AirCraft"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasTypeFood">
    <rdfs:domain rdf:resource="#Food"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Reliability">
    <rdfs:domain rdf:resource="#ExecutionQuality"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasNumberPhone">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Phone"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasDepartureCity">
    <rdfs:subPropertyOf rdf:resource="#hasInputs"/>
    <rdfs:domain rdf:resource="#flightReservation"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="departureDate">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
    <rdfs:domain rdf:resource="#flight-Lookup"/>
    <rdfs:subPropertyOf rdf:resource="#hasInputs"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasQualityDescription">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#OperationQuality"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasMoment">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#time"/>
    <rdfs:domain rdf:resource="#moment"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="flightNumber">
    <owl:equivalentProperty>
      <owl:DatatypeProperty rdf:ID="hasFlightNumber"/>
    </owl:equivalentProperty>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#flight-Lookup"/>
    <rdfs:subPropertyOf rdf:resource="#hasOutputs"/>
```

```
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasReturnDate">
    <rdfs:subPropertyOf rdf:resource="#hasInputs"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
    <rdfs:domain rdf:resource="#flightReservation"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasFlightNumber">
    <owl:equivalentProperty rdf:resource="#flightNumber"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:subPropertyOf rdf:resource="#hasOutputs"/>
    <rdfs:domain rdf:resource="#flightReservation"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="CreditCardNumber">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Buy-flight-ticket"/>
    <rdfs:subPropertyOf rdf:resource="#hasInputs"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasFlightSeatNumber">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#flightReservation"/>
    <rdfs:subPropertyOf rdf:resource="#hasOutputs"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasFSynonymes">
    <rdfs:subPropertyOf rdf:resource="#hasFunction"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasSpecialisation">
    <rdfs:domain rdf:resource="#SabstractOperation"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Confidentiality">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#SecurityQuality"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasProtocole">
    <rdfs:domain rdf:resource="#OperationBinding"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="responseTime">
    <rdfs:domain rdf:resource="#ExecutionQuality"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasTotalSeatNumber">
    <rdfs:domain rdf:resource="#Seat"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="arrivalDate">
    <rdfs:domain rdf:resource="#flight-Lookup"/>
    <rdfs:subPropertyOf rdf:resource="#hasInputs"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="BuyTicketConfirm">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Buy-flight-ticket"/>
    <rdfs:subPropertyOf rdf:resource="#hasOutputs"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasAirlineName">
```

```
    <rdfs:domain rdf:resource="#flightReservation"/>
    <rdfs:subPropertyOf rdf:resource="#hasOutputs"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <owl:equivalentProperty>
      <owl:DatatypeProperty rdf:ID="airlineName"/>
    </owl:equivalentProperty>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasNumberOfSeats">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#AirCraft"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasTypeBeverage">
    <rdfs:domain rdf:resource="#Beverage"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasOrgNames">
    <rdfs:domain rdf:resource="#Organisation"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="destination">
    <rdfs:domain rdf:resource="#flight-Lookup"/>
    <rdfs:subPropertyOf rdf:resource="#hasInputs"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#airlineName">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#flight-Lookup"/>
    <rdfs:subPropertyOf rdf:resource="#hasOutputs"/>
    <owl:equivalentProperty rdf:resource="#hasAirlineName"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="HasRegistrationNumber">
    <rdfs:domain rdf:resource="#Vehicle"/>
  </owl:DatatypeProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.1, Build 284)
http://protege.stanford.edu -->
```