PROTOCOLE DE DECOUVERTE SENSIBLE AU CONTEXTE POUR LES SERVICES WEB SEMANTIQUES

Ana Roxin

Numéro 125 Version du 25 novembre 2009

THESE

Préparée à

L'Université de Technologie de Belfort-Montbéliard

Pour obtenir le grade de

Docteur de l'Université de Technologie de Belfort-Montbéliard et l'Université de Franche-Comté

Spécialité Informatique

Par

Ana ROXIN

PROTOCOLE DE DECOUVERTE SENSIBLE AU CONTEXTE POUR LES SERVICES WEB SEMANTIQUES

Soutenue publiquement le 30 novembre 2009

Composition du jury:

Rapporteurs :	**Daniel JOLLY**, Professeur, Université d'Artois, Béthune (France)
	Jean-Marie PINON, Professeur, INSA, Lyon (France)
Examinateurs :	**Alexandre CAMINADA**, Professeur, UTBM, Belfort (France)
	Jacques SAVOY, Professeur, UNINE, Neuchâtel (Suisse)
Directeur de thèse :	**Maxime WACK**, Maître de conférences HDR, UTBM, Belfort (France)
Co-encadrant :	**Jaafar GABER**, Maître de conférences, UTBM, Belfort (France)

REMERCIEMENTS

Le présent travail a été mené au sein du laboratoire Systèmes et Transports (SeT), de l'Université de Technologie de Belfort-Montbéliard (UTBM), sous la direction de Monsieur Maxime WACK, pour la partie scientifique, en collaboration avec la société PIMENTIC, spécialisée dans les systèmes embarqués, présidée par Monsieur Jean-Claude RAGRIS, pour l'industrialisation des résultats scientifiques obtenus. Je leur adresse ma profonde gratitude pour m'avoir permis d'intégrer leurs structures respectives dans le cadre de mon contrat CIFRE.

Nous sommes particulièrement sensibles à l'honneur que nous font Messieurs les Professeurs Jacques SAVOY et Alexandre CAMINADA en acceptant d'être membres du jury.

Je remercie chaleureusement Messieurs les Professeurs Jean-Marie PINON et Daniel JOLLY pour avoir accepté la charge de travail qu'implique la lecture critique du présent mémoire. Leurs remarques pertinentes et conseils avisés nous ont amené à de nouvelles réflexions et perspectives de recherche.

Je remercie vivement mon directeur de thèse, Monsieur WACK, Maître de Conférences HDR à l'UTBM, ainsi que mon co-encadrant, Monsieur Jaafar GABER, Maître de Conférences, pour la qualité de leur encadrement. Tous deux m'ont initié à la recherche et m'ont guidé tout au long de l'élaboration de ce travail. Ils m'ont encouragé à m'impliquer dans leurs thématiques de recherche sur la découverte de services Web sémantiques.

Je remercie également les enseignants et administratifs de l'UTBM, notamment Monsieur Ahmed NAIT-SIDI-MOH pour sa relecture attentive et ses conseils avisés. Je remercie également mes collègues de travail de PIMENTIC pour leur chaleureux accueil. Ils m'ont permis d'effectuer mes recherches dans une ambiance conviviale.

Monsieur Ioan SZILAGY, doctorant à l'Université de Franche-Comté, m'a en outre été d'un immense secours et je l'en remercie à cet égard.

Enfin, mes parents et mon fiancé m'ont supporté tout au long de ces trois années et ont volontiers relu mon mémoire avec l'œil du néophyte.

Ana ROXIN

RESUME

Le Web d'aujourd'hui représente un espace où les utilisateurs recherchent, découvrent et partagent des informations. Dans ce cadre, les processus de découverte de services Web jouent un rôle fondamental. Un tel processus permet de faire le lien entre des informations publiées par des fournisseurs de services et des requêtes créées par les internautes. Généralement, un tel processus repose sur une recherche « textuelle » ou à base de « mots-clés ». Or, ce type de recherche ne parvient pas à toujours identifier les services les plus pertinents. Notre idée est de concevoir un système plus « intelligent », permettant d'utiliser, lors du processus de découverte, une base de connaissances associées aux informations, comme c'est le cas pour le Web sémantique.

Cette thèse présente un prototype pour la découverte de services Web sémantiques, utilisant des caractéristiques non-fonctionnelles (descriptives) des services. Notre approche emploie le langage OWL-S (Web Ontology Language for Services) pour définir un modèle de description des paramètres non-fonctionnels des services. Ce modèle a pour but de faciliter la découverte de services Web sémantiques. Ce modèle représente le centre de notre contribution, étant utilisé pour la conception des interfaces et des requêtes. Deux interfaces sont développées, l'une s'adressant aux fournisseurs de services, alors que la deuxième interface est conçue pour l'utilisateur final. L'algorithme de recherche présenté dans cette thèse a pour but d'améliorer la précision et la complétude du processus de découverte de services.

MOTS-CLES

Web sémantique, RDF, OWL, services Web sémantiques, découverte de services, découverte de services Web sémantiques.

ABSTRACT

The Web of today represents a broad space where users research, discover and share information. In this contexte, the process of Web service discovery plays a significant role. Indeed, such process allows linking the information published by service providers and the requests of users, looking for information. Generally, such process involves text or keyword-based techniques. Still, this kind of discovery fails in retrieving only relevant information. Our approach is to design a more « intelligent » system that allows using a knowledge base during the service discovery process, as it is done in the Semantic Web vision.

This thesis presents a framework for semantic Web service discovery using non-functional service characteristics. The framework relies on the Web Ontology Language for Services (OWL-S) to design a template for describing non-functional service parameters in a way that facilitates service discovery. Our model is designed to facilitate the discovery of semantic Web services. This service description model is used as a core for designing the interfaces and the querries. Two interfaces are developped : one for the service providers, and one for the final users. The search algorithm presented in this thesis is designed to maximize precision and completeness of service discovery.

KEYWORDS

Semantic Web, RDF, OWL, semantic Web service, service discovery, semantic Web service discovery.

TABLE DES MATIERES

TABLE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE 1.
INTRODUCTION

Lorsque nous parcourons le Web, lorsque nous remplissons un formulaire ou lorsque nous effectuons un achat, nous prenons partie dans une opération distribuée, dont nous connaissons que très peu les composants. Les services Web sont intéressants en cela qu'ils fournissent une approche pour construire et déployer de telles opérations distribuées, tout en améliorant la productivité des programmeurs, mais aussi celle des utilisateurs.

La plupart des chercheurs et des « praticiens » sont d'accord pour dire que le Web d'aujourd'hui, même s'il est une réussite, a de nombreuses limitations. L'information n'est pas organisée sur le Web; elle peut être imprécise, incomplète ou même, pire, incompréhensible. Les techniques actuelles pour rechercher des informations manquent à découvrir certaines informations pertinentes.

Le Web est sans autorité centrale et implique des composants hétérogènes et autonomes. Aujourd'hui ces composants sont principalement des pages Web, mais, avec le temps, ce seront des programmes en général. En d'autres mots, le Web actuel fournit à la fois du contenu et des services. Le Web est dynamique étant donné les changements arbitraires que peuvent subir ses composants. Toutefois, cette dynamique est limitée, puisque les composants ne peuvent négocier l'ensemble des aspects définissant leurs interactions.

La principale vision de l'avenir du Web est la vision du *Web Ubiquitaire*. En effet, Mark Weiser a défini la vision dite de l'informatique ubiquitaire (ou *ubiquitous computing*) : l'ère de l'informatique ubiquitaire est caractérisée par l'enfouissement des différentes technologies d'accès à l'information[1], dans notre environnement de tous les jours (WEISER 1991). L'une des promesses de l'informatique ubiquitaire est de permettre aux utilisateurs d'être en permanence connectés avec d'autres utilisateurs, mais aussi avec des applications et des services Internet (BAKHOUYA 2008). Dans la vision de Weiser, le Web actuel représente une transition des connexions du type un-à-un (entre utilisateurs et ordinateurs) vers des connexions du type un-à-plusieurs, qui permettront à une multitude d'ordinateurs de répondre aux besoins d'un utilisateur, à tout moment et quelque soit son environnement. Le *Web ubiquitaire* est une vision à moyen terme, centrée sur l'utilisateur, et transparente pour celui-ci.

Dans la vision du *Web sémantique*, la page Web contient des annotations concernant les aspects de présentation, mais aussi des annotations fournissant une représentation, à part, du sens du contenu de la page Web. Ceci permet de passer d'une approche syntaxique (ne capturant que la structure de l'information) à une approche sémantique (capturant le sens de l'information).

L'évolution des paradigmes de communication est une autre idée gagnant en popularité (BAKHOUYA 2008). Actuellement, le Web est utilisé à travers des interactions du type client-serveur: l'information est stockée du côté serveur et l'intelligence est stockée du côté client. Cette asymétrie entre les deux

[1] « *The most profound technologies are those that disappear* » (WEISER 1991)

parties dénote le fait que l'information a tendance à être stockée dans de grands serveurs, qui deviennent vitaux pour le fonctionnement du système entier. Si ces serveurs sont amenés à subir une panne, l'ensemble du système est affecté. Si les serveurs sont compromis, la sécurité du système l'est aussi.

La vision du *Web pair-à-pair* (de l'anglais *peer-to-peer* - P2P) considère les différentes entités (ou composants) du Web en tant que pairs, chaque composant étant l'égal d'un autre. Chaque entité comprend à la fois des aspects serveur et client. Dans cette vision, le Web n'est plus un ensemble de pages statiques auxquelles des programmes accèdent. Le Web pair-à-pair est constitué de programmes actifs, qui communiquent entre eux. Ces programmes peuvent mener des négociations entre eux ou encore se transmettre des suggestions entre eux.

Toutefois, les approches P2P actuelles n'intègrent pas l'approche sémantique (BAKHOUYA 2008) : les applications P2P doivent spécifier leurs interactions à travers des codages « en dur ». Les applications P2P ne peuvent pas mener des négociations flexibles, à la « volée ». C'est pour cette raison qu'actuellement, les approches P2P se limitent à des applications simples, dans lesquelles les utilisateurs fournissent les sémantiques, telles des applications de partage de fichiers.

Lorsque les visions présentées ci-dessus sont rassemblées, nous obtenons la vision plus globale d'un *Web permettant de supporter des processus dans un contexte*. Cette vision est celle d'un Web actif, pouvant être utilisé, à la fois, par les humains et les ordinateurs. Les principales caractéristiques de cette vision du Web sont (ROXIN et al. 2008a):

- ***L'automatisation*** pour permettre le passage de l'humain à l'ordinateur.
- ***Les annotations riches*** pour capturer les sémantiques (ou le sens formel) des documents et des structures.
- ***Les nouveaux paradigmes d'interaction*** pour passer du modèle client/serveur vers un modèle coopératif.
- ***La prise en compte du contexte***, à travers la sémantique, pour fournir une compréhension mutuelle entre les entités du Web.

Cette vision du Web rejoint la vision plus générale de Weiser. Dans les deux visions, « *il s'agit de passer de systèmes et d'interfaces fixes et à priori connus, à des environnements nécessairement personnels à chaque utilisateur* » (VALLEE 2006). Or ces nouveaux environnements doivent faire preuve d'une grande flexibilité et d'une grande résistance aux pannes. Pour ce faire, il faut pouvoir fournir aux utilisateurs des services prenant en compte leur contexte d'utilisation. Ces services doivent intégrer des informations définissant le contexte de l'utilisateur (sa position, ses préférences, les caractéristiques de son terminal mobile, etc.) afin de s'y adapter, et ce quelque soit le type de réseau ou le type de terminal mobile utilisé.

Afin d'illustrer cette idée, nous pouvons prendre un exemple de notre vie quotidienne. En effet, lors de conversations, il nous arrive souvent d'omettre certaines informations pouvant être déduites des circonstances dans lesquelles la conversation se déroule. Par exemple, nous allons rarement dire « *Il pleut. Il faut prendre un parapluie.* », mais « *Je vais prendre un parapluie* ». C'est la conscience que l'on a de notre environnement (ou contexte), qui permet un échange efficace des idées. Nous retrouvons ici l'idée de base de la vision du Web futur: la prise en compte du contexte permet une compréhension mutuelle entre des entités.

De la même manière, les systèmes informatiques doivent être réceptifs aux désirs de l'utilisateur, notamment en déduisant les intentions de l'utilisateur à partir d'informations auxiliaires mais pertinentes par rapport au but envisagé. Ces informations sont appelées informations contextuelles et les applications réagissant en fonction de ce type d'information sont appelées *applications sensibles au contexte*. Il s'agit d'applications réseau fournissant aux utilisateurs des services sensibles au contexte.

Les services sensibles au contexte sont par définition de nature hétérogène. Chaque service a des propriétés, des capacités, des interfaces et des schémas d'invocation qui lui sont propres. La fourniture de tels services pose de nombreuses questions: comment standardiser la description de ces services, comment capturer leur sémantique, comment prendre en compte les besoins utilisateur et comment les traiter? Répondre à ces questions est une tâche d'autant plus difficile que les services sensibles au contexte possèdent des caractéristiques bien particulières (LASSILA 2005) :

- ***Mobilité*** - Il s'agit d'avoir une capacité de calcul continue lors des déplacements d'une position à l'autre. Les besoins associés sont ceux de l'informatique mobile sur des terminaux portables avec des logiciels embarqués.
- ***Sensibilité au positionnement*** - Il s'agit de pouvoir détecter et identifier des positions de terminaux et/ou personnes. Les besoins associés sont ceux du positionnement à l'intérieur et à l'extérieur (ROXIN et al. 2007).
- ***Interopérabilité*** - Il s'agit de permettre des opérations interopérables entre différents standards d'échanges de ressources, de composition et d'intégration de services. Les besoins associés sont ceux des standards de contenu, services et protocoles de communication.
- ***Sans coutures*** - Il s'agit de proposer des sessions de service continues, avec n'importe quel type de connexion et sur n'importe quel type de terminal mobile.
- ***Sensibilité à la situation*** - Il s'agit de pouvoir détecter et identifier des scénarios de situations. Les besoins impliquent la capacité de connaître l'activité d'une personne et son contexte (quand, où, avec qui, etc.) (ROXIN et al. 2008).
- ***Adaptation dans le temps*** - Il s'agit de permettre un ajustement dynamique du service et/ou de son contenu, en fonction des besoins de l'utilisateur. Les besoins associés impliquent la connaissance du niveau d'accessibilité et des préférences de l'utilisateur.
- ***Pervasivité*** - Il s'agit de permettre un accès transparent au service et/ou à son contenu. Les besoins associés renvoient à la capacité de déterminer les besoins de l'utilisateur avant que celui-ci ne les ait exprimés.

Ces caractéristiques posent des défis considérables et soulèvent de nombreuses questions:

- Comment découvrir les ressources nécessaires ?
- Comment les utiliser ?
- Comment vérifier qu'elles répondent à telle ou telle contrainte ?

Parmi ces questions, celle soulevant le problème de la découverte des ressources dans un environnement ouvert et, de par ce fait, hétérogène, constitue la problématique centrale de cette thèse. Elle est expliquée plus en détail dans la partie suivante.

1.1 PROBLEMATIQUE

La vision du Web futur est centrée sur l'utilisateur et non-obstrusive. Avec l'augmentation du nombre de capteurs et de terminaux mobiles, cette vision est devenue une vision à moyen terme. La technologie existante aujourd'hui permet à l'informatique de rentrer dans nos vies, de manière transparente, et donc de mettre en place un Web tel que décrit plus haut.

Or, un des principaux défis d'un tel environnement ubiquitaire est le développement de protocoles de découverte de services permettant aux applications et utilisateurs de découvrir et interagir avec les services qui répondent le mieux à leurs requêtes. La difficulté réside non seulement dans la multitude de fournisseurs de services présents aujourd'hui sur le marché, mais aussi dans la dynamique des environnements ubiquitaires (ZHU et al. 2005). En effet, un service donné peut ne pas être disponible à tout moment, et un utilisateur peut à tout moment déconnecter son terminal du réseau. De plus, les services disponibles ont chacun des propriétés, des capacités, des interfaces et des schémas d'invocation différents. Les différents standards pour la description des services ne réussissent pas à capturer l'ensemble de ces différences. Ceci est une des principales raisons pour lesquelles, les protocoles de découverte actuels ne suffisent plus pour les services de demain.

Dans (ZHU et al. 2005), les auteurs étudient les différents protocoles de découverte de services proposés, conçus et implémentés. Leur analyse est que même s'ils partagent tous le même but de fournir un mécanisme pour la publication et la découverte de services, ils sont très différents par rapport à des aspects tels l'architecture déployée ou l'environnement dans lequel ils fonctionnent (réseaux LAN, réseaux mobiles, Internet, etc.). Les protocoles de découverte actuels ne correspondent pas aux environnements ubiquitaires, et ce pour les raisons suivantes:

- Premièrement, ***plusieurs descriptions de services peuvent être syntaxiquement différentes, mais sémantiquement équivalentes*** (synonymes) (Synonymes 2009), (ZHU et al. 2005). Or, les protocoles de découverte actuels sont basés sur une description syntaxique des services et n'intègrent que des mécanismes de recherche du type « mots-clés ». Par conséquent, en cherchant un service « *Restaurant* », l'utilisateur ne pourra pas trouver un service « *Fast food* ». Les requêtes ainsi construites fournissent des résultats peu précis et ne permettent de découvrir qu'un faible nombre de résultats pertinents parmi l'ensemble des résultats correspondants.
- Deuxièmement, ***plusieurs descriptions de services peuvent être équivalentes du point de vue de la syntaxe, mais différentes du point de vue de la sémantique***. En effet, deux mots peuvent avoir la même forme graphique (s'écrivent de la même manière) et des sens différents; dans ce cas, ils sont dits homographes (Homographe 2009). Un protocole basé sur une recherche de type « mots-clés » ne prend pas en compte ce type de situation et de ce fait peut retourner des résultats non-pertinents par rapport à une requête utilisateur. Par exemple, en cherchant un service « *Bois* » (nom propre), le protocole de découverte peut identifier comme pertinents des services incluant le mot « *bois* » (verbe « *boire* » conjugué à l'impératif).
- Troisièmement, ***les protocoles de découverte actuels ne sont pas sensibles au contexte***, c'est-à-dire qu'ils n'intègrent pas des informations contextuelles dans le processus de découverte de services. C'est pour cette raison qu'ils ne permettent pas de découvrir les services les plus pertinents et les plus appropriés (CUDDY 2005), (ZHU et al. 2005). Or, ce problème est crucial pour les environnements ubiquitaires. Si l'on considère un utilisateur

recherchant un restaurant, le protocole de découverte devrait prendre en compte la position et les préférences de l'utilisateur, afin d'identifier les services les plus pertinents. Il s'agit en l'occurrence, de permettre la découverte d'un restaurant situé à proximité de l'utilisateur, proposant un style de cuisine apprécié de l'utilisateur. Si des informations contextuelles sont ignorées lors du processus de découverte, la tâche d'identifier les services les plus pertinents revient à l'utilisateur. Par rapport à l'exemple précédent, si l'utilisateur reçoit une liste de restaurants situés plus ou moins loin et proposant divers styles de cuisine, il va devoir lui-même trier la liste en fonction de ses critères, afin de choisir le restaurant qui lui convient le mieux.

La réponse à ces problèmes se trouve dans la *description sémantique des services*. Afin d'illustrer les avantages d'une telle approche, prenons l'exemple suivant: un utilisateur souhaite trouver un restaurant à proximité de sa position actuelle. Il recherche un restaurant proposant des spécialités mexicaines. Il souhaiterait y réserver une table pour six personnes. Aussi, il aimerait savoir si le restaurant dispose d'un parking, et, si oui, combien de places sont disponibles sur ce parking. L'utilisateur a la possibilité de définir des pondérations pour chacune des contraintes définies ci-dessus.

En utilisant un protocole de découverte de services basé sur une recherche par mots-clés, l'utilisateur reçoit une liste de restaurants, avec pour chaque restaurant une description contenant des informations telles son adresse, son style de cuisine ou encore l'existence d'un parking propre au restaurant. C'est à l'utilisateur de parcourir l'ensemble des descriptions de la liste, afin d'identifier le restaurant qui correspond le mieux à sa requête. De plus, l'utilisation d'un tel protocole de découverte ne garantit pas la découverte de tous les services pertinents. Si, un service ne contient pas le mot « *restaurant* » dans sa description, il ne sera pas inclus dans la liste finale. Or, il se peut très bien qu'au lieu de « *restaurant* », la description du service contienne des mots tels « *brasserie* » ou « *taverne* » ou « *fast-food* ».

Si par contre, le protocole de découverte utilise des informations contextuelles (de l'utilisateur et du service), il peut, par exemple, déterminer la position de l'utilisateur, ainsi que le fait qu'il préfère les spécialités mexicaines (voir Figure 1). Ce type d'information peut être obtenu à partir du profil utilisateur. Les informations relatives au restaurant (style de cuisine, position, parking) seront obtenues à partir du profil de service. Lors de la création de la liste de services répondant à la requête de l'utilisateur, les services seront triés en fonction des critères utilisateur. Cela revient à dire que tous les services dans la liste seront des restaurants (même si le mot « *restaurant* » n'apparaît pas dans la description du service) situés à proximité, proposant des spécialités mexicaines et ayant six places disponibles. Pour les restaurants disposant d'un parking, le nombre de places disponibles sera aussi indiqué. Si l'utilisateur a défini comme principale contrainte le nombre de places disponibles dans le restaurant, les premiers éléments de la liste seront des restaurants remplissant ce critère, même s'ils ne sont pas situés à proximité de l'utilisateur. Il est dès lors évident que le premier service de cette liste a de fortes chances d'être LE service recherché par l'utilisateur. Ce dernier passera donc moins de temps à rechercher le service qui lui convient le mieux, puisque des informations décrivant son contexte et celui du service ont été prises en compte, analysées et comparées.

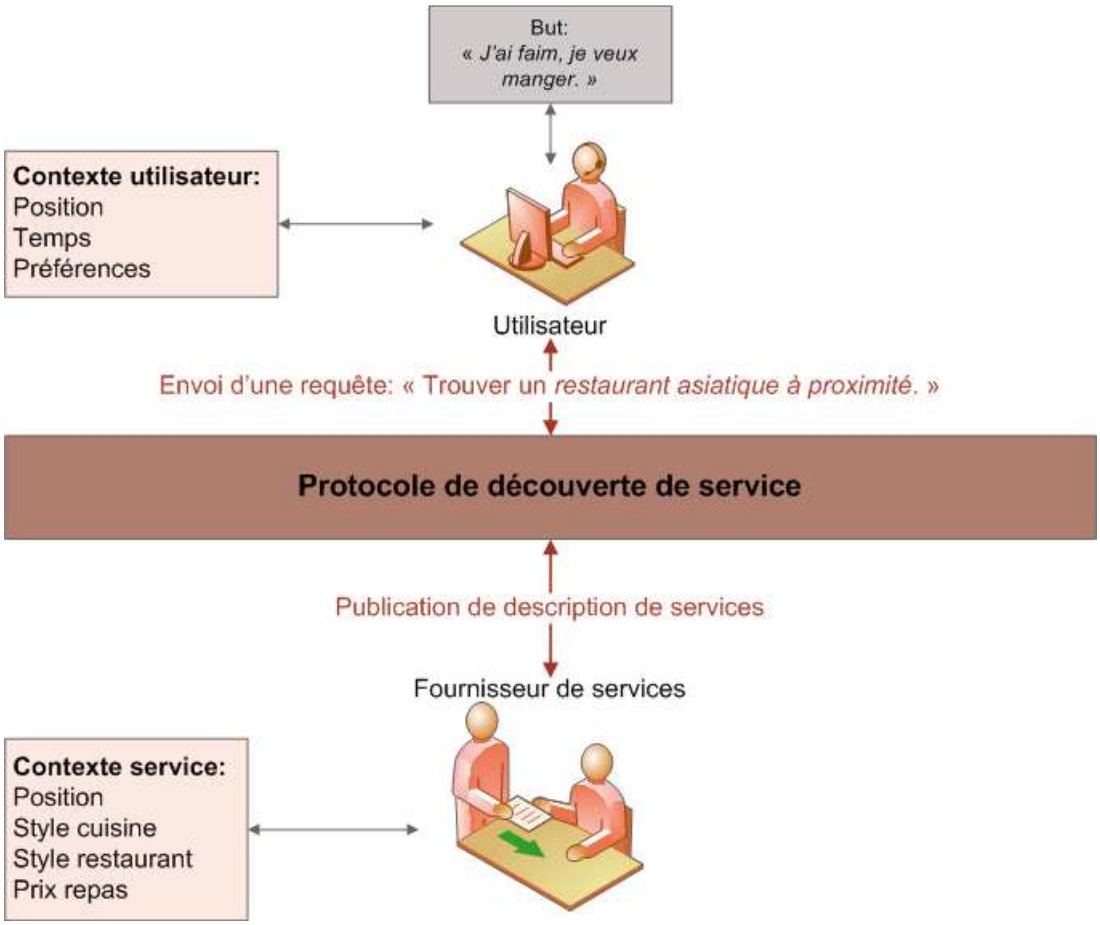

Figure 1.Découverte de services intégrant des informations contextuelles.

Ce scénario a permis d'illustrer les lacunes d'une découverte de services basée sur une recherche par mots-clés. Il nous a aussi permis d'identifier les avantages de la prise en compte d'informations contextuelles du service et de l'utilisateur. Dans un environnement mobile, où tout est en constant changement, la prise en compte du contexte d'un service permet la découverte de services plus pertinents et plus proches des besoins de l'utilisateur. La gestion des informations contextuelles définissant un service apparaît alors comme essentielle, dans le cas de l'informatique pervasive. Un protocole de découverte de services, intégrant le contexte de ces services, repose sur les éléments suivants:

- ***Une description de service bien définie*** permet aux utilisateurs de rechercher des services selon des caractéristiques qualitatives (ou non-fonctionnelles), telles la position, le prix ou le style de cuisine, comme illustré par l'exemple ci-dessus.
- ***Un vocabulaire*** (ou des *sémantiques*), décrivant les caractéristiques des services, permettent de réduire le nombre de résultats non-pertinents et donc d'identifier un maximum de résultats répondant au besoin de l'utilisateur.
- ***Une architecture unifiée du moteur de recherche*** – elle doit permettre une recherche simultanée de services fournis par différents fournisseurs de services, et ce sans avoir connaissance des moyens d'accès à ces fournisseurs de services.

Le Tableau 1 reprend les éléments clés d'un tel moteur de découverte, et indique les fonctionnalités supportées actuellement par les standards existants, ainsi que celles qui restent à implémenter.

Élément	*Fonctionnalités supportées*	*Fonctionnalités non-supportées*
Description de service	Description fonctionnelle bien définie à travers le langage WSDL. Description non-fonctionnelle limitée (à travers des propriétés du registre UDDI, telles "`provider`" ou "`category`").	Pas de support pour les informations non-fonctionnelles.
Sémantiques	Recherche à base de mots-clés, sur un ensemble limité de paramètres.	Peu de support pour les descriptions sémantiques.
Recherche unifiée	Chaque fournisseur de service dispose de son propre registre de services. Ces registres peuvent être répliqués.	Il faut effectuer des recherches séparées sur chaque registre. Il n'existe pas de registre universel de ces registres.

Tableau 1. Eléments de la découverte de services et fonctionnalités associées

Les objectifs de cette thèse concernent les fonctionnalités non-adressées par les protocoles de découverte actuels. Ces objectifs sont présentés dans les paragraphes suivants.

1.2 OBJECTIFS

Actuellement, la grande majorité des protocoles de découverte existants ne concerne que les interfaces fonctionnelles des services (CUDDY 2005). Les standards actuels des services Web ne fournissent qu'un support limité pour la prise en compte de requêtes non-fonctionnelles, telles qu'illustré par l'exemple de la recherche de restaurant, cité précédemment. Nous souhaitons donc mettre en place un *prototype pour la découverte de services Web, intégrant des informations contextuelles*[2]. Afin d'atteindre cet objectif, nous devons développer deux principaux composants:

- ***Un modèle de description de service***, qui utilise des outils sémantiques pour décrire les aspects non-fonctionnels d'un service;
- ***Un moteur de découverte***, qui permet une recherche parmi l'ensemble des descriptions de services et qui emploie un algorithme utilisant le modèle de description de service proposé. Ce moteur implique le développement de deux interfaces: la première permet aux fournisseurs de services de créer des descriptions de services conformes à notre modèle, et la deuxième est destinée aux utilisateurs finaux, utilisée pour la création de requêtes, toujours basées sur notre modèle de description de services.

L'objectif principal de notre travail de recherche est divisé en deux sous-objectifs, présentés dans ce qui suit. Le but est de tirer parti des outils existants, en les étendant et les combinant de manière originale et innovante. Les objectifs clés du prototype ainsi spécifié sont la simplicité de conception, d'implémentation et de déploiement.

[2] Le présent travail de recherche a été mené dans le cadre d'un contrat CIFRE avec l'entreprise belfortaine PIMENTIC. Malheureusement, cette entreprise n'existe plus depuis le mois d'avril 2009. Nous n'avons donc pas pu bénéficier d'une aide pour l'industrialisation des résultats scientifiques présentés ici.

1.2.1 OBJECTIFS CONCERNANT LA DESCRIPTION DE SERVICES

Avant de pouvoir découvrir des services, ceux-ci doivent être décrits d'une manière qui facilite leur recherche et découverte. La plupart des algorithmes de recherche de service (tels la recherche Web ou UDDI) est basée sur une recherche par mots-clés. Une telle recherche n'est pas suffisante pour la découverte de services, puisque deux mêmes mots peuvent avoir des sens différents (*homographes*) (Homographe 2009) ou encore des mots différents peuvent avoir le même sens (*synonymes*) (Synonymes 2009). A cela, nous pouvons ajouter les problèmes générés par les fautes d'orthographe lors de la saisie ou encore le sens des mots qui évolue. Lors du choix du modèle de description de service, il est donc important d'outrepasser les mots-clés et de tirer parti des descriptions sémantiques. Un autre facteur à prendre en compte est la flexibilité du modèle de description choisi. Ce modèle devra pouvoir être étendu, par exemple en y intégrant des schémas décrivant de nouveaux types de services.

1.2.2 OBJECTIFS CONCERNANT LE MOTEUR DE DECOUVERTE

Les protocoles de découverte de services peuvent être classés dans deux catégories (ZHU et al. 2005):

- ***Protocoles sans registre*** – dans ce type de protocole de découverte, les messages (notamment les publications et requêtes de services) s'échangent directement entre les fournisseurs de services et les utilisateurs.
- ***Protocoles à base de registre*** – ce type de protocole implique pour les fournisseurs de service d'enregistrer leurs services dans un registre ou annuaire. C'est le type d'architecture proposé dans cette thèse. L'utilisateur crée une requête de service, qui est ensuite adressée au registre. Un protocole de découverte est dès lors nécessaire pour parcourir ces registres et identifier les services répondant à la requête de l'utilisateur.

En général, les protocoles de découverte traitent la requête utilisateur puis renvoient une liste de services jugés pertinents par rapport à cette requête. Ensuite, c'est à l'utilisateur de parcourir cette liste et de choisir le service qui répond le mieux à sa requête. Si le service ainsi choisi n'est pas satisfaisant, l'utilisateur parcourt à nouveau la liste des services et en choisit un autre. Ce processus de sélection de service est long et fastidieux. De plus un utilisateur peut être novice et ne pas arriver à différentier les services retournés par le protocole de découverte (ZHU et al. 2005).

Le moteur de découverte de services développé dans cette thèse devra proposer à l'utilisateur une liste de services, ordonnés selon leur degré de pertinence. Pour ce faire, le moteur de découverte devra calculer ce degré de pertinence pour chaque service répondant à la requête initiale de l'utilisateur. Le but est de faciliter au maximum la recherche de l'utilisateur et de toujours lui retourner en premier les résultats les plus pertinents. Les interfaces développées devront être claires et permettre la création de requêtes différentes (simples ou complexes).

1.3 CONTRIBUTIONS

Dans cette partie, le but est d'exposer les principales contributions apportées par cette thèse. Ces contributions concernent deux principaux domaines de recherche, à savoir la description de services et la découverte de services.

1.3.1 CONTRIBUTION CONCERNANT LA DESCRIPTION DE SERVICES

Traditionnellement, les descriptions de services Web sont des documents WSDL, définissant l'interface et les fonctionnalités du service. Ce type de description n'est pas suffisamment expressif en ce qui concerne les capacités non-fonctionnelles d'un service. Afin de rendre plus efficace la recherche et la découverte de services, il est nécessaire d'intégrer la sémantique dans ces processus. La sémantique doit être utilisée pour exprimer les informations non-fonctionnelles caractérisant un service. Toutefois, dans les traitements classiques, l'emploi de la sémantique est souvent synonyme de problèmes d'interopérabilité lors de l'échange de données (WELTY 2000). Dans des domaines tels que la modélisation et la représentation des connaissances, de nombreuses recherches ont été menées afin d'identifier des outils et méthodes permettant d'exprimer cette sémantique de manière explicite et non-ambiguë (GUARINO 1997).

Parmi les différentes approches ainsi identifiées, l'utilisation des ontologies pour la représentation des connaissances a été identifiée comme la plus prometteuse (STRANG 2004). Les ontologies peuvent répondre au besoin d'exprimer les attributs non-fonctionnels ou contextuels d'un service. En effet, une ontologie permet de définir une structure de données complexe. Une ontologie modélise un ensemble de concepts, existants dans un domaine de connaissance donné, ainsi que les différentes relations reliant ces concepts.

Des travaux de recherche ont abouti à la mise en place de plusieurs ontologies décrivant un service Web. Les différentes initiatives dans ce domaine sont listées dans la sous-section 2.2.3.2. Dans cette thèse, nous avons choisi de baser notre approche sur le modèle OWL-S. La principale raison motivant ce choix découle du fait que le schéma de description de services défini par le langage OWL-S (MARTIN 2004) encapsule la description WSDL (CHINNICI 2007) du service et facilite la définition d'attributs service non-fonctionnels.

Dans le cadre de notre étude, nous proposons un schéma structuré pour la description d'attributs service non-fonctionnels, sur la base de la trame OWL-S. L'idée sous-jacente est d'étendre ainsi l'expressivité d'une description de service, afin d'obtenir des réponses plus pertinentes, par rapport à une requête initiale. Une hiérarchie de catégories de services sera définie, permettant d'étendre la définition donnée par le modèle OWL-S. De plus, nous étendons le modèle OWL-S en définissant un nouveau concept, regroupant les informations contextuelles relatives aux différentes catégories de services.

1.3.2 CONTRIBUTION CONCERNANT LA DECOUVERTE DE SERVICES

Comme précisé plus haut, la deuxième contribution de cette thèse concerne la définition et l'implémentation d'un prototype d'application, qui utilise la trame de description de service définie. Une fois la trame de description de services définie, elle sera intégrée dans un prototype global, permettant la découverte de services sur la base de notre modèle.

Une interface utilisateur intuitive et simple sera développée. Cette interface permettra à l'utilisateur de saisir sa requête service, puis traduira cette requête en description de service ou en requête service. A travers cette interface, l'utilisateur peut décrire quel serait le service idéal répondant à sa demande. Cette description de service servira de référence lors de la recherche de services correspondant à la requête utilisateur. L'interface utilisateur est présentée dans le Chapitre 5 (sous-section 5.2.2) et son développement est expliqué dans le Chapitre 6 (sous-section 6.3.1).

Une deuxième interface sera développée. Elle s'adresse aux fournisseurs de service souhaitant créer des descriptions de services conformes à notre modèle ontologique. Tout en gardant un lien avec les descriptions WSDL (supposées existantes pour chaque service), cette interface permettra de décrire un service en utilisant le vocabulaire défini dans notre ontologie. Nous la décrivons dans la sous-section 5.2.1, du Chapitre 5 et son implémentation est détaillée dans la sous-section 6.3.2, du Chapitre 6.

Le moteur de découverte a pour but de trouver, dans le registre, les services correspondant à la requête utilisateur. La sous-section 4.4 présente en détail l'algorithme qui est à la base de ce moteur. Le développement du moteur est présenté dans le Chapitre 6 (sous-section 6.2).

Chacun des composants utilise une combinaison innovante d'outils, choisis de manière à être les plus adaptés par rapport la description des services définie. Le moteur de découverte exploite efficacement les propriétés de la trame de description des services. L'interface utilisateur est structurée afin de permettre des requêtes détaillées (portant sur un grand nombre d'attributs service), mais aussi des requêtes plus simples. En se rapportant à notre modèle ontologique (voir le sous-section 4.3), le moteur de découverte peut prendre en compte plusieurs types de services, qui auraient été ignorés dans le cas d'une recherche à base de mots-clés. Grâce au modèle de description de services, le moteur de découverte peut déterminer qu'un utilisateur recherchant un « *restaurant* » peut être intéressé par un établissement de type « *relais* » ou encore « *auberge* ». Une recherche basée uniquement sur des mots-clés n'aurait pris en compte que les services incluant le mot « *restaurant* » dans leur description.

1.4 PLAN DU RAPPORT

Cette thèse comprend sept chapitres :

- Chapitre 1 – Introduction et contributions générales;
- Chapitre 2 – Etat de l'art des technologies impliquées ;
- Chapitre 3 – Approches pour la découverte de services sensible au contexte ;
- Chapitre 4 – Approche contextuelle pour la découverte de services Web sémantiques ;
- Chapitre 5 – Présentation du prototype ;
- Chapitre 6 – Développement et implémentation du prototype ;
- Chapitre 7 – Evaluation du prototype
- Chapitre 8 – Conclusion générale et perspectives.

Ce premier chapitre a été dédié à présentation de la problématique et des principales contributions de ce travail de recherche.

Le deuxième chapitre est un état de l'art des technologies et concepts auxquels il est fait référence dans le restant de cette thèse. Il traite des technologies impliquées dans la définition et l'implémentation des services Web, mais aussi des technologies du Web sémantique. Finalement, le chapitre présente le concept des services Web sémantiques et les technologies qui s'y relatent.

Le troisième chapitre est une analyse des travaux précédemment réalisés dans le domaine de la découverte de services sensible au contexte. Après une introduction aux principes et concepts liés à ce type de services, nous allons investiguer les différentes approches décrites dans la littérature pour la découverte sensible au contexte de ces services. Ces travaux sont comparés ensuite à l'approche

proposée dans cette thèse. Les points forts de cette approche seront mis en évidence par rapport aux solutions déjà existantes.

Le quatrième chapitre discute le rôle d'une description de services et formalise la trame proposée pour la description de paramètres service non-fonctionnels, ou autrement dit « *informations contextuelles* ». Ce modèle repose sur la trame OWL-S, mais augmente cette dernière en introduisant de nouveaux concepts. Ce chapitre présente aussi notre contribution concernant le moteur découverte, notamment l'algorithme sous-jacent.

Le cinquième chapitre présente le prototype que nous avons développé, afin de mettre en œuvre notre modèle de description de services et notre « *Moteur Découverte* ». Y sont présentés l'architecture du prototype, les différentes interfaces, ainsi que l'architecture proposée pour le moteur découverte.

Le sixième chapitre présente l'implémentation de notre approche. Notre prototype repose sur la plate-forme mobile Android™ ; nous commençons donc par présenter les caractéristiques de cette dernière. Nous nous intéressons ensuite au développement du moteur découverte, puis au développement des deux interfaces, que nous appelons « *Interface utilisateur final* » et « *Interface fournisseur de services* ».

Le septième chapitre présente une méthode d'évaluation de notre prototype, ainsi que les résultats associés. Il s'agit d'évaluer le prototype en fonction du temps nécessaire à l'exécution d'une requête de l'utilisateur. Nous y discutons aussi l'évaluation de notre prototype par rapport à la satisfaction des utilisateurs.

Le dernier chapitre résume les recherches présentées et traite des travaux futurs nécessaires pour améliorer notre prototype. Nous discutons aussi les principales perspectives (applicatives et académiques) concernant nos travaux de recherche.

Finalement, et à titre d'illustration, les différentes annexes présentent quelques exemples de descriptions de services, ainsi que quelques fragments des ontologies développées dans cette thèse. Nous y listons aussi des fragments de code Java, extraits des fichiers développés pour notre prototype.

CHAPITRE 2.

ETAT DE L'ART DES TECHNOLOGIES IMPLIQUEES

Ce chapitre présente les technologies et les standards qui ont été choisis comme base de départ pour ce travail de recherche. Il s'agit notamment des standards des services Web et des outils du Web sémantique. Nous souhaitons ainsi définir l'arrière-plan (technique et scientifique) nécessaire pour la compréhension de résultats présentés.

2.1 LES SERVICES WEB

2.1.1 DEFINITION

Les services représentent un concept très « *en vogue* » dans l'informatique d'aujourd'hui, mais peu de personnes peuvent en donner une définition claire et exacte. Non seulement les définitions varient en fonction du domaine d'application, mais même pour un même domaine d'application, plusieurs définitions similaires existent. Il est communément sous-entendu qu'un service représente « *un travail effectué pour une autre partie, avec ou sans contrepartie* » (Service 2009). Par contre dans des domaines bien précis, tels l'économie, la cuisine ou encore le sport, un service ne correspond pas à cette définition (Service 2009).

Dans le domaine du Web, le W3C (*World Wide Web Consortium*) définit un service comme étant une application logicielle, identifiée par une URI (*Unified Ressource Identifier*), dont les interfaces et connexions sont définies, décrites et découvertes par des artéfacts XML (Service 2009a). Le W3C spécifie que les interactions entre services sont directes et reposent sur des messages de type XML, envoyés via des protocoles Internet.

Un des points communs parmi ces définitions est le fait qu'un service représente une fonctionnalité fournie et exploitée, souvent, mais pas toujours, à distance. A partir de là, un service Web est défini en tant que fonctionnalité pouvant être engagée (invoquée, recherchée, exécutée, etc.) à travers le Web. Nous allons donner plus d'explications de cette définition dans les paragraphes suivants.

2.1.2 COMPOSANTS DE BASE D'UN SERVICE WEB

Le modèle architectural des services Web est illustré par la Figure 2.

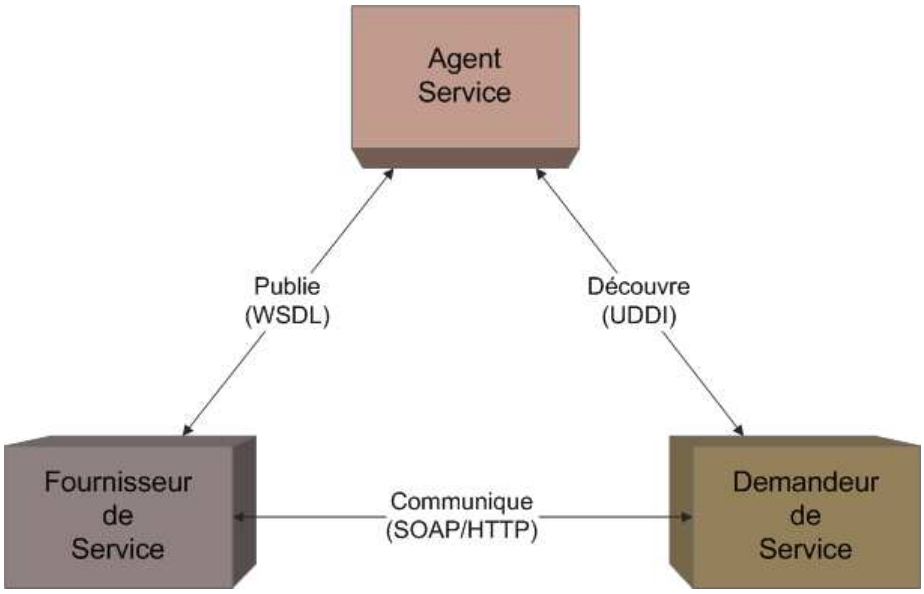

Figure 2.Modèle architectural des services Web.

Trois entités distinctes peuvent être identifiées:

- ***Les fournisseurs de services***, qui créent et publient des services Web pour des utilisateurs potentiels.

- ***Les demandeurs de services***, qui recherchent à travers les registres des « courtiers » de services, pour trouver des fournisseurs de services répondant à leur demande.
- ***Les « agents » de services*** (« *service broker* » en anglais), qui gèrent un registre de publications de services, et dont la mission est de mettre en relation les fournisseurs de services avec les demandeurs de services.

Le modèle architectural des services Web est basé sur les principes suivants (Figure 2):

- ***Connexion*** – afin que les fournisseurs et demandeurs de services puissent se connecter et échanger des informations entre eux, un langage commun est nécessaire. Il s'agit du langage *XML* (voir § 2.1.2.1).
- ***Communication*** – la communication entre les différentes entités décrites plus haut, nécessite elle aussi un protocole commun. Il s'agit principalement des protocoles *XML-RPC* (voir § 2.1.2.2.1) et *SOAP* (voir § 2.1.2.2.2).
- ***Description*** – la description des services (noms des fonctions, paramètres, résultats, etc.) doit utiliser un langage commun, afin de pouvoir être interprétée et « comprise » par l'ensemble des entités du système. Le langage permettant de décrire les services est le langage *WSDL* (voir § 2.1.2.3).
- ***Découverte*** – les demandeurs de services doivent pouvoir rechercher les services, à travers une structure accessible à tous. Ceci est réalisé par le registre *UDDI* (voir § 2.1.2.5), qui spécifie un registre pour les services.

Mis à part ces composants de base, il est nécessaire d'établir une entente préalable, concernant les sémantiques à utiliser. Différentes approches existent pour la représentation des sémantiques des services. Ces aspects seront traités plus loin dans ce chapitre (voir § 2.2). Le but de cette partie est d'introduire les éléments clés des services Web, et c'est ce que nous détaillons dans les paragraphes suivants.

2.1.2.1 LE LANGAGE XML

Le langage XML représente le langage fondamental pour le déploiement de services Web. Le XML régularise la syntaxe HTML, en la rendant plus facile à analyser et traiter.

Le langage XML offre les avantages suivants (BRAY 2008):

- Il peut être étendu à de nouvelles applications;
- Il supporte, à la fois, les données non-structurées (par exemple les documents) et les données structurées (par exemple bases de données);
- Il permet des requêtes en fonction de la structure d'un document - il est possible d'écrire une requête pour un document XML en précisant sa structure ou alors la valeur de certains de ses champs.
- Les données étiquetées à base XML peuvent être validées mécaniquement.

Le langage XML fournit un formatage de données (structurées ou non), mais ne précise pas les sémantiques de ce formatage. Afin de partager des informations et des connaissances entre différentes applications, il faut un ensemble partagé de termes, décrivant le domaine de l'application. La sous-section 2.2.3 présente des langages formels, pouvant être utilisés en ce sens. Ces langages peuvent être exprimés en XML, mais conceptuellement parlant, ils sont situés à un niveau supérieur au XML.

2.1.2.2 PROTOCOLES DE COMMUNICATION

Il existe deux méthodes pour dialoguer avec un service Web, le protocole XML-RPC et le protocole SOAP.

2.1.2.2.1 LE PROTOCOLE XML-RPC

XML-RPC est le plus simple des formats d'échange. Le principe de base est le suivant (WINER 1999):

- Sur le poste client, une bibliothèque encode les paramètres de la requête en XML ;
- Sur le poste serveur, une (autre) bibliothèque les décode et les transmet à l'application.

La procédure inverse a lieu lors de l'envoi de la réponse à la requête vers le poste client. En définitive, le programmeur n'a jamais besoin de coder lui-même le format de sortie en XML, puisque, dans le cadre du langage de programmation qu'il utilise, des fonctions se chargent des opérations à sa place. Il n'en voit que le résultat final.

Le système a malheureusement des limites. En théorie, par exemple, les seuls transferts autorisés se font sous le format ASCII, même si des extensions — non officielles — autorisent les transferts en Unicode. De plus, ce format n'est pas normalisé par un organisme indépendant et neutre (comme le W3C).

2.1.2.2.2 LE PROTOCOLE SOAP

Il s'agit du protocole actuellement le plus en vogue; il est promu par le W3C lui-même, ainsi que par Microsoft. La première recommandation date de juillet 2002 (GUDGIN 2007).

Le principe est le même que pour le XML-RPC (GUDGIN 2007) : le programmeur ne voit jamais le fichier XML que son poste émet ou reçoit, car tout est géré par une bibliothèque de fonctions et procédures, dont il ne perçoit que le résultat final, avec le format habituel de son langage de programmation favori.

Un message SOAP peut être comparé à une lettre postée. Ceci est illustré par la Figure 3.

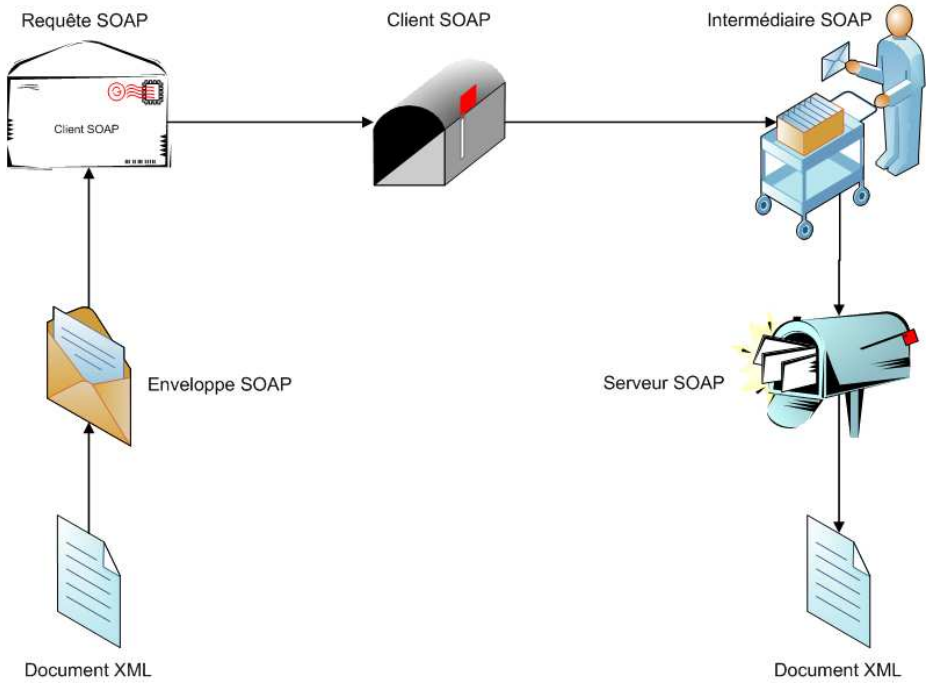

Figure 3.Principe de fonctionnement du protocole SOAP.

En effet, lorsque nous écrivons une lettre, nous la mettons dans une enveloppe sur laquelle nous indiquons le nom et l'adresse du destinataire, sans oublier d'y joindre un timbre. Lorsque ce processus est comparé au processus d'envoi d'un message SOAP, nous remarquons plus d'un aspect commun. En effet, dans le cas de SOAP, la lettre devient un document XML qui est inclus dans une enveloppe SOAP. Cette enveloppe SOAP contient donc le document XML, mais aussi une en-tête avec des informations de routage et/ou de sécurité. Une fois cette enveloppe créée, elle est convertie en une requête SOAP et envoyée à un client SOAP, afin d'être acheminée vers sa destination. Cela revient à poster une lettre dans une boîte aux lettres. Le client SOAP reçoit le message et l'achemine vers sa destination. Lors de cette étape, le message peut « traverser » plusieurs serveurs SOAP intermédiaires. Le routage du message SOAP dépend des informations de routage renseignées dans l'en-tête du message. Lorsque le message SOAP arrive à destination, il est reçu par un serveur SOAP. Le serveur SOAP « ouvre » l'enveloppe et transfère le message qu'elle contient au service Web concerné.

Comme nous l'avons indiqué ci-dessus, un message SOAP est un document XML. Ce document XML contient les éléments suivants (schématisés par la Figure 4) (GUDGIN 2007) :

- Une ***Enveloppe*** (`Envelope`) permet d'identifier le document XML en tant que message SOAP;
- Une ***En-tête*** (`Header`) contient des informations concernant la requête définie dans le corps du message SOAP. Elle peut contenir d'éventuelles informations de sécurité ou routage, ou encore des informations contextuelles ou en lien avec le profil utilisateur;
- Le ***Corps*** (`Body`) du message contient le document à envoyer, c'est-à-dire la requête et une trame XML pour la réponse;
- L'élément ***Faute*** (`Fault`) contient des informations sur les erreurs pouvant être rencontrées lors du traitement du message.

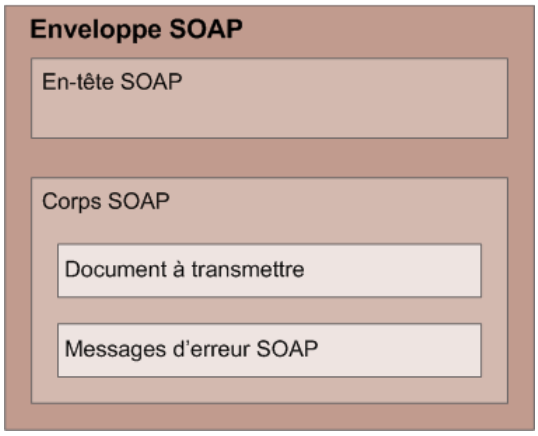

Figure 4.Structure d'un message SOAP.

La spécification SOAP est en perpétuelle révision. Elle ne décrit pas encore la communication bidirectionnelle ou multi-parties, aspects qui pourraient être utiles à la composition de services Web en provenance de différents fournisseurs. De plus, SOAP ne permet pas le transfert des sémantiques de la transaction. Ce protocole n'est donc efficace que pour une interopérabilité simple, impliquant un seul client et un seul serveur.

2.1.2.3 WSDL

Le modèle architectural des services Web (décrit dans la sous-section 2.1.2) suppose que les services peuvent être trouvés et utilisés. Ceci suppose en retour que chaque service est décrit avec « précision ». Ceci est réalisé grâce au langage WSDL qui est un langage XML permettant de décrire des interfaces de programmation pour un service Web. Une description WSDL inclut des définitions de types de données, de messages d'entrée et sortie, d'opérations fournies par le service, d'adresses réseau, etc.

En d'autres termes, le langage de description de services Web (WSDL) décrit une interface abstraite pour les services Web, tout en permettant d'y relier un protocole de transport spécifique, tel HTTP. Ceci permet d'avoir plusieurs implémentations système pour une même interface abstraite. La description d'un service Web peut donc être comprise et interprétée par des systèmes différents.

A l'image d'un message SOAP, qui est encapsulé dans un élément `<enveloppe>`, les documents WSDL sont encapsulés dans un élément appelé `<definitions>`. Un document WSDL peut ainsi être vu comme un ensemble de définitions (CHRISTENSEN 2001). Les documents WSDL contiennent cinq parties distinctes, tel qu'illustré par la Figure 5.

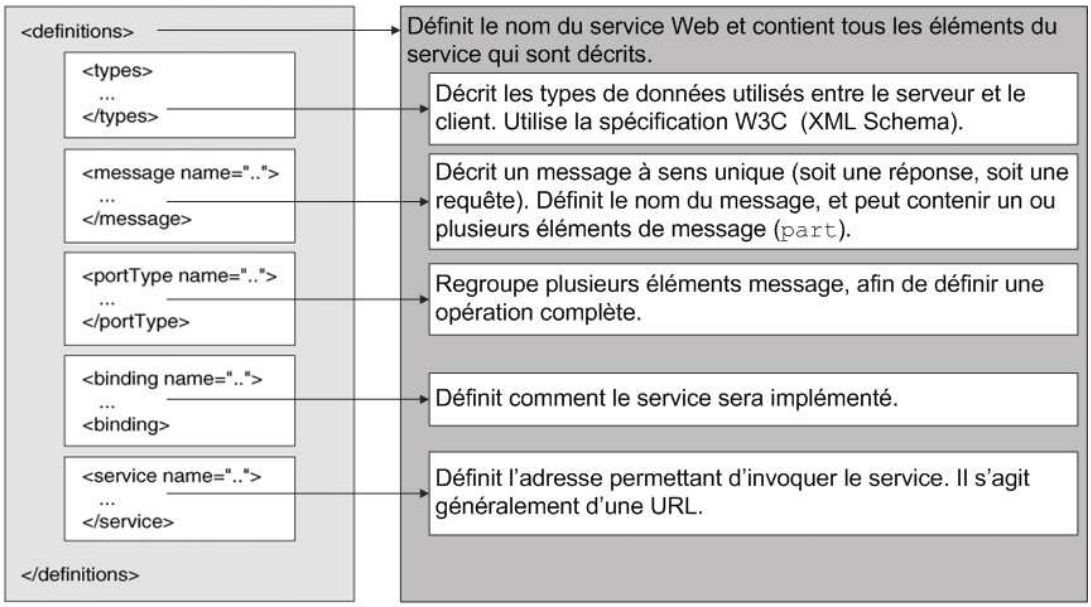

Figure 5.Structure d'un document WSDL.

Un des éléments importants de cette description est l'élément `portType`. Cet élément peut être apparenté à une interface Java, puisqu'il permet de définir un ensemble d'opérations. La Figure 6 met en évidence les similitudes entre l'élément `portType` et une classe Java. L'élément `portType` contient un ensemble d'éléments « `operation` », chacun de ces éléments définissant une fonction spécifique de l'élément `portType`. Ceci peut être apparenté à une méthode dans une classe.

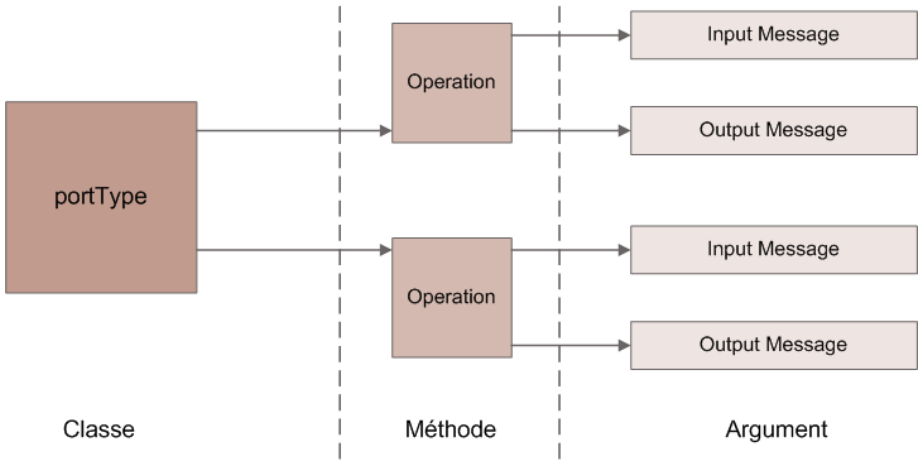

Figure 6.Liens entre les éléments WSDL et une classe Java.

Le standard WSDL définit quatre types d'opérations[3]:

- ***Opération à sens unique*** (one way) - recevoir un message;
- ***Opération de notification*** (« notification) - envoyer un message;
- ***Opération requête/réponse*** (Request-response) - recevoir une requête, puis émettre la réponse correspondante;
- ***Opération sollicitant une réponse*** (Solicit-response) - émettre une requête et recevoir la réponse correspondante.

Chaque élément <operation> a des éléments <message> associés. Un élément <message> est une définition abstraite de données et types de données. Un élément <message> décrit un message à sens unique: soit une requête (input), soit une réponse (output). Un élément <message> définit le nom du message et peut contenir zéro ou plusieurs éléments de partie de message (se référant à des paramètres du message ou à des valeurs de retour du message). Les éléments <message> peuvent être apparentés à des arguments d'une méthode Java (voir Figure 6).

2.1.2.4 LES SERVICES REGISTRE

Le but d'un service de registre est de permettre à des composants ou des participants de se localiser les uns les autres. Les composants ou participants peuvent être des applications, des agents, des fournisseurs de services Web ou des demandeurs de services Web. Les registres rassemblent et organisent des informations d'identification (adresses sur le réseau, descriptions, etc.), puis les mettent à disposition des clients.

Les registres reposent sur deux principaux standards: ebXML (ebXML 2009) et UDDI (BELLWOOD 2004). Malheureusement, aucun de ces standards ne prend en compte les descriptions sémantiques des services. Ils ne supportent pas non plus les recherches sémantiques de fonctionnalités. Les seules méthodes de recherche implémentées sont basées sur des mots-clés, tels le nom du service ou du fournisseur de service. Seul le standard ebXML permet d'effectuer des recherches de type requête SQL, mais ces recherches ne considèrent que les mots-clés.

[3] Dans sa version 2.0, le WSDL (CHINNICI 2007) offre un ensemble plus riche de primitives, permettant entre autres de recevoir ou d'envoyer des réponses multiples à une seule requête. Mais ces détails ne sont pas importants par rapport au but de ce chapitre.

Dans ce qui suit, nous allons nous intéresser de plus près aux fonctionnalités du standard UDDI.

2.1.2.5 UDDI

La spécification UDDI (BELLWOOD 2004) décrit un mécanisme permettant de référencer des services Web. Le standard UDDI a été publié par Ariba, Microsoft, IBM et une trentaine d'autres entreprises, en septembre 2000. Le schéma XML de l'UDDI définit quatre principaux types de données pour les entreprises et les informations services. Pour chacun de ces types, il existe une structure XML précisant les champs obligatoires et les champs optionnels. Les quatre entités ainsi définies sont: `businessEntity`, `businessService`, `bindingTemplate` et `tModel`. La Figure 7 illustre ces quatre entités de base, ainsi que leurs relations.

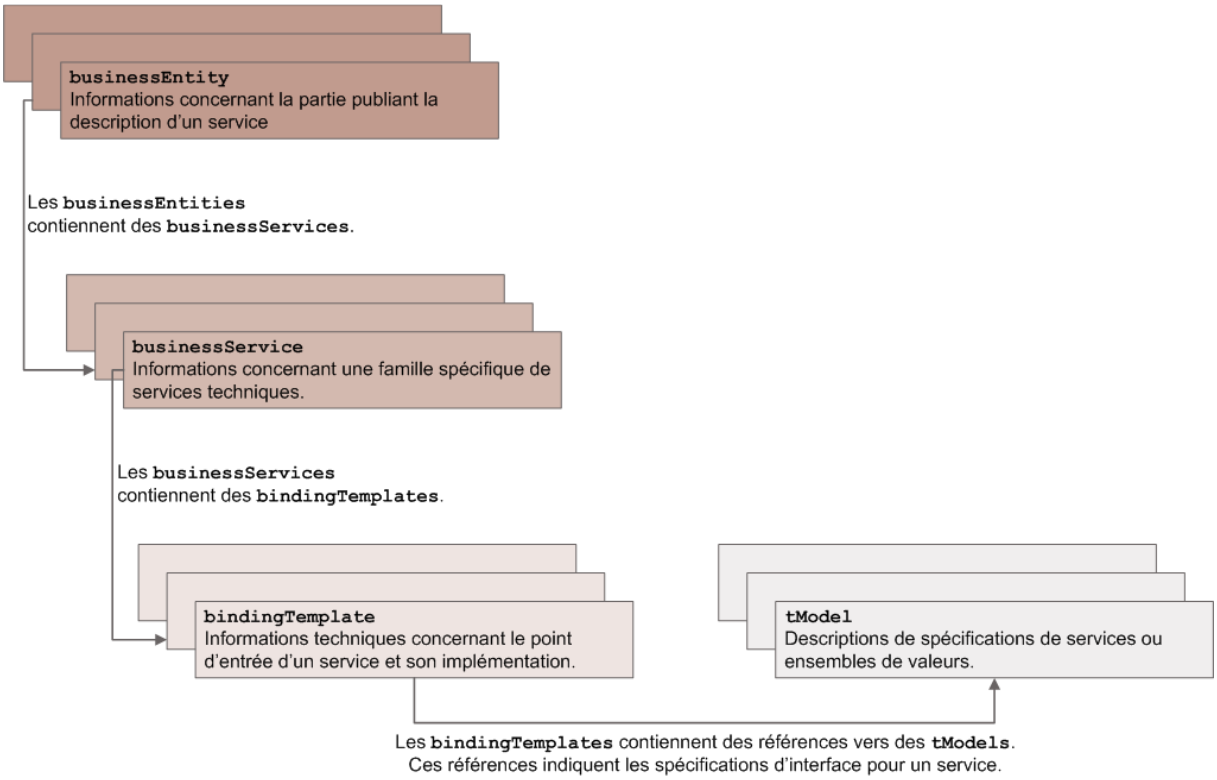

Figure 7.Entités de base du registre UDDI.

Ces quatre entités permettent de représenter trois types d'information:

- Les « *Pages Blanches* » contiennent des adresses, des contacts et d'autres informations générales, concernant des entreprises ou des individus.
  Par exemple, les « Pages Blanches » peuvent être utilisées afin de rechercher une entreprise dont on connaît le nom. C'est la structure l'entité `businessEntity` qui est utilisée afin de représenter ce niveau d'information. Un registre UDDI de type « Pages Blanches » contient les champs d'information suivants:
  - Nom de l'entreprise ;
  - Description textuelle de l'entreprise ;
  - Des informations de contact (noms, numéros de téléphone, sites Web, etc.) ;

- Des identifiants de l'entreprise (D-U-N-S[4] ou Thomas Register[5]).

- Les « *Pages Jaunes* » contiennent des classifications industrielles pour les entreprises, en fonction de taxonomies standard telles le NAICS (NAICS 2009) ou la classification de Nice (Classfication de Nice 2009). C'est la structure de l'entité `businessService` qui est utilisée afin de représenter ce niveau de classification. Une entité `businessEntity` peut référencer plusieurs structures `businessService`. Un registre UDDI de type « Pages Jaunes » contient les champs d'information suivants:
  - Un identifiant dans une classification internationale pour l'entreprise (par exemple un code NAICS ou un code de la classification de Nice);
  - Un identifiant dans une classification internationale pour chaque type de service et/ou produit offert par l'entreprise;
  - Une position géographique respectant un format standard[6].
- Les « *Pages Vertes* » contiennent des informations techniques concernant les services fournis par les entreprises. Il s'agit, entre autres, de références à des spécifications d'interfaces pour les services Web. Le modèle des « Pages Vertes » est plus compliqué, car il contient des informations plus variées (processus métier, descriptions de services, informations de liaison, etc.).

Afin de mieux comprendre le fonctionnement du modèle UDDI, nous allons prendre l'exemple d'une entreprise nommée « Mon Entreprise » qui offre un service d'actualités. Ceci est illustré par la Figure 8. L'élément `businessEntity` fournit une clé (« `ABCD` ») et un nom (« `Mon Entreprise` »). La structure de l'élément `businessService` représente le service actualités de « Mon Entreprise ». Elle référence l'élément `businessEntity` en utilisant sa clé, puis définit une clé pour le service lui- même (« `EFGH` »). La structure de l'élément `businessService` est référencée à travers deux éléments `bindingTemplate`. Le premier d'entre eux est une page Web disponible à l'adresse `"http://infos.monentreprise.com"`. Le deuxième d'entre eux est un service Web disponible à l'adresse: `"http://monentreprise.com/infos"`. Les deux éléments `bindingTemplate` référencent l'élément `serviceTemplate` en utilisant sa clé de service, tout en fournissant eux-mêmes des clés de liaison. L'élément `bindingTemplate` du service Web fournit plus d'informations relatives aux méthodes d'invocation pour ce service. Ceci est réalisé référençant une entité `tModelInstanceInfo`, qui fournit une clé de type `tModel` (« `QRST` »). Cette entité `tModelInstanceInfo` est contenue dans une structure `tModelInstanceDetails` qui permet le référencement de plusieurs clés de type `tModel`. La structure `tModel` dont la clé est « `QRST` » contient une URL indiquant l'emplacement de la description WSDL du service (`"http://monentreprise.com/infos?wsdl"`).

[4] « *Le Numéro D-U-N-S est une séquence d'identification unique à neuf chiffres qui garantit une identification sûre et unique de chaque entreprise et permet de découvrir des structures d'apparentement à l'échelle mondiale.* » (D&B 2009)

[5] Considéré comme le plus grand annuaire « pages vertes » au monde, ce registre couvre plus de 650000 fournisseurs de services, classés dans plus de 67000 catégories industrielles (Thomas Register 2009).

[6] La norme ISO 3166 « *définit un code pour la quasi totalité des pays du monde* » (ISO 3166 2009).

Figure 8.Fonctionnement du modèle UDDI.

La spécification UDDI offre un ensemble limité de moyens pour permettre aux utilisateurs finaux de rechercher des services. Afin d'illustrer les contraintes de la recherche dans un registre UDDI, la Figure 9 donne un exemple de requête pour le service d'information défini plus haut.

```
<find_business>
        <findQualifiers>
                <findQualifier>
                        Uddi :uddi.org :findQualifier :exactMatch
                </findQualifier>
        </findQualifiers>
        <name>
                Mon Entreprise
        </name>
</find_business>
```

Figure 9.Exemple de requête service UDDI.

La requête ainsi décrite illustre une structure `find_business` qui filtre la requête afin de déterminer une correspondance exacte sur le nom de l'entreprise. Une telle requête est acheminée vers la structure `businessEntity` du registre UDDI. Le protocole UDDI définit des requêtes équivalentes pour les trois autres structures (`businessService`, `bindingTemplate` and `tModel`).

Le modèle de découverte UDDI est basé sur le principe du parcours du registre: un utilisateur peut parcourir l'ensemble des services mis à disposition par une entreprise donnée ou alors l'ensemble des services existant dans une catégorie donnée. Ce type de recherche a deux principaux inconvénients:

- Elle peut retourner un nombre ingérable de résultats;
- Elle nécessite une grande implication/participation de la part de l'utilisateur final.

Un autre problème de la recherche dans un registre UDDI est constitué par le fait que les fournisseurs de services interprètent de manière erronée les champs dans les structures de données UDDI (ERL 2005). Les informations manquantes, couplées à des paramètres de recherche hautement structurés et nécessaires pour effectuer une recherche dans un registre UDDI, jouent au détriment de la complétude de la découverte de services.

2.1.3 CONCLUSION

Telle que nous l'avons présentée ci-dessus, la notion de « *service Web* » repose sur des standards bien connus. Le modèle ainsi décrit est déjà mis en œuvre par une multitude d'acteurs. Ce modèle repose sur des technologies et protocoles issus du monde de l'Internet, et définit une architecture distribuée. L'implémentation de ce modèle est relativement peu complexe et donc aisément accessible aux fournisseurs de services.

Le principal inconvénient de ce modèle est que les technologies inhérentes ne prennent pas en compte les aspects non-fonctionnels des services Web. Des efforts de recherche et de standardisation existent et sont menés à travers le monde, avec pour but d'étendre la description des composants technologiques des services Web en utilisant des vocabulaires précis de termes. La partie suivante présente ces approches plus en détail.

2.2 LES PRINCIPES DU WEB SEMANTIQUE

Le but de cette partie est de permettre la compréhension de la nature des données et contenus du Web sémantique, ainsi que de discuter les approches permettant de représenter des métadonnées. Le principal but du Web sémantique est de se construire, au dessus du Web actuel (ou Web syntaxique), afin de donner à chaque donnée un sens bien défini, pouvant être interprété par un ordinateur. Dans ce qui suit, nous allons présenter le rôle que jouent les descriptions de métadonnées et les ontologies dans l'atteinte de ce but.

2.2.1 VISION DU WEB SEMANTIQUE

C'est en 2001 que les auteurs Berners-Lee, Hendler et Lassila publient un article révolutionnaire, dans le journal « *Scientific American* », intitulé « *The Semantic Web: A New Form of Web Content That Is Meaningful to Computers Will Unleash a Revolution of New Possibilities* » (BERNERS-LEE et al. 2001). Dans cet article, les auteurs définissent la vision du Web sémantique, à travers des scénarios.

Un des scénarios présente Lucy qui a besoin de programmer une série de consultations médicales pour sa mère. Plusieurs contraintes doivent être prises en compte, dont des contraintes géographiques, des contraintes concernant les qualifications des médecins, ou encore des contraintes liées au calendrier de Lucy. Afin d'aider Lucy à trouver une solution, un agent peut mener des négociations entre les différentes parties: les médecins, l'agenda de Lucy et le registre de services médicaux, entre autres. L'idée clé derrière cet exemple est d'affirmer que, même si chacune de ces parties code son information d'une manière différente, c'est la couche sémantique qui rend possibles les interactions et les échanges de données entre ces parties. De plus, les données

échangées peuvent être comprises par un programme informatique (l'agent). Les auteurs appellent Web sémantique la technologie permettant d'atteindre cette vision[7].

Afin d'organiser le contenu du Web actuel, les chercheurs dans l'intelligence artificielle ont proposé une série de modèles conceptuels. L'idée centrale est de structurer l'information en catégories, de manière à faciliter son accès. Or, cette idée est similaire à la solution utilisée pour classifier les êtres vivants en biologie. En effet, les biologistes utilisent une taxonomie bien définie, la taxonomie de Linné (Taxinomie 2009). En s'inspirant de cette approche, les chercheurs en informatique essayent de trouver un modèle similaire permettant de structurer le contenu du Web actuel.

Toutefois, il a toujours été dit que le Web a connu un grand succès justement à cause du degré de liberté qu'il permet. Il est en effet possible de trouver, dans un même environnement, des pages Web sophistiquées, créées par des spécialistes, et des pages Web personnelles, créées par des personnes ayant peu de connaissances en informatique. Il n'existe pas de « censure » quant à la qualité de l'information présente dans une page Web. Dans un tel environnement, chacun est libre d'exprimer ce qu'il veut et ce concernant n'importe quel sujet[8]. Il est dès lors difficile d'imaginer avoir un unique modèle organisationnel pour l'ensemble du Web.

D'après ces quelques considérations, il est évident que les métadonnées forment un aspect clé dans l'application de la vision du Web sémantique. Toutefois, les descriptions à travers les métadonnées doivent respecter des règles précises. Le Web sémantique a besoin de modèles permettant d'organiser les connaissances d'une manière souple et non-contraignante. Les technologies permettant d'atteindre cette vision sont décrites dans ce qui suit.

2.2.2 LES METADONNEES

Nous venons de voir que, dans la vision du Web sémantique, il s'agit de cataloguer une quantité énorme de ressources, virtuelles pour la plupart d'entre elles, réparties à travers le monde, et codées avec des langages différents. Une solution à ce problème est représentée par l'usage de métadonnées.

Les métadonnées sont définies en tant que données ou informations concernant d'autres données[9]. Berners-Lee donne toutefois une définition des métadonnées qui est mieux adaptée au cadre du Web sémantique[10]: « *(dans la conception Web)… les métadonnées représentent des informations compréhensibles par des ordinateurs et qui concernent des ressources Web ou d'autres choses* » (BERNERS-LEE 1998). Cette définition nous amène vers le concept de données compréhensibles par un ordinateur. En effet, le but ultime des métadonnées identifié par Berners-Lee est la production d'informations qui s'auto-décrivent. Pour ce faire, les métadonnées doivent contenir une référence vers une spécification des sémantiques qui leurs sont associées (BERNERS-LEE 1998). La sémantique

[7] Toutefois, les auteurs précisent que les actions décrites dans les scénarios peuvent être réalisées en utilisant les technologies du Web actuel (syntaxique), mais avec des efforts considérables. La promesse du Web Sémantique est de libérer les utilisateurs des tâches encombrantes et pénibles.

[8] Il s'agit en fait du slogan AAA: « *Anyone can say Anything about Any topic* » (ALLEMANG 2008).

[9] La fédération IFLA donne la définition suivante: « *Metadata is data about data. The term refers to any data used to aid the identification, description and location of networked electronic resources. Many different metadata formats exist, some quite simple in their description, others quite complex and rich* » (METADATA 2005).

[10] (BERNERS-LEE 1998) *"(in Web design)… Metadata is machine understandable information about Web resources or other things"*.

des ressources du Web actuel doit donc être rendue compréhensible pour les ordinateurs. Pour ce faire, il faut déterminer une représentation formelle et standardisée qui puisse être utilisée par les machines lors de l'échange d'informations. Il s'agit donc de spécifier un vocabulaire (ou des *sémantiques*) à utiliser. Dans le cas du Web sémantique, cette spécification repose sur l'usage d'ontologies.

2.2.3 LES ONTOLOGIES - VOCABULAIRES POUR LES METADONNEES

2.2.3.1 DEFINITIONS

Le mot « *ontologie* » vient du grec « *ontos* » (être) et « *logos* » (mot). Au 19ème siècle, il est introduit en philosophie par des philosophes allemands afin de pouvoir différencier l'étude de l'être en tant que tel, de l'étude, dans les sciences naturelles, de différents êtres vivants.

Dans (SOWA 1997), Sowa définit l'ontologie en tant que discipline philosophique: « *Le sujet de l'Ontologie est l'étude des catégories de choses/objets existant ou pouvant exister dans un domaine donné. Le produit d'une telle étude, appelé ontologie, est un catalogue des types d'objets existant dans un domaine D du point de vue d'une personne utilisant un langage L pour parler de D*[11]*.* »

En informatique, les ontologies ont été adoptées dans le domaine de l'intelligence artificielle pour faciliter le partage et la réutilisation de connaissances (ALESSO 2006). Aujourd'hui, leur usage se répand dans divers domaines tels l'intégration d'informations intelligentes, les systèmes d'information coopératifs, le commerce électronique ou encore dans le développement de logiciels à base d'agents. Les ontologies sont des modèles conceptuels capturant et rendant explicite le vocabulaire utilisé dans des applications sémantiques. Les ontologies garantissent une communication sans ambiguïtés. Elles sont considérées en tant que *lingua franca*[12] du Web sémantique.

Dans (ALESSO 2006), les auteurs définissent une ontologie en tant qu'entente entre des agents échangeant des informations. Cette entente porte sur un modèle pour la structure et l'interprétation des informations échangées, ainsi que sur un vocabulaire délimitant les échanges[13].

La définition donnée par le W3C (MCGUINESS 2004) est la suivante: « *Le terme d'ontologie est un terme emprunté à la philosophie qui fait référence à la science de décrire les types d'entités du monde et les relations qu'elles ont entre elles*[14] ». Les « entités » composant une ontologie sont les suivantes:

- ***Des concepts*** dans un domaine d'intérêt donné;
- ***Des relations*** entre ces objets;
- ***Des propriétés*** (ou attributs) que possèdent les objets ;

---

[11] "*The subject of Ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called ontology, is a catalogue of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D.*"

[12] « *Le terme de lingua franca désigne une langue véhiculaire utilisée par une population donnée pour communiquer.* » (Lingua franca 2009).

[13] « *An ontology is an agreement between agents that exchange information. The agreement is a model for structuring and interpreting exchanged data and a vocabulary that constrains these exchanges.* »

[14] « *Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related.* »

- ***Des instances*** représentant des objets concrets (nommés et identifiables) dans le domaine d'intérêt visé.

Ces « entités » constituent un vocabulaire ontologique, pour un domaine d'intérêt donné. Une ontologie peut dès lors être vue en tant qu'un ensemble d'énoncés, exprimés en utilisant les termes de ce vocabulaire. Ces énoncés sont souvent appelés ***axiomes*** (GRIMM et al. 2007a). L'énoncé « *Madame X est une employée* » constitue un exemple d'axiome simple, impliquant un concept (« *employée* ») et une instance (« *Madame X* »).

Dans (MAEDCHE et STAAB 2001) les auteurs définissent une ontologie en tant qu'un ensemble $O = \{C, R, CH, rel, OA\}$, où:

- C représente l'ensemble des concepts ;
- R représente l'ensemble des relations ;
- $CH \subseteq C \times C$représente une hiérarchie de concepts ou encore une taxonomie (Taxinomie 2009). La notation $CH(C_1, C_2)$ définit le concept C_1 en tant que sous-concept de C_2 ;
- $rel: R \rightarrow C \times C$ est une fonction reliant les concepts d'une manière non-taxonomique ;
- OA représente l'ensemble des axiomes de l'ontologie, exprimés en utilisant un langage logique et décrivant des contraintes supplémentaires pour l'ontologie.

Indépendamment de la définition adoptée, il est important de noter que les ontologies sont utilisées afin de décrire une grande variété de modèles. Généralement, les ontologies sont différenciées des bases de connaissances, dont le rôle est de décrire la connaissance en termes de taxonomies conceptuelles et d'énoncés généraux (GRIMM et al. 2007a). Dans notre vision, une ontologie représente un ensemble de connaissances pouvant être utilisé par une application à base de connaissances, et ce parmi d'autres ensembles de connaissances (notamment d'autres ontologies ou des métadonnées). A chaque fois que l'application à base de connaissances doit consulter l'ontologie, elle charge une partie des spécifications de l'ontologie dans une base de connaissances. Une base de connaissances est dès lors constituée d'un ensemble d'ontologies (ou fragments d'ontologies), chacune concernant un domaine de connaissance différent.

2.2.3.2 LANGAGES DE DESCRIPTION D'ONTOLOGIES

Nous venons de définir les principaux composants d'une ontologie comme étant les concepts, les relations, les propriétés et les instances. Ces éléments permettent de constituer un vocabulaire ontologique pour un domaine donné. Une ontologie peut alors être vue comme un ensemble de propositions exprimées utilisant les termes de ce vocabulaire.

A première vue, modéliser une ontologie semble être similaire à la modélisation de logiciels orientés-objet ou encore à la conception de diagrammes de type entité-relation pour les bases de données. Il y a cependant deux différences majeures (GRIMM et al. 2007a) :

- Premièrement, ***les langages de description d'ontologies fournissent des sémantiques formelles plus riches*** que celles fournies par les formalismes orientés-objet ou liés aux bases de données. Une ontologie spécifie une « axiomatisation » de connaissances dans un certain domaine, plutôt qu'un modèle de données ou qu'un modèle objet.
- Deuxièmement, ***les ontologies sont généralement développées à des fins différentes*** par rapport aux modèles orientés-objet ou par rapport aux diagrammes entités-relations.

Alors que les derniers décrivent des composants d'un système d'information, ou alors un schéma pour le stockage de données, une ontologie capture la connaissance d'un certain domaine en tant que telle et permet de raisonner à propos de cette connaissance.

Afin de rendre les ontologies disponibles pour des systèmes d'information, plusieurs langages d'ontologie ont été conçus et proposés à des fins de standardisation. Un langage de description d'ontologie est conçu pour définir des ontologies. Ces langages reçoivent une attention croissante depuis l'émergence de la vision du Web sémantique.

L'architecture en couches du Web sémantique (ALESSO 2006) (voir Figure 10) permet d'avoir une vue d'ensemble de la relation entre les différents langages de description d'ontologies. La couche inférieure contient les mécanismes de référencement (URI) et d'encodage des caractères (Unicode). La deuxième couche introduit le langage XML en tant que standard pour l'échange de documents. La troisième couche comprend les langages RDF (*Ressource Description Framework*) et RDF Schema en tant que mécanismes pour décrire les ressources disponibles sur le Web. Ces deux langages sont ainsi considérés en tant que langages simples. Les langages complets apparaissent dans la quatrième couche, en permettant de capturer plus de sémantiques. Enfin, la dernière couche introduit les langages à base de règles, dont l'étude est en dehors des objectifs de cette thèse.

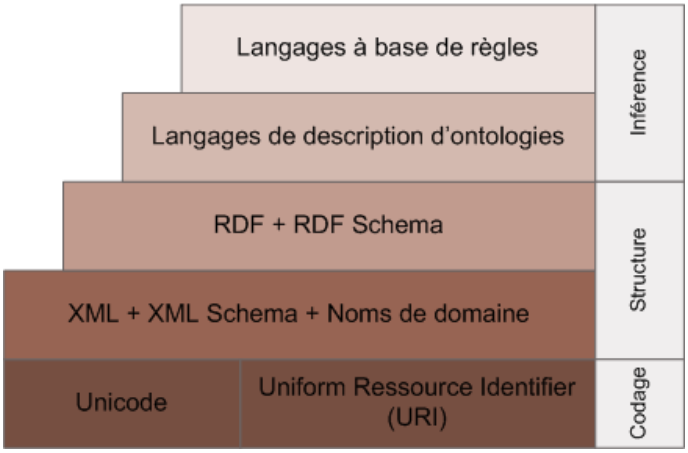

Figure 10. Architecture du Web sémantique (adapté depuis (ALESSO 2006)).

Dans ce qui suit, nous allons présenter un bref aperçu des langages RDF et RDF Schema (RDFS), puis nous allons décrire les concepts de base du langage OWL (*Ontology Web Language*), actuellement considéré comme le standard pour la description des ontologies Web.

2.2.3.2.1 Les langages RDF et RDFS

Une des premières initiatives de standardisation d'un langage permettant les annotations sémantiques de ressources Web, a été celle initiée par le W3C et qui a donnée naissance aux langages RDF et RDF Schema. Publiés le 10 Février 2004, ces deux langages concernent plus particulièrement les annotations de ressources Web identifiables, telles le titre ou l'auteur d'une page Web. Le langage RDFS est une extension du langage RDF, introduisant les notions de classes de ressources et de hiérarchie de classes. L'usage combiné de RDF et RDFS est souvent référencé en tant que RDF(S) et fournit un langage d'ontologie simple pour des modélisations conceptuelles.

L'approche RDF(S) utilisée pour la représentation d'annotations de ressources est basée sur les idées suivantes (BREITMANN 2007) :

- ***Les ressources sont identifiées grâce à des URI (Uniform Resource Identifier).*** Une ressource dispose ainsi d'un identifiant unique sur le Web. Ceci permet une organisation décentralisée de la connaissance des ressources. Une URI permet d'identifier les types suivants de ressources: des individus, des types d'objets, ou encore des propriétés de ces objets.
- ***Un énoncé RDF est un triplet*** (S, P, O)***.*** Un triplet (S, P, O) permet de relier un Sujet S à un Objet O, par l'intermédiaire d'un Prédicat P. Ces éléments *(Sujet, Prédicat, Objet)* représentent des ressources, identifiées par des URI. Le Sujet S représente la ressource à décrire; le Prédicat P représente une propriété particulière de cette ressource. L'Objet O d'un énoncé RDF peut être une URI (et dans ce cas il est appelé objet de l'énoncé) ou une valeur littérale (et dans ce cas il représente la valeur de la propriété). Le Tableau 2 illustre plusieurs énoncés RDF. L'énoncé E1 définit l'auteur du document dont l'URI est `"http://www.gsem.fr/documents#D42305"`. Les énoncés E2 et E3 définissent respectivement le titre et la date[15] de ce document (« *Liste Publications Roxin* » et « *11-11-2009* »).

Enoncé E1	Sujet	http://www.gsem.fr/documents#D42305	*Le document D42305…*
	Prédicat	http://purl.org/dc/elements/1.1/creator	*…a été créé par…*
	Objet (une URI)	http://www.gsem.fr/auteur#ROX639	*l'auteur ROX639.*
Enoncé E2	Sujet	http://www.gsem.fr/documents#D42305	*Le document D42305…*
	Prédicat	http://purl.org/dc/elements/1.1/title	*…a pour titre…*
	Objet (une valeur)	Liste Publications Roxin	*« Liste Publications Roxin ».*
Enoncé E3	Sujet	http://www.gsem.fr/documents#D42305	*Le document D42305…*
	Prédicat	http://purl.org/dc/elements/1.1/date	*…a pour date…*
	Objet (une valeur)	11-11-2009	*le 11 Novembre 2009.*

Tableau 2. Exemples d'énoncés RDF.

- ***Plusieurs triplets RDF forment un graphe RDF.*** Les nœuds d'un graphe RDF représentent des URI de ressources et les arcs des propriétés de ces ressources. Les graphes RDF peuvent contenir des nœuds vides, représentant des ressources anonymes (des ressources qui n'ont pas d'URI). La Figure 11 présente un exemple de graphe RDF. Ce graphe décrit les trois énoncés illustrés par le Tableau 2.

[15] Ici nous utilisons une PURL (*Persistent Uniform Ressource Locator*) qui est un identificateur permanent de resource. Il s'agit d'URIs qui ne décrivent pas directement l'emplacement de la resource, mais un emplacement intermédiaire permanent. Lorsque la resource est appelée à travers une PURL, il y a une redirection vers son emplacement actuel (PURL 2009).

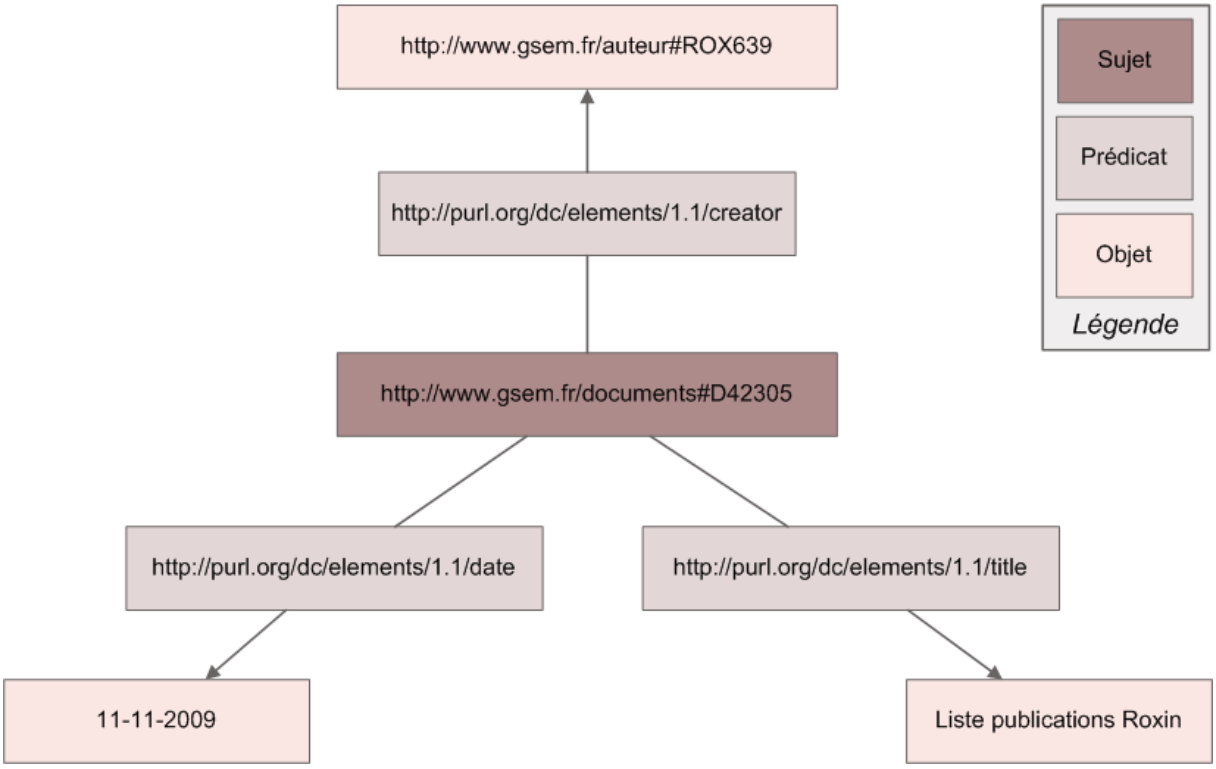

Figure 11. Graphe RDF décrivant trois énoncés RDF.

- ***Les graphes RDF sont encodés suivant le format XML*** permettant leur traitement et transport à travers le réseau. La Figure 12 présente le fichier XML associé au graphe RDF présenté dans la Figure 11. Les descriptions de ressources sont encodées en utilisant des étiquettes XML particulières, issues du vocabulaire RDF prédéfini.

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description
            rdf:about="http://www.gsem.fr/documents#D42305">
            <dc:creator>rdf:resource="http://www.gsem.fr/auteur#ROX639"
            </dc:creator>
      </rdf:Description>
      <rdf:Description>
                  rdf:about="http://www.gsem.fr/documents#D42305
                  <dc:title>Liste Publications Roxin</dc:title>
      </rdf:Description>
      <rdf:Description>
                  rdf:about="http://www.gsem.fr/documents#D42305">
                  <dc:date>11-11-2009</dc:date>
      </rdf:Description>
</rdf:RDF>
```

Figure 12. Notation XML pour la description des énoncés RDF.

- Le langage RDF permet de formuler des énoncés concernant d'autres énoncés. Ce procédé est appelé ***réification*** (Réification 2009). La réification est particulièrement intéressante dans le contexte du Web sémantique, puisqu'elle permet de formuler des énoncés concernant des objets ayant été énoncés ailleurs, en y faisant référence en tant que

ressource (MCGUINESS 2004). RDF dispose des termes suivants pour le procédé de réification : `rdf:Statement`, `rdf:subject`, `rdf:predicate` et `rdf:objet`. La Figure 13 illustre l'usage de ces termes, à travers la réification de l'énoncé présenté au Tableau 2.

```
Enc:E1 rdf:type rdf:Statement .
Enc:E1 rdf:subject docid:D42305 .
Enc:E1 rdf:predicate dc:creator .
Enc:E1 rdf:object authid:ROX639 .
```

Figure 13. Vocabulaire RDF pour la réification.

En résumé, le langage RDF(S) définit les bases de la représentation et du traitement des métadonnées. Le modèle de données RDF(S) est à base de graphes. Ses concepts clés sont les ressources, les propriétés et les énoncés (ALESSO 2006). La syntaxe du langage RDF(S) est basée sur le langage XML. Le XML permet une *interopérabilité syntaxique*, alors que le RDF(S) permet une *interopérabilité sémantique*; les deux langages sont donc complémentaires.

2.2.3.2.2 LE LANGAGE OWL

Au dessus des standards RDF(S), les efforts de standardisation du W3C ont produit la famille de langages OWL (*Web Ontology Language*). Le langage OWL est actuellement le langage le plus populaire pour la création d'ontologies. OWL a été construit sur la base du langage RDF(S). Le langage OWL est défini de la même manière qu'a été défini le langage RDF(S), mais intègre des sémantiques plus riches. Une ontologie OWL est un ensemble de triplets RDF(S), utilisant le vocabulaire OWL (HERMAN 2007). Les classes et propriétés fournies par RDF(S) peuvent donc être utilisées pour créer un document OWL.

Toutefois, comparé à RDF(S), OWL permet d'exprimer des relations plus complexes et plus riches. C'est pour cette raison qu'OWL est beaucoup plus utilisé pour le développement d'ontologies. RDF(S) demeure un choix valide, mais ses limitations évidentes par rapport à l'OWL en font un « deuxième choix ».

Le langage OWL est organisé sous la forme de trois sous-langages avec des niveaux d'expressivité différents. Ces sous-langages sont (MCGUINESS 2004) :

- ***OWL Full*** est destiné aux utilisateurs recherchant un maximum d'expressivité couplé à la liberté syntaxique offerte par le RDF. OWL Full est compatible avec RDF(S), à la fois du point de vue sémantique que syntaxique: chaque document RDF(S) est un document OWL Full.
- ***OWL DL*** prend en compte l'ensemble des constructions OWL, mais leur utilisation doit respecter certaines restrictions. OWL DL emploie des logiques de description (*description logic*) [22322_9], et c'est pour cette raison qu'il porte ce nom. OWL DL est un sous-langage d'OWL Full. OWL DL restreint l'usage des constructeurs OWL et RDF. L'avantage de ce langage est de permettre un raisonnement efficace. Son désavantage est de ne pas être totalement compatible avec RDF. Un document RDF devra être étendu de certains points de vue, et restreint à partir d'autres points de vue, et ce afin d'être considéré comme un document OWL DL. Par contre, tout document OWL DL est un document RDF.
- ***OWL Lite*** restreint le langage OWL DL à un ensemble limité de constructeurs, notamment en excluant les énumérations de classes ou les cardinalités arbitraires. OWL DL comprend les hiérarchies de classes et de propriétés, ainsi que les contraintes simples, permettant de

modéliser des thésaurus ou des ontologies simples. Le principal avantage de ce langage est d'être facile à comprendre pour les utilisateurs et facile à implémenter pour les programmeurs. Son désavantage est, bien sûr, son expressivité restreinte.

Nous présentons dans ce qui suit un aperçu de la syntaxe OWL.

2.2.3.2.2.1 EN-TETE D'UNE ONTOLOGIE OWL

Les documents OWL sont généralement appelés ontologies OWL et sont des documents RDF. Une ontologie OWL débute généralement avec un ensemble de déclarations de noms de domaines et inclut un ensemble de propositions concernant l'ontologie elle-même (regroupées sous l'étiquette `owl:Ontology`). De telles propositions contiennent des informations de versionnage et d'éventuels commentaires. Elles peuvent indiquer si l'ontologie courante importe d'autres ontologies.

Lorsqu'une ontologie O_1 importe une autre ontologie O_2, alors l'ensemble des déclarations constituant O_2 est ajouté à O_1. L'import d'ontologies se fait à travers une déclaration de nom de domaine pointant vers l'URI référençant O_2, de façon à ce qu'O_1 puisse utiliser le vocabulaire d'O_2 dans ses propres déclarations. Il est à noter que les noms de domaines sont utilisés afin de limiter l'ambigüité, alors que les ontologies importées fournissent des définitions pouvant être utilisées.

Considérons le fragment d'ontologie suivant:

```
1.    <?xml version="1.0"?>
2.    <!DOCTYPE rdf:RDF [
3.          <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
4.    <rdf:RDF
5.          xml:base="http://www.gsem.fr/owl-schema/"
6.          xmlns:dc="http://purl.org/dc/elements/1.1/"
7.          xmlns:owl="http://www.w3.org/2002/07/owl#"
8.          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
9.          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
10.         <owl:Ontology rdf:about="">
11.               <rdfs:comment>Hierarchie de profils service
12.               </rdfs:comment>
13.               <rdfs:label>Hierarchie services</rdfs:label>
14.               <owl:priorVersion
15.                     rdf:resource="http://www.gsem.fr/schema/"/>
16.               <owl:imports
17.                     rdf:resource="http://www.gsem.fr/auth/"/>
18.         </owl:Ontology>
```

A la ligne 3, nous observons la définition de `"xsd"` en tant qu'abréviation pour le vocabulaire XML Schema. Vient ensuite l'élément racine d'une ontologie OWL, à savoir l'élément `rdf:RDF`, à l'intérieur duquel sont définis les noms de domaine utilisés par l'ontologie (lignes 5 à 9). La valeur de l'attribut `rdf:about` de l'élément `owl:ontology` indique l'URI servant de référence à l'ontologie. Si cette valeur est vide, comme dans l'exemple ci-dessus, l'URI de référence est considérée comme étant l'URI de base du document. La ligne 11 définit un commentaire sur l'ontologie. La ligne 13 définit un nom plus « lisible » pour l'ontologie. Les lignes 14 et 15 indiquent l'URI référençant une version précédente de l'ontologie. Enfin, les lignes 16 et 17 définissent l'import dans l'ontologie actuelle de l'ontologie référencée par l'URI fournie dans l'élément `owl:imports`. Il est à noter que la propriété owl:imports est transitive: si une ontologie O_1 importe une ontologie

O_2 et que l'ontologie O_2 importe elle aussi une ontologie O_3, alors l'ontologie O_1 importe aussi l'ontologie O_3.

2.2.3.2.2.2 LES CLASSES OWL

En OWL, les classes sont définies en utilisant l'élément `owl:Class`. Une classe peut être définie comme disjointe d'une autre classe, en utilisant l'élément `owl:disjointWith`. L'élément `owl:equivalentClass` permet de définir une relation d'équivalence entre des classes.

Il existe seulement deux classes prédéfinies: la classe `owl:Thing` et la classe `owl:Nothing`. La classe `owl:Thing` est la classe la plus générale, puisque c'est la classe qui contient l'ensemble des objets (tout objet est un objet). La classe `owl:Nothing` est une classe vide. Pour résumer, toute classe OWL est une sous-classe de `owl:Thing` et une superclasse de `owl:Nothing`.

L'exemple de code suivant définit la classe `"Vehicule"`, disjointe de la classe `"Velo"`. La classe `"Vehicule"` a pour classe équivalente la classe `"Voiture"`.

```
<owl:Class rdf:ID="Vehicule">
   <rdfs:subClassOf rdf:resource="#MoyensDeTransport"/>
</owl:Class>
<owl:Class rdf:about="#Vehicule">
   <owl:disjointWith rdf:resource="#Velo"/>
</owl:Class>
<owl:Class rdf:ID="Voiture">
   <owl:equivalentClass rdf:resource="#Vehicule"/>
</owl:Class>
```

Le Tableau 3 regroupe l'ensemble des constructeurs de classes OWL:

Terme	*Théorie des ensembles*	*Description*
`owl:Thing`	$\mathcal{U}$	L'ensemble de tous les individus.
`owl:Nothing`	$\emptyset$	L'ensemble vide.
`owl:oneOf`	$\{x_1, \ldots, x_n\}$	L'ensemble $x_1, \ldots, x_n$.
`rdfs:subClassOf`	$A \subseteq B$	A est un sous-ensemble de B.
`<owl:Restriction>` …R… `</owl:Restriction>`	$\{x/R\}$	L'ensemble des objets satisfaisant la condition R.
`owl:equivalentClass`	$A = B$	A est égal à B.
`owl:intersectionOf`	$A \cap B$	A intersection B
`owl:unionOf`	$A \cup B$	A union B
`owl:complementOf`	$\sim B$ $A \sim B$	Le complément de B dans U. Le complément de B dans A.
`owl:disjointWith`	$A \cap B = \emptyset$	A et B sont disjoints.

Tableau 3. Constructeurs de classe du langage OWL.

2.2.3.2.2.3 LES PROPRIETES OWL

Le langage OWL définit deux types de propriétés:

- ***Les propriétés objet*** sont des propriétés reliant des objets entre eux. Le code suivant présente un exemple d'une telle propriété. Les éléments `rdfs:domain` et `rdfs:range` correspondent au *Sujet* et respectivement *Objet* d'un triplet RDF(S). La définition d'une propriété objet correspond donc à un triplet RDF, où la propriété joue le rôle de *Prédicat*.

```
<owl:ObjectProperty rdf:ID="estEnseignePar">
        <rdfs:domain rdf:ressource="#cours"/>
        <rdfs:range rdf:resource="#membrePersonnelEnseignant"/>
</owl:ObjectProperty>
```

- ***Les propriétés « données »*** ou "data" (datatype) relient des objets à des valeurs de types de données. L'OWL ne définit pas des types de données et ne fournit pas non plus de outils pour définir ces types. L'OWL utilise les types de données définis par le langage XML Schema (en utilisant de cette manière l'architecture par couches du Web sémantique) (SPERBERG-MCQUENN 2008). Le code suivant illustre la définition d'une propriété data.

```
<owl:DatatypeProperty rdf:ID="age">
        <rdfs:range          rdf:resource=          "http://www.w3.org/2001/XMLSchema
#nonNegativeInteger"/>
</owl:DatatypeProperty>
```

Le langage OWL permet de définir des propriétés de propriétés (aussi appelées méta-propriétés ou caractéristiques). Ces types de propriétés sont présentés dans le tableau suivant (Tableau 4):

Terme	Définition
owl:TransitiveProperty	$P\ est\ transitive\ ssi\ \forall x, \forall y, \forall z, si\ P(x,y) et\ P(y,z), alors\ P(x,z).$
owl:SymmetricProperty	$P\ est\ symétrique\ ssi\ \forall x, \forall y, P(x,y) ssi\ P(y,x).$
owl:FunctionalProperty	$P\ est\ fonctionnelle\ ssi\ \forall x, \forall y, \forall z, si\ P(x,y) et\ P(y,z), alors\ y = z$
owl:InverseFunctionalProperty	$P\ est\ inversement\ fonctionnelle\ ssi\ \forall x, \forall y, \forall z, si\ P(y,x) et\ P(z,x),\ \ alors\ y = z$
owl:InverseOf	$R\ est\ l'inverse\ de\ P\ ssi\ \forall x, \forall y, P(x,y) ssi\ R(y,x).$

Tableau 4. Vocabulaire OWL pour la définition de propriétés.

2.2.3.2.2.4 LES INDIVIDUS OWL

En OWL, les instances de classes sont appelées individus. Leur définition respecte les mêmes règles qu'en RDF(S). L'exemple suivant définit un individu de la classe "Vehicule":

```
<rdf:Description rdf:ID="5641-XK-25"
   <rdf:type
rdf:resource="#Vehicule"/>
</rdf:Description>
```

```
<Vehicule rdf:ID="5641-XK-
25"/>
```

Contrairement aux systèmes de bases de données, ce n'est pas parce que deux URI sont différentes qu'elles font référence à des individus forcément différents. L'OWL définit donc des propriétés supplémentaires permettant de statuer sur l' « *égalité* » entre deux concepts. Ces propriétés sont listées dans le tableau suivant (Tableau 5):

Nom propriété	Description	Exemple

owl:sameAs	Permet de déclarer deux URI faisant référence à un même individu.	`<owl:Thing rdf:about="#Rio-de-Janeiro-Brazil">`
		`  <owl:sameAs rdf:resource="#Cidade-Maravilhosa" />`
		`</owl:Thing>`

owl:differentFrom	Permet de déclarer deux URI faisant référence à des individus différents.	`<owl:Thing rdf:about="#Rio-de-Janeiro-Brazil">`
		`  <owl:differentFrom rdf:resource="#Rio-de-Janeiro-Peru"/>`
		`</owl:Thing>`

owl:AllDifferent	Permet de déclarer un ensemble d'URI toutes différentes.	<owl:AllDifferent> <owl:distinctMembers rdf:parseType="Collection"> <cs:City rdf:about="#Rio-de-Janeiro-Brazil"/> <cs:City rdf:about="#Rio-de-Janeiro-Peru"/> <cs:City rdf:about="#Rio-de-Janeiro-Colombia"/> </owl:distinctMembers> </owl:AllDifferent>

Tableau 5. Propriétés OWL permettant de statuer sur l' « égalité » entre concepts.

Les constructeurs OWL (comme `owl:Class`, `owl:DatatypeProperty` et `owl:ObjectProperty`) sont des spécialisations de leurs équivalents en RDF. La Figure 14 illustre les relations de classe entre ces constructeurs.

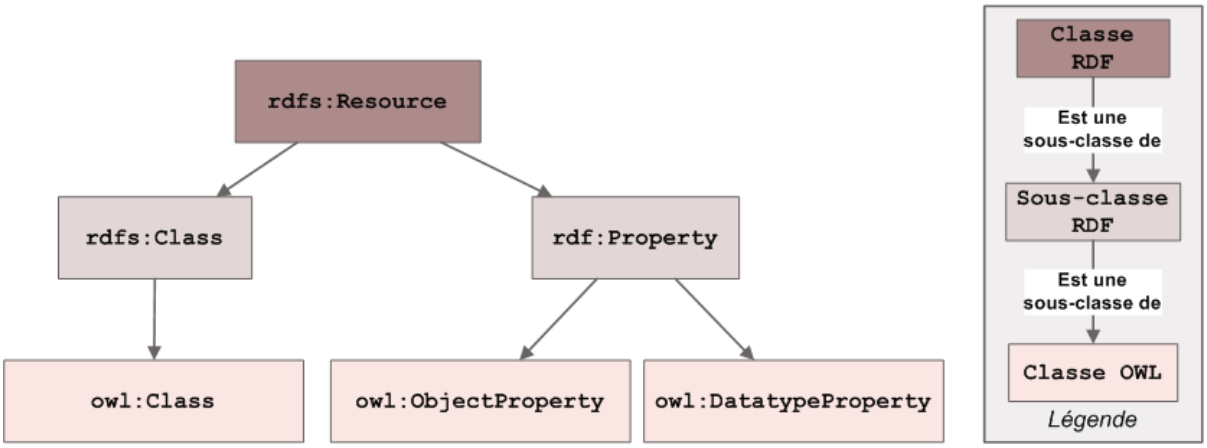

Figure 14. Relations de classe entre les constructeurs OWL et RDF(S).

Pour résumer, le langage OWL est le standard proposé pour les ontologies Web. Ce langage a été construit au dessus du langage RDF(S). C'est pour cette raison qu'il emploie la syntaxe à base de XML, utilisée par le langage RDF(S). Les instances ou individus sont définis en utilisant les descriptions RDF et la majorité des primitives RDF sont aussi utilisées. Le langage OWL supporte les sémantiques et les raisonnements formels grâce aux prédicats logiques et aux logiques descriptives. Ces éléments permettent de définir des règles entre les concepts d'une ontologie OWL.

2.2.3.2.3 AUTRES LANGAGES

Il existe d'autres langages permettant la description d'ontologies. Ici, nous n'allons en donner qu'un bref aperçu, car l'étude de ces langages est en dehors des objectifs de cette thèse.

Dans la recherche concernant les formalismes de représentation de la connaissance, une tendance actuelle est d'intégrer à des ontologies employant des logiques descriptives (*DL-style ontologies*) des règles de programmation logique (*LP-style rules*). Une des tentatives dans ce domaine est le langage ***SWRL (Semantic Web Rule Language)*** (HORROCKS et al. 2004) qui étend l'ensemble des axiomes OWL en incluant des règles de Horn (*Horn-like rules*). L'interopérabilité avec les ontologies OWL est assurée en référençant les concepts et propriétés OWL à l'intérieur des règles SWRL. Le langage ***DL-safe rules*** (BAADER 2002) représente une autre tentative de mélanger les ontologies OWL et les règles (MOTIK 2005).

Le langage ***WSML (Web Service Modeling Language)*** (DE BRUIJN 2008) constitue la tentative la plus récente de standardisation des langages d'ontologie pour le Web. Ce langage concerne plus particulièrement l'annotation de services Web sémantiques. Il s'agit en fait d'une famille de langages, chaque langage couvrant un formalisme différent de représentation de la connaissance. Les membres de cette famille sont:

- *Le modèle conceptuel WSMO (Web Service Modeling Ontology)* considéré en tant qu'ontologie de haut-niveau pour les services Web sémantiques;
- *Le language WSML (Web Service Modeling Language)* permettant de décrire formellement les éléments définis dans le modèle WSMO;
- *L'environnement d'exécution WSMX (Web Service Execution Model)*.

La Figure 15 (GRIMM et al. 2007a) illustre les différents langages de description d'ontologies et les relations qui les relient. Puisque certains langages sont construits à partir d'autres langages et sur la base de précédents standards, cette illustration peut être perçue comme une hiérarchie des langages du Web sémantique. Toutefois, cette illustration ignore certains détails de ces langages et est expressément imprécise.

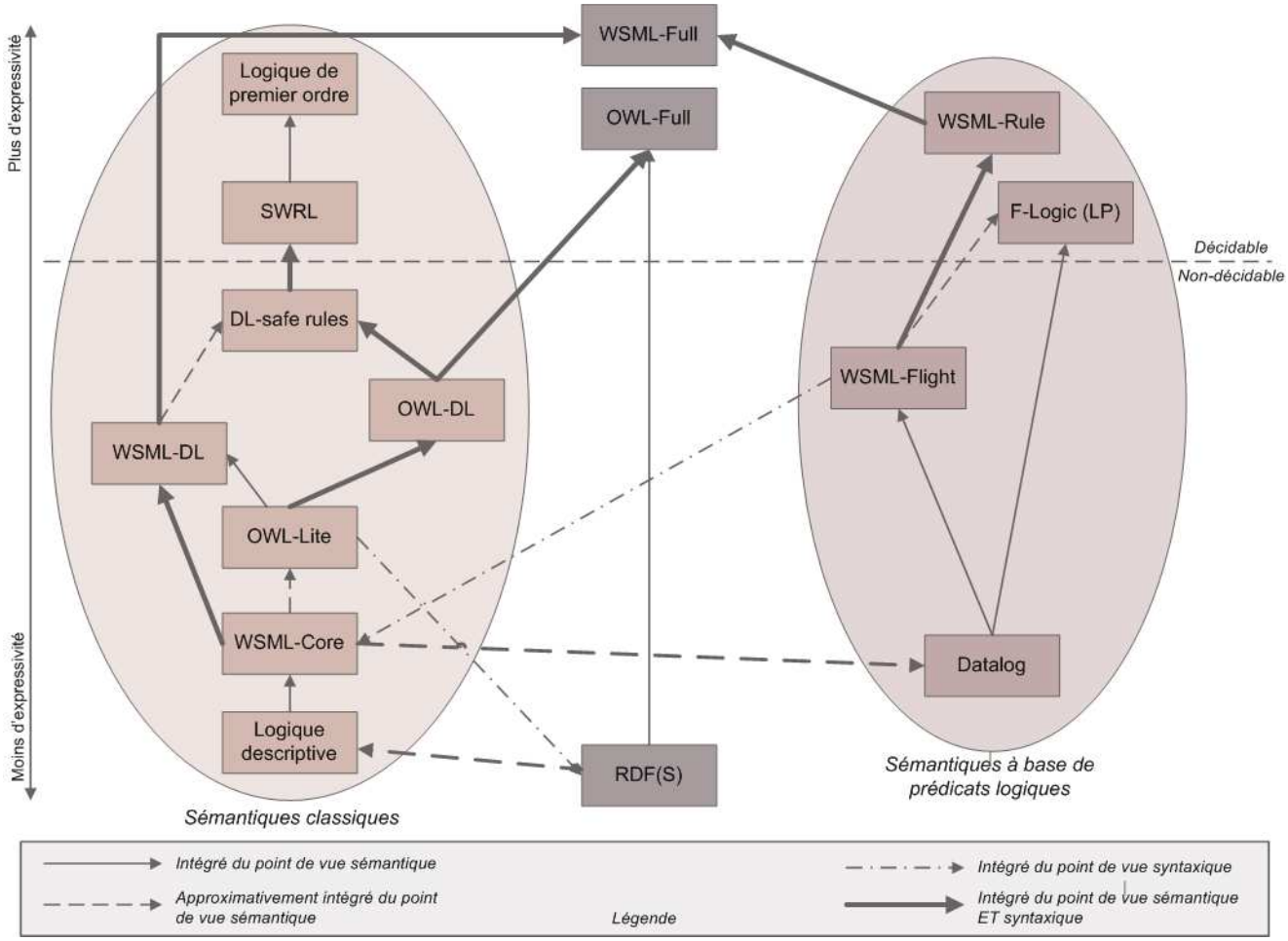

Figure 15. Relations entre les différents langages de description d'ontologies (GRIMM et al. 2007a).

2.2.4 Conclusion

Dans cette partie, nous avons discuté la vision du Web sémantique par rapport au Web actuel. Cette analyse nous a permis d'identifier le rôle fondamental joué par les métadonnées. Nous avons ensuite défini les ontologies comme étant un type de métadonnées, spécifiques à un domaine de connaissance donné. Deux principaux langages permettant de définir des ontologies, RDF(S) et OWL, ont été présentés.

Pour résumer, le Tableau 6 illustre les extensions du Web sémantique par rapport au Web actuel.

Web actuel	***Web sémantique***
Recherche à base de mots-clés	Recherche intégrant raisonnement et sémantiques
Interprétable par un humain	Interprétable par des ordinateurs
HTML/HTTP URL	XML/RDF(S)/OWL URI
Ressources (pages Web, services, etc.)	Ressources sémantiquement annotées

Tableau 6. Extensions du Web sémantique par rapport au Web actuel.

Nous remarquons l'annotation sémantique des ressources. Le modèle de métadonnées RDF et des ontologies au format OWL sont utilisés pour encoder ces sémantiques. Cette annotation des ressources permet des recherches plus efficaces, souvent utilisant des raisonnements sur les concepts définis dans les ontologies.

Dans notre cas, la recherche d'informations est synonyme de découverte de services. Or, l'utilisation d'annotations sémantiques implique l'utilisation de standards et procédés différents de ceux utilisés par le Web actuel. Nous allons donc nous intéresser aux services Web sémantiques et aux technologies sous-jacentes.

2.3 Les services Web semantiques

Comme nous l'avons vu dans les paragraphes précédents, les technologies liées au Web sémantique œuvrent vers un Web dont le sens pourrait être interprété par les ordinateurs. D'autre part, les technologies inhérentes aux services Web ont pour but de créer un environnement permettant aux fournisseurs de services de rendre leurs compétences accessibles sur la Toile. Pour ce faire, l'approche utilisée est d' « envelopper » les capacités des services Web dans une interface. C'est cette interface que les utilisateurs vont rechercher (via un registre de type UDDI) et c'est avec cette interface qu'ils vont interagir (via WSDL). La vision des *services Web sémantiques* est de combiner ces deux technologies de manière à rendre possible l'interaction automatique et dynamique entre différents systèmes logiciels (BREITMANN 2007).

D'abord, nous allons nous intéresser aux raisons déterminant l'existence des services Web sémantiques, ainsi qu'aux principales améliorations qu'ils apportent par rapport aux services Web

classiques. Ensuite, nous allons investiguer les approches de développement de services Web sémantiques, s'inspirant du Web sémantique[16].

2.3.1 SEMANTIQUES DES SERVICES WEB

Nous avons vu que les technologies inhérentes aux services Web décrivent une interface standard. Ces technologies permettent de créer un service Web, mais ne suffisent pas à atteindre l'interopérabilité et l'automation entre services. En effet, le standard WSDL ne permet qu'une spécification syntaxique des fonctionnalités d'un service Web. Le sens de ces fonctionnalités n'est pas décrit, et donc il est impossible pour d'autres applications d'utiliser cette information sans l'intervention d'un humain.

Ces lacunes peuvent être comblées grâce aux technologies du Web sémantique. Les fonctionnalités des services peuvent être décrites dans un langage compréhensible par un ordinateur. Les ontologies sont le chaînon manquant; elles constituent des descriptions de services, pouvant être publiées, recherchées, mais surtout interprétées par des ordinateurs.

La vision des services Web sémantiques repose sur les concepts suivants (BREITMANN 2007) :

- ***Un service Web sémantique est défini comme la réalisation d'une ou plusieurs actions par une entité, et ce pour une autre entité.*** Un service est défini dans un domaine donné (transport, e-commerce, etc.); ce domaine s'appelle le domaine de valeur du service. L'entité réalisant les actions du service est appelée fournisseur de service, l'autre entité étant le demandeur de service. Ces définitions correspondent à celles des services Web.
- ***Un service Web sémantique peut être considéré à différents niveaux d'abstraction.*** Un service concret est la réalisation spécifique de certaines actions d'une entité pour une autre. Un service abstrait correspond à un ensemble de services concrets. Les services abstraits permettent de faire référence à des services hypothétiques, sans avoir à spécifier l'ensemble des aspects les définissant.
- ***Une représentation de service pouvant être interprétée par un ordinateur est appelée description de service.*** Différents formalismes ont été définis pour les descriptions de services. Il s'agit d'ontologies, chacune offrant des vocabulaires différents. Le processus de médiation consiste à traduire une description de service d'un vocabulaire à un autre.

Dans la suite, nous étudions les technologies permettant de créer de telles descriptions de services.

2.3.2 DESCRIPTION DES SERVICES WEB SEMANTIQUES

Plusieurs approches existent pour la description de services Web sémantiques. Il s'agit principalement de soumissions de membres du W3C: WSMO (DE BRUIJN et al. 2005), OWL-S (MARTIN 2004), SWSF (BATTLE 2005) et WSDL-S (AKKIRAJU 2005). Ces propositions différent, tant dans le but visé, que dans l'approche de modélisation ou dans les langages utilisés pour les descriptions de services. Dans ce qui suit, nous allons donner une brève présentation pour chaque approche, mais seule l'approche OWL-S sera détaillée.

[16] D'autres technologies alternatives permettent de capturer les sémantiques d'un service, mais leur étude est en dehors du sujet de cette thèse.

L'approche WSMO (*Web Service Modeling Ontology*) (DE BRUIJN et al. 2005) est une initiative pour créer une ontologie décrivant les différents aspects en lien avec les services Web sémantiques. Le but est de résoudre le problème d'intégration de ces services. Cette approche a été développée par le groupe de travail Semantic Web Services[17] du cluster ESSI[18]. L'initiative WSMO a pour but la fourniture d'une plate-forme complète pour manipuler les services Web sémantiques. Cette plate-forme comprend un langage de modélisation, le WSML et un environnement d'exécution, le WSMX. Un modèle conceptuel, l'ontologie WSMO, est aussi fourni. Le langage WSML est utilisé pour décrire formellement les éléments définis dans l'ontologie WSMO. Le principal avantage de cette approche est le langage WSML, qui fournit des termes spécifiques aux services Web sémantiques, tels que : "goal", "web service", "interface", "choreography" ou "capability". A la manière du langage OWL, le langage WSML possède plusieurs variantes, chacune respectant différents formalismes de représentation des connaissances. Les inconvénients de cette approche résident dans sa relative nouveauté ; en effet, il existe peu d'outils de développement permettant la manipulation et la création d'ontologies WSML (*Web Service Modeling Language*) (LAUSEN 2007).

L'approche SWSF (*Semantic Web Services Framework*) (BATTLE 2005) est une tentative relativement récente, dont le but est de fournir une plate-forme d'annotation de services Web sémantiques. Cette approche s'est grandement inspirée de travaux précédents, notamment ceux concernant l'OWL-S et le langage PSL[19] (*Process Specification Language*). Cette plate-forme est une proposition du *Semantic Web Services Language Comittee*, et a été soumise au W3C en septembre 2005. Cette approche repose sur l'ontologie SWSO (*Semantic Web Service Ontology*), qui peut être vue comme une extension de l'ontologie OWL-S. La principale différence réside dans le fait que l'ontologie SWSO repose sur un langage plus riche du point de vue sémantique par rapport à l'OWL, à savoir le SWSL (*Semantic Web Service Language*). Toutefois, contrairement aux autres approches, nous n'avons pas connaissance d'efforts d'implémentation basés sur l'approche SWSF (LAUSEN 2007).

L'approche WSDL-S (*Web Service Semantics*) (AKKIRAJU 2005) est plutôt minimaliste, en ce sens qu'elle propose une extension des descriptions WSDL « traditionnelles », en utilisant des étiquettes d'annotation spécifiques. L'approche est similaire à l'approche WSMO ou encore à l'OWL-S, puisqu'elle prend en compte les pré-conditions et les effets d'un service. Le principal inconvénient de cette approche est d'être indépendante de tout langage de représentation d'ontologie. WSDL-S ne préconise aucun formalisme pour les descriptions sémantiques. Il s'agit seulement d'ajouter quelques attributs utiles aux étiquettes XML du WSDL, de manière à référencer des annotations sémantiques.

L'approche OWL-S définit une ontologie pour l'annotation sémantique des services Web. Le but d'OWL-S est de rendre possible une découverte, invocation, composition et exécution automatiques de services Web (MARTIN 2004).

Nous avons choisi de baser notre approche sur cette ontologie, car elle est suffisamment détaillée et générale pour permettre la description de tout service. De plus, elle représente aujourd'hui l'ontologie standard pour la description des propriétés et fonctionnalités des services Web. Dans les paragraphes suivants, nous investiguons les concepts de haut niveau de l'ontologie OWL-S.

[17] Voir http://www.wsmo.org
[18] Voir http://www.essi-cluster.org
[19] Standardisé par ISO 18269.

2.3.3 L'APPROCHE OWL-S

OWL-S est une ontologie OWL, permettant de décrire les différents aspects d'un service Web. A l'origine de cette initiative se trouve l'ontologie service DAML (DAML-S), publiée pour la première fois au mois de mai 2001. A l'époque, cela représentait le premier effort vers l'annotation sémantique des services Web. OWL-S en est actuellement à sa version 1.2 (MARTIN 2008), la première version ayant été soumise pour acceptation au W3C en novembre 2004 (MARTIN 2004).

L'approche OWL-S tient compte des recommandations actuelles du Web sémantique, tout en gardant des liens avec le monde du Web traditionnel, notamment en reliant les descriptions OWL-S aux descriptions WSDL existantes. C'est aussi une des raisons qui a motivé notre choix d'utiliser cette ontologie comme point de départ de notre contribution.

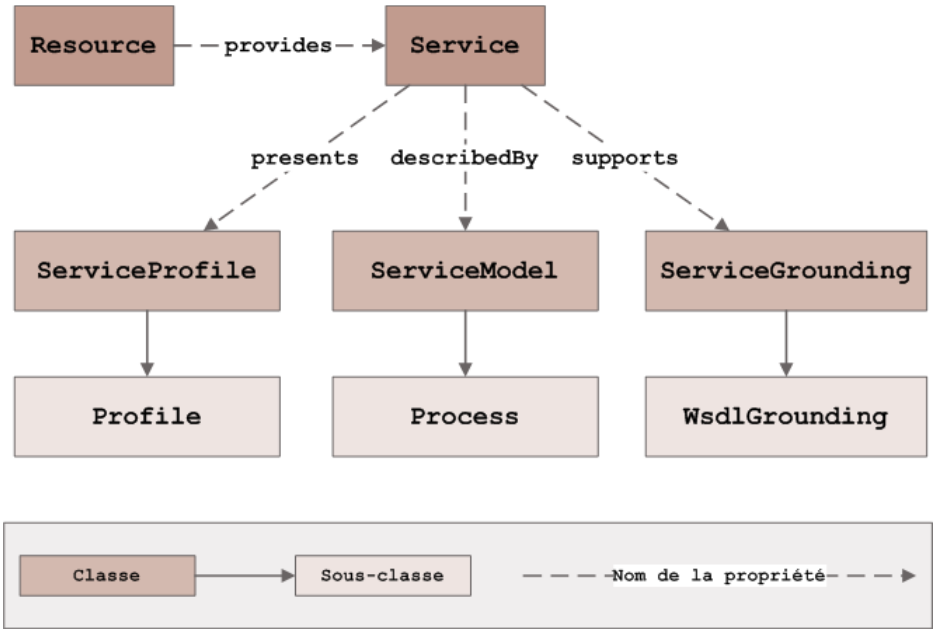

Figure 16. Concepts de base de l'ontologie OWL-S.

L'ontologie OWL-S définit le concept de `"Service"`[20] comme étant le concept de plus haut niveau (MARTIN 2004). Le concept `"Service"` est utilisé en tant que référence dans l'organisation des déclarations de services Web. Chaque service est déclaré en créant une instance de ce concept. Le concept `"Service"` est lui-même divisé en trois sous-concepts, listés ci-dessous et illustrés par la Figure 16 (MARTIN 2004).

- Le concept de `"Service Profile"`[21] représente une ontologie qui décrit ce que fait le service, notamment à travers ses fonctionnalités et ses propriétés non-fonctionnelles. C'est ce concept qui est utilisé pour rendre automatique la découverte des services Web. L'ontologie sous-jacente est détaillée au paragraphe 2.3.3.1.
- Le concept de `"Service Model"`[22] représente aussi une ontologie qui décrit comment le service réalise ses fonctionnalités. Les différents processus constituant le service sont décrits au paragraphe 2.3.3.2. C'est ce concept qui est utilisé pour rendre automatique la composition de services Web.

[20] http://www.daml.org/services/owl-s/1.2/Service.owl
[21] http://www.daml.org/services/owl-s/1.2/Profile.owl
[22] http://www.daml.org/services/owl- s/1.2/Process.owl

- Le concept de "Service Grounding"[23] est une ontologie dont les concepts décrivent comment utiliser un service, autrement dit comment un client peut invoquer le service. Cette ontologie est utilisée pour automatiser l'invocation de services Web. Elle est détaillée au paragraphe 2.3.3.3.

Le concept de "Resource" représente le fournisseur de service. Dans le formalisme OWL-S, une *Ressource* fournit un ou plusieurs *Services*.

Chacun des concepts du formalisme OWL-S définit une ontologie (ou un vocabulaire de termes) permettant de décrire différents aspects du service. Dans ce qui suit, nous allons présenter chacune de ces ontologies.

2.3.3.1 L'ONTOLOGIE SERVICEPROFILE

Nous avons indiqué précédemment que chaque service est déclaré en tant qu'instance du concept "Service". Chacune de ces instances présente (presents) zéro ou plusieurs profils de service (ServiceProfile). Un profil de service exprime « *ce que fait le service* » et est utilisé pour la publication du service dans un annuaire. Un profil de service est utilisé en tant que modèle pour les requêtes de service. C'est cette description du service qui est utilisée pour la découverte et la comparaison de services. La Figure 17 illustre les principales classes de l'ontologie ServiceProfile (MARTIN 2004).

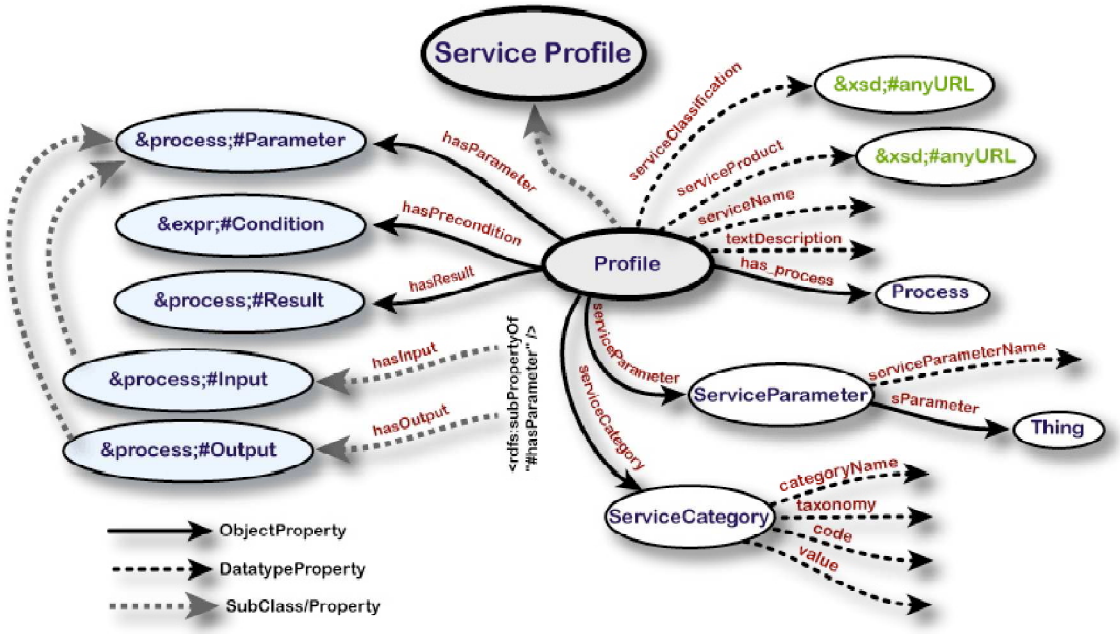

Figure 17. Classes de l'ontologie ServiceProfile (MARTIN 2004).

Un profil de service contient des informations relatives, à la fois, aux aspects fonctionnels et non-fonctionnels du service. Les informations non-fonctionnelles comprennent notamment des références vers des classifications existantes (par exemple NAICS (NAICS 2009) ou la classification de Nice (Classfication de Nice 2009) – voir § 4.3.1), des informations concernant le fournisseur du service et des informations concernant l'évaluation qualitative du service. Dans l'approche OWL-S, le profil d'un service offre une description non-fonctionnelle de ce service par rapport à trois types

[23] http://www.daml.org/services/owl-s/1.2/Grounding.owl

d'information (MARTIN 2008) : quelle entreprise ou organisation fournit le service, quelles opérations sont réalisées par le service et quelles sont les caractéristiques du service. La description des caractéristiques du service (voir Figure 17) comprend la catégorie du service (`ServiceCategory`), une description textuelle (`textDescription`), des informations de contact et un ensemble non-restreint de paramètres service (`ServiceParameters`). Ces paramètres service ne font pas référence aux entrées et sorties du service, mais concernent, par exemple, la qualité du service ou son temps de réponse.

L'information la plus importante demeure toutefois la spécification des éléments *fonctionnels du service* (ou IOPEs). Ces IOPEs représentent un sous-ensemble des IOPEs définis par la classe `ServiceModel` (MARTIN 2004). Les *entrées* et *sorties* du service représentent les opérations effectuées par le service. Les *pré-conditions* et les *effets* du service indiquent les changements, produits suite à l'exécution du service. Les entrées et les sorties du service font référence à des concepts OWL (voir 2.2.3.2.2) décrivant les types d'instances envoyées au service et, respectivement les réponses fournies par le service. Les pré-conditions et les effets ne doivent pas respecter un format imposé. Il peut s'agir d'expressions SWRL (HORROCKS et al. 2004), DRS (MCDERMOTT 2004) ou encore KIF (GENESERETH 1994).

Un des problèmes soulevés par cette approche est le fait de ne pas spécifier les sémantiques des pré-conditions et effets en utilisant l'ontologie OWL-S (LAUSEN 2007). Ce sont les langages cités ci-dessus qui permettent de spécifier ces conditions. Durant le processus de découverte de service, les parties utilisant OWL-S doivent se mettre d'accord sur le langage à utiliser pour l'expression des conditions et des comparaisons de services.

2.3.3.2 L'ONTOLOGIE SERVICEMODEL

Un service peut être *décrit par* (`described by`) un modèle de service, spécifiant « *comment le service fonctionne* ». Un modèle de service rend possible l'invocation, la composition, ou encore la récupération de services. Le modèle de service définit les interactions du service en tant que processus. Ces interactions sont exprimées à l'aide des concepts d' « *entrée* », « *sortie* », « *pré-condition* » et « *effet* » du service. Ces concepts sont appelés IOPEs (*Inputs, Outputs, Preconditions, Effects*). Pour connecter ces IOPEs à un processus donné, le formalisme OWL-S définit les propriétés : `hasInput`, `hasOutpupt`, `hasPrecondition`, et `hasResult` (voir Figure 17) (MARTIN 2004).

Un processus (`Process`) n'est pas nécessairement un programme à exécuter, mais plutôt une spécification des méthodes qu'un client peut utiliser pour interagir avec un service. L'approche OWL-S fait la distinction entre des processus atomiques, des processus simples et des processus composés (MARTIN 2008) :

- ***Les processus atomiques*** (`AtomicProcess`) sont des opérations simples. Ces processus peuvent être invoqués, n'ont pas de sous-processus, et, du point de vue du client, ils s'exécutent en une étape. Etant une sous-classe de la classe `Process`, chaque processus atomique peut spécifier ses propres IOPEs. L'exemple suivant illustre la définition d'un processus atomique permettant de sélectionner un restaurant depuis une liste de restaurants.

```
<process:AtomicProcess rdf:ID="ChoixRestaurant">
        <process:hasInput rdf:resource="#RestaurantsDisponibles" />
        <process:hasOutput rdf:resource="#RestaurantChoisi" />
```

```
</process:AtomicProcess>
```

- ***Les processus simples*** (`SimpleProcess`) représentent des « vues » de processus atomiques ou complexes. Ces processus ne peuvent pas être invoqués, et sont aussi perçus par le client comme s'exécutant en une seule étape. Ces processus sont utilisés en tant qu'éléments d'abstraction. Un tel processus est utilisé dans le cas où il représente le seul processus du service et lorsqu'il n'a pas de description WSDL associée.
- ***Les processus composés*** (`CompositeProcess`) sont construits à partir de processus atomiques ou simples. Le formalisme OWL-S fournit des méthodes pour contrôler les flux à l'intérieur de tels processus, mais leur étude est en dehors des objectifs de cette thèse.

Un processus a des résultats et des sorties, qui peuvent être fonction d'une condition, encore une fois exprimée utilisant un langage logique tel SWRL, KIF ou DRS. Ceci amène encore le problème des sémantiques de ces conditions (LAUSEN 2007).

2.3.3.3 L'ONTOLOGIE SERVICEGROUNDING

Afin de garder un lien vers le monde des services Web, un service OWL-S supporte (`supports`) une implémentation (`grounding`) qui relie les constructions du modèle de processus à des spécifications détaillées des formats de messages, de protocoles, etc. L'approche OWL-S permet donc de définir des liens entre (MARTIN 2008) :

- des processus atomiques et des opérations WSDL;
- des entrées/sorties de processus et des messages WSDL.

Il est à noter que même si le langage WSDL est la seule implémentation définie en OWL, l'approche OWL-S n'est pas restreinte au WSDL en tant que technologie service. L'approche OWL-S propose une description de service générale, tout en restant extensible à d'autres mécanismes d'implémentation (MARTIN 2004).

2.4 CONCLUSION

Dans ce chapitre, nous avons présenté les technologies de base concernant notre domaine de recherche. Il s'agit des principes et des technologies du Web sémantiques, notamment l'architecture du Web sémantiques, les ontologies, et les langages de description d'ontologies. Nous avons ensuite étudiés les services Web sémantiques, notamment les langages permettant leur description sémantique. Nous avons justifié pourquoi notre choix de modélisation s'est porté sur le langage OWL-S, et nous avons présenté les principes sous-jacents à cette approche.

Dans ce qui suit, nous nous intéressons à la définition des services sensibles au contexte. Nous étudions comment les technologies décrites précédemment peuvent être employées pour modéliser un contexte, puis pour prendre en compte ce contexte lors du processus de découverte de services. Le but de cette thèse étant de proposer un protocole de découverte pour des services sensibles au contexte, nous devons d'abord d'examiner les travaux existants dans ce domaine, avec leurs avantages et inconvénients.

CHAPITRE 3.
APPROCHES POUR LA DECOUVERTE DE SERVICES SENSIBLES AU CONTEXTE

Dans ce chapitre, nous présentons une analyse des travaux existants dans le domaine de la découverte de services sensible au contexte. Ce chapitre introduit les concepts de service sensibles au contexte (*service SC*) et de découverte de services sensibles au contexte (*découverte SC*). Ce chapitre définit ce qu'est le contexte et explique comment la sensibilité au contexte peut améliorer la découverte de services. Sont ensuite étudiées les principales approches décrites dans la littérature pour la découverte SC. Ces travaux sont comparés ensuite à l'approche proposée dans cette thèse. Les points forts de notre approche seront mis en évidence par rapport aux solutions présentées dans ce chapitre.

3.1 SERVICES SENSIBLES AU CONTEXTE

3.1.1 DEFINITIONS DU CONTEXTE

Lorsque nous cherchons une définition du mot « *contexte* », nous pensons presqu'instantanément à des mots tels « *temps* » et « *position* ». Ce sont en effet deux exemples presqu'évidents de contexte. Il en existe toutefois de nombreux autres, plus ou moins intuitifs. Donner une définition précise du mot « contexte » n'est pas une tâche facile, et a été un sujet de recherche dans de nombreux domaines scientifiques (intelligence artificielle, informatique ubiquitaire, etc.) (SCHILIT 1994), (DEY 2001).

Dans le cadre de cette thèse, nous adoptons une des définitions les plus citées dans le domaine de l'informatique pervasive. Il s'agit de la définition donnée par Abowd (ABOWD 1999) selon laquelle le contexte représente toute « *information pouvant être utilisée pour caractériser la situation d'une entité. Une entité peut être une personne, un endroit ou un objet, qui est considéré comme relevant par rapport à l'interaction entre un utilisateur et une application, en incluant l'utilisateur et l'application*[24] ».

Toujours dans (ABOWD 1999), la sensibilité au contexte est définie comme étant « *une propriété d'un système, utilisant le contexte pour fournir des informations et/ou des services pertinents par rapport à l'utilisateur, où la pertinence dépend de la tâche réalisée par l'utilisateur*[25] ».

Plusieurs chercheurs ont tenté de classifier les différents types de contextes pouvant être pertinents pour un utilisateur au moment d'accéder à un service d'information. Dans (DEY 2001), les auteurs soulignent trois composantes importantes du contexte:

- ***Le contexte spatial*** – où est-ce que l'utilisateur se trouve ;
- ***Le contexte social*** – quelles sont les autres personnes qui se trouvent à proximité ;
- ***Le contexte informatif*** – quelles sont les ressources disponibles à proximité.

Dans (ROXIN et al. 2008a), nous avons proposé une classification plus récente des informations contextuelles. Deux principaux types d'informations contextuelles sont définis, tel qu'illustrés par la Figure 18:

- ***L'information contextuelle primaire***, ou de base, est constituée par l'information spatiale: identité, temps et position. Ce type d'information contextuelle peut être utilisé afin d'indexer des entités.
- ***L'information contextuelle secondaire*** contient des aspects supplémentaires pouvant caractériser une entité. Ces informations sont classées en fonction du type de contexte auquel elles font référence (selon la classification donnée dans (ABOWD 1999)): contexte personnel, technique, spatial, social ou encore physique.

[24] "*Information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to interaction between a user and an application including the user and application themselves*". (ABOWD 1999)

[25] "*a property of a system that uses context to provide relevant information and/or service to the user, where relevancy depends on the user's task*". (ABOWD 1999)

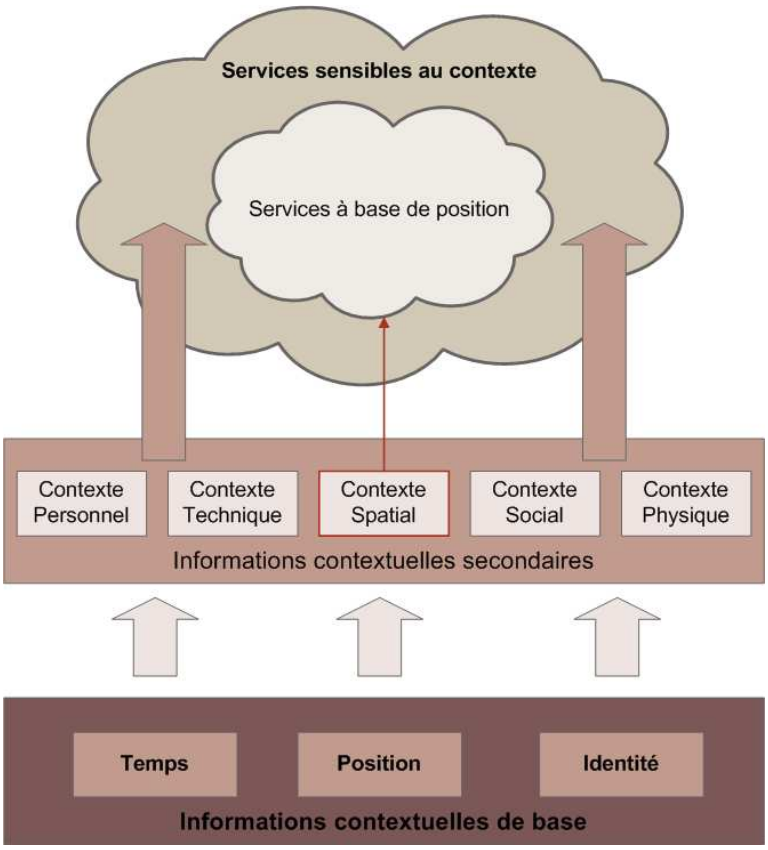

Figure 18. Types d'informations contextuelles.

La Figure 18 illustre comment le type d'informations contextuelles utilisées par un service en détermine l'appellation. Les services à base de position utilisent seulement les informations du contexte spatial, alors que les services sensibles au contexte font usage de l'ensemble des informations contextuelles,

Étant donné la diversité d'informations contextuelles, une modélisation du contexte est nécessaire et ce afin de pouvoir définir et stocker les données dans une forme pouvant être interprétée par un ordinateur. La partie suivante présente les différentes modélisations de contexte présentes dans la littérature.

3.1.2 MODELISATION DU CONTEXTE

Les modèles contextuels reposent sur différents types de structures de données. Les auteurs de (STRANG 2004) listent les principales approches de modélisation du contexte. Ce sont ces approches que nous présentons dans les paragraphes suivants.

Les modèles « clé/valeur » représentent la structure la plus simple permettant de modéliser un contexte. Les capacités d'un service sont modélisées à travers des paires du type « clé/valeur ». Les services sont ensuite découverts en effectuant des recherches sur ces paires « clé/valeur ». L'une des premières approches dans ce domaine a été celle de Schilit (SCHILIT 1994), qui a fait appel aux paires « clé/valeur » pour associer une valeur à un type d'information contextuelle. Ces modèles « clé/valeur » sont simples à gérer, mais ne permettent pas une structure sophistiquée, nécessaire à l'application d'algorithmes efficaces pour le renvoi de contexte.

Les modèles utilisant des schémas de balisage utilisent une structure de données hiérarchique, constituée d'étiquettes de balisage avec leurs attributs et contenus. Les profils constituent un exemple typique de ces modèles. Ils reposent principalement sur des dérivées du langage XML. D'autres exemples incluent le modèle capacités/préférences CC/PP (*Composite Capability/Preference Profiles*) (KLYNE 2004) ou encore le modèle UAProf (*User Agent Profile*) (UAProf 2001) Le but de l'approche CC/PP est de permettre au client d'exprimer les capacités logicielles et matérielles de son terminal, ainsi que des préférences. Le modèle UAProf a pour but de créer des spécifications en vue de développer des protocoles orientés mobilité. Ces deux modèles atteignent le même niveau d'expressivité que pour le langage RDF(S), puisqu'ils reposent sur le langage RDF. Dans ce domaine, une des approches les plus innovantes est celle présentée dans (HENRICKSEN et al., Generating Context Management Infrastructure from High-Level Context Models 2003) Les auteurs y définissent des profils contextuels compréhensibles et structurés (*CSCP - Comprehensive Structure Context Profiles*). Cette approche est innovante car les auteurs n'y définissent pas de structure hiérarchique fixe. Le but est de tirer avantage de la flexibilité du langage RDF(S) afin d'exprimer les informations de profil ou contextuelles. De manière similaire, les auteurs de (INDULSKA 2003) proposent d'étendre le modèle CC/PP. Leur approche, *CC/PP Context Extension*, permet d'étendre le vocabulaire de base des modèles CC/PP et UAProf, et ce en intégrant des arbres du type composant-attribut en lien avec des aspects contextuels.

Les modèles orientés-objet utilisent les techniques de la programmation orientées-objet afin de tirer parti de leurs avantages, notamment l'encapsulation, la réutilisation des ressources ou encore l'héritage. Les approches existantes représentent les éléments contextuels sous la forme d'objets, encapsulant ainsi les détails relatifs au traitement et à la représentation du contexte. Ce type de modélisation repose souvent sur un formalisme UML, textuel ou graphique. En effet, la structure de l'UML est adaptée à la modélisation du contexte, et, dans la littérature, il existe plusieurs approches reposant sur ce principe (HENRICKSEN et INDULSKA 2006), (GRIMM 2007). Un des exemples les plus connus de ce type de modélisation est constitué par le système Hydrogen (HOFER 2003). Les auteurs y définissent le contexte en tant que modèle UML, employant plusieurs classes. La Figure 19 illustre cette modélisation du contexte.

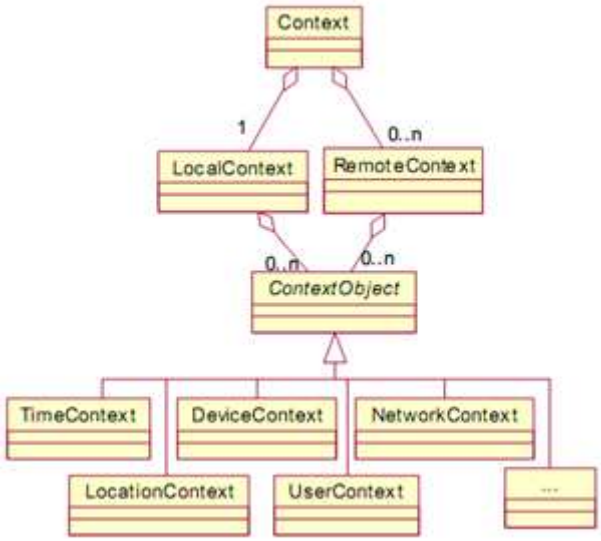

Figure 19. Modélisation UML du contexte dans le cadre du système Hydrogen (HOFER 2003).

Les modèles à base de logique emploient des expressions et des règles pour définir des modèles contextuels. Une logique permet de définir les conditions utilisées pour déduire une information à partir d'un ensemble d'informations existant. Ces conditions sont définies dans le cadre d'un système formel. L'information contextuelle est donc modélisée de manière formelle, en utilisant des expressions ou des conditions. L'ensemble des systèmes reposant sur ce type de modélisation présente un haut degré de formalisme. (MCCARTHY 1997), (GHIDINI 2001) et (GRAY 2001) présentent quelques exemples d'approches de modélisation appliquées dans ce domaine. Leur étude est en dehors des buts poursuivis dans cette thèse.

Les modèles à base d'ontologies définissent le contexte en utilisant des concepts et des relations entre concepts. Les ontologies constituent une alternative très prometteuse pour la modélisation des informations contextuelles. Elles permettent d'atteindre un haut niveau d'expressivité, mais aussi d'utiliser des techniques de raisonnement particulières. Il existe aujourd'hui plusieurs plateformes sensibles au contexte reposant sur ce type de modélisation (WANG 2004), (BROENS 2004).

Dans (STRANG 2004), les auteurs évaluent les approches listées ci-dessus. Ils définissent une liste des besoins clés, adaptés aux environnements ubiquitaires, auxquels les modélisations de contexte doivent répondre. Il s'agit, entre autres, de la richesse et la qualité de l'information, l'ambiguïté, le niveau de formalisme et l'applicabilité aux environnements existants. Dans leur analyse, les auteurs identifient la modélisation à base d'ontologies comme l'approche la plus expressive et répondant au plus grand nombre de besoins (STRANG 2004).

En effet, le principal avantage des modèles « clé/valeur » est leur simplicité (notamment au niveau de la gestion et du risque d'erreur), mais cette simplicité devient un inconvénient lorsqu'il s'agit prendre en compte des métadonnées ou de l'ambiguïté. La seule « force » de ce type de modélisation demeure en son applicabilité aux environnements ubiquitaires existants. Les modèles reposant sur des schémas de balisage présentent les mêmes lacunes, notamment dues aux contraintes liées à l'usage de l'XML et du RDF(S). Leur principal avantage est leur applicabilité dans le cadre d'infrastructures existantes et reposant sur des langages à base de balises (notamment dans le domaine des services Web). Les modèles orientés-objet atteignent un haut niveau de formalisme, mais sont difficilement intégrables dans les environnements existants (car ils demandent beaucoup de ressources machine). Ils sont généralement utilisés pour décrire la structure du contexte Le niveau de formalisme atteint est encore plus haut pour les modèles logiques, mais il est difficile d'envisager leur application aux terminaux mobiles d'aujourd'hui. En effet, les modèles logiques ont besoin d'un raisonneur permettant de valider les informations contextuelles, or ceci est difficilement intégrable dans les terminaux mobiles actuels.

Les modèles à base d'ontologies constituent donc l'approche la plus prometteuse pour les environnements ubiquitaires. C'est pour cette raison que l'approche présentée dans cette thèse utilise des ontologies pour la modélisation des informations contextuelles (et plus particulièrement le modèle OWL-S). Ces informations sont utilisées afin de délivrer des services sensibles au contexte. Dans ce qui suit, nous allons définir ce que sont ces services, puis nous allons étudier les approches existantes permettant la découverte de tels services.

3.1.3 SERVICES SENSIBLES AU CONTEXTE

Nous venons de voir que les services traditionnels sont basés sur des entrées explicites et ne tiennent pas compte du contexte. Si on considère un service traditionnel de type « *Pages Jaunes* », il

ne prend en compte que l'information saisie par l'utilisateur (par exemple un nom de restaurant). Le service retournera une liste de restaurants qui correspondent à la requête de l'utilisateur (soit ils ont le même nom, soit proposent le style de cuisine recherché). Le protocole de découverte sous-jacent ne prend en compte qu'un nombre limité d'informations définissant le service. Il ne prendra pas en compte le nombre de tables disponibles dans le restaurant ou encore la distance qui sépare l'utilisateur du restaurant. Un service sensible au contexte peut prendre en compte ces informations et donc fournir une réponse plus adaptée aux besoins de l'utilisateur. La Figure 20 schématise les différences entre ces deux types de services.

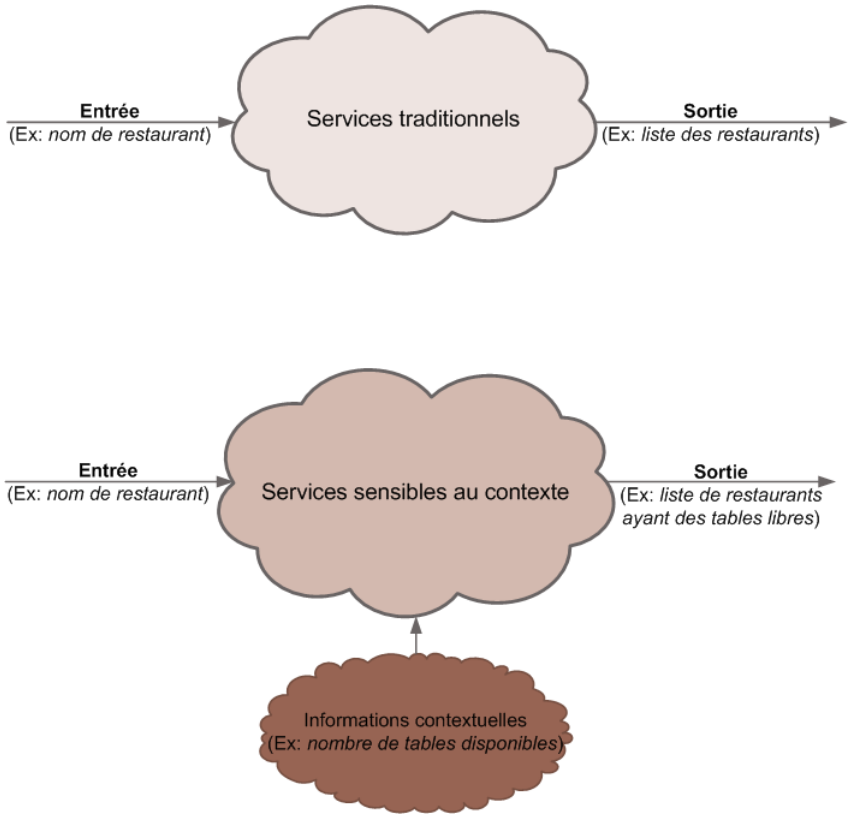

Figure 20. Services traditionnels et services sensibles au contexte.

Les services sensibles au contexte considèrent à la fois les entrées fonctionnelles et contextuelles (ou non-fonctionnelles). La sortie d'un tel service dépend donc des informations contextuelles fournies. Ces informations sont fournies au service soit à travers des fournisseurs de contexte (tel un équipement de positionnement, des capteurs, etc.), soit par l'utilisateur lui-même (généralement en remplissant un profil utilisateur ou un formulaire spécifique), soit par le service lui-même (un service parking peut diffuser le nombre de places restantes dans le parking).

La recherche de service confronte deux acteurs, chacun ayant un problème différent : le fournisseur de service essaie d'annoncer du mieux possible ses services et l'utilisateur ne sait pas où trouver le service voulu. L'introduction des services sensibles au contexte ne résoud pas le problème de la recherche de services. De plus, les mécanismes utilisés pour la découverte de services Web traditionnels ne permettent pas la découverte de services sensibles au contexte. *C'est le processus de découverte qui doit intégrer cette sensibilité aux informations contextuelles*. Ceci constitue l'idée maîtresse de notre travail de recherche. La partie suivante étudie les principaux protocoles de découverte utilisant des informations contextuelles.

3.2 PROTOCOLES DE DECOUVERTE DE SERVICES SENSIBLES AU CONTEXTE

Différentes approches ont été proposées dans la littérature pour permettre la découverte de tels services et l'intégration d'informations contextuelles (CHEN et KOTZ 2000), (MITCHELL 2002), (KLYNE 2004), (SHENG 2005). Comme nous l'avons précisé plus haut, ce sont les modélisations à base d'ontologies qui permettent de mieux utiliser et intégrer les informations contextuelles. Or, actuellement, il n'existe que peu d'architectures qui prennent en compte les informations contextuelles dans le processus de découverte de services (GRIMM 2007). La prise en compte du contexte lors du processus de découverte permet le développement d'applications spécifiques à des environnements et des conditions donnés. De telles applications ne sont pas conçues en vue d'être flexibles et « extensibles ». Elles reposent par contre sur des infrastructures génériques permettant l'accès aux données contextuelles, leur découverte et leur publication.

Dans ce qui suit, nous présentons les principales approches proposées pour la découverte de services sensibles au contexte, reposant sur une modélisation du contexte à travers des ontologies. Ces approches seront étudiées en fonction des aspects suivants:

- ***Modélisation du contexte*** – comment sont modélisées les informations contextuelles ;
- ***Publication de service*** – quels mécanismes sont utilisés pour publier l'offre d'un fournisseur de services ;
- ***Découverte de service*** – quels mécanismes permettent à l'utilisateur de découvrir des services.

Les paragraphes suivants étudient les réponses à ces questions dans le cadre de chaque approche.

3.2.1 CONTEXT AWARE SERVICE DISCOVERY (CASD)

3.2.1.1 PRESENTATION DE L'APPROCHE

Dans (PAWAR 2006), les auteurs présentent une approche dans le cadre de laquelle les informations contextuelles sont fournies par des *sources de contexte*. Chaque service et chaque client peuvent avoir une ou plusieurs sources de contexte. Les services et les sources de contexte sont référencés dans un registre services, afin d'être découverts. La description d'un service contient des références à ses sources de contexte.

La découverte de service est réalisée par le biais du service CASD. Ce service renvoie, depuis le registre de services, une liste de services correspondant au type de service spécifié par le client dans sa requête. Les services composant cette liste sont appelés *services de base*. Ce sont des services découverts en utilisant des protocoles basiques, c'est-à-dire des protocoles qui ne prennent pas en compte les informations contextuelles.

Une fois cette liste construite, le service CASD collecte les informations contextuelles relatives aux services dans la liste, mais aussi celles qui sont relatives au client. La liste de services de base est ensuite filtrée en fonction de ces informations contextuelles, et seul le service le plus adapté est retourné au client (PAWAR 2006).

Le contexte est modélisé à travers une ontologie OWL. Les informations contextuelles des services de base sont stockées dans un graphe, appelé *graphe service*. Ce sont les sources de contexte qui transforment les informations contextuelles de l'ontologie en graphe service (semblable à un graphe

RDF – voir § 2.2.3.2.1.) Pour ce faire, une source de contexte transforme les individus de l'ontologie en nœuds du graphe service. Ces nœuds sont reliés entre eux par des arcs, représentant les propriétés reliant les individus dans l'ontologie.

3.2.1.2 Discussion de l'approche

Le Tableau 7 résume les principales caractéristiques de cette approche.

Modélisation du contexte	Le contexte est modélisé à travers une ontologie OWL. Les informations contextuelles sont ensuite extraites sous la forme d'un graphe.
Publication de service	Les services sont publiés dans un registre de services.
Découverte de service	Le service CASD est responsable du traitement de la requête client, de la constitution de la liste de services basiques, puis du tri de cette liste en fonction des informations contextuelles fournies par les sources de contexte de chaque service.

Tableau 7. Caractéristiques de l'approche CASD.

Le principal avantage de cette solution est de faciliter la conception de la découverte de services dans les environnements pervasifs. Il ne s'agit plus de rechercher les meilleurs services à chaque changement de contexte. L'utilisateur est averti, de manière dynamique, lorsque les meilleurs services correspondant à sa requête sont disponibles. Les auteurs introduisent ainsi le concept de d*écouverte persistante*. Le graphe modélisant la requête utilisateur est gardé en mémoire. A chaque changement de contexte, l'algorithme de découverte s'y réfère pour découvrir de meilleurs services.

Cette approche présente toutefois des inconvénients importants. En effet, l'algorithme de découverte de services ne renvoie qu'un seul service à l'utilisateur. De plus, si aucun service, correspondant à la fois à la requête et au contexte utilisateur, n'est disponible au moment de la requête, l'utilisateur n'obtient aucune réponse. Par contre, lorsque son contexte change, l'algorithme de découverte recherche de nouveaux services, mieux adaptés à la requête de l'utilisateur et à son nouveau contexte. La découverte du meilleur service est conditionnée par la présence de l'utilisateur dans le meilleur contexte. Or cette approche semble incompatible avec le comportement d'un utilisateur dans un environnement pervasif. Si celui-ci cherche un restaurant à l'instant *t*, et si le programme lui en propose un, il ne sera très certainement pas intéressé d'en découvrir un autre lorsqu'il sera déjà en train de déjeuner.

3.2.2 A Context- And QoS-Aware Service Discovery (ConQo)

3.2.2.1 Presentation de l'approche

Le protocole de découverte SC présenté ci-dessous est basé sur l'approche présentée dans le cadre d'un projet de recherche européen, le projet DIP (*Data, Information, and Process Integration with Semantic Web Services*). Le but de ce projet a été de développer et d'étendre les technologies du Web sémantique et des services Web, afin de produire une nouvelle infrastructure technologique pour les services Web sémantiques (DIP 2004).

Le projet a choisi de se baser sur l'architecture proposée par le modèle WSML. L'architecture ConQo emploie donc l'environnement d'exécution WSMX de l'ontologie WSMO. Ce choix est justifié par le fait que l'approche WSMO intègre un environnement nécessaire à la publication, la recherche et la

découverte de services Web. Les descriptions de services sont basées sur le langage WSML et sont référencées dans un registre de services Web.

L'approche ConQo (BRAUN 2008) utilise des ontologies pour modéliser les propriétés fonctionnelles et non-fonctionnelles des services. L'ontologie WSMO est utilisée en tant qu'ontologie service de haut niveau et elle représente la référence à utiliser pour la modélisation des sémantiques du service. D'autres ontologies, spécifiques à des domaines particuliers, sont utilisées afin d'exprimer les fonctionnalités d'un service, en termes de domaine de valeur d'un service.

L'idée de base est de considérer que seules ces informations sont primordiales dans la découverte de services. En effet, le protocole de découverte ne prend pas en compte des informations concernant l'invocation du service, par exemple. Les services sont modélisés en termes de fonctionnalités réalisées et en termes de domaines de valeur, c'est-à-dire les domaines d'application associés à chaque fonctionnalité.

Ces descriptions de service sont utilisées en tant qu'entrée lors du processus de découverte. L'idée est encore une fois de comparer la description fournie par le demandeur de service à celles publiées par les fournisseurs de services. Ces derniers disposent d'une interface leur permettant de publier et de mettre à jour les descriptions des services qu'ils offrent.

Le protocole de découverte ainsi conçu, comporte quatre phases distinctes:

a. ***Définition d'un but*** – il s'agit de rechercher dans un registre un but prédéfini (exprimé en WSMO) qui servira en tant que critère de recherche. La description de ce but pourra être parachevée par l'utilisateur. Ce processus de recherche de buts prédéfinis repose sur des techniques de recherche à base de mots-clés dans les ontologies.
b. ***Découverte de services*** – il s'agit de comparer une description abstraite de service (autrement dit un élément but WSMO) avec des descriptions de services présentes dans le registre (autrement dit un élément web service WSMO). Le processus de découverte prend en compte uniquement les fonctionnalités des services. Le protocole de découverte ne traite que des services abstraits.
c. ***Définition/sélection de service*** – après la phase précédente, un ensemble de services candidats a été construit. Dans cette phase sont pris en compte les paramètres des services, leurs interfaces. Il s'agit ici d'essayer de définir un service concret (avec une configuration concrète des paramètres), et ce parmi l'ensemble de candidats. Or, ce n'est pas parce que des services candidats ont été identifiés qu'ils peuvent réellement être délivrés à l'utilisateur final. En effet, il se peut que, parmi l'ensemble des services candidats, aucun service ne puisse définir un service concret convenant à la fois au fournisseur et au demandeur de service.
d. ***Exécution du service*** – cette phase comprend l'ensemble des étapes nécessaires à l'invocation et l'exécution du service concret sélectionné précédemment.

3.2.2.2 DISCUSSION DE L'APPROCHE

Le Tableau 8 résume les caractéristiques de cette approche.

Modélisation du contexte	Le contexte est modélisé à travers une ontologie WSMO-Context. Chaque élément contextuel est classifié en l'une des deux catégories suivantes: données mesurables et données non-mesurables. Une deuxième classification verticale définit des données statiques ou dynamiques. L'ontologie WSMO-QoS contient des concepts permettant aux fournisseurs de services de spécifier des besoins spécifiques quant à l'exécution du service (bande passante minimum, nombre de couleurs de l'écran, etc.)
Publication de service	Les services sont publiés dans un registre de services. Chaque concept de l'ontologie WSMO-Context est utilisé dans les descriptions abstraites de services, afin de formuler les offres et les besoins de services.
Découverte de service	La découverte de services est basée sur la comparaison de descriptions abstraites de services. Suite à une série de comparaisons, les services répondant à la requête de l'utilisateur sont ordonnés dans une liste. Chaque service présent dans la liste répond aux critères contextuels et QoS définis par l'utilisateur

Tableau 8. Caractéristiques de l'approche ConQo.

La principale différence par rapport aux approches basées sur l'ontologie OWL réside dans la modélisation des services en termes de buts à atteindre. De cette manière, l'utilisateur peut définir sa requête en parcourant un registre de buts ou alors en utilisant un masque de saisie permettant d'affiner cette spécification. Pour ce faire, l'approche ConQo repose sur des techniques de recherche et de navigation dans les ontologies, basées sur des mots-clés. L'idée est d'obtenir un but spécifié de manière abstraite, qui est ensuite utilisé en tant que critère de recherche pour la découverte de services.

Or, c'est dans cette utilisation de buts « abstraits » en tant que critères de recherche que réside le principal inconvénient de cette approche. Comme nous l'avons indiqué plus haut, même si une liste de services compatibles est construite à partir de la requête, rien ne garantit que ces services puissent être invoqués et exécutés. Pour l'utilisateur final, ceci représente un grand désavantage, puisque le programme lui indique l'existence de services répondant à sa requête, mais ne lui indique pas lesquels sont compatibles avec son terminal mobile.

3.2.3 Context-Aware, Ontology-based, Semantic Service Discovery (COSS)

3.2.3.1 Presentation de l'approche

Dans (BROENS 2004), l'auteur présente une architecture qui permet d'intégrer des informations contextuelles dans le processus de découverte de services sémantiques. Cette approche utilise des ontologies OWL pour modéliser les requêtes utilisateur et les services existants. Chaque service et requête sont modélisés en fonction du type de service, des sorties et des entrées du service, et en fonction des attributs du service. La description d'un service et la description d'une requête sont toutes les deux des ontologies OWL, qui doivent être comparées afin de déterminer leur degré de compatibilité (voir la Figure 21).

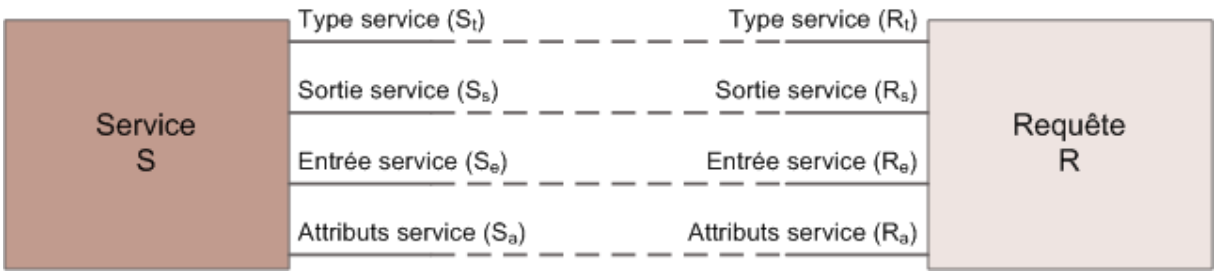

Figure 21. Comparaison des propriétés service aux propriétés requête.

Dans l'approche développée ici, l'auteur crée ses propres ontologies OWL (voir Figure 22): une ontologie modélise la structure d'un service, une autre modélise les types de services, une autre les attributs d'un service, etc. L'approche COSS repose donc sur un ensemble d'ontologies, chacune modélisant un concept participant dans le processus de découverte.

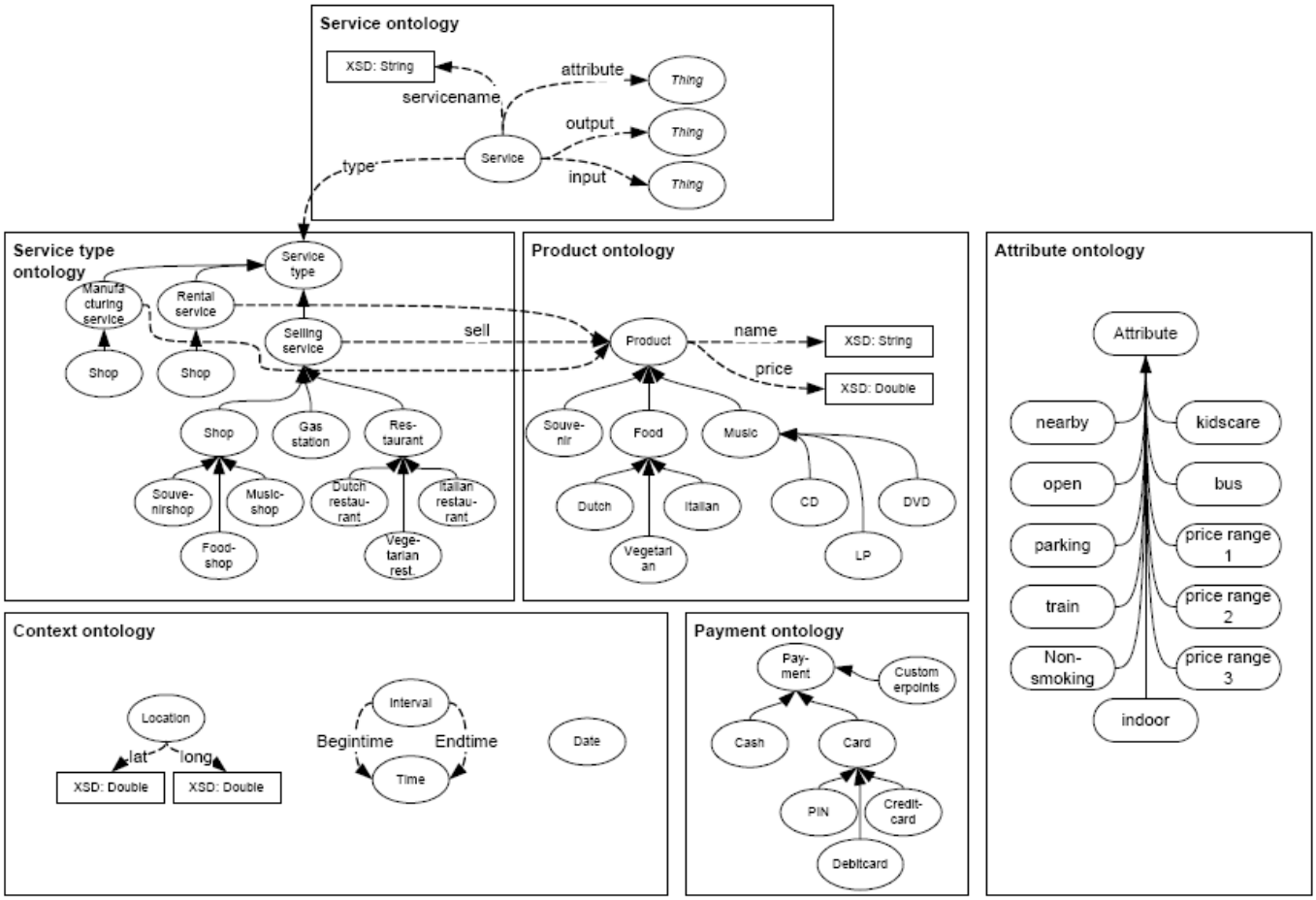

Figure 22. Ontologies utilisées par l'approche COSS (BROENS 2004).

3.2.3.2 Discussion de l'approche

Le Tableau 9 résume les caractéristiques de cette approche.

Modélisation du contexte	Le contexte est modélisé à travers des ontologies OWL, créées par l'auteur.
Publication de service	Les services sont stockés dans un registre de services.
Découverte de service	Le protocole de découverte envoie d'abord une requête MySQL au registre de services, afin de récupérer l'ensemble des services correspondant à un type de service donné. Cette requête retourne l'ensemble des descriptions OWL des services correspondants. Ces descriptions sont fusionnées en un seul modèle d'ontologie en utilisant la plate-forme Jena. Les descriptions de services contenues dans cette ontologies sont ensuite parcourues une par une, afin d'identifier les services présentant une correspondance exacte ou approximative. Les services sont stockés dans les listes de correspondance selon l'ordre dans lequel ils ont été découverts.

Tableau 9. Caractéristiques de l'approche COSS.

Le choix de créer ces ontologies « *à partir de rien* » se révèle être assez limitant. En effet, seule une dizaine d'attributs contextuels est définie et le concept de contexte utilisateur se base seulement sur les concepts de position, intervalle de temps et date. Non seulement les ontologies définissent un nombre restreint de concepts, mais en plus ces concepts sont exploités de manière binaire lors du processus de comparaison. Les ontologies créées par l'auteur permettent d'affirmer si un service est à proximité de l'utilisateur, mais il est impossible de quantifier cette proximité. Donc lorsqu'il faudra choisir entre deux services tous les deux situés à proximité de l'utilisateur, il sera impossible de déterminer lequel est le plus proche de l'utilisateur. De la même manière, cette modélisation permet d'affirmer qu'un service dispose d'un parking, mais elle ne permet pas de statuer sur la capacité du parking ou le nombre de places disponibles sur ce parking.

L'algorithme de comparaison permet seulement de différencier les correspondances exactes des correspondances approximatives. Une correspondance approximative entre un service et une requête signifie que le service ne présente pas l'ensemble des entrées de la requête. L'algorithme de comparaison renvoie alors une liste composée de deux parties. La première partie (affichée en premier) est constituée des services présentant une correspondance exacte avec la requête de l'utilisateur. La deuxième partie est constituée des services ayant des correspondances approximatives. Les services sont donc ordonnés en fonction de deux niveaux de correspondance: exacte ou approximative.

Le principal inconvénient de cette approche se situe donc au niveau des ontologies et au niveau du moteur de comparaison. Les ontologies limitent l'utilisateur lors de la formulation de sa requête. Par exemple, l'utilisateur peut demander les restaurants à proximité (selon la définition), dont le prix se situe dans un des trois intervalles de prix définis dans l'ontologie. Il ne peut pas spécifier une autre valeur ou définir une distance maximale à ne pas dépasser. Il ne peut pas non plus spécifier que pour lui, le critère primordial est la distance.

3.3 PROBLEMES NON ADRESSES PAR LES APPROCHES ACTUELLES

Dans ce chapitre, nous avons examiné les principales références présentes dans la littérature par rapport à la modélisation et la découverte des services sensibles au contexte. L'étude de ces approches nous a permis de déterminer un ensemble de caractéristiques et techniques à prendre en compte dans notre approche.

Lors de l'étude des approches permettant de décrire des services sensibles au contexte, nous avons identifié les modélisations à base d'ontologies comme étant les plus prometteuses. L'ontologie OWL-S (présentée dans 2.3.3.) étant considérée comme l'ontologie standard pour la description des propriétés et fonctionnalités des services Web, nous allons baser notre description des services sur cette modélisation. De plus, comme précisé précédemment, elle est suffisamment détaillée et générale pour permettre la description de tout service, notamment des services sensibles au contexte.

En étudiant les principales initiatives dans le domaine de la découverte de services sensibles au contexte, nous avons isolé les problèmes suivants:

- ***Les utilisateurs ont des difficultés à spécifier une requête*** – il leur est impossible de définir des critères de recherche spécifiques, ou de définir une pondération pour un critère en particulier;
- ***Les moteurs de comparaison ne permettent pas d'ordonner les services découverts*** – en fonction de leur relevance par rapport à la requête initiale, ou alors en fonction de critères spécifiés par l'utilisateur;
- ***Les éléments contextuels sont souvent interprétés de manière binaire*** – les ontologies utilisées sont souvent construites à partir de zéro, donc elles ne sont pas assez générales pour permettre une recherche précise.

Notre solution répond à ces problèmes, en proposant une approche simple et efficace qui permet une recherche plus précise. En se basant sur l'ontologie OWL-S et en l'étendant, notre solution a su tirer parti des inconvénients et des lacunes d'approches précédentes. La partie suivante présente les contributions de cette thèse, à savoir la conception d'un modèle de description de services et la définition d'un protocole de découverte tirant parti de cette description.

CHAPITRE 4.
APPROCHE CONTEXTUELLE POUR LA DECOUVERTE DE SERVICES WEB SEMANTIQUES

Dans ce chapitre, nous allons présenter en détail les contributions amenés par notre recherche, notamment concernant la description de services et son rôle dans la découverte de services. Nous allons ainsi mettre en évidence le fait que les spécifications actuelles telles le WSDL, l'UDDI ou l'OWL-S, supportent mal l'intégration d'éléments contextuels dans la recherche de services.

La partie 4.1 identifie les buts visés par les différents composants d'une description de service. Il s'agit de présenter comment ces buts sont réalisés à travers les actuels standards des services Web, tout en mettant en évidence la différence entre les composants fonctionnels et non-fonctionnels.

La section 4.2 étudie plus en détail les rôles joués par une description de services lors du processus de découverte de services. Pour ce faire, nous allons souligner les différents types de recherches pouvant être réalisés dans un registre de services, puis nous identifions les lacunes de chaque solution. Nous discutons en détail les avantages des descriptions sémantiques et des ontologies dans la découverte de services, tout en identifiant l'approche OWL-S comme étant un outil prometteur, mais pas assez détaillé.

La section 4.3 présente notre extension de l'ontologie OWL-S. Notre approche repose sur la définition d'une description sémantique des informations contextuelles associées aux services. Il s'agit de modéliser à la fois les informations dynamiques et statiques définissant le contexte d'un service.

La section 4.4 définit l'algorithme utilisé dans la découverte de services utilisant notre modèle. Cet algorithme, ainsi que le modèle de description de services défini, constituent la base de notre protocole de découverte.

4.1 ROLE DE LA DESCRIPTION DE SERVICE

Les descriptions de services visent plusieurs buts. Certains de ces buts pourvoient les humains: en effet, les descriptions de services décrivent des attributs et des fonctionnalités du service en utilisant des termes compréhensibles par des humains. D'autres buts visent l'usage des services par les machines: les descriptions de services décrivent l'interface service et facilitent l'exécution des services Web. Généralement, les descriptions de services visent à faciliter les fonctions suivantes (MARTIN 2008) : la découverte, l'invocation et la composition de services Web.

Dans ce qui suit, nous examinons les éléments composant une description de service, ainsi que le rôle qu'ils jouent dans la découverte de services. Il s'agit de différencier les attributs fonctionnels et non-fonctionnels d'un service, étudier leur rôle dans la description du service, puis de comprendre comment ces rôles sont remplis par les standards actuels.

4.1.1 COMPOSANTS FONCTIONNELS ET NON-FONCTIONNELS

Les descriptions de services contiennent des éléments fonctionnels et des éléments non-fonctionnels.

Les éléments fonctionnels comprennent les informations nécessaires à l'exécution du service: la spécification de l'interface, les protocoles associés, ainsi que les IOPEs (voir § 2.3.3.2) du service.

Les éléments non-fonctionnels comprennent des informations descriptives, qui ne relèvent pas directement de l'exécution du service. Il s'agit d'informations concernant le fournisseur du service, le coût du service, la zone géographique à laquelle le service s'applique, ainsi que des descriptions abstraites telles « *disponible en Français, Anglais et Allemand* » ou encore « *fournit une résolution GPS de 3m* ».

4.1.2 FONCTIONNALITES COUVERTES PAR LES DESCRIPTIONS STANDARD DE SERVICES WEB

La description standard de service, comprenant à la fois des éléments fonctionnels et non-fonctionnels, a plusieurs niveaux d'abstraction: le langage de base commun, les interfaces, les *protocoles métier*[26], ainsi que les propriétés et sémantiques du service. Ces niveaux sont illustrés par la Figure 23 (LAUSEN 2007) et décrits ci-dessous.

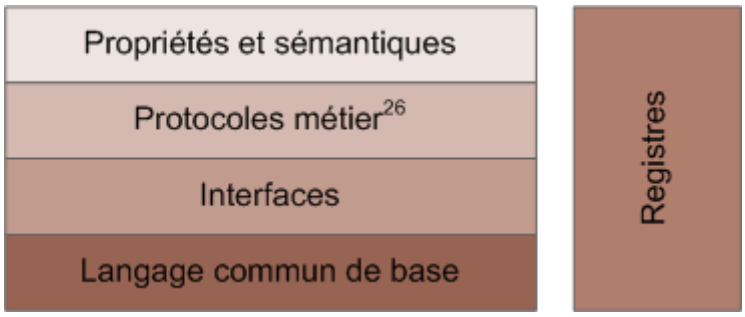

Figure 23. Niveaux d'abstraction d'une description de service.

Généralement, pour les descriptions de services, ***le langage commun de base*** est le langage XML.

[26] En anglais: "*business protocols*".

Les interfaces sont communément décrites en utilisant le langage WSDL. Ce niveau d'abstraction définit comment les utilisateurs finaux doivent interagir avec le service. Ce niveau est principalement utilisé lors de l'invocation du service. Il est aussi utile lorsqu'il s'agit d'étudier la composition des services.

La couche protocole métier est utilisée pour définir des flux de services métiers, de manière à aider la composition et la chorégraphie des services. Il s'agit de langages tels WSCL (*Web Service Conversation Language*), BPEL (*Business Process Execution Language for Web Services*) ou encore WS-CDL (*Web Services Choreography Description Language*) (KAVANTZAS 2004).

La couche « Propriétés et sémantiques » définit les propriétés non-fonctionnelles d'un service, telles le coût ou la qualité du service. Ces propriétés sont essentielles lors du processus de découverte. C'est au niveau de cette couche que notre approche va intervenir.

Les trois premiers niveaux d'abstraction (le langage commun de base, les interfaces et les protocoles métier) peuvent être vus en tant que composants fonctionnels d'une description de service. La couche « *Propriétés et sémantiques* » représente à la fois le plus haut niveau d'abstraction et le seul niveau contenant des propriétés non-fonctionnelles.

Les standards actuels des services Web définissent surtout des composants fonctionnels décrivant des interfaces. Il s'agit de standards tels SOAP (voir § 2.1.2.2.2) et WSDL (voir § 2.1.2.3), dont le but est de fournir des descriptions des mécanismes de transport de messages et des interfaces utilisées par chaque service. Toutefois, l'approche UDDI (voir § 2.1.2.5) n'est pas suffisamment flexible dans la description des composants non-fonctionnels d'un service. L'approche UDDI fournit seulement un moyen de traitement rudimentaire de ces composants, puisqu'elle ne prend en compte que des éléments tels la catégorie ou le fournisseur d'un service. De manière similaire, ni le protocole SOAP, ni le protocole WSDL ne permettent la découverte de services Web sur la seule base de leurs fonctionnalités ou capacités (OWL-S Matchmaker 2002).

Comme c'est le cas pour UDDI et WSDL, la grande majorité des composants de l'ontologie OWL-S (les ontologies `ServiceModel`, `ServiceGrounding` ou encore les IOPEs inclus dans l'ontologie `ServiceProfile`) concerne la définition des attributs fonctionnels d'un service. Toutefois, comme souligné dans la sous-section 2.3.3.1, l'ontologie `ServiceProfile` offre quelques fonctions primitives permettant une description non-fonctionnelle du service. Les quelques éléments prédéfinis (`serviceName`, `textDescription`, `contactInformation` et `serviceCategory`) offrent peu de flexibilité en plus de ce que permet l'approche UDDI.

De plus, la majorité des composants non-fonctionnels d'une description de service Web sont destinés aux utilisateurs humains. La plupart d'entre eux ne peuvent donc pas être compris par une machine. Afin de rendre la découverte de services plus efficace et plus pertinente, ces composants non-fonctionnels doivent être décrits en utilisant des sémantiques. Ceci permettra à un ordinateur de comprendre ces informations, et donc d'en tirer avantage lors du processus de découverte.

Dans notre approche, *l'ensemble des éléments non-fonctionnels décrivant un service constitue le contexte de ce service*. Pour décrire le contexte d'un service de manière compréhensible pour un ordinateur, nous utilisons le standard OWL-S. Nous nous intéressons à l'extension de ce standard, afin de fournir plus de possibilités pour décrire le contexte des services. Pour ce faire, nous utilisons les outils sémantiques fournis par les langages OWL et RDF. Dans la suite de ce chapitre, nous

montrons comment la sémantique appliquée aux composants non-fonctionnels d'un service peut améliorer la découverte du service.

4.2 ROLE DE LA DESCRIPTION DE SERVICE DANS LE PROCESSUS DE DECOUVERTE

L'un des rôles les plus importants joués par une description de service est d'être stockée dans un registre afin de permettre la découverte du service. Le processus de découverte de services peut être réalisé:

- soit au moment de l'implémentation, en parcourant le registre et en identifiant les services pertinents,
- soit au moment de l'exécution, en utilisant des techniques d'attachement dynamique de la programmation orientée-objet (Dynamic binding 2009).

Dans cette thèse, nous nous intéressons au processus de découverte de services initié par l'utilisateur et basé sur des composants non-fonctionnels décrivant le service. En effet, lors de la recherche d'un service, il est peu probable que l'utilisateur puisse spécifier sa requête en utilisant des composants fonctionnels (par exemple en spécifiant des interfaces service ou des types d'entrées du service). Au contraire, il est vraisemblable que l'utilisateur spécifie sa requête en utilisant des éléments qualitatifs, tel qu'illustré par la Figure 24.

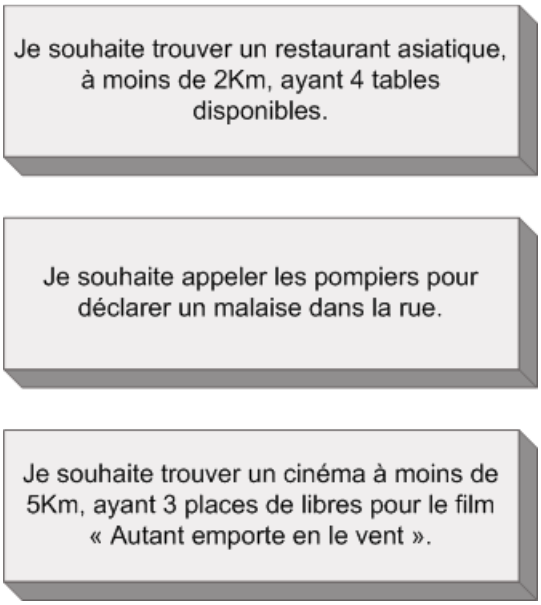

Figure 24. Exemples de requêtes qualitatives.

Les exemples de requêtes qualitatives illustrent le fait que les services Web représentent non seulement des entités logicielles, mais aussi une manière d'interagir avec des occurrences de ces entités dans le monde réel (un service d'achat de places de cinéma représente un cinéma du monde réel). Notre objectif est de proposer une application permettant de spécifier des requêtes service, tout en utilisant des éléments non-fonctionnels définissant le contexte du service. De cette façon, l'utilisateur peut définir des requêtes semblables à celles présentées par la Figure 24.

Mais avant de traiter le problème de la spécification des requêtes, nous allons discuter des différents types de recherche pouvant être effectués sur un registre de services.

La recherche à base de mots-clés représente une recherche textuelle. Il s'agit de retrouver les services contenant un ou plusieurs mots-clés. Ce type de recherche est principalement utilisé par les moteurs de recherche Internet.

La recherche de type « browsing » ou « survol » consiste à laisser l'utilisateur parcourir manuellement l'ensemble des services présents dans un registre. L'ensemble des services à parcourir peut être réduit en utilisant des tris selon des attributs donnés (par exemple en fonction du nom du fournisseur de service ou alors en fonction de la catégorie du service).

La recherche à base de couples « attribut-valeur » revient à comparer les descriptions de services entre elles, sur la base d'un ensemble de couples du type « attribut-valeur ».

La recherche à base de sémantiques est un mélange entre la recherche à base de mots-clés et la recherche à base de couples « attribut-valeur ». Ce type de recherche repose sur des règles sémantiques pour déterminer si un service est pertinent par rapport à une requête donnée.

Lorsqu'il s'agit de découverte de service, l'approche UDDI se limite à une recherche à base de mots-clés, ou encore une comparaison de descriptions de service. L'UDDI ne supporte aucun type de traitement sémantique, ce qui permettrait une comparaison flexible entre les services publiés et les requêtes de services (BALZER 2004). Comme précisé dans le Chapitre 2, la recherche dans un registre UDDI se résume au parcours d'un ensemble de descriptions de services, limité à un fournisseur de service ou à une catégorie de service donnée.

Étant donné un ensemble bien connu de catégories, ainsi qu'un ensemble limité de services, réduire la recherche à une catégorie de service peut s'avérer suffisant pour une première étape du processus de découverte. Toutefois, lorsqu'il s'agit de parcourir un grand registre, la recherche par catégorie peut se révéler soit trop limitée, soit trop étendue.

Une telle recherche s'avère être trop limitée, notamment dans le cas où plusieurs systèmes de classification des services sont utilisés. Cela fait que des services similaires peuvent être classés dans des catégories ayant des noms différents. Dans un tel cas, il est préférable de rechercher un service en fonction de ses propriétés, ou alors d'implémenter des règles pouvant relier les catégories de services entre elles. De la même manière, une recherche par catégories peut renvoyer un ensemble de services trop étendu, puisqu'une telle recherche peut retourner des services ayant la bonne catégorie, mais pas les propriétés désirées. Les résultats pertinents se perdent alors parmi les résultats qui ne le sont pas.

4.2.1 DESCRIPTION SEMANTIQUE ET DECOUVERTE DE SERVICE

Lors de la mise au point d'une description de service, le but est de la rendre suffisamment détaillée et structurée afin qu'un utilisateur n'ait pas à parcourir l'ensemble des services avant de découvrir le service désiré. Afin de permettre une découverte de service orientée utilisateur qui soit plus sophistiquée et plus flexible, nous devons trouver l'équilibre entre la flexibilité d'une recherche ad-hoc à base de mots-clés et la précision d'une recherche hautement structurée, comme c'est le cas pour les registres UDDI (voir § 2.1.2.5).

L'impact de la sémantique sur le vocabulaire des descriptions de service présente deux aspects importants, chacun affectant l'efficacité d'une recherche à base d'attributs: l'aspect limitant et l'aspect élargissant.

L'aspect limitant consiste en l'utilisation d'une ontologie commune pour le vocabulaire de la description de service. L'ontologie commune permet de définir un contexte de service (tel que défini plus haut), mais elle limite le vocabulaire aux seuls termes définis dans l'ontologie. L'usage d'une ontologie limite la recherche à base d'attributs, en éliminant les attributs hors contexte. L'usage d'URIs et de noms de domaine XML fournit les outils permettant d'atteindre l'aspect limitant. En effet, chaque concept est identifié de manière unique par une URI, et ce dans un domaine de travail donné (identifié par un nom de domaine XML).

L'aspect élargissant consiste à fournir une relation entre différents concepts du vocabulaire. Pour un concept donné, ceci permet de définir des synonymes. Cet aspect élargit la recherche à base d'attributs en incluant dans les résultats un ensemble spécifique de concepts liés entre eux (par exemple, en utilisant la relation d'inclusion entre les concepts de l'ontologie). L'implémentation de cet aspect élargissant demande des outils plus sophistiqués permettant d'exprimer des relations entre mots-clés, ou entre concepts. Il s'agit d'outils tels les langages RDF et OWL.

Afin de tirer avantage de l'expressivité du langage OWL, tout en limitant la description de service à un format standard, nous choisissons de baser notre modèle sur l'approche OWL-S. Dans ce qui suit, nous allons étudier les limitations du standard OWL-S, de manière à mettre en évidence le besoin d'une description plus détaillée.

4.2.2 LIMITES DE LA DESCRIPTION SEMANTIQUE AVEC OWL-S

Comme décrit plus haut (voir § 2.3.3), le langage OWL-S définit une ontologie pour les services Web. Etant donnée notre problématique, les deux principaux avantages de ce langage, en rapport avec celle-ci, sont (MARTIN 2008) :

- ***La découverte automatique de services Web*** – ceci est rendu possible par la prise en charge, à travers le langage OWL-S, de requêtes sémantiques.
- ***L'invocation automatique de services Web*** – ceci est rendu possible à travers l'ensemble d'APIs fourni avec le langage OWL-S.

Toutefois, l'ontologie OWL-S n'est pas assez détaillée. Elle contient trop d'ambiguïté, et manque de précision quant à certains concepts. En effet, le concept `ServiceModel` (voir § 2.3.3.2) décrit un processus, et ce processus possède des pré-conditions, des effets, des entrées et des sorties, à la manière du concept `ServiceProfile` (voir § 2.3.3.1). Les concepts `ServiceProfile` et `ServiceModel` semblent être des entités non-disjointes. En réalité, le concept `ServiceProfile` est souvent « écrit à la main », donc ne contient pas l'intégralité des fonctionnalités du service, alors que c'est le cas pour le concept `ServiceModel`. Ceci s'explique par le fait que le concept `ServiceProfile` est surtout utilisé à des fins de découverte. C'est pour cette raison que cette classe ne contient pas l'ensemble des fonctionnalités du service, mais seulement celles qui le caractérisent.

Par contre, ce manque de précision de l'OWL-S fait que cette ontologie, dans sa version originale, ne permet pas une description rigoureuse des éléments non-fonctionnels d'un service, tels sa localisation par exemple. Les principaux inconvénients du standard OWL-S sont donc les suivants:

- ***Faible degré d'axiomatisation*** – de nombreuses relations et/ou concepts pointent vers la classe `owl:Thing` (voir § 2.2.3.2.2).

- ***Trop d'ambiguïté et manque de rigueur*** dans la définition de certains de ses concepts – en particulier, une approche plus structurée est nécessaire pour la conception et l'organisation des ontologies reposant sur le modèle OWL-S. Afin de permettre un meilleur traitement et raisonnement automatiques, des liens doivent être définis entre l'ontologie OWL-S et les ontologies de base (par exemple l'ontologie DOLCE (DOLCE 2006)). Ainsi, à la fois les propriétés fonctionnelles et non-fonctionnelles d'un service pourront être décrites avec plus de rigueur.
- ***Les services ne sont pas associés à une zone géographique donnée***, et peuvent être invoqués de n'importe quel endroit – il n'existe aucun concept dans l'ontologie OWL-S permettant de définir une « *zone d'action* » ou une « *zone de valabilité* » pour un service. Ajouter ce type d'information permet de répondre à des requêtes spécifiant une zone géographique, comme par exemple « *Lister l'ensemble des restaurants italiens à moins de 2 Km* ».
- ***Pas de description sémantique d'informations décrivant le contexte du service*** – seule la propriété `serviceParameters` permet de définir une liste expansible de propriétés pouvant accompagner une description de service.

Pour palier à ces lacunes, notre approche est d'étendre l'ontologie OWL-S afin de tirer parti des informations contextuelles d'un service lors du processus de découverte de services. Notre approche consiste à:

a. Définir une hiérarchie de types de services;
b. Définir une description sémantique d'attributs contextuels pour chaque catégorie de services;
c. Développer une interface permettant de renseigner des valeurs pour ces attributs contextuels ou encore de définir de nouveaux attributs;
d. Développer une interface permettant de spécifier une requête tout en utilisant les attributs contextuels définis dans notre modèle;
e. Définir un algorithme permettant d'utiliser les attributs contextuels des services pour calculer un score de pertinence du service par rapport à la requête de l'utilisateur.

Dans ce qui suit, nous présentons notre modèle de description de services, autrement dit les améliorations apportées à l'ontologie OWL-S. Ensuite, nous étudions les avantages de ce modèle en rapport avec le processus de découverte de services.

4.3 MODELE DE DESCRIPTION DE SERVICES SENSIBLES AU CONTEXTE AVEC OWL-S ETENDU

Comme nous l'avons précisé plus haut, l'ontologie OWL-S comporte plusieurs lacunes qui ne la rendent pas adaptée à la prise en compte, lors du processus de découverte, d'informations contextuelles caractérisant le service. En vue de rendre ceci possible, notre approche se décompose en deux phases.

Dans un premier temps, il s'agit de définir une hiérarchie de types de services. Ceci permet d'avoir des catégories de services bien distinctes, chacune étant caractérisée par des propriétés différentes. Dans un deuxième temps, il s'agit de spécifier les caractéristiques ou attributs contextuels pour chaque catégorie de services. Il s'agit ensuite de spécifier comment le modèle ainsi défini et construit va être utilisé lors du processus de découverte de services.

4.3.1 CREATION D'UNE HIERARCHIE DE SERVICES

4.3.1.1 CLASSIFICATIONS STANDARD DES SERVICES

Comme nous l'avons expliqué plus haut, le profil d'un service est utilisé pour le caractériser à des fins de publication, découverte et sélection. Les profils de services peuvent être publiés dans différents types de registres, découverts en utilisant différents outils et sélectionnés grâce à de multiples techniques de comparaison. L'ontologie OWL-S ne limite ou n'impose pas la manière d'utiliser les profils de service, mais cherche à fournir une base pour leur construction, une base qui soit suffisamment flexible pour s'adapter aux divers contextes et méthodes d'usage de ces profils de service.

Généralement, une telle caractérisation de service doit situer le service parmi le vaste ensemble de services disponibles, que ce soit dans un domaine donné ou pas. La construction d'une hiérarchie de classes, avec un héritage de propriétés entre classes, est une des techniques les plus évidentes qui permette ce type de positionnement des services. A l'origine, cette technique est utilisée dans la conception et la programmation orientée-objet, mais les langages à base de logique descriptive permettent aussi de l'appliquer. Le but est de construire une catégorisation de services semblable à celle des annuaires de type « *Pages Jaunes* », mais dont la structure serait plus formelle et supporterait des requêtes plus complexes.

Nous cherchons donc une caractérisation de service permettant de situer le service parmi le vaste ensemble de services disponibles, que ce soit dans un domaine donné ou pas. Plusieurs standards de classification des services, ou des taxonomies de services, existent aujourd'hui. Parmi elles, nous pouvons citer:

- ***Standard Industrial Classification (SIC)*** - il s'agit de la classification normalisée des industries, établie par le gouvernement des États-Unis dans les années 1930. Ce modèle utilise cinq chiffres pour identifier un service (SIC 2008) : les deux premiers chiffres définissent la catégorie principale du service, le troisième chiffre définit la catégorie industrielle, le quatrième chiffre le groupe de services, et finalement le cinquième chiffre identifie le service lui-même. Aujourd'hui, cette classification a été remplacée par le modèle ***NAICS (North American Industry Classification System)*** établi en 1997. Similaire au modèle SIC, le modèle NAICS est basé sur une classification des services en utilisant 6 chiffres. La classification NAICS est utilisée dans le standard OWL-S.
- La ***Nomenclature Statistique des Activités Économiques (NACE)*** est l'équivalent européen du système NAICS. Ce système utilise un code formé d'une lettre, d'un nombre et plusieurs chiffres (NACE 2009).
- La classification ***United Nations standard Products and Services Code (UNSPSC)*** (UNSPSC 2009) permet de classifier à la fois les produits et les services, en visant une utilisation dans les systèmes de commerce électronique. Créée en 1998, ce système propose des codes disponibles en plusieurs langues, dont le français. Cette classification qui est utilisée par le standard OWL-S.

Il existe donc plusieurs modèles de classification et le standard OWL-S utilise deux d'entre eux. Nous allons brièvement expliquer comment ceci est réalisé et quelles ont été les raisons motivant notre choix de créer une autre classification des services.

4.3.1.2 CLASSIFICATIONS UTILISEES PAR LE STANDARD OWL-S

Dans le standard OWL-S, c'est la classe `ServiceCategory` qui permet de décrire des catégories de services sur la base d'une classification standard, telles celles listées plus haut. La propriété objet (voir 2.2.3.2.2.3) `serviceCategory` relie la classe `ServiceProfile` à la classe `ServiceCategory` (la valeur de la propriété `serviceCategory` est une instance de la classe `ServiceCategory`). Une catégorie de services ainsi définie dispose des propriétés « données » suivantes, telles qu'illustrées par la Figure 25 (MARTIN 2004):

- `categoryName` – cette propriété permet de définir le nom de la catégorie, qui peut être soit une chaîne de caractères soit une URI;
- `taxonomy` – cette propriété définit une référence vers un schéma de taxonomie. Il peut s'agir soit de l'URI de la taxonomie, soit d'une URL vers cette taxonomie, ou encore du nom de la taxonomie;
- `value` – cette propriété pointe vers une valeur dans une taxonomie donnée;
- `code` – cette propriété définit le code d'une catégorie de service, par rapport à une taxonomie donnée.

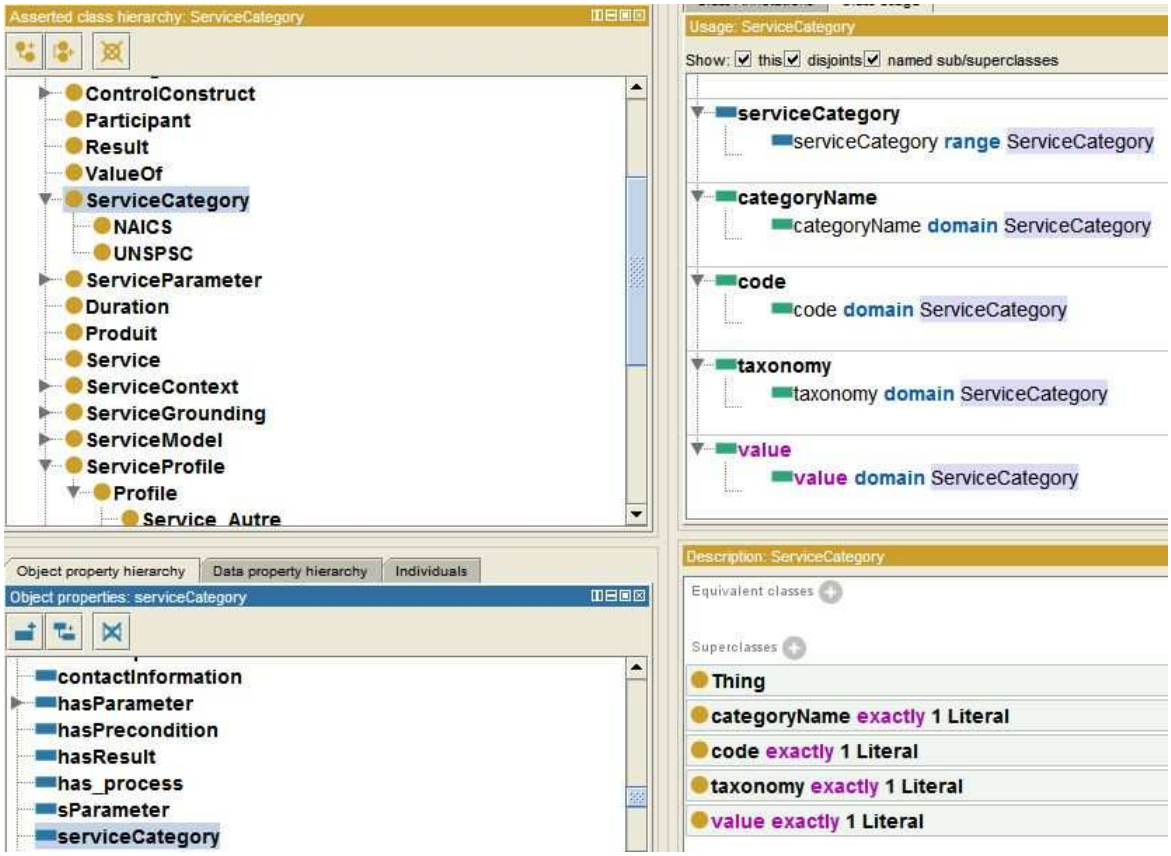

Figure 25. Illustration des propriétés de la classe `ServiceCategory`.

Par exemple, un service de collection des déchets peut être classé selon les codes NACE, en spécifiant les valeurs suivantes pour les propriétés listées ci-dessus:

```
taxonomy = "http://ec.europa.eu/competition/mergers/cases/index/nace_all.html"
value = "Treatment and disposal of non-hazardous waste"
code = "E38.2.1"
```

Le fait d'affecter une valeur aux propriétés listées ci-dessus, crée une instance de la classe `ServiceCategory`, autrement dit cela définit une catégorie de services. Cependant, le standard OWL-S utilise principalement les classifications NAICS et UNSPSC. La classe `ServiceCategory` contient deux sous-classes `"NAICS"` et `"UNSPSC"`, définies par une instanciation des propriétés listées ci- dessus, comme indiqué sur la Figure 26 et la Figure 27.

De plus, deux autres propriétés définies par le standard OWL-S, `serviceClassification` et `serviceProduct`, peuvent être utilisées afin de spécifier le type d'un service et du produit concerné. Ces propriétés sont similaires aux propriétés décrites plus haut, mais elles différent au niveau des valeurs possibles. En effet, les valeurs des ces propriétés sont des instances OWL, au lieu d'être des chaînes de caractères faisant référence à des taxonomies non-OWL (MARTIN 2004) :

- La propriété `serviceClassification` définit un lien entre un profil de service (`Profile`) et une ontologie des services OWL, telle la spécification OWL de la classification NAICS.
- La propriété `serviceProduct` définit un lien entre un profil de service (`Profile`) et une ontologie de produits OWL, telle la spécification OWL de la classification UNSPSC.

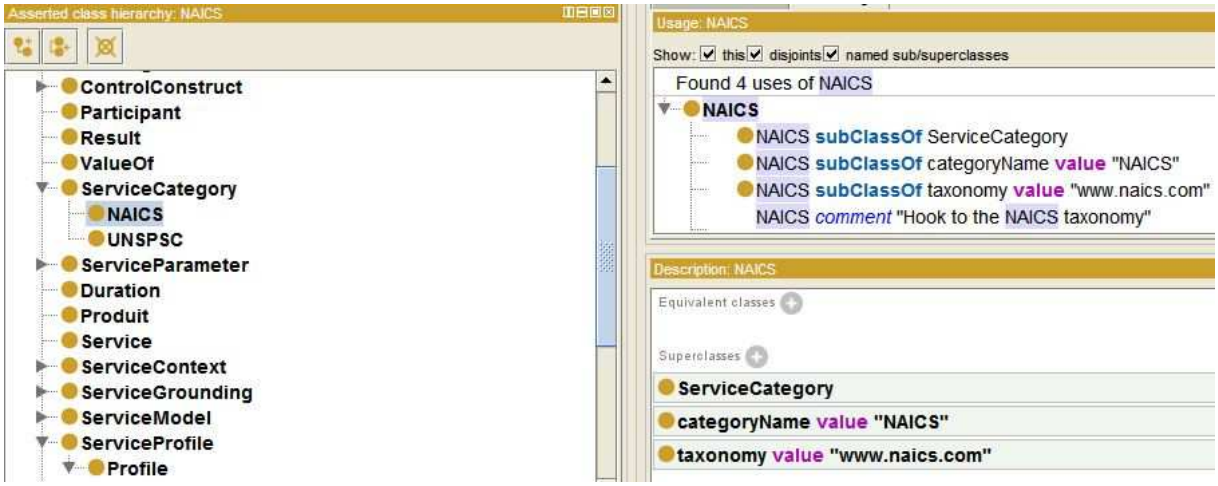

Figure 26. Classe NAICS du modèle OWL-S.

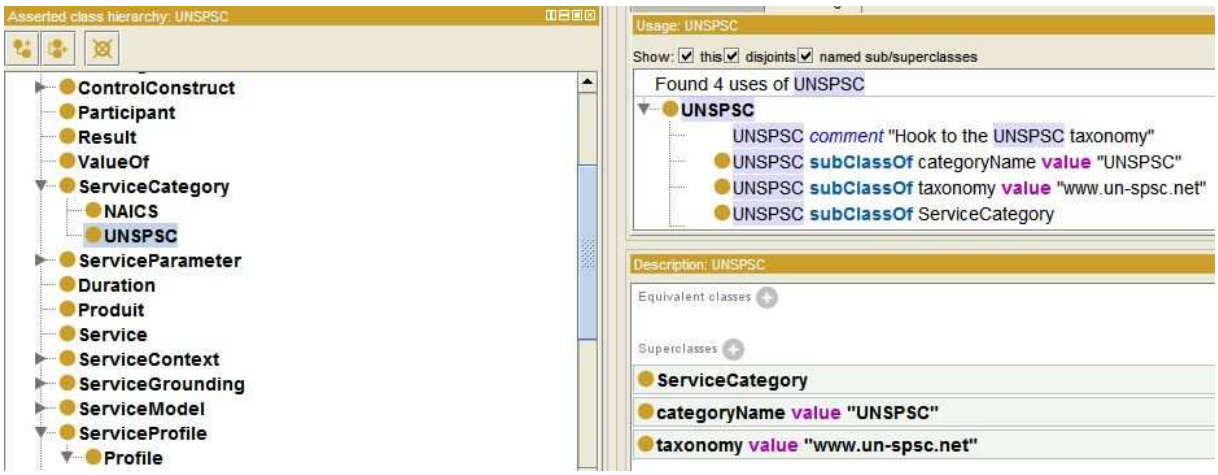

Figure 27. Classe UNSPSC du modèle OWL-S.

L'ensemble de ces classes et propriétés permettent en effet de classifier des services et produits existants, mais cette classification ne peut pas être utilisée lors du processus de découverte. En effet,

le parcours d'une classification aussi complexe que le système NAICS ou UNSPSC se révèle impossible, et ce pour plusieurs raisons:

- ***Le nombre de codes à parcourir est énorme*** - Un utilisateur mobile disposant d'un temps limité, ainsi que d'un terminal mobile avec une capacité de traitement limitée, perdra patience avant d'avoir trouvé le service recherché.
- ***Les classifications ne sont pas intuitives pour un utilisateur non-averti*** - ces systèmes définissent de nombreuses catégories dont le nom n'est pas toujours évocateur pour un utilisateur quelconque.
- ***Les classifications concernent surtout des services de l'industrie*** - la grande majorité des catégories définies par les systèmes cités plus haut, concernent le monde de l'industrie et de l'agriculture. Il est peu probable que des catégories telles « *l'industrie du poisson* » (code NACE `152Z`) ou « *l'extraction d'ardoise* » (code NACE `141E`) représentent des catégories de services fréquemment recherchées par les utilisateurs.

A partir de ces quelques remarques, nous pouvons conclure quant à la non-adéquation de ces systèmes de classification par rapport à notre but initial, à savoir classer les services afin de faciliter leur découverte. Dans ce qui suit, nous présentons notre approche et ses avantages par rapport aux solutions existantes.

4.3.1.3 NOTRE APPROCHE

Généralement, une telle caractérisation de service doit situer le service parmi le vaste ensemble de services disponibles, que ce soit dans un domaine donné ou pas. Nous cherchons donc à identifier une manière de caractériser les services, afin de pouvoir les découvrir parmi l'ensemble des services disponibles. Notre approche est de nous baser sur une catégorisation de services semblable à celle des annuaires de type « *Pages Jaunes* », mais dont la structure serait plus formelle et supporterait des requêtes plus complexes. La construction d'une hiérarchie de classes, avec un héritage de propriétés entre classes, est une des techniques les plus évidentes qui permette ce type de positionnement des services. A l'origine, cette technique est utilisée dans la conception et la programmation orientée-objet, mais les langages à base de logique descriptive permettent aussi de l'appliquer.

L'idée de définir une hiérarchie de types de services dans le standard OWL-S a été une initiative de la *Coalition OWL-S* (MARTIN 2008), un groupe de 14 chercheurs internationaux, travaillant dans le domaine du Web sémantique. A partir de l'ontologie de base OWL-S, ils ont étendu la classe `ServiceProfile`, en créant une hiérarchie de catégories de services. Le but recherché était de permettre la spécification d'informations non-fonctionnelles caractérisant les services, informations qui s'avèrent utiles lors du processus de découverte. Cette initiative a débouché sur la création d'une ontologie, appelée `ProfileHierarchy.owl` et disponible en ligne[27]. Comme le précisent les auteurs, cet exemple est volontairement simpliste et limité. Son but est juste d'illustrer l'héritage de propriétés entre classes, ainsi que la catégorisation de deux exemples de services de commerce en ligne (*e-commerce*).

[27] Voir http://www.ai.sri.com/daml/services/owl-s/1.2/ProfileHierarchy.owl

En se basant sur cet exemple, nous avons défini une classification hiérarchique de profils de services plus avancée et plus détaillée. Cette classification est illustrée par la Figure 28. Les catégories définies sont les suivantes:

- ***Service ECommerce*** – cette catégorie regroupe l'ensemble des services de vente, dans le sens général du terme. Les informations contextuelles d'un service E-Commerce sont la marchandise concernée (ou le produit à vendre) et le type de commerce. Les différents types de commerce envisagés donnent naissance à des sous-catégories de services, chacune caractérisant un type de commerce. Ces sous-catégories définissent des services plus spécifiques.
  - ***Service Achat*** – le contexte d'un tel service est définit par des attributs concernant le produit à vendre (coût produit, garantie produit) et la livraison de ce produit (mode de livraison, coût de livraison, durée de livraison et pays concernés par la livraison). Il n'est pas nécessaire de spécifier ici le type de produit concerné. En effet, cette information est déjà dans les attributs d'un service ECommerce, et le service Achat hérite de cette propriété.
  - ***Service Location*** – le contexte d'un tel service est défini par des attributs concernant la durée de la location et les conditions de location. Il n'est pas nécessaire de spécifier ici le type de produit concerné. En effet, cette information est déjà dans les attributs d'un service ECommerce, et le service Location hérite de cette propriété.
  - ***Service Vente aux Enchères*** – le contexte d'un tel service est définit par des attributs concernant le type de vente aux enchères (enchère ascendante ou enchère descendante), le nombre de participants à la vente, le prix de départ du produit, le prix limite et l'enchère en cours.
- ***Service Information*** – cette catégorie regroupe les services informatifs. Les informations contextuelles d'un tel service sont la date, la source et le sujet de l'information. Il s'agit aussi de la catégorie de l'information (nationale, internationale, sciences, sport ou météo) et du type de l'information (de dernière minute, communiqué de presse, etc.)
- ***Service Urgence*** – cette catégorie regroupe l'ensemble des services en lien avec des situations d'urgence. Il s'agit principalement des services hospitaliers, ainsi que des services de pompiers et de police. Ces services doivent permettre à l'utilisateur de remonter rapidement des informations importantes pour les secours. Les informations contextuelles définissant cette catégorie de services sont la position de l'utilisateur et le niveau d'urgence.
  - ***Service Police*** – les informations contextuelles définissant un service police concernent le type d'infraction/agression dont l'utilisateur a été victime. L'idée est de permettre à l'utilisateur de pouvoir choisir dans une liste d'infractions celle dont il a été victime. Il se peut aussi que l'utilisateur ne puisse pas préciser le type d'infraction dont il a été victime. il s'agit pour lui alors de définir la catégorie générale. La classification des infractions et délits a été construite à partir de celle fournie par Jeremy Bentham, dans son « *Traité de législation civile et pénale* » (BENTHAM 2009). Dans ce document, plusieurs niveaux de classification sont définis. Dans notre approche, nous avons volontairement réduit ces niveaux, en n'en gardant que trois. Il s'agit des catégories suivantes: « *Crimes et délits contre la personne* », « *Vols/recels* » et « *Autres infractions* ». Chaque catégorie a un niveau d'urgence

associé et contient plusieurs types d'infractions et/ou délits. La Tableau 10 illustre notre classification.

Niveau Urgence	Nom Niveau Urgence	Description Urgence
1	Autres infractions	Infractions liées aux stupéfiants
1	Autres infractions	Destruction/dégradation de biens
1	Autres infractions	Délits divers
2	Vols / recels	Vol à main armée
2	Vols / recels	Vol avec violence sans arme à feu
2	Vols / recels	Vol avec entrée par ruse
2	Vols / recels	Cambriolage
2	Vols / recels	Vol lié à l'automobile
2	Vols / recels	Vol lié aux deux roues à moteur
2	Vols / recels	Vol à la tire
2	Vols / recels	Vol à l'étalage
2	Vols / recels	Vol sur chantier
3	Crimes et délits contre la personne	Homicide
3	Crimes et délits contre la personne	Tentative d'homicide
3	Crimes et délits contre la personne	Coups et blessures volontaires
3	Crimes et délits contre la personne	Séquestration
3	Crimes et délits contre la personne	Violation de domicile
3	Crimes et délits contre la personne	Atteinte aux mœurs
3	Crimes et délits contre la personne	Infractions contre la famille et l'enfant

Tableau 10. Classification des urgences de police.

- ***Service Hôpital*** – de la même manière, nous avons classé en trois catégories les différents « maux » dont un patient peu souffrir. Étant donnée la diversité de ces « maux », les catégories portent le nom du niveau d'urgence. Il s'agit des catégories « *normal ou assez élevé* », « *élevé* » et « *très élevé* ». Cette classification se base sur la classification clinique des malades des urgences en France (CCMU 2009). Sans être aussi exhaustive, elle a pour but d'informer les secours sur la nature du problème dont souffre le patient. Le Tableau 11 présente notre classification plus en détail.

Niveau Urgence	Nom Niveau Urgence	Description Urgence
1	Normal	Intoxication médicamenteuse
1	Normal	Intoxication éthylique
1	Assez Elevé	Angine
1	Assez Elevé	Gastro-entérite simple

1	Assez Elevé	Otite
1	Assez Elevé	Plaie sans suture
1	Assez Elevé	Piqure d'insecte
1	Assez Elevé	Accouchement
2	Elevé Médical	Lombo-sciatique
2	Elevé Médical	Broncho-pneumopathie
2	Elevé Chirurgical	Plaie à suturer
2	Elevé Chirurgical	Entorse
2	Elevé Chirurgical	Fracture fermée
2	Elevé Chirurgical	Luxation
2	Elevé Chirurgical	Fracture des côtes
2	Elevé Chirurgical	Brûlure 2ème degré
3	Très Elevé Médical	Douleur thoracique
3	Très Elevé Médical	Crise d'asthme
3	Très Elevé Chirurgical	Fracture ouverte

Tableau 11. Classification des urgences hôpital.

- ***Service Pompiers*** – à l'image des deux services définis précédemment, les informations contextuelles du service visent à aider les pompiers et secouristes dans leur intervention. Nous avons donc classé les différentes interventions des sapeurs-pompiers en trois catégories d'urgence (listées ici par niveau d'urgence décroissant): « *Secours à la personne* », « *Incendies* » et « *Opérations diverses* ». Cette classification se base sur l'édition 2008 des statistiques des services d'incendie et de secours (Pompiers 2009). Le Tableau 12 présente le détail de cette classification.

Niveau Urgence	*Nom Niveau Urgence*	*Description Urgence*
1	Opérations diverses	Faits d'animaux
1	Opérations diverses	Fuite d'eau
1	Opérations diverses	Inondation
1	Opérations diverses	Fuite/odeur de gaz
2	Incendies	Feux d'habitations
2	Incendies	Feu de véhicules
2	Incendies	Feux de végétations
2	Incendies	Feux sur voie publique
3	Secours à la personne	Malaise / maladie à domicile
3	Secours à la personne	Accident sur voie / lieux publics
3	Secours à la personne	Malaise sur voie publique
3	Secours à la personne	Accident à domicile
3	Secours à la personne	Accident lieu de travail
3	Secours à la personne	Accidents de circulation

Tableau 12. Classification des urgences pompiers.

- ***Service Personnel*** – cette catégorie regroupe l'ensemble des services concernant l'individu. Nous y avons classé les services permettant de trouver des restaurants, des hôtels, des garages ou encore des centres de lavage. Les informations contextuelles sont différentes pour chaque service. Nous allons détailler seulement les informations caractérisant le service restaurant.
    - ***Service Restaurant*** – ce type de service est caractérisé par les informations contextuelles suivantes: la catégorie du restaurant (brasserie, cafétéria, traditionnel, etc.), le style de cuisine (cuisine africaine, européenne, asiatique, etc.), le prix du repas, le nombre de places parking disponibles et le nombre de tables disponibles. Un tel service est aussi caractérisé par la présence ou non d'un accès handicapés ou d'une climatisation.
    - ***Service Divertissement*** – cette catégorie regroupe l'ensemble des services permettant à l'utilisateur de se divertir. Nous y avons classé les services permettant de trouver des salles de cinéma, de théâtre ou de concert. Les informations contextuelles de ces services concernent le type du spectacle (film, pièce de théâtre, concert, exposition, etc.), l'heure de début et l'heure de fin du spectacle, le nombre de places disponibles dans la salle, le nombre de places disponibles dans le parking,
    - ***Service Transport*** – cette catégorie regroupe l'ensemble des services permettant à l'utilisateur de se déplacer d'un point A vers un point B. Les informations contextuelles caractérisant ces services sont le point de départ, la destination, le prix du voyage, et la durée du voyage. D'autres informations contextuelles sont définies pour chaque type de service.

Ceci constitue donc notre hiérarchie de types de services (Figure 28).

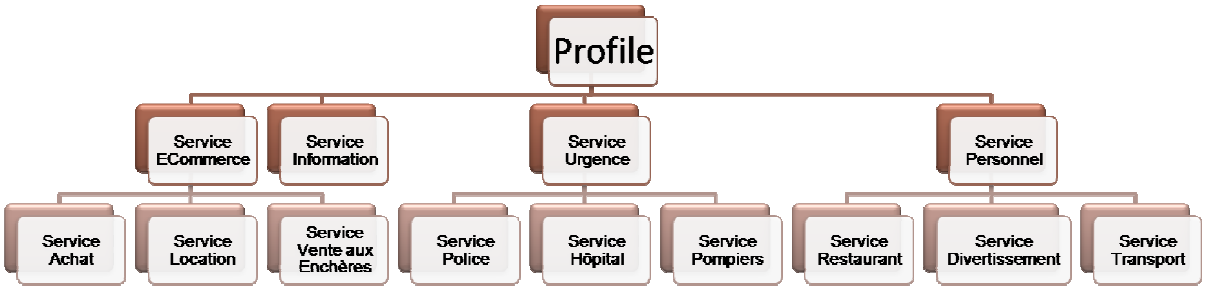

Figure 28. Notre classification hiérarchique de profils service.

Elle n'est volontairement pas exhaustive, car cela reviendrait à construire une taxonomie de services, ce qui n'est pas le but de cette thèse. Cette classification peut être facilement étendue à d'autres domaines et/ou services. En effet, cette hiérarchie est définie sur le principe d'une ontologie, donc il est aisé d'y inclure des références ou des liens vers d'ontologies classifiant les services. Dans la suite, nous présentons les outils permettant de renseigner des services par rapport à cette hiérarchie, mais aussi d'y ajouter des catégories services qui n'existent pas encore. Mais avant cela, nous allons étudier la construction de cette hiérarchie par rapport au standard OWL-S.

4.3.2 Integration dans le modele OWL-S

Nous avons mis en évidence les lacunes du modèle OWL-S quant à la prise en compte d'informations contextuelles caractérisant un service donné (voir 4.3.1.2). Nous avons aussi présenté notre

proposition pour une hiérarchie de profils service, chacun étant caractérisé par différentes informations contextuelles (voir 4.3.1.3). Dans ce qui suit, nous allons détailler l'intégration de notre approche dans le modèle OWL-S. Pour ce faire, deux étapes sont nécessaires:

Dans un premier temps, il s'agit de créer les classes correspondant à notre hiérarchie. Ces classes sont créées en tant que sous-classes de la classe `ServiceProfile`, puisque chacune d'entre elles définit un profil de service. Ensuite, il est nécessaire de définir les informations contextuelles associées à chaque profil de service. Nous détaillons la réalisation de ces deux étapes dans les paragraphes suivants.

4.3.2.1 EXTENSION DE LA CLASSE SERVICEPROFILE

L'extension de la classe `ServiceProfile` revient à y créer des sous-classes, chacune d'entre elles correspondant à un type de profil de service défini au paragraphe 4.3.1.3. Pour ce faire, nous utilisons l'éditeur Protégé (Protégé 2009) et sa fonction permettant de créer des hiérarchies de classes. La Figure 29 illustre l'interface proposée par cet éditeur.

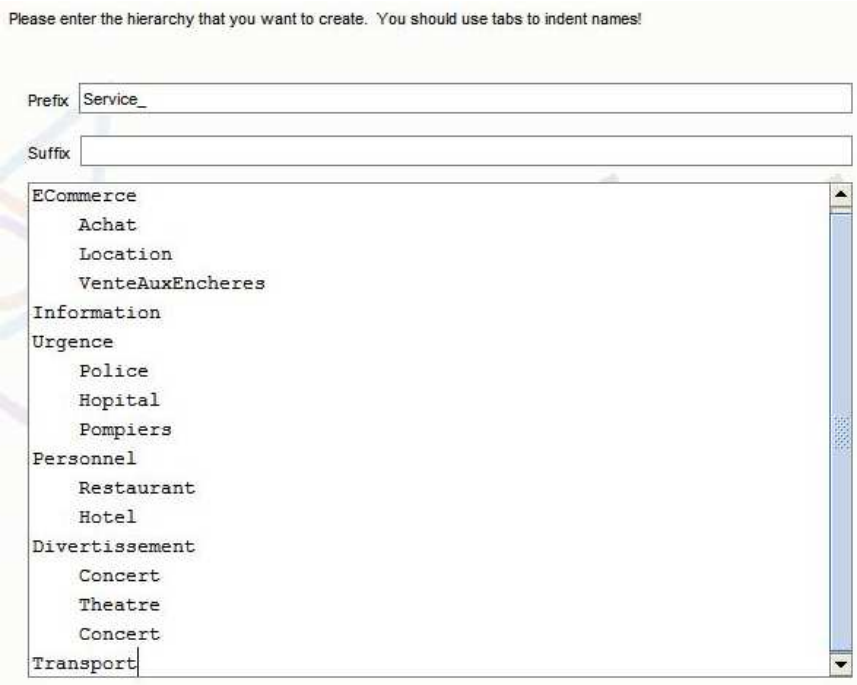

Figure 29. Interface de l'éditeur Protégé pour la création de hiérarchies de classes.

Il s'agit d'entrer le nom des différentes classes, en utilisant des tabulations pour définir des relations d'inclusion entre classes. La fonction « *préfixe* » permet de définir un préfixe pour l'ensemble des classes ainsi créées. Les caractères renseignés dans le champ `"Prefix"` vont précéder le nom de chaque classe. L'ensemble des classes ainsi créées sont définies comme étant disjointes. Cela veut dire que l'intersection entre deux classes prises au hasard est vide. En d'autres termes, un service ne peut appartenir qu'à un type de profil de service.

Une fois cette hiérarchie de classes créée, nous définissons les deux propriétés suivantes pour la classe `Profile` (classe contenant l'ensemble des classes listées plus haut):

- `aAdresse` – chaque service est associé à une adresse dans le monde réel, du type *Numéro, Rue, Ville, Code Postal, Pays*. La propriété `aAdresse` est une propriété de type « données », qui n'accepte que des chaînes de caractères (voir Figure 30).

```
<owl:DatatypeProperty rdf:about="#aAdresse">
    <rdfs:comment xml:lang="fr">Cette propriete permet de definir l'adresse d'un
service.</rdfs:comment>
    <rdfs:domain rdf:resource="&Service;ServiceProfile"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

Figure 30. Définition de la propriété `aAdresse`.

- `aTelephone` – chaque service est associé à un numéro de téléphone. De la même manière, la propriété `aTelephone` est une propriété de type « données », qui n'accepte que des chaînes de caractères (voir Figure 31).

```
<owl:DatatypeProperty rdf:about="#aTelephone">
    <rdfs:comment xml:lang="fr">Cette propriete permet de definir le numero de
telephone d'un service.</rdfs:comment>
    <rdfs:domain rdf:resource="&Service;ServiceProfile"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

Figure 31. Définition de la propriété `aTelephone`.

La Figure 32 illustre l'extension de la classe `ServiceProfile`. Nous y remarquons des catégories de services supplémentaires (par rapport à celles illustrées par la Figure 28). Ceci prouve la flexibilité du modèle et la facilité avec laquelle de nouvelles catégories de services peuvent être ajoutées à notre hiérarchie.

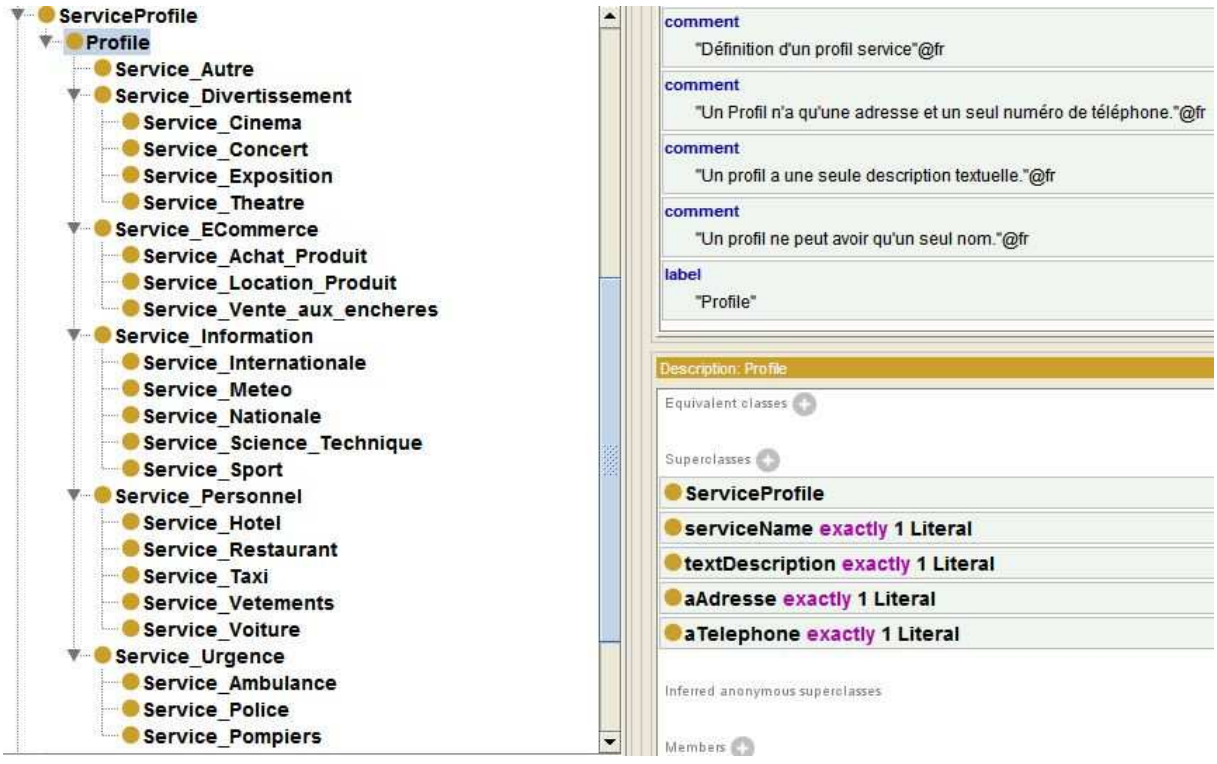

Figure 32. Notre extension de la classe `Profile`.

4.3.2.2 CREATION DE LA CLASSE SERVICECONTEXT

Une fois notre hiérarchie créée, il s'agit de définir, pour l'ensemble des classes la constituant, les attributs contextuels qui les caractérisent. Plusieurs options s'offrent à nous:

- Nous pouvons définir chaque attribut contextuel en tant que propriété « données » pour chaque profil de service.
- Nous pouvons définir ces attributs en tant qu'éléments d'une classe à part.

Nous avons choisi la deuxième option, et ce pour les raisons suivantes:

- Nous avons déjà étendu la classe `Profile`, en y créant notre hiérarchie de profils. Nous ne voulons pas « charger » encore cette classe.
- Nous souhaitons pouvoir utiliser les attributs contextuels des services lors du processus de découverte des services. En outre, nous souhaitons pouvoir manipuler ces attributs, notamment leur associer des pondérations (indiquant leur importance dans la recherche). Or, ceci est impossible si ces attributs sont définis en tant que propriétés « données » OWL. En effet, OWL ne permet pas de définir des propriétés pour des propriétés.

Les attributs contextuels d'un service sont donc définis dans une classe à part, que nous appelons `ServiceContext`.

```
<owl:Class rdf:about="&Service;ServiceContext">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>
```

Cette nouvelle classe contient les attributs contextuels associés aux services, classés par classes, chaque classe correspondant à un profil de service. Pour ce faire, nous utilisons à nouveau l'outil Protégé permettant de créer une hiérarchie de classes. Nous obtenons une hiérarchie similaire à celle créée précédemment, tel qu'illustré par la Figure 33.

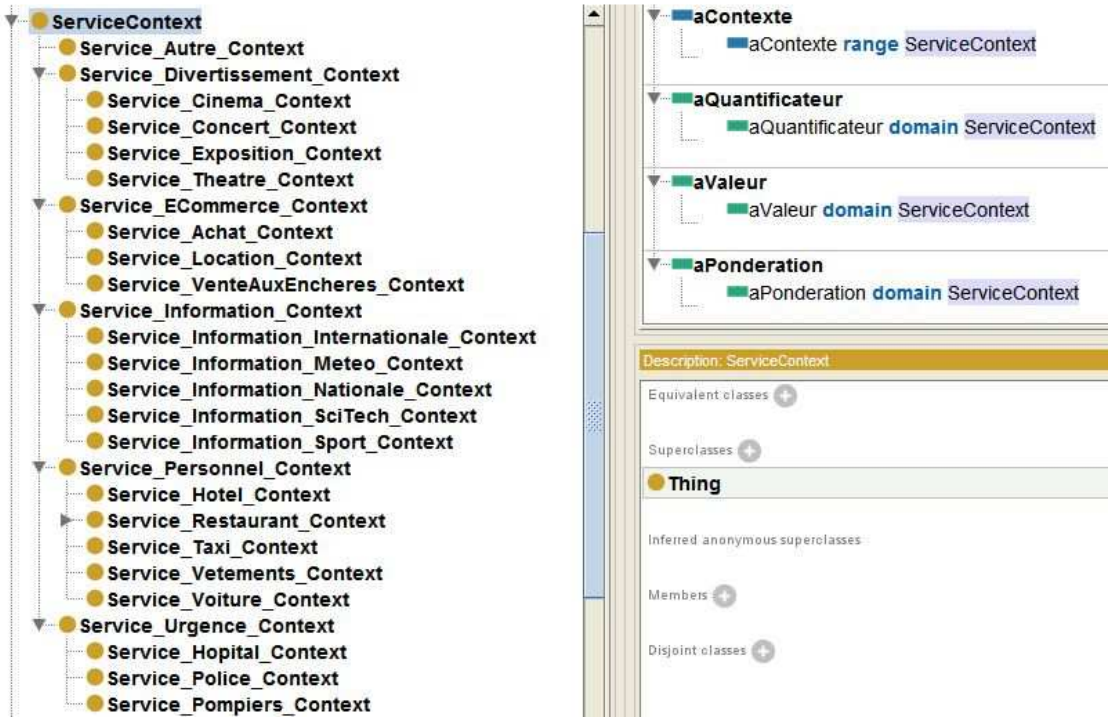

Figure 33. Contenu de la classe `ServiceContext`.

Cette nouvelle classe, `ServiceContext`, est reliée à la classe `ServiceProfile` par la propriété `aContexte`. Cette propriété objet permet d'associer une ou plusieurs informations contextuelles à un profil de service. Les restrictions OWL permettent de limiter les valeurs de cette propriété à une sous-classe donnée de la classe `ServiceContext`. Une restriction de classe est définie en OWL en utilisant le mot-clé `owl:onProperty` (MCGUINESS 2004). Ainsi on spécifie quelle propriété doit être utilisée dans la définition de la restriction de classe. L'appartenance à une restriction de classe est soumise au respect des conditions déterminées par le type de restriction (`owl:allValuesFrom`, `owl:someValuesFrom` ou `owl:hasValue`) et la spécification `onProperty`. Dans notre cas, nous utilisons la restriction `owl:allValuesFrom`, qui produit des restrictions de la forme « *l'ensemble des individus pour lesquels l'ensemble des valeurs de la propriété P appartiennent à la classe C* ». Une telle restriction est définie comme suit (MCGUINESS 2004) :

```
[ a owl:Restriction;
    owl:onProperty P;
    owl:allValuesFrom C]
```

Nous utilisons donc ce type de restriction pour affirmer que l'ensemble des services de type restaurant ont des attributs contextuels appartenant à la classe `Service_Restaurant_Context` (voir Figure 34). Nous appliquons ce type de restriction pour l'ensemble des profils service définis.

```
<owl:Class rdf:about="&Service;Service_Restaurant">
    <rdfs:subClassOf rdf:resource="&Service;Service_Personnel"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&Service;aContexte"/>
            <owl:allValuesFrom rdf:resource="&Service;Service_Restaurant_Context"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Figure 34. Restriction de la propriété `aContexte`.

Nous avons dit vouloir pondérer les différents attributs d'un service lors de la formulation d'une requête. Nous voulons aussi pouvoir définir des valeurs pour ces attributs et avoir des indications quant à l'ordre de grandeur de ces attributs. Pour ce faire, pour chaque élément de la classe ServiceContext nous définissons les propriétés suivantes:

- `aQuantificateur` - cette propriété indique la valeur souhaitée de l'attribut contextuel considéré. Cette propriété est de type booléen. Lorsque la valeur de cette propriété est `"VRAI"`, cela indique que l'on souhaite une valeur élevée pour cette propriété, et vice et versa lorsqu'elle a la valeur `"FAUX"`. Par exemple, si la propriété `aQuantificateur` est définie `"VRAI"` pour l'attribut contextuel `PlacesParkingDisponibles`, cela veut dire que pour cet attribut contextuel on souhaite avoir des valeurs élevées. La valeur de cette propriété est définie par le fournisseur de service, au moment de la création de la description du service.

```
<owl:DatatypeProperty rdf:about="&Service;aQuantificateur">
    <rdfs:domain rdf:resource="&Service;ServiceContext"/>
    <rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>
```

- aValeur - cette propriété indique la valeur actuelle de l'attribut contextuel. Il est à noter que dans la définition de cette propriété, nous ne spécifions pas le type de valeurs qu'elle peut prendre (ou rdfs:range). Ceci est réalisé en utilisant les restrictions OWL, et ce pour chaque attrobut contextuel. De cette manière, il est possible de restreindre la valeur d'un attribut contextuel à un type de données. Par exemple, pour l'attribut contextuel PlacesParkingDisponibles, la propriété aValeur peut seulement prendre des valeurs entières. C'est la restriction "aValeur **only** integer" qui permet cela.

```
<owl:DatatypeProperty rdf:about="&Service;aValeur">
       <rdfs:domain rdf:resource="&Service;ServiceContext"/>
</owl:DatatypeProperty>
```

- aPonderation - cette propriété permet à l'utilisateur de définir les attributs contextuels qui ont le plus d'importance dans sa recherche. Elle permet de définir une pondération, allant de 1 à 5 (du moins important au plus important).

```
<owl:DatatypeProperty rdf:about="#aPonderation">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="&Service;ServiceContext"/>
    <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>
```

Une fois définies notre hiérarchie de classes d'attributs contextuels et les propriétés de ces attributs, il s'agit d'y définir les attributs contextuels proprement dit. Nous prenons toujours l'exemple d'un service de type restaurant. Nous avons défini une restriction indiquant que les propriétés d'un service de ce type sont contenues dans la classe Service_Restaurant_Context. Nous y définissons donc les attributs contextuels identifiés dans la sous-section 4.3.1.3, à savoir:

- ***Le nom du restarant***
- ***La catégorie du restaurant*** – plusieurs catégories de restaurants sont définies ici, dont restaurant brasserie, restaurant à thème, restaurant traditionnel, restaurant rapide, etc. Les valeurs de la propriété aValeur de cet élément sont restreintes à des chaînes de caractères.
- ***Les places de parking disponibles*** – les valeurs de la propriété aValeur de cet élément sont restreintes à des entiers.
- ***Les tables disponibles*** – les valeurs de la propriété aValeur de cet élément sont restreintes à des entiers.
- ***Le style cuisine du restaurant*** – plusieurs styles de cuisine sont définis ici, parmi lesquels la cuisine asiatique, la cuisine européenne ou la cuisine africaine. Les valeurs de la propriété aValeur de cet élément sont restreintes à des chaînes de caractères.
- ***L'accès pour personnes handicapées*** – les valeurs de la propriété aValeur de cet élément sont restreintes à des booléens.
- ***La climatisation*** – les valeurs de la propriété aValeur de cet élément sont restreintes à des booléens.
- ***Le prix du repas*** – les valeurs de la propriété aValeur de cet élément sont restreintes à des entiers.

La Figure 35 illustre le résultat final. Nous pouvons remarquer le fait que les classes CategorieRestaurant et StyleCuisine contiennent chacune un ensemble de sous-classes.

Cette spécification des attributs contextuels d'un service de restauration a l'avantage de permettre la création de requêtes simples ou complexes. Si l'utilisateur souhaite préciser un style de cuisine ou une catégorie de restaurant, il peut ainsi créer des requêtes complexes. Par contre, il lui est aussi possible de ne pas spécifier ces attributs, et donc d'effectuer des requêtes plus simples.

Figure 35. Attributs contextuels de la classe `Service_Restaurant_Context`.

Nous venons de spécifier et de construire notre modèle étendu de description de services. Ce modèle permet d'avoir des catégories de services, chacune avec des attributs contextuels différents. Les restrictions définies sur les différentes propriétés assurent la cohérence du modèle. Nous devons maintenant spécifier comment ce modèle est utilisé lors du processus de découverte.

4.4 PROTOCOLE DE DECOUVERTE DE SERVICE SENSIBLES AU CONTEXTE AVEC OWL-S ETENDU

Dans cette partie, il s'agit de spécifier le protocole de découverte qui permet d'utiliser le modèle de description de service présenté ci-dessus. Nous proposons un protocole de découverte permettant d'ordonner les services découverts selon leur degré de pertinence par rapport à la requête de l'utilisateur. Les différentes étapes du processus de découverte sont illustrées par la Figure 36.

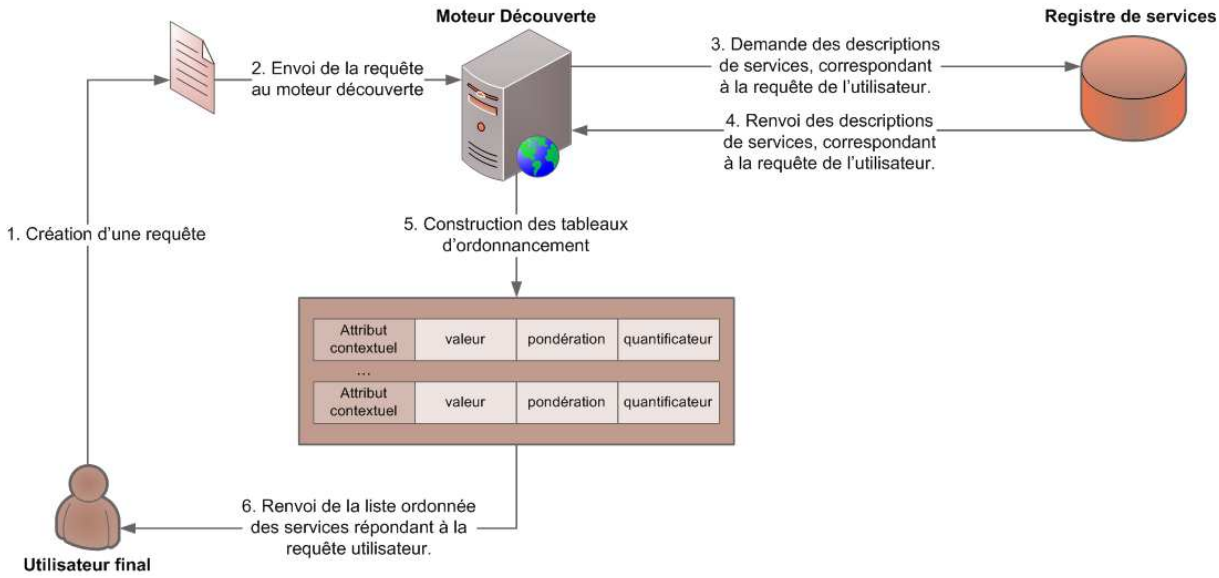

Figure 36. Notre protocole de découverte de services sensibles au contexte.

Lors de la première étape, l'on considère la requête de l'utilisateur en tant que description du service parfait. En effet, dans sa requête l'utilisateur précise des valeurs et des pondérations pour les attributs contextuels définissant un service donné. L'ensemble de ces attributs, accompagnés de leurs valeurs et pondérations respectives, forme une description définissant le service idéal recherché par l'utilisateur. Les services renseignés dans le registre seront comparés à cette description idéale. Cette description de service est ensuite transformée sous la forme d'un tableau, récapitulant les informations contenues dans la requête initiale. Ceci est illustré par le Tableau 13. Il s'agit de récupérer les valeurs des propriétés `aQuantificateur`, `aPonderation` et `aValeur` de chaque attribut contextuel défini dans l'ontologie pour le type de service recherché par l'utilisateur. Pour rappel, les valeurs de la propriété `aQuantificateur` sont définies lors de l'enregistrement du service dans le registre (voir 4.3.2.2). Les valeurs des propriétés `aPonderation` et `aValeur` sont définies par l'utilisateur, lors de la formulation de sa requête (voir 4.3.2.2).

Attribut contextuel	`aQuantificateur`	`aPonderation`	`aValeur`
`CategorieRestaurant`	-	-	*Brasserie*
`Places_Parking_Disponibles`	OUI	5	1
`Places_Tables_Disponibles`	OUI	5	2
`PrixRepas`	NON	3	15
`Climatisation`	-	1	OUI
`AccesHandicapes`	-	1	-
`StyleCuisine`	-	1	-

Tableau 13. Requête utilisateur.

La deuxième étape consiste à obtenir une liste de services répondant à la requête utilisateur. Pour ce faire, nous renvoyons les services ayant la même catégorie que celle définie par l'utilisateur dans sa requête. Par exemple, si l'utilisateur recherche un service de type restauration, seuls les services ayant cette même catégorie seront considérés à l'intérieur de cette liste de services. Si cette liste contient plus d'un élément, le « *Moteur Découverte* » extrait les informations contextuelles de chaque service. Si l'on considère l'exemple de requête présenté par le Tableau 13, il s'agit ici de retourner la liste des services restaurant dont la catégorie est « *Brasserie* ».

La troisième étape vise à construire, pour chaque service, un tableau contenant la valeur de chaque attribut contextuel du service. Le Tableau 14 illustre un exemple d'un tel tableau. Il n'est pas nécessaire ici de récupérer les valeurs de la propriété `aQuantificateur`, car ces valeurs sont identiques à celles de la requête utilisateur. En effet, ces valeurs étant définies lors de la création du service par le fournisseur de service, elles sont les mêmes pour un type de service donné. Il n'est pas non plus possible de récupérer les valeurs de la propriété `aPonderation`, car ces valeurs sont définies par l'utilisateur, au moment de la requête. Les propriétés `aQuantificateur` et `aPonderation` sont exclusivement utilisées lors du calcul du niveau de pertinence d'un service. Elles permettent de situer un service par rapport à une requête.

Attribut contextuel	*Service_restaurant_1 aValeur*	*Service_restaurant_2 aValeur*
CategorieRestaurant	Brasserie	Brasserie
Places_Parking_Disponibles	6	0
Places_Tables_Disponibles	5	2
PrixRepas	30	10
Climatisation	OUI	NON
AccesHandicapes	OUI	NON
StyleCuisine	-	-

Tableau 14. Valeurs des attributs contextuels de deux services.

Une fois ce tableau construit, les informations contenues dans la requête utilisateur sont comparées aux informations de chaque service. Il s'agit de calculer un score pour chaque service présent dans la liste de services construite lors de la deuxième étape. Ce score est calculé en fonction des pondérations définies par l'utilisateur, et en fonction des valeurs que présente chaque service pour les différents attributs contextuels considérés par l'utilisateur. Pour le calcul du degré de pertinence d'un service par rapport à une requête, nous définissons l'algorithme ci-dessous (Figure 37). L'algorithme diffère selon le type de valeur qu'un attribut contextuel peut prendre.

```
Si valeur booléenne
       Si Valeur(Service)=Valeur(Requete)
       -> Pondération
       Sinon
       -> (-1) × Pondération
Si valeur numérique
       Si (aQuantificateur = « VRAI » ET Valeur(Service)/Valeur(Requete) ≥ 1) OU
(aQuantificateur = « FAUX » ET Valeur(Service)/Valeur(Requete) ≤ 1)
       -> Pondération × Valeur(Service)/Valeur(Requete)
       Sinon
       -> (-1) × Pondération × Valeur(Service)/Valeur(Requete)
Si valeur ontologique
       Si Valeur(Service) = Valeur(Requete)
       -> Pondération
       Si Valeur(Service) subClassOf Valeur(Requete),
       -> Pondération/2
```

Figure 37. Algorithme pour le calcul du niveau de pertinence d'un attribut service.

Le niveau de pertinence d'un service par rapport à une requête est défini comme la somme des scores du service par rapport à chaque attribut contextuel. Si l'on suppose n comme étant que le nombre d'attributs contextuels pour un service donné, le score du $n^{ième}$ attribut est S_n. Le niveau de pertinence du service est défini par l'équation suivante:

$$SCORE(Service\ S) = \sum_{i=1}^{n} SCORE(A_i), \text{où } A_i \text{ représente le } i^{ème} \text{attribut du service.}$$

Par exemple, pour les services listés dans le Tableau 14, il s'agit de déterminer leur pertinence par rapport la requête initiale. Le calcul du score de chaque service est réalisé en appliquant l'algorithme de la Figure 37 à chacun des attributs. Pour chacun des deux services listés dans le Tableau 14, nous calculons donc leur niveau de pertinence (ou leur score) par rapport à la requête utilisateur :

$$\begin{aligned} &SCORE(Service\ R1) \\ &= \sum SCORE(Parking), SCORE(Tables), SCORE(Repas), SCORE(Clim), SCORE(Handicap), SCORE(Style) \\ &= \sum((5 \times 6), (5 \times 2{,}5), \big(3 \times (-2)\big), 1) \\ &= 37{,}5 \end{aligned}$$

$$\begin{aligned} &SCORE(Service\ R2) \\ &= \sum SCORE(Parking), SCORE(Tables), SCORE(Repas), SCORE(Clim), SCORE(Handicap), SCORE(Style) \\ &= \sum(0, (5 \times 1), (3 \times 0{,}67), -1) \\ &= 6 \end{aligned}$$

Ceci permet de construire une liste ordonnée de services, par ordre décroissant du niveau de pertinence. Cette liste est ensuite retournée à l'utilisateur, qui n'a plus qu'à choisir un service. Ce protocole de découverte permet de toujours avoir en tête de la liste le service répondant le mieux à la requête utilisateur. L'algorithme utilisé par ce protocole est décrit dans la Tableau 15.

Algorithme pour la comparaison et l'ordonnancement de services
1. Réception de la requête utilisateur, définissant le profil de service recherché S_P.
2. Traitement de la requête et création d'une requête SPARQL *Q*
3. Exécution de la requête *Q*, et obtention de la liste des services S_R répondant à la requête.
4. Si (card(S_R)>1)
4.1. Pour chaque service S_i dans S_R
4.1.2. Pour chaque attribut contextuel $A_i(S)$
4.1.2.1. Obtenir la valeur de $A_i(S)$
4.1.3. Construire le tableau de valeurs pour S_i
4.1.4. Calculer le score de S_i
4.2. Retourner la liste de services S_R ordonnée

Tableau 15. Algorithme pour la comparaison et l'ordonnancement des services.

Maintenant que nous avons présenté notre modèle de description de service et notre protocole de découverte, nous allons étudier la réalisation d'un prototype permettant ce genre de recherche. Comme décrit dans l'introduction, notre contribution vise à proposer une interface utilisateur claire et intuitive (à la fois pour l'utilisateur final, mais aussi pour le fournisseur de services qui souhaite référencer un service dans le registre), ainsi qu'un moteur de découverte de services. Le chapitre suivant présente l'implémentation de ces deux éléments, dans le cadre de notre prototype.

CHAPITRE 5.

PRESENTATION DU PROTOTYPE

Dans ce chapitre, nous présentons notre prototype, qui comprend une interface permettant de référencer un service dans un registre, une interface permettant de spécifier une requête service et un moteur découverte permettant une découverte sensible au contexte des services. L'architecture envisagée est présentée en premier, puis nous détaillons les fonctionnalités que devront supporter les deux interfaces et le moteur découverte.

5.1 L'ARCHITECTURE

Le prototype envisagé ici doit permettre la réalisation de deux principales fonctionnalités:

- ***Référencer un service dans le registre de services (F1)*** - cette première fonctionnalité s'adresse aux fournisseurs de services. Le but est de permettre la création de nouvelles descriptions de services, sur la base du modèle défini dans la sous-section 4.3. Nous faisons ici l'hypothèse que le fournisseur de services dispose déjà d'un fichier WSDL décrivant son service. A partir de ce fichier WSDL et de notre modèle, le fournisseur de services doit pouvoir créer une description de service intégrant à la fois les informations contenues dans le fichier WSDL et les informations définissant le contexte du service. Pour ce faire, nous devons mettre au point une interface permettant la création de telles descriptions de service et leur référencement dans le registre de services.
- ***Déterminer un ensemble de services répondant à une requête créée par l'utilisateur (F2)*** - cette fonctionnalité s'adresse à l'utilisateur final. Le but est de lui permettre de créer une requête de service, puis de visualiser la liste de services répondant à cette requête. Les actions disponibles pour les services présents dans cette liste sont définies dans la sous-section 5.2.2. Nous devons donc développer une interface assurant ces fonctionnalités, et ce d'une manière intuitive pour l'utilisateur.

Notre prototype peut donc être décomposé en deux parties que nous appelons la partie « *Fournisseur de services* » et la partie « *Utilisateur final* ». L'architecture du prototype, ainsi que les éléments le composant sont illustrés par la figure ci-dessous (Figure 38).

Les deux parties impliquent une interface utilisateur. Dans les deux cas, il s'agit de formulaires dynamiques, générés à partir d'informations contenues dans des fichiers passés en paramètre. Le formulaire « *Fournisseur de service* » permet de générer une description de service conforme à notre modèle OWL-S étendu. Le formulaire « *Utilisateur final* » permet de générer une requête de service, qui n'est en fait qu'une description d'un service idéal, toujours en conformité avec le modèle OWL-S étendu défini plus haut. Ces deux interfaces sont détaillées dans la partie suivante de ce chapitre.

Nous pouvons observer que le registre de services représente l'élément commun entre la partie « *Fournisseur de services* » et la partie « *Utilisateur final* ». Les descriptions de services générées par les fournisseurs de services y sont référencées. Le moteur de découverte récupère depuis ce même registre les services correspondant à une requête donnée.

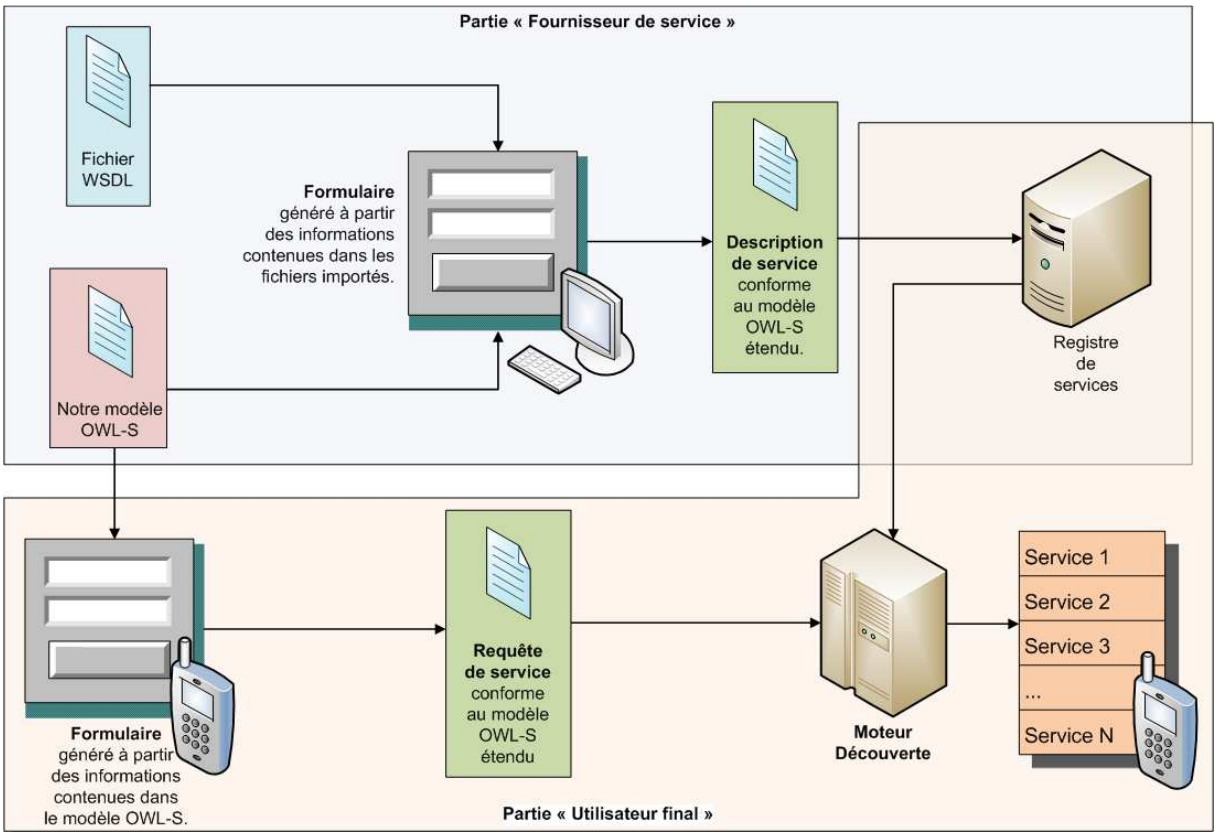

Figure 38. Architecture de notre prototype.

Dans ce qui suit, nous étudions les principales fonctionnalités que doivent supporter les deux interfaces. Leur réalisation sera détaillée au Chapitre 6.

5.2 LES INTERFACES

Comme précisé dans l'introduction, le but ici est de créer des interfaces utilisateur claires et intuitives. Le but de ces interfaces est de rendre la publication et la requête de services aussi simples que possible du point de vue de l'utilisateur.

Dans cette partie, *une publication de service* consiste en un document OWL-S décrivant un service selon le modèle défini au paragraphe 4.3. Dans un tel document, le fournisseur de service définit des valeurs pour les propriétés `aQuantificateur` et `aValeur`, de chaque attribut contextuel.

Une requête de service est aussi un document OWL-S correspondant au modèle défini, mais à la différence de la publication de service, ce document spécifie aussi la pondération des attributs contextuels (en définissant des valeurs pour la propriété `aPonderation`).

5.2.1 INTERFACE POUR LA PUBLICATION DE SERVICE

Lors de la publication d'un service par un fournisseur de service, nous avons voulu préserver la description WSDL du service, et l'encapsuler dans la description OWL-S de ce même service. Comme décrit dans la sous-section 2.1.2.3, le langage WSDL décrit de manière abstraire les opérations et les messages d'un service Web. WSDL permet d'invoquer un service sans que le fournisseur de service et le client aient connaissance l'un de l'autre. Un deuxième avantage est le fait que la plupart des

fournisseurs de services disposent d'ores et déjà de descriptions WSDL pour leurs services. Le standard WSDL étant antérieur au standard OWL-S, la plupart des services Web disposent aujourd'hui d'une description WSDL. Il convient donc de tirer parti des avantages d'une description WSDL, lors de la création de la description OWL-S.

Pour ce faire, nous adoptons la démarche suivante, illustrée par la Figure 39.

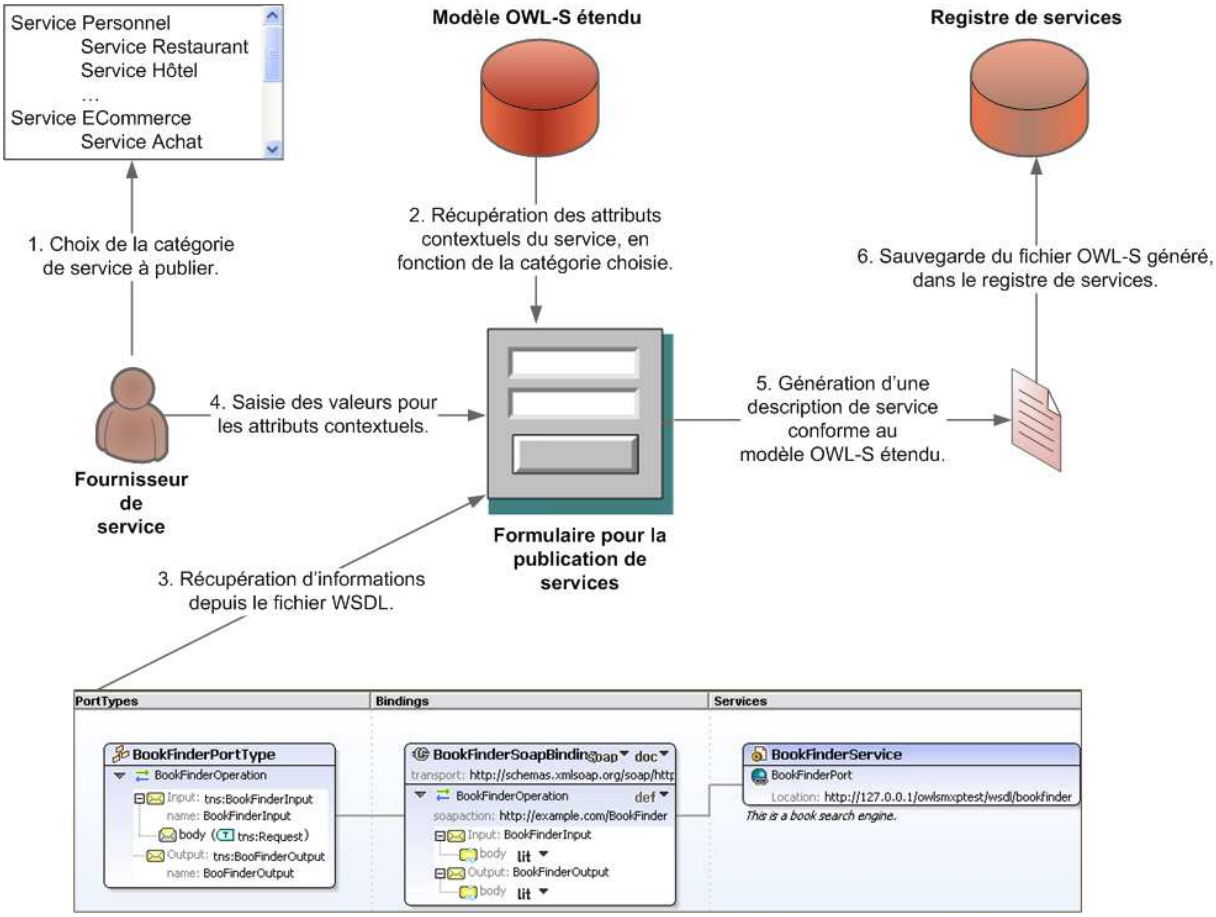

Figure 39. Etapes de la publication d'une description de service.

Sont récupérées en premier les informations décrivant le service de manière abstraite. Il s'agit de lire ces informations depuis un fichier WSDL fourni par l'utilisateur. Ces informations définissent des caractéristiques du service, tel son nom, ses entrées et sorties, etc.

Deuxièmement, il faut lire notre modèle de description de service et d'en extraire la liste des attributs contextuels définis pour le type de service considéré. Pour chaque attribut contextuel, il faut aussi extraire le type de valeur qu'il accepte en entrée. Ceci est défini à travers la propriété `aValeur`. Pour chaque attribut, cette propriété peut prendre des types de valeurs différents. Ceux-ci sont spécifiés dans notre modèle de description de service. Une fois les attributs contextuels identifiés, le fournisseur de service n'a plus qu'à leur définir une valeur.

Dans l'idéal, la valeur de ces attributs contextuels peut être *dynamique* (par exemple le nombre de tables disponibles) ou *statique* (par exemple le prix moyen d'un repas). Dans la pratique, nous avons choisi de considérer l'ensemble de ces attributs comme étant statiques. Il est aisément imaginable d'avoir des capteurs permettant de récupérer la valeur d'attributs contextuels dynamiques, tels le nombre de places disponibles sur un parking. Notre objectif n'étant pas la collecte des informations

contextuelles, mais bien leur traitement et utilisation, nous ne traitons pas le cas des attributs dynamiques.

Une fois les informations WSDL récupérées et les valeurs des attributs contextuels définies, il faut générer une description de service OWL-S englobant l'ensemble de ces informations. Ce fichier constitue une publication de service. La Figure 40 ci-dessous illustre un exemple d'un formulaire permettant de générer un tel fichier.

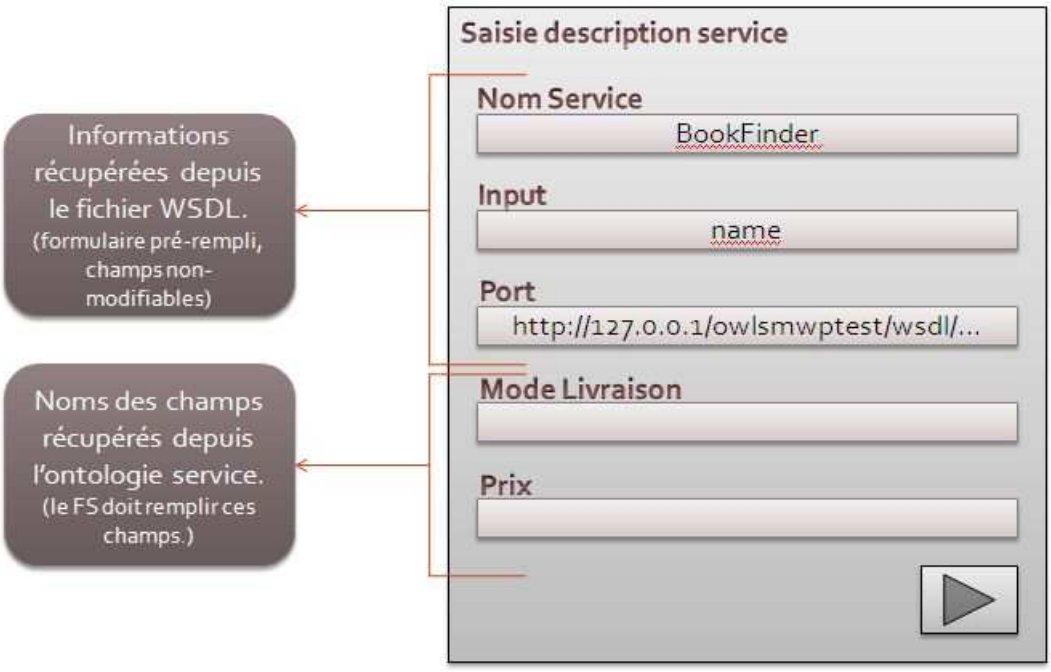

Figure 40. Exemple d'un formulaire pour la publication de services.

Les différentes étapes décrites ci-dessus sont résumés dans la figure suivante (Figure 41):

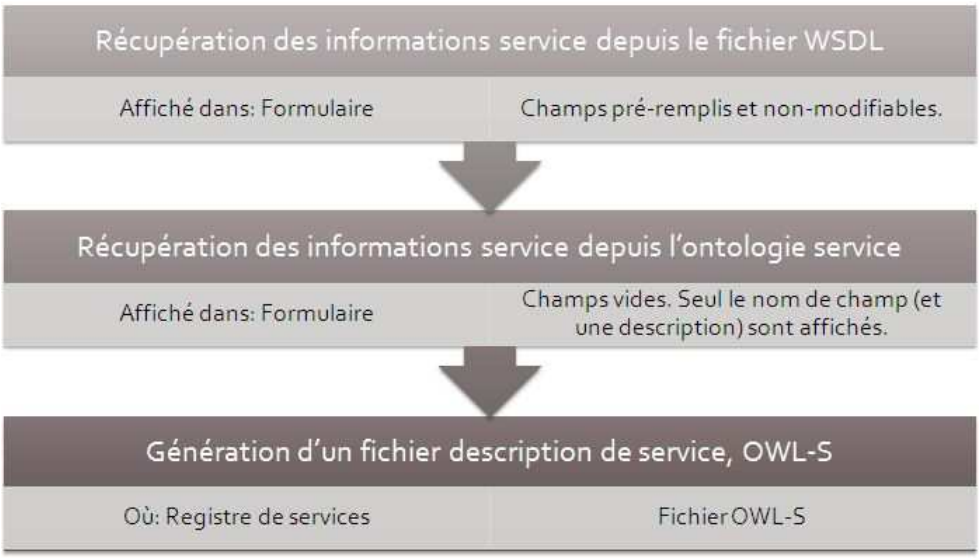

Figure 41. Etapes du processus de publication de service.

Une fonctionnalité importante de cette interface est la possibilité, pour le fournisseur de service, de modifier notre modèle en ajoutant de nouvelles catégories de services. En effet, lors du processus de référencement d'un service dans l'annuaire, le fournisseur de service commence par balayer la liste

des catégories de services disponibles. Il en choisit une, à l'intérieur de laquelle son service sera référencé. Si le fournisseur de service ne trouve pas de catégorie correspondant à son service, il peut en créer une nouvelle. Dans ce cas, il devra définir les attributs contextuels de cette nouvelle catégorie, ainsi que le type de valeur que ces attributs peuvent prendre. Grâce à l'usage des ontologies, il peut même spécifier une ontologie définissant la nouvelle catégorie de services. Il est aisément imaginable d'avoir à référencer un service spécifique à un domaine donné. Dans ce cas, des ontologies spécifiques à ce domaine peuvent être utilisées dans la définition du contexte du ce service. Ce sont de telles ontologies, spécifiques à un domaine, qui constituent une base de connaissances.

L'implémentation de l'interface et du moteur permettant la réalisation des opérations sus mentionnées seront détaillées dans le chapitre suivant (Chapitre 6).

5.2.2 INTERFACE POUR LA REQUETE DE SERVICE

Une requête de service doit satisfaire les conditions suivantes:

- ***Une requête de service doit être exprimée de manière simple et claire.*** Le protocole de découverte ne doit pas imposer à l'utilisateur la construction de requêtes trop longues ou trop complexes. Par exemple, si les descriptions de services sont stockées dans une base de données relationnelle, il est incommode pour l'utilisateur de créer des requêtes SQL complexes pour découvrir des services. Idéalement, les utilisateurs doivent pouvoir spécifier les attributs/propriétés des services à travers une interface simple et intuitive. C'est au protocole de découverte de transformer la demande de l'utilisateur en une requête correspondant au modèle de description de services.
- ***Une requête de service doit être flexible.*** Elle doit permettre à l'utilisateur de rechercher un service en combinant plusieurs critères. Le système ne devra pas imposer des restrictions quant au format de la requête.
- ***Une requête de service doit utiliser des sémantiques***, afin d'améliorer la précision du processus de découverte.

Dans notre approche, l'utilisateur commence par parcourir la liste des catégories de services, telle que définie dans notre hiérarchie de profils service (voir 4.3.1.3). Pour ce faire, le programme analyse notre modèle de description de services, afin d'en extraire la liste des profils service. Lorsque l'utilisateur choisit une catégorie, le programme vérifie si cette catégorie présente des sous-catégories ou pas. Si oui, l'utilisateur le programme renvoie la liste des sous-catégories, et l'utilisateur en choisit une. Il continue ainsi jusqu'à arriver à une catégorie qui ne possède pas de sous-catégories. A tout moment, l'utilisateur peut arrêter ce processus et passer à l'étape suivante. Cela veut dire que l'utilisateur n'est pas obligé de spécifier intégralement la catégorie de service qu'il recherche. Il est possible de définir une catégorie spécifique (telle la catégorie « *Brasserie* ») ou une catégorie générale (en précisant seulement la catégorie de service, « *Restaurant* » par exemple). Une fois une catégorie de service spécifiée, les attributs contextuels de cette catégorie sont affichés. Pour chaque attribut contextuel, l'utilisateur dispose d'un champ texte, lui permettant de saisir une valeur, ainsi que d'un « *slider* », lui permettant de définir une pondération à l'attribut considéré. L'utilisateur valide ensuite sa requête. Ceci a pour effet de générer un fichier OWL-S, contenant les spécifications de l'utilisateur. La Figure 42 ci-dessous illustre les étapes de ce processus.

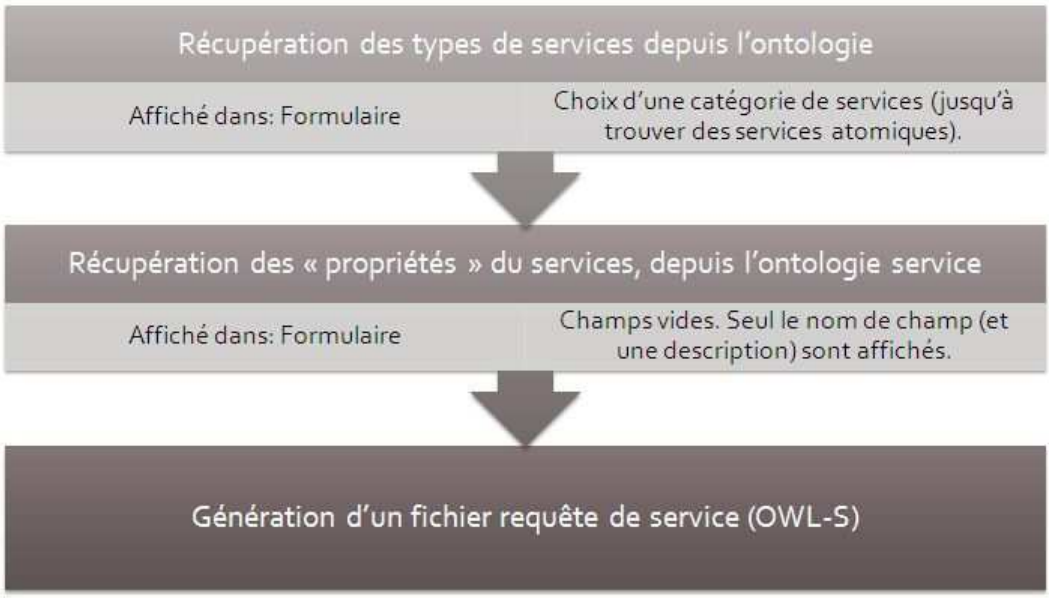

Figure 42. Etapes du processus de requête de service.

Notre approche tient compte des conditions énoncées plus haut, puisqu'elle permet d'exprimer une requête de service de manière simple et intuitive. L'utilisateur effectue des sélections, mais peut aussi en ignorer, ce qui rend la requête plus rapide. L'utilisation de pondérations pour les différents attributs contextuels permet la construction de requêtes flexibles. L'ensemble du processus de requête est basé sur notre extension de l'ontologie OWL-S, ce qui permet de construire une requête de service en utilisant des sémantiques.

5.2.3 AVANTAGES DE CETTE APPROCHE

Un des principaux avantages de notre approche est sa simplicité. Les interfaces décrites plus haut sont intuitives pour l'utilisateur:

- Pour le fournisseur de service, notre interface lui évite d'écrire une description de service « *à la main* ».
- Pour l'utilisateur final, la construction d'une requête à travers notre interface est de loin plus aisée que la création d'une requête en utilisant des langages tels XQuery (BOAG 2007), XPath (CLARK 1999) ou même SQL (SQL 2009).

De plus, notre approche est transparente: les différentes étapes et l'ensemble de choix que l'utilisateur doit faire reflètent la structure de notre modèle ontologique. Ceci a aussi une valeur éducative ou pédagogique. En effet, cela permet de réduire le temps d'apprentissage pour l'utilisateur.

En outre, la hiérarchie de profils service est facilement transposable dans une interface Web ou encore dans une interface d'application mobile. Cette interface peut être affichée dans un navigateur web, un terminal mobile ou un ordinateur.

Une fois les publications et requêtes de service ainsi construites, elles peuvent être utilisées par le moteur découverte. Ceci est présenté dans le chapitre suivant.

5.3 LE MOTEUR DECOUVERTE

Nous avons vu comment les deux interfaces « *Fournisseur de service* » et « *Utilisateur final* » permettent de créer des descriptions de services, que ce soient des publications de services ou des requêtes de services. Le « *Moteur Découverte* » a pour principale fonction la comparaison de descriptions de services. Pour ce faire, il repose sur l'algorithme défini dans la sous-section 4.4. La Figure 43 présente le fonctionnement de ce moteur.

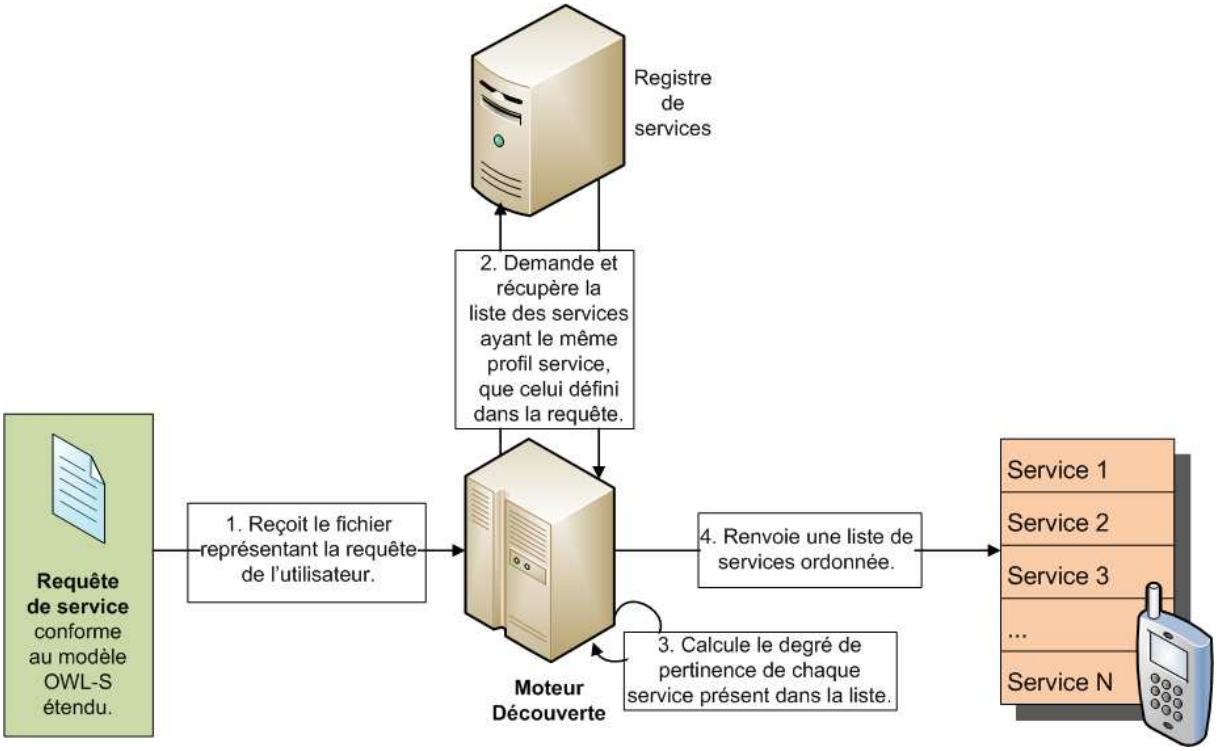

Figure 43. Fonctionnement du « Moteur Découverte ».

Le « *Moteur Découverte* » ainsi que le registre de services représentent la partie *serveur* de notre prototype. Ces composants doivent en effet manipuler des fichiers d'ontologie, et cela implique l'utilisation d'outils particuliers, dont l'implémentation sur une plate-forme mobile serait trop lourde. Les détails relatifs à l'implémentation de ces composants sont donnés dans le chapitre suivant.

CHAPITRE 6.

DEVELOPPEMENT ET IMPLEMENTATION DU PROTOTYPE

Ce chapitre a pour but de détailler l'implémentation du prototype décrit précédemment. L'implémentation du prototype a été effectuée en Java. La partie « *Utilisateur final* » (ou terminal mobile) a été développée sur la base de la plate-forme Android™ de Google. La partie « *Fournisseur de service* » a été implémentée sous la forme d'un formulaire Web. Dans un premier temps, nous étudions les détails d'implémentation relatifs au « *Moteur Découverte* ». Ce moteur, tout comme l'interface « *Fournisseur de service* », repose sur l'API Jena, qui permet de manipuler des ontologies. Dans un deuxième temps, nous examinons le développement et l'implémentation des deux interfaces correspondant à chacune des deux parties susmentionnées. Nous y présentons les technologies sur lesquelles ces interfaces reposent. Il s'agit, entre autres, de présenter les caractéristiques de la plate-forme Android™ et de justifier pourquoi notre choix s'est porté vers cette plate-forme.

6.1 La plate-forme Android™

6.1.1 Presentation generale

Android™ est un système d'exploitation lancé par Google, en 2007, qui a été conçu pour fonctionner sur des plateformes mobiles, telles les Smartphones ou les PDA. Le système est basé sur un noyau Linux et est disponible via une licence Apache version 2. Il s'agit d'une plate-forme logicielle, qui n'est pas liée à un appareil donné. Au contraire, elle est vouée à être intégrée dans différents appareils mobiles, par différents constructeurs.

Les raisons qui ont porté notre choix sur cette plate-forme concernent principalement sa grande flexibilité, son accessibilité à tous les intégrateurs et développeurs, ainsi que le fait qu'elle fasse profiter l'utilisateur de la convergence mobile/Web. De plus, cette plate-forme est facilement programmable, et Google a fourni un SDK Java, un plugin Eclipse, des émulateurs pour tests, ainsi qu'une documentation étendue. De plus, la stratégie de Google vise un grand nombre d'utilisateurs, de la même manière que notre application. Pour résumer, la figure suivante (Figure 44) présente les avantages de cette plate-forme, du point de vue des utilisateurs et des développeurs.

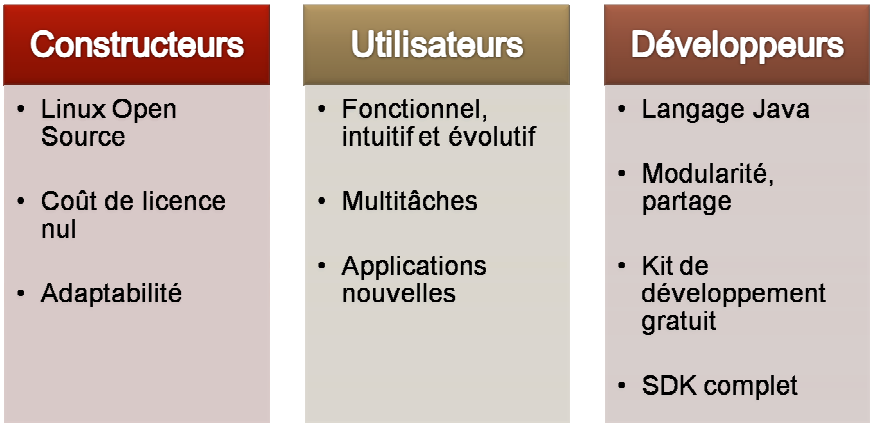

Figure 44. Points forts de la plate-forme Android™

Ces sont ces éléments qui nous ont fait choisir cette alternative. De plus, l'intégration de composants Google tels GoogleMaps, facilitent la localisation et l'accès aux services.

La plate-forme Android™ a une architecture en couches, tel qu'illustré par la Figure 45 ci-dessous.

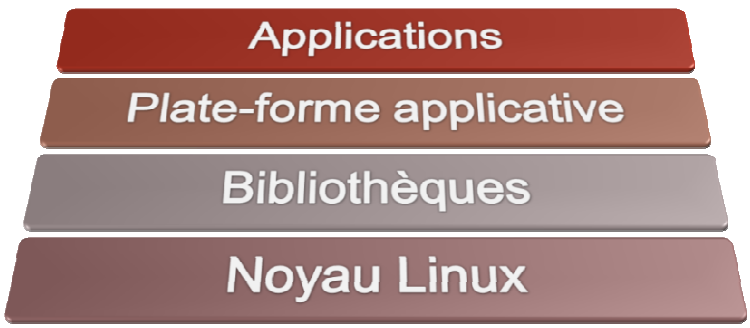

Figure 45. Architecture Android™.

Le système d'exploitation se situe au plus près de la partie matérielle, et représente un noyau Linux 2.6 amélioré par Google, aux niveaux de la gestion de l'énergie et de la communication interprocessus. La seconde couche est la couche des bibliothèques, notamment la bibliothèque C bionic, écrite par Google spécialement pour Android™ et dont la taille réduite est adaptée à un environnement embarqué (MEIER 2009). Le runtime Android™, c'est-à-dire la machine virtuelle Java et ses bibliothèques, est situé au-dessus de la couche bibliothèques. Il s'agit d'une machine virtuelle spécialement conçue pour les environnements embarqués, la Dalvik Virtual Machine (MEIER 2009). La troisième couche est la couche « *plate-forme applicative* » qui contient, entre autres, la boîte à outils graphiques, permettant l'affichage de boîtes de dialogue, de menus, etc. C'est à cette couche qu'il faudra accéder afin de développer l'interface utilisateur pour notre prototype. Ceci est réalisé à travers un ensemble d'APIs. La dernière couche est la couche application. Une application est représentée par un paquet `*.apk`, ce qui permet une installation/désinstallation facilitées.

La réalisation des interfaces sous Android™ est présentée dans ce qui suit.

6.1.2 ANATOMIE D'UNE APPLICATION ANDROID™

Une application Android™ est écrite en Java. Le code compilé, les données et les fichiers ressources requis par l'application sont packagés dans un fichier archive — Paquet Android™— ayant pour suffixe « `.apk` ». Par défaut, chaque application a son processus Linux et chaque processus possède sa propre machine virtuelle Java.

Une application Android™ repose sur les concepts suivants (MURPHY 2009):

- ***Le concept d' « Activité »*** (`Activity`) est le concept de base d'une interface utilisateur. Une activité peut être vue en tant qu'analogie d'une fenêtre ou d'une boîte de dialogue dans une application bureautique. C'est en utilisant des activités qu'il est possible de définir des interfaces graphiques permettant à l'utilisateur d'interagir avec l'application.
- ***Le concept de « Fournisseur de contenu »*** (`Content Provider`) fournit un niveau d'abstraction pour tous types de données stockées sur le terminal mobile, et accessible par plusieurs applications. En effet, le modèle de développement Android™ encourage la réutilisation des données entre applications. La création d'éléments « *Fournisseurs de contenu* » permet de réaliser ceci, tout en gardant le contrôle quant au mode d'accès aux donnés.
- ***Le concept d' « Intention »*** (`Intent`) représente des messages système, s'exécutant à l'intérieur du terminal mobile, et notifiant les applications de divers événements, allant de modifications hardware (insertion d'une carte SD) à des événements d'application (lancement d'une activité depuis le menu principal du terminal mobile). Dans une application Android™ il est non seulement possible de répondre à des intentions, mais aussi d'en créer des nouvelles. Les intentions permettent de lancer d'autres activités ou d'informer sur des situations spécifiques (par exemple, lorsque l'utilisateur se rapproche à moins de 100m d'une position donnée).
- ***Le concept de « Services »*** (`Service`) se différencie des autres concepts listés ci- dessus par sa durée de vie. Les activités, fournisseurs de contenu et intentions sont censés avoir une courte durée de vie et peuvent être quittés ou fermés à tout moment. Les services sont au contraire conçus pour s'exécuter en continu, et si nécessaire, indépendamment de toute autre activité. Il est par exemple possible d'utiliser un service pour vérifier des mises à jour

de flux RSS ou alors pour continuer à jouer de la musique même lorsque l'activité contrôlant le lecteur (contrôle du volume, passage au morceau suivant, etc.) n'est plus en exécution.

Ces concepts sont étudiés plus en détail dans la partie « Présentation des concepts Android™ ». Pour le moment, nous nous intéressons à la structure d'un projet Android™. Cette structure reprend une structure d'arbre, comme c'est le cas pour un projet Java. Lors de la création d'un nouveau projet Android, plusieurs éléments sont présents à la racine du projet (MURPHY 2009). Le répertoire `/bin/notreapp-debug.apk` ou `/bin/notreapp-unsigned.apk` constitue la véritable application Android™.

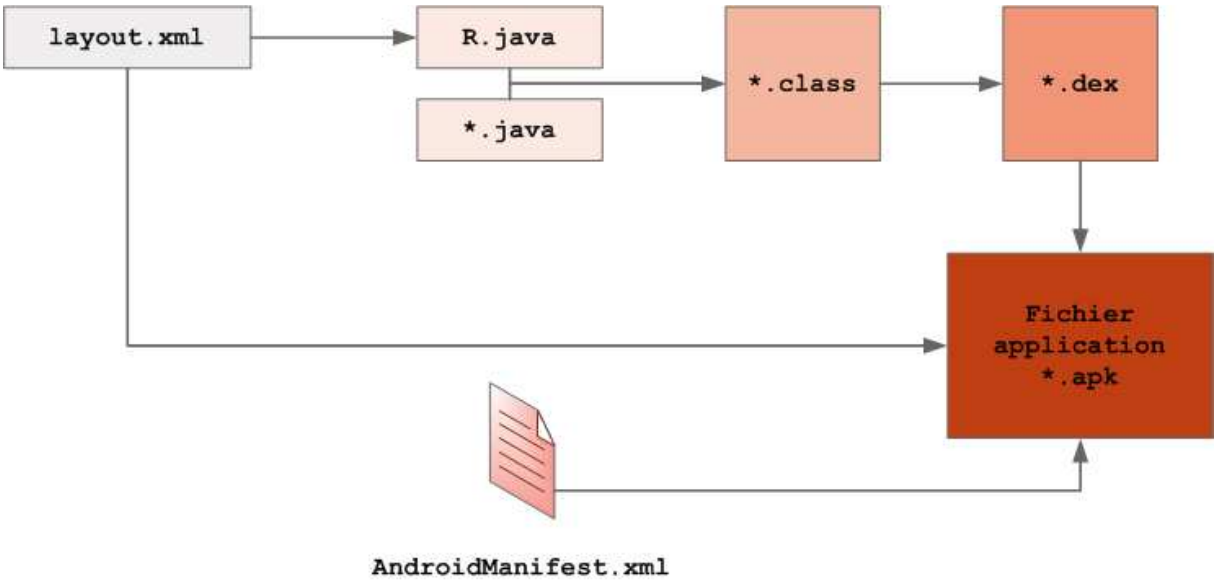

Figure 46. Eléments d'une application Android™.

Le fichier `.apk` est une archive `ZIP` contenant un fichier `.dex`, qui est le résultat de la compilation des ressources (`resources.arsc`), des ressources non-compilées (contenues dans le répertoire `/res/raw/`) et du fichier `AndroidManifest.xml`. Ce dernier fichier représente la base d'une application Android™. C'est à l'intérieur de ce fichier qu'il faut déclarer les éléments composant notre application (les activités, les services, les intentions, etc.). Il faut aussi y déclarer comment ces éléments interagissent entre eux et comment ces éléments sont attachés au système Android™ en général. Il s'agit par exemple de spécifier quelles activités doivent apparaître sur le menu principal du terminal mobile. La Figure 46 ci-dessus illustre les éléments entrant dans la composition d'une application Android™.

6.2 DEVELOPPEMENT DU MOTEUR DECOUVERTE

Ayant présenté les principes de développement Android™, nous nous intéressons maintenant au développement du « *Moteur Découverte* ». Comme indiqué précédemment, le « *Moteur Découverte* » a pour but de comparer la requête utilisateur aux descriptions de services présentes dans le registre. Cela revient à comparer des fichiers ontologies entre eux. Pour cette première implémentation de notre prototype, nous avons choisi d'utiliser l'API Jena (Jena 2009). Notre choix a été principalement motivé par la gratuité de cette API, mais aussi par le fait qu'elle repose sur le langage Java (tout comme la plate-forme Android™).

L'API Jena permet la création, la gestion et la manipulation des ontologies à travers les graphes RDF. Tel que précisé dans la sous-section 2.2.3.2.2, toute ontologie OWL est une ontologie RDF. Les formalismes OWL utilisés dans notre modèle de description de services sont construits au dessus du langage RDF et peuvent être manipulés grâce à cette API (Jena 2009). Jena permet, entre autres, de récupérer les URIs des classes et des propriétés définies dans notre modèle. Nous utilisons donc cette API afin d'extraire les informations nécessaires des fichiers décrivant la requête utilisateur et les différents services, afin de les comparer entre eux.

Or, les outils nécessaires pour la comparaison d'ontologies ne sont pas adaptés à une plate-forme mobile, telle la plate-forme Android™. Nous avons donc développé une couche logicielle qui fournit un ensemble de méthodes pour la partie interface. De cette façon, la partie « *utilisateur final* » (ou « *terminal utilisateur* ») de l'application ne manipule pas directement le concept d'ontologie. L'interface utilisateur emploie seulement des chaînes de caractères et des URIs. De cette manière, le concept d'ontologie est transparent pour les développeurs d'interfaces. Nous avons donc implémenté l'architecture suivante (Figure 47):

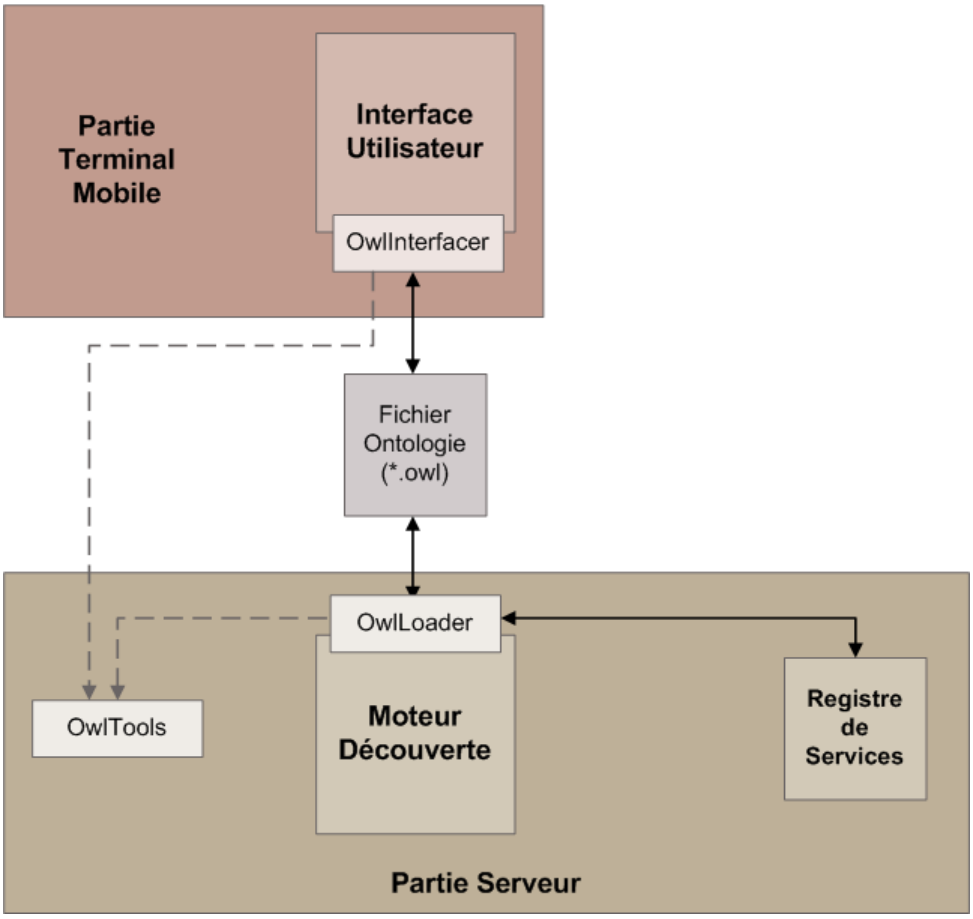

Figure 47. Architecture du « Moteur Découverte ».

Nous présentons les détails relatifs à la conception de ces classes dans les paragraphes suivants.

6.2.1 LES CLASSES JAVA

La Figure 48 illustre le diagramme des classes implémentées. Pour plus de simplicité, la classe `OwlServer` n'est pas représentée sur cette figure.

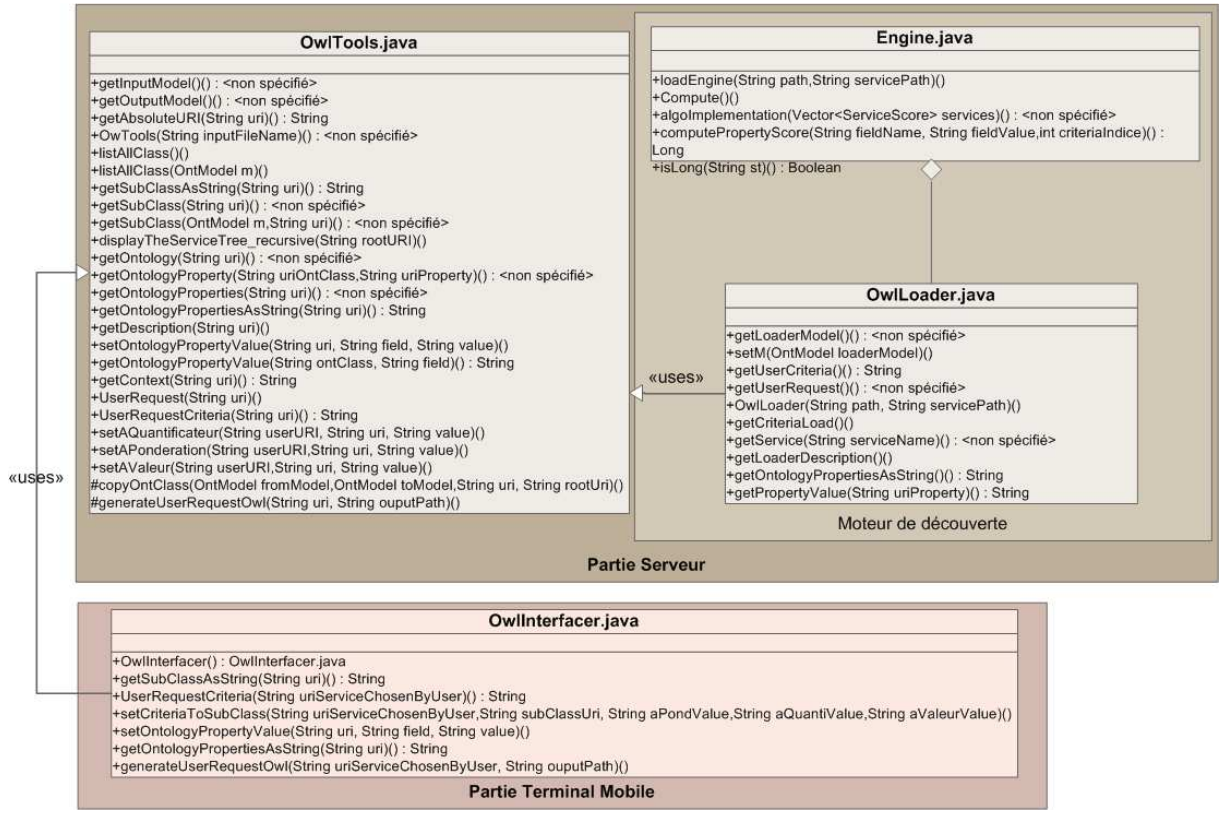

Figure 48. Diagramme des classes du « Moteur Découverte ».

Comme indiqué par la Figure 48, les classes `OwlLoader` et `OwlInterfacer` héritent de la classe `OwlTools`. Cette classe contient les méthodes nécessaires à la manipulation des ontologies. Les méthodes de cette classe sont détaillées dans le Tableau 16.

Nom classe	*Description / Rôle*
`private OntModel inputModel`	Référence le modèle d'entrée Jena (autrement dit le fichier `*.owl` passé en entrée).
`public OntModel getInputModel()`	Permet d'accéder à l'`inputModel`.
`private OntModel OutpoutModel`	Référence le modèle de sortie Jena (autrement dit le fichier `*.owl` généré).
`public OntModel getOutputModel()`	Permet d'accéder à l'outputModel.
`public OwlUtils(String inputFileName)`	Constructeur de la classe, auquel il faut passer en paramètre le fichier `*.owl` représentant la requête utilisateur.
`public void listAllClass()`	Liste l'ensemble des classes du modèle.
`public Vector<String> getSubClassAsString(String uri)`	Renvoie, sous la forme d'un vecteur, l'ensemble des sous-classes d'une classe dont l'URI est passée en paramètre.
`public Vector<OntClass> getSubClass(String uri)`	Renvoie, sous la forme d'un vecteur, l'ensemble des sous-classes d'une classe dont l'URI est passée en paramètre.
`public void`	Affiche, sous forme d'arbre, l'ensemble des sous-

displayTheServiceTree_recursive(String rootURI)	classes d'une classe dont l'URI est passée en paramètre.
public OntClass getOntology(String uri)	Renvoie l'`OntClass` asscoiée à l'URI passée en paramètre.
public OntProperty getOntologyProperty(String uriOntClass,String uriProperty)	Renvoie l'`OntProperty` associée aux URI passées en paramètre.
public LinkedList<OntProperty> getOntologyProperties(String uri)	Renvoie l'ensemble des propriétés d'une classe ou d'une ontologie, dont l'URI est passée en paramètre.
public LinkedList<String> getOntologyPropertiesAsString(String uri)	Renvoie une liste chaînée d'URIs de propriétés.
public void getDescription(String uri)	Affiche la description d'une classe, dont l'URI est passée en paramètre.
public String getOntologyPropertyValue(String ontClass, String field)	Renvoie une chaîne de caractères représentant la valeur d'une propriété, dont le nom est passé en paramètre.
public void setOntologyPropertyValue(String uri, String field, String value)	Permet de définir la valeur d'une propriété.
public String getContext(String uri)	Cette méthode permet de renvoyer le contexte d'un profil de service, dont l'URI est passée en paramètre.
public LinkedList<String> UserRequestCriteria(String uri)	Cette méthode renvoie, sous la forme d'une liste chaînée, l'ensemble des informations contextuelles pouvant être modifiées par l'utilisateur lors de la recherche de services.
public void setAQuantificateur(String userURI, String uri, String value)	Permet de définir une valeur pour la propriété `aQuantificateur` d'une classe dont l'URI est passée en paramètre.
public void setAPonderation(String userURI,String uri, String value)	Permet de définir une valeur pour la propriété `aPondération` d'une classe dont l'URI est passée en paramètre.
public void setAValeur(String userURI,String uri, String value)	Permet de définir une valeur pour la propriété `aValeur` d'une classe dont l'URI est passée en paramètre.
protected void copyOntClass(OntModel fromModel,OntModel toModel,String uri, String rootUri)	Permet de copier une `ontClass` d'un modèle à un autre.
protected void generateUserRequestOwl(String uri, String ouputPath)	Génère un fichier `*.owl`, vers une adresse passée en paramètre.

Tableau 16. Liste des méthodes de la classe `OwlTools`.

La classe `OwlInterfacer` hérite de la classe `OwlTools`. Cette classe est utilisée lors de la création de l'interface utilisateur, pour la plate-forme Android™ (voir 6.3.1). Les méthodes fournies par cette classe sont détaillées dans le Tableau 17.

Nom classe	Description / Rôle
public OwlInterfacer(String inputFile)	Constructeur de la classe, auquel il faut passer en paramètre le fichier `*.owl` représentant notre modèle de description de service.
public Vector<String> getSubClassAsString(String uri)	Renvoie, sous la forme d'un vecteur, l'ensemble des sous-classes d'une classe dont l'URI est passée en paramètre.
public LinkedList<String> UserRequestCriteria(String uriServiceChosenByUser)	Renvoie, sous la forme d'une liste chaînée, les informations contextuelles paramétrables par l'utilisateur, dans sa requête. Elle reçoit en paramètre l'URI du type de service choisi par l'utilisateur.
public void setCriteriaToSubClass(String uriServiceChosenByUser,String subClassUri, String aPondValue,String aQuantiValue,String aValeurValue)	Permet d'affecter le triplet de valeurs utiles `<aPonderation, aQuantificateur, aValeur>` pour une propriété dont l'URI est passée en paramètre.
public void generateUserRequestOwl(String uriServiceChosenByUser, String ouputPath)	Génère le fichier `*.owl` correspondant à la requête utilisateur.

Tableau 17. Liste des méthodes de la classe `OwlInterfacer`.

La classe `OwlLoader` permet de communiquer avec le registre de services. Elle permet de récupérer les descriptions de services correspondant aux critères définis par l'utilisateur dans sa requête, puis d'en extraire les valeurs des différentes informations contextuelles (sous la forme d'un vecteur). Les méthodes de cette classe sont détaillées dans le Tableau 18.

Nom classe	Description / Rôle
public OwlLoader(String path, String servicePath)	Constructeur de la classe, auquel il faut passer en paramètre l'adresse du fichier `*.owl` représentant la requête utilisateur.
public void getCriteriaLoad()	Permet de récupérer les valeurs et pondérations des différentes informations contextuelles présentes dans la requête utilisateur. Cette fonction est utilisée par le constructeur de la classe.
public ResultSet getService(String serviceName)	Il s'agit de la requête SPARQL adressée au registre de services. Cette méthode permet de récupérer les services dont le nom de la catégorie est passé en paramètre.

Tableau 18. Liste des méthodes de la classe `OwlLoader`.

La classe `SimpleEngine` contient l'algorithme de comparaison. Elle contient le cœur du moteur de découverte. Les méthodes de cette classe sont détaillées dans le Tableau 19.

Nom classe	Description / Rôle
`public SimpleEngine()`	Constructeur de la classe.
`public void loadEngine(String path,String servicePath)`	Permet de lancer le moteur découverte, en lui passant l'adresse du fichier requête utilisateur, ainsi que l'adresse du service auquel il faut la comparer.
`public void Compute()`	Calcule le niveau de pertinence d'un service.
`public Vector<ServiceScore> algoImplementation(Vector<ServiceScore> services)`	Implémente l'algorithme de comparaison défini au paragraphe 4.4. Cette méthode parcourt la liste des services chargés par le moteur découverte, depuis le registre de services. Pour chaque service, elle fait appel à la méthode computePropertyScore, afin de déterminer le niveau de pertinence du service.
`public Long computePropertyScore(String fieldName, String fieldValue,int criteriaIndice)`	Permet de calculer le niveau de pertinence pour une propriété donnée, comme défini dans l'algorithme de comparaison défini au paragraphe 4.4.

Tableau 19. Liste des méthodes de la classe `SimpleEngine`.

Dans ce qui suit, nous présentons l'implémentation du moteur découverte par rapport à la plate-forme Android™, notamment côté serveur et côté client.

6.2.2 L'IMPLEMENTATION

Pour pouvoir utiliser le « *Moteur Découverte* » décrit précédemment, il faut le voir en tant qu'un « *Service Découverte* ». L'utilisateur interroge ce service, en lui adressant sa requête, et ce « *Service Découverte* » répond à l'utilisateur en lui renvoyant la liste des services correspondant à sa requête. L'application utilisateur doit donc savoir où trouver ce « *Service Découverte* », comment y accéder et comment l'utiliser correctement.

La plate-forme Android™ ne propose aucune bibliothèque permettant de communiquer avec un service Web, que ce soit pour le protocole SOAP (voir § 2.1.2.2.2) ou pour le protocole XML-RPC (voir § 2.1.2.2.1). Il est donc nécessaire de palier à ce problème, soit en utilisant des bibliothèques déjà existantes, soit en en codant une « *à partir de rien* ».

La raison de l'absence de bibliothèques dans Android™ est très probablement liée au très fort intérêt que Google porte aux « *REST-based services* » qui utilisent JSON pour l'encapsulation des données (FIELDING 2000). À l'heure actuelle, travailler avec des services Web basés sur du XML n'est pas évident, puisque la machine virtuelle Java (JVM) présente dans Android™ ne permet pas d'utiliser facilement les bibliothèques disponibles.

6.2.2.1 COTE SERVEUR

L'application a donc pour objectif de se connecter à un service Web afin de récupérer l'ontologie nécessaire à son fonctionnement. Il s'agit du fichier `*.owl` définissant notre modèle de description de service. Par simplicité, c'est la méthode SOAP qui a été retenue. Du côté serveur, il est nécessaire de réceptionner les requêtes émises par l'application et d'y répondre. Une seule fonction est nécessaire pour la réalisation de ce service Web. Ce service Web est défini par le fichier WSDL suivant :

```
<?xml version="1.0"?>
<definitions name="CSP"
              targetNamespace="urn:CSP"
              xmlns:tns="urn:CSP"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:typens="urn:CSP"
              xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
              xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
              xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
       <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:CSP">
              <xsd:complexType name="MyResults">
                     <xsd:all>
                            <xsd:element name="owl" type="xsd:string"/>
                     </xsd:all>
              </xsd:complexType>
       </xsd:schema>
</types>
<message name="getowl">
              <part name="none" type="xsd:int"/>
</message>
<message name="getowlResponse">
              <part name="value" type="typens:MyResults"/>
</message>
<portType name="CspPorts">
              <operation name="getowl">
                     <input message="getowl"/>
                     <output message="getowlResponse"/>
              </operation>
</portType>
<binding name="MyBinding" type="typens:CspPorts">
       <soap:binding style="rpc"
              transport="http://schemas.xmlsoap.org/soap/http" />
              <operation name="getowl">
                     <soap:operation
                     soapAction="http://www.gsem.fr/webservice/WS.wsdl"/>
                     <input>
                            <soap:body use="literal"/>
                     </input>
                     <output>
                            <soap:body use="literal"/>
                     </output>
              </operation>
</binding>
<service name="getOntology">
       <documentation>Service web permettant d identifier l ontologie</documentation>
       <port name="CspPorts" binding="typens:MyBinding">
              <soap:address location="http://www.gsem.fr/webservice/index.php"/>
       </port>
</service>
</definitions>
```

Le serveur est réalisé en PHP. Son fonctionnement est basé sur la librairie "SoapServer" présente dans PHP5. Le fonctionnement de ce serveur se décompose en une classe "handler" qui implémente les fonctions mises à disposition par le service Web. Dans notre cas, elle fournira l'adresse de l'ontologie à utiliser. Cette classe est ensuite transmise au "SoapServer" qui se charge de gérer les requêtes de l'utilisateur ainsi que de la formulation des réponses.

```
<?
// Definition de la classe handler
class ServiceWeb
{
	public function getowl()
	{
		return array('owl'=>'http://..../ontologie.owl');
	}
}
// Desactivation du cache pouvant causer des erreurs en reception client
ini_set("soap.wsdl_cache_enabled", "0");

// Instantiation du serveur defini par le schema wsdl
$server = new SoapServer('WS.wsdl');

// Association de la classe "handler"
$server->setClass('ServiceWeb');

// Gestion des requetes en POST
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
	$server->handle();
}
// Affichage des fonctions disponibles
else {
	echo '<strong>Ce serveur SOAP peut traiter les fonctions suivantes: </strong>';
	echo '<ul>';
	foreach($server -> getFunctions() as $func)
		echo '<li>'.$func.'</li>';
		echo '</ul>';
}
?>
```

6.2.2.2 COTE CLIENT

Nous avons dû effectuer le portage de deux librairies permettant d'effectuer des requêtes à travers un service Web, selon le protocole XML. Il s'agit des librairies RPC et SOAP.

Pour la librairie XML-RPC, nous avons choisi la librairie kXML-RPC (kXML-RPC 2002). Cette librairie est conçue pour Java ME et donc « relativement » légère. De plus cette bibliothèque est disponible sous licence GPL et permet donc la réutilisation, la modification, la redistribution. Parmi ses principaux avantages, nous pouvons citer le fait qu'elle ait été spécialement conçue pour les applications mobiles (GABHART 2007). Nous avons aussi besoin des sources de la bibliothèque kXML (kXML 2005), dont dépend kXML-RPC (kXML-RPC 2002). Là encore on bénéficie d'une licence GPL. Afin de permettre à ces deux librairies de fonctionner avec la machine virtuelle Java proposée par Android™, il a été nécessaire de réaliser un certain nombre de patchs les rendant ainsi pleinement opérationnelles.

Pour ce qui est de la communication SOAP, il suffit d'importer la bibliothèque "ksoap" (kSOAP 2003). Contrairement au XML-RPC, il n'est pas nécessaire de modifier le code de cette librairie, en vue de l'adapter à la plate-forme Android™.

6.2.2.3 RECUPERATION DE L'ONTOLOGIE

Par soucis de cohérence avec le côté serveur, nous avons retenu la méthode SOAP. De plus, pour la réalisation du service Web, nous avons utilisé le protocole SOAP. En outre, vu la simplicité de la requête finale, nous avons choisi de ne pas déployer une librairie complète.

La requête auprès du service Web s'effectue donc de la manière suivante :

```
public class Client {
	private static final String URL = "http://urlduwebservice";
	private static final String WSDL = "http://urlduwsdl";
	private String owl = "";
	private String url = "";
	public Client() {
		String request = "<?xml version=\"1.0\" encoding=\"UTF-8\"?><SOAPENV:
Envelope xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\"><SOAPENV:
Body><SOAP-ENV:getowl><none/></SOAP-ENV:getowl></SOAP-ENV:Body></SOAPENV:
Envelope>";
		HttpClient c = new DefaultHttpClient();
		HttpPost m = new HttpPost(URL);
		String _owl = "";
		try {
			m.setHeader("SOAPAction", WSDL);
			m.setHeader("Content-Type", "text/xml; charset=utf-8");
			m.setEntity(new StringEntity(request));
			ResponseHandler<String> responseHandler = new BasicResponseHandler();
			_owl = c.execute(m, responseHandler);
			c.getConnectionManager().shutdown();
		}
		catch (Exception E) {
		return;
		}

		// On recupere l'url de l'ontologie
		if (_owl.length() > 0) {
			url = _owl.substring(_owl.indexOf("<owl>") + 5,
			_owl.indexOf("</owl>"));
		}

		// On telecharge cette ontologie
		c = new DefaultHttpClient();
		HttpGet g = new HttpGet(url);
		try {
			ResponseHandler<String> responseHandler = new BasicResponseHandler();
			owl = c.execute(g, responseHandler);
			c.getConnectionManager().shutdown();
		}
		catch (Exception E) {
		}
	}

	public String getOWL() {
		return owl;
	}

	public String getURL() {
		return url;
	}
}
```

Cette requête permet de récupérer le fichier ontologie correspondant à notre modèle de description de services. Dans le paragraphe suivant, nous étudions comment cette ontologie est envoyée au moteur découverte.

6.2.2.4 Envoi de l'ontologie au moteur decouverte

Le moteur de découverte n'est en mesure que de lire un fichier. Nous avons donc modifié la classe `OwlTools` (voir ci-dessous) afin de rendre possible la transmission d'un XML sous forme de chaîne de caractères, et donc d'éviter le stockage de fichiers sur le mobile.

```
public OwlTools(String xml, String URL) {
	try {
		inputModel = ModelFactory.createOntologyModel();
		OutpoutModel = ModelFactory.createOntologyModel();
		inputModel.read( new StringReader( xml ), URL, "RDF/XML" );
	}
	catch (Exception e) {
	}
}
```

L'ontologie, une fois traitée par le parseur, permet de générer l'arborescence des menus de l'interface utilisateur. Dans ce qui suit, nous allons découvrir comment cette interface est réalisée.

6.3 Developpement des interfaces

6.3.1 Interface « Utilisateur final »

6.3.1.1 La creation d'interfaces utilisateur sous Android™

Sous Android™, les interfaces utilisateur peuvent être conçues de deux manières différentes:

- ***De manière procédurale***, c'est-à-dire en écrivant du code, Java par exemple, pour créer et manipuler les objets d'interface;
- ***De manière déclarative***, à travers des fichiers XML spécifiant les éléments que l'on désire voir apparaître dans l'interface.

Les deux manières sont valides, mais Google avise d'utiliser les déclarations XML autant que possible. Les raisons motivant ce choix concernent la complexité du code Java et la stabilité du XML. En effet, il est beaucoup plus facile et plus rapide de comprendre un fichier XML, plutôt que le code Java correspondant. De plus, comparée à la syntaxe Java, la syntaxe XML est moins sujette à des changements. Nous utilisons donc la programmation à base de déclarations XML, comme indiqué au paragraphe présentant le concept de « Vue » (§ 6.3.1.1.1.2).

Android™ introduit aussi de nouveaux concepts pour la création d'interfaces:

- **`Activities`** - Les activités représentent la fenêtre ou l'écran affiché l'utilisateur. Les Activités sont les équivalents Android™ d'un formulaire. Pour afficher une interface utilisateur, il faut affecter une vue à une activité.
- **`Views`** - Les vues représentent le type de base d'éléments d'interface utilisateur. Il s'agit d'éléments visuels d'interface, aussi appelés *widgets*. L'ensemble des contrôles interface utilisateur, ainsi que les classes de *layout* dérivent de la classe `Views`.
- **`ViewGroups`** - Les ViewGroups représentent des extensions de la classe `Views` pouvant contenir plusieurs « vues » enfants. En étendant la classe `ViewGroups` il est possible de créer des contrôles composés, formés par un ensemble de « vues » enfants interconnectées.

6.3.1.1.1 PRESENTATION DES CONCEPTS ANDROID™

6.3.1.1.1.1 LE CONCEPT D' « ACTIVITE »

Les activités permettent de créer des écrans d'interface utilisateur. Chaque activité représente une fenêtre que l'application peut présenter à l'utilisateur (comparable à un formulaire dans un environnement bureautique).

Il s'agit donc de créer des activités pour chaque fonctionnalité de l'application. Cela nécessite au moins une fenêtre d'interface de base pour gèrer ces fonctionnalités. Les fonctionnalités représentent des activités telles la saisie de texte ou l'affichage des résultats. Pour passer d'une fenêtre à une autre, il faut soit démarrer une nouvelle activité, soit revenir depuis une activité.

Une activité peut être dans plusieurs états différents :

- ***Activée*** quand elle est au premier plan sur l'écran.
- ***Mise en pause*** quand elle est recouverte par une autre activité qui n'occupe pas l'intégralité de l'écran ou qui est transparente. L'état et les données internes sont conservés mais le système pourra « tuer » l'activité si les ressources système sont extrêmement faibles.
- ***Stoppée*** quand elle est n'est plus visible pour l'utilisateur, l'état et les données internes sont conservées mais le système pourra tuer l'activité s'il a besoin de mémoire pour d'autres tâches.

Le passage d'un état à un autre est géré grâce à des méthodes protégées :

```
void onCreate(Bundle savedInstanceState)
void onStart()
void onRestart()
void onResume()
void onPause()
void onStop()
void onDestroy()
```

La Figure 49 représente le cycle de vie d'une activité.

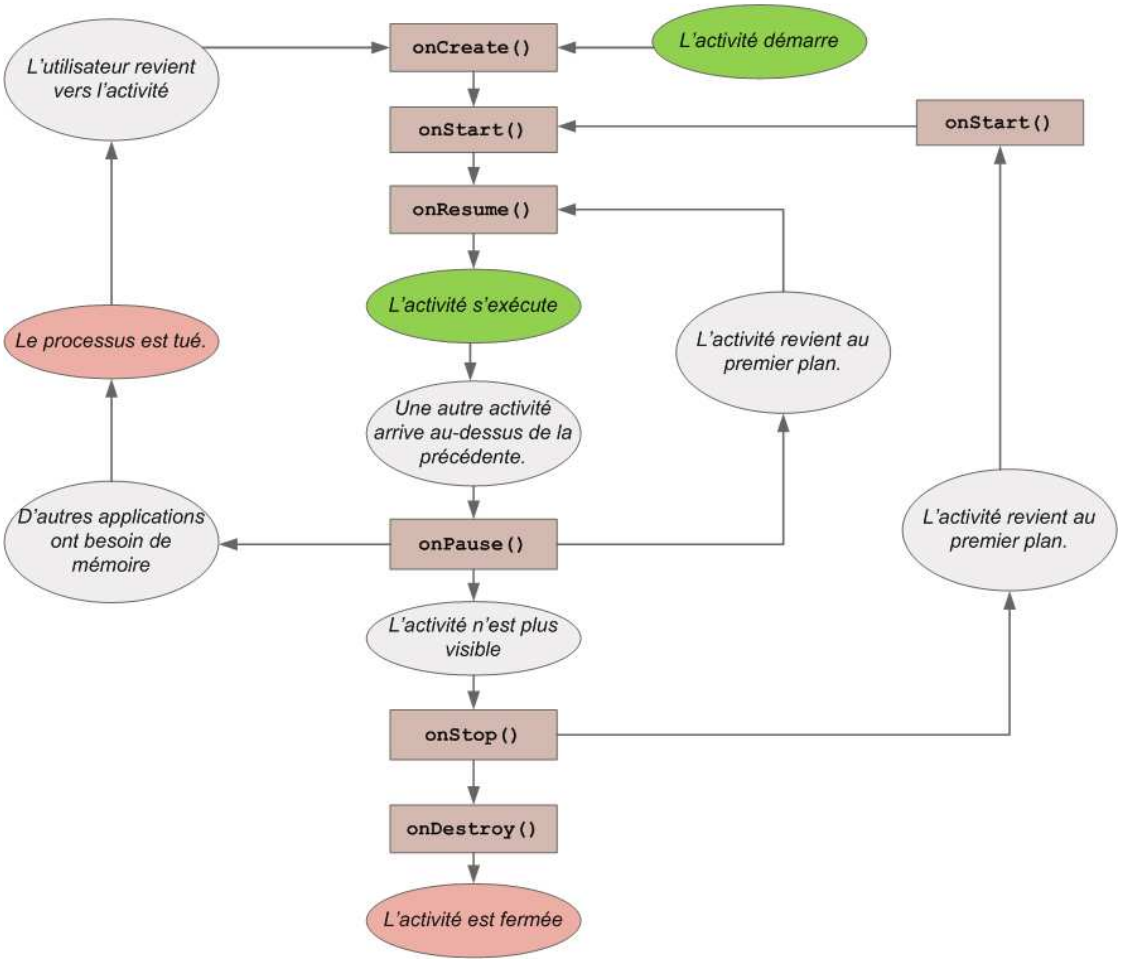

Figure 49. Cycle de vie d'une Activité.

Pour créer une nouvelle activité, il faut étendre la classe `Activity`, en définissant l'interface utilisateur et en implémentant nos fonctionnalités. Un squelette de code permettant de définir une nouvelle activité est présenté ci-dessous.

```
import android.app.Activity ;
import android.os.Bundle ;

public class MyActivity extends Activity {
	//appelée lorsque l'activité est créée pour le première fois
	@Override
	public void onCreate(Bundle icicle) {
		super.onCreate(icicle) ;
	}
}
```

Ceci permet de créer un écran vide, encapsulant les fonctionnalités manipulant l'affichage de la fenêtre. Une activité vide n'était pas très utile, nous devons définir l'interface de l'écran en utilisant des « *Vues* ». C'est ce que nous étudions dans ce qui suit.

6.3.1.1.1.2 Le concept de « Vue »

L'interface graphique d'une activité est définie grâce à des objets de type `View` et `ViewGroup`. Les vues sont des contrôles d'interface utilisateur, qui permettent d'afficher des données et de fournir des interactions avec l'utilisateur. L'ensemble des éléments visuels d'une interface proviennent de la classe `View` et sont appelés « *vues* ».

La classe `ViewGroup` est une extension de la classe `View` conçue pour contenir des vues multiples. Généralement, ces groupements de vues sont utilisés:

- pour construire des composants atomiques réutilisables (ou des *widgets*), ou
- pour gérer la disposition (ou le `layout`) des vues enfants. Les `ViewGroup` utilisés en ce sens sont appelés *dispositions* ou `layouts`.

La Figure 50 illustre la hiérarchie entre ces différents éléments. *Un contrôle* est une extension de la classe `View` implémentant une fonctionnalité basique ou relativement simple. *Un widget* représente à la fois des contrôles composés et des extensions plus complexes de la classe `View`.

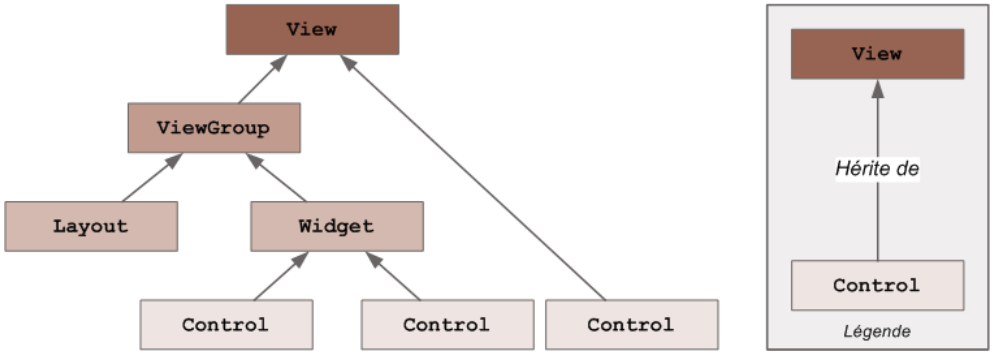

Figure 50. Illustration des relations entre les différents composants d'une Vue.

Pour notre application, la manière la plus simple de créer l'interface est de décrire un layout composé de différents objets `View` dans un fichier XML. Nous donnons ci-dessous un exemple de fichier XML permettant de créer une disposition simple, qui place un élément `TextView` au dessus d'un contrôle `EditText`. La disposition ainsi décrite est basée sur celle d'un `LinearLayout` orienté verticalement.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Saisissez votre texte ci-dessous"
        />
        <EditText
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Le texte est ici"
        />
</LinearLayout>
```

Cependant, si l'on a besoin de rajouter un élément d'interface dynamiquement, on peut le faire directement dans le code Java. C'est ce qui est décrit dans la partie « *Réalisation de l'interface utilisateur* » (voir 6.3.1.2).

6.3.1.1.1.3 Le concept d' « Intention »

Maintenant que nous avons présentés les concepts de base de la création d'interfaces sous Android, nous allons nous intéresser à comment ces éléments sont reliés à l'application et comment ils communiquent entre eux.

Les intentions sont des mécanismes permettant de déclarer l'action à réaliser par rapport à des données. Les intentions sont communément utilisées pour démarrer de nouvelles activités:

- soit de ***manière explicite*** (en spécifiant la classe à charger),

```
Intent intent = new Intent(MyActivity.this, MyOtherActivity.class);
startActivity(intent);
```

- soit de ***manière implicite*** (en demandant d'effectuer une action sur des données)

```
if (quelqueChoseEtrange && caNeSemblePasCorrect) {
       Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:555-2368"));
       startActivity(intent);
}
```

L'utilisation d'intentions pour propager des activités – même à l'intérieur d'une même application – représente un concept fondamental d'Android™. Ceci encourage le découplage des composants, afin de permettre un remplacement sans coutures des éléments de l'application. Ceci représente aussi la base d'un modèle simple permettant étendre les fonctionnalités d'une application.

6.3.1.2 Realisation de l'interface utilisateur

Pour le développement de notre application, nous avons besoin de plusieurs fenêtres, chacune correspondant à une fonctionnalité de l'application. Ces fenêtres sont illustrées par la Figure 51, et sont détaillées dans ce qui suit:

- Une fenêtre pour naviguer dans l'ontologie, notamment pour choisir une catégorie de service - Fenêtres 1A, 1B et 1C;
- Une fenêtre permettant à l'utilisateur de pondérer et de définir des valeurs pour les différentes informations contextuelles associées au type de service choisi (par exemple, pour un service restaurant, l'utilisateur doit pouvoir définir un prix, un type de restaurant, etc.) - Fenêtres 2A et 2B;
- Une fenêtre pour afficher la liste des résultats, triés en fonction de leur niveau de pertinence - Fenêtre 3;
- Une fenêtre pour afficher l'itinéraire vers une adresse donnée - Fenêtre 4;
- Une fenêtre pour afficher les détails concernant un résultat en particulier (par exemple, pour un service restaurant, il faut afficher son adresse, numéro de téléphone, ainsi que ses informations contextuelles) - Fenêtre 5;
- Une fenêtre pour lancer l'appel d'un numéro de téléphone donné – cette fenêtre n'est pas illustrée par la Figure 51, mais elle correspond à l'interface Android™ de base, pour le module « Téléphonie » (voir § 0).

Figure 51. Illustration des différents écrans composant l'interface « *Utilisateur final* ».

Comme nous l'avons précisé, chacune de ces fenêtres implémente une ou plusieurs fonctionnalités bien distinctes. Les fenêtres 1A, 1B et 1C concernent la navigation dans une liste, alors que les fenêtres 2A et 2B impliquent la définition de *layouts* personnalisés et reposent sur l'utilisation de *sliders*. Nous détaillons la réalisation de ces fonctionnalités dans les paragraphes suivants.

6.3.1.2.1 NAVIGATION DANS UNE LISTE

Il existe une activité spéciale qui permet de gérer un affichage par liste. Notre activité doit hériter de l'activité `ListActivity`, afin d'avoir accès à un ensemble de méthodes permettant la gestion des interactions de l'utilisateur avec les différents éléments de la liste. Il est ainsi possible de définir le résultat d'un clic sur un élément de la liste.

Les données sont importées depuis l'ontologie sous la forme de chaînes de caractères.

Pour pouvoir afficher ces données, il faut utiliser un adapter qui va mettre les données sous une forme convenant à l'affichage.

```
ArrayAdapter<String> itemsList = new ArrayAdapter<String>(
        this,
        android.R.layout.simple_list_item_1,
        items);
setListAdapter(itemsList);
```

Le code affiché ci-dessus utilise les éléments suivants:

- `this` correspond à l'activité en cours.
- `android.R.layout.simple_list_item_1` correspond au *layout* que l'on souhaite utiliser pour l'interface. Ici, il s'agit d'un *layout* standard, correspondant à un mode liste.
- `items` représente notre tableau de chaînes de caractères contenant les catégories de services.

Une fois l'adapter créé, on affiche les données dans la liste grâce à la méthode `setListAdapter()`. La Figure 52 illustre le résultat obtenu.

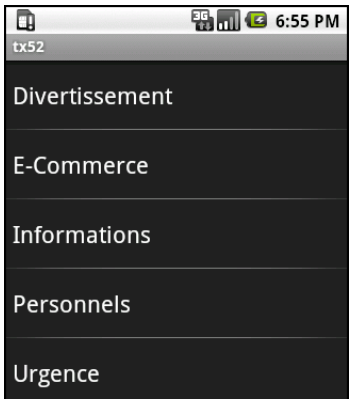

Figure 52. Liste des catégories de services définies dans notre modèle.

Notre activité hérite de la classe `ListActivity` ce qui nous permet d'utiliser les méthodes de cette classe, notamment pour capturer les événements de clic. Avec la méthode `onListItemClick()` il est possible de savoir, par exemple, sur quel élément l'utilisateur clique. Il est dès lors possible de naviguer dans la hiérarchie de profils service définie par notre modèle ontologique (voir Figure 53).

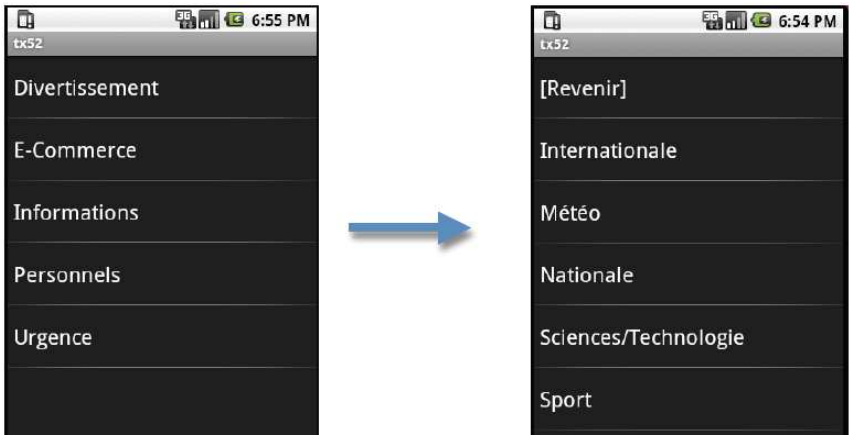

Figure 53. Navigation parmi les catégories de services - ici, affichage des différents types d'informations (nous sommes ici dans la même activité, nous avons seulement rafraîchi la liste).

Dans le déroulement de l'application, l'utilisateur doit pouvoir choisir les détails vis-à-vis d'un service donné comme par exemple l'importance du prix d'un restaurant. Le comportement de cette activité est différent de la navigation dans l'ontologie, nous allons donc créer une activité supplémentaire.

6.3.1.2.2 AJOUT ET APPEL D'UNE NOUVELLE ACTIVITE

Pour ajouter une activité, il faut tout d'abord créer une nouvelle classe qui hérite de la classe `Activity`. Il faut aussi modifier le fichier `AndroidManifest.xml` dans lequel sont répertoriées toutes les activités et les autorisations de chacune des activités.

Le code ci-dessous présente un extrait de ce fichier. Le fichier complet est présent dans l'Annexe D. Fichiers de l'interface « *Utilisateur final* ».

```
<manifest ...
	<application ...>
		<activity ... android:name=".activity.ServiceList"> ... </activity>
		<activity android:name=".activity.ServiceDetail"></activity>
	</application>
</manifest>
```

On peut voir ici que notre application contient pour le moment deux activités : `ServiceList` et `ServiceDetail`.

Pour appeler cette nouvelle activité, on utilise une intention (appelée de manière implicite) et la méthode `startActivity()` :

```
Intent i = new Intent(this, ServiceDetail.class);
startActivity(i);
```

Ainsi, nous avons créée une nouvelle activité, appelée `ServiceDetail` et qui est démarrée par l'intention *i*. Dans ce qui suit, nous expliquons la création d'un *layout* personnalisé pour cette activité.

6.3.1.2.3 CREATION D'UN LAYOUT PERSONNALISE

Pour notre nouvelle activité `ServiceDetail`, nous avons besoin de créer une nouvelle interface, et ce à partir de « *rien* ». En effet, nous devons créer un affichage personnalisé, comprenant une liste composée de champs de texte et de sliders (comme l'indique l'Ecran 2A, dans la Figure 51). Pour ce faire, nous décrivons notre interface dans un nouveau fichier `*.xml`, stocké dans le répertoire `/res/layout/details.xml`.

Le code ci-dessous présente un extrait de ce fichier. Le fichier complet est présent en annexe (voir Annexe D. Fichiers de l'interface « *Utilisateur final* »).

```
<LinearLayout
	<TextView android:text="TextView01"
		android:id="@+id/TextView01"
		android:layout_width="wrap_content"
		android:layout_height="wrap_content"></TextView>
	<fr.utbm.android.view.Slider
		android:id="@+id/slider"
		android:layout_width="fill_parent"
		android:layout_height="wrap_content"
		android:text="Slide me..." />
</LinearLayout>
```

Nous voyons ici que notre layout est constitué d'un objet de type `TextView` et d'un objet de type `Slider`. Ces deux objets ont pour classe mère la classe `View`. Nous définissons les attributs de chaque objet, notamment son identifiant (propriété `id`), sa largeur (propriété `layout_width`) et son hauteur (propriété `layout_height`).

À partir de notre activité, il nous suffit maintenant d'appliquer ce *layout*, à travers la commande :

```
setContentView(R.layout.details);
```

6.3.1.2.4 LE SLIDER

Android™ implémente par défaut une vue de type `SeekBar`, dont le fonctionnement s'apparente à un *slider*. Après avoir instancié notre *slider*, il est possible de capturer les événements dus aux mouvements et ainsi récupérer la valeur correspondant à la position du curseur. Ceci est illustré par le code suivant:

```
// On active l'écoute des événements
mSlider.setOnPositionChangedListener( new OnPositionChangedListener() {

	// Pendant le mouvement
	public void onPositionChanged(Slider slider, int oldPosition, int newPosition) {
		int newPos = newPosition; // position en temps réel
	}

	// Quand le mouvement est terminé
	public void onPositionChangeCompleted() {
		int newPos = mSlider.pos; // position à la fin du mouvement
	}
});
```

La Figure 54 affiche le résultat ainsi obtenu.

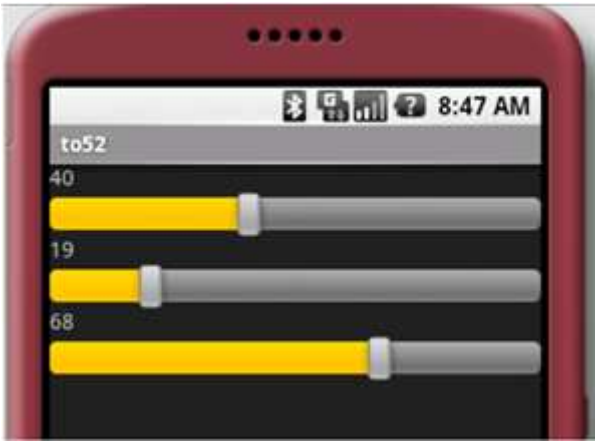

Figure 54. Récupération de la valeur du *slider*.

Nous observons que les valeurs sont comprises entre 0 et 100. Or, nous n'avons nul besoin d'une telle précision, car nous n'avions défini que 5 niveaux de pondération, pouvant être associés à une information contextuelle. Nous normalisons donc les valeurs selon 5 paliers de 20 unités. Une valeur de « 40 » va correspondre à un niveau de pondération de « 2 », et une valeur de « 68 » va correspondre à un niveau « 4 » de pondération (voir Fenêtre 2B, dans la Figure 51).

6.3.1.2.5 INTEGRATION DU MODULE "AFFICHAGE CARTE"

Grâce aux API fournies par le SDK Android™, il est possible d'utiliser l'application GoogleMaps®, pour afficher une adresse ou un itinéraire vers une adresse donnée.

Il faut d'abord obtenir la longitude et la latitude correspondant à une adresse donnée. Pour ce faire, nous utilisons le geocoder intégré à l'application GoogleMaps®:

```
if (address != null) {
	Geocoder geocoder = new Geocoder(context, Locale.getDefault());
	try {
		addressResult = geocoder.getFromLocationName(address, 1);
		if (!addressResult.isEmpty()) {
			Address resultAddress = addressResult.get(0);
			location.setLatitude(resultAddress.getLatitude());
			location.setLongitude(resultAddress.getLongitude());
		}
	} catch (IOException e) {
		Log.d("Lieu non déterminé", e.getMessage());
	}
}
```

Il reste ensuite à appeler l'activité `map`, qui permet d'afficher la position correspondant à l'adresse initiale:

```
try {
	final Intent myIntent = new Intent(android.content.Intent.ACTION_VIEW,
	Uri.parse("geo:"+location.getLatitude + "," + location.getLongitude));
	startActivity(myIntent);
}
catch (URISyntaxException e) { }
```

La Figure 55 illustre le résultat obtenu.

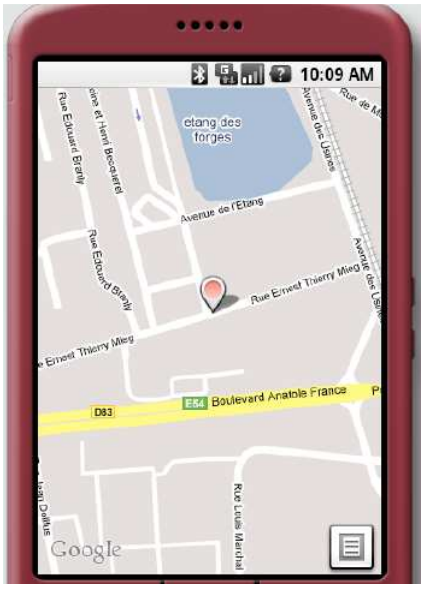

Figure 55. Affichage d'une adresse grâce au module GoogleMaps®.

6.3.1.2.6 Integration du module « Telephonie »

Le SDK fourni par Google offre une activité déjà prête pour pouvoir simplement appeler un numéro de téléphone. Le fonctionnement est le même que pour les autres modules, nous utilisons une intention pour appeler cette activité.

```
public void callNumber(String number) {
	Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:" + number));
	startActivity(intent);
}
```

Ici, l'action exécutée est ACTION_DIAL. Quand cette fonction est exécutée, le module de téléphonie est lancé et le numéro est composé directement (voir Figure 56).

Figure 56. Composition d'un numéro de téléphone.

6.3.1.3 Conclusions concernant le developpement de l'interface « Utilisateur Final »

Le principal avantage du système Android™ est d'être très modulaire, ce qui facilite grandement l'intégration des différents composants et fonctionnalités. Toutefois, au moment de la rédaction de ce chapitre, le développement de l'application n'est pas encore terminé. Nous sommes en train de terminer la description de l'affichage personnalisé correspondant à la création de la requête utilisateur. Par la suite, nous n'aurons plus qu'à réaliser différents tests pour vérifier le bon fonctionnement de l'application. La durée restante est estimée à environ 6 semaines.

6.3.2 Interface « Fournisseur de services »

Nous avons présenté l'interface permettant la création de requêtes utilisateur. Ces requêtes sont adressées à un registre de services, contenant un ensemble de descriptions de services. En d'autres termes, nous avons développé une interface permettant de rechercher à travers le registre, mais nous ne nous sommes pas intéressés à la manière de remplir ce registre avec des descriptions de services conformes à notre modèle. Pour le moment, il n'existe pas d'outil permettant de créer une telle description de service. Il faut donc développer une interface claire et intuitive, qui puisse permettre aux fournisseurs de services de créer de telles descriptions, puis de les enregistrer dans le registre de services.

Traditionnellement, la découverte et l'invocation de services Web reposent sur l'usage de fichiers WSDL. Comme défini précédemment (voir § 2.1.2.3), un document WSDL spécifie de manière abstraite, les opérations et messages définissant un service Web. Des nos jours, lors du processus de

découverte de services Web, le client recherche les services pertinents parmi un ensemble des documents WSDL publiés dans un annuaire de type UDDI. Chaque document WSDL correspond à la spécification d'un service Web. Donc, il apparaît évident d'intégrer les descriptions WSDL dans notre approche. En effet, la plupart des fournisseurs de services disposent aujourd'hui de descriptions de leurs services, conformément au standard WSDL. Nous souhaitons nous baser sur ces descriptions, puis de les compléter, selon notre modèle de description.

Il s'agit donc de proposer au fournisseur de service un formulaire, lui permettant de préciser le fichier WSDL contenant les informations service, telles le nom ou l'adresse du service (voir Figure 39). Nous supposons que le fournisseur de service possède déjà un tel fichier WSDL. Une fois ce fichier chargé, les informations qu'il contient sont extraites et remplissent les champs correspondants du formulaire. Les autres champs du formulaire sont obtenus par requêtes SPARQL depuis notre ontologie de description de service. Il s'agit donc d'un formulaire dynamique, construit à partir du fichier WSDL et de l'ontologie. Dans les paragraphes suivants, nous présentons sa réalisation et son fonctionnement.

6.3.2.1 Recuperation des informations WSDL

Pour récupérer les informations service contenues dans un fichier WSDL, il faut pouvoir analyser ce fichier. Or, pour ce faire, il existe plusieurs programmes. De plus, il existe même des programmes permettant de traduire une description WSDL en une description OWL-S.

Nous avons choisi d'utiliser le programme WSDL2OWL-S (GIAMPAPA 2005), développé par le laboratoire Softagent Lab, de l'université Carnegie Mellon[28]. Ce programme fournit une traduction partielle entre les descriptions WSDL et OWL-S d'un même service. Les résultats d'une telle traduction consistent en une spécification complète de la classe `ServiceGrounding`, une spécification partielle des classes `ServiceProcess` et `ServiceProfile`. Cette spécification partielle s'explique par les différences des informations contenues dans une description OWL-S et une description WSDL. En effet, une description WSDL ne fournit aucune information quand à la composition de processus, donc la classe `ServiceProcess` générée ne pourra contenir de telles informations. De la même manière, les fonctionnalités du service ne sont pas décrites dans un fichier WSDL. Cela veut dire que, pour être complète, la classe `ServiceProfile` générée doit être complétée par l'utilisateur (PAOLUCCI 2003).

Nous reprenons donc les sources de ce programme, puis nous les adaptons à notre application. Ayant étendu la description de la classe `ServiceProfile`, à travers la création de catégories de services, il est nécessaire de préciser quel type de profil de service est visé. Pour ce faire, le fournisseur de services commence par choisir le type de service qu'il souhaite spécifier. Une fois défini le profil de service, le fournisseur spécifie le fichier WSDL décrivant son service. Nous analysons donc ce fichier, puis en extrayons les informations nécessaires, telles le nom et l'adresse du service. La Figure 57 présente la traduction en OWL-S des informations contenues dans un fichier WSDL.

[28] Voir : http://www.cs.cmu.edu/~softagents/

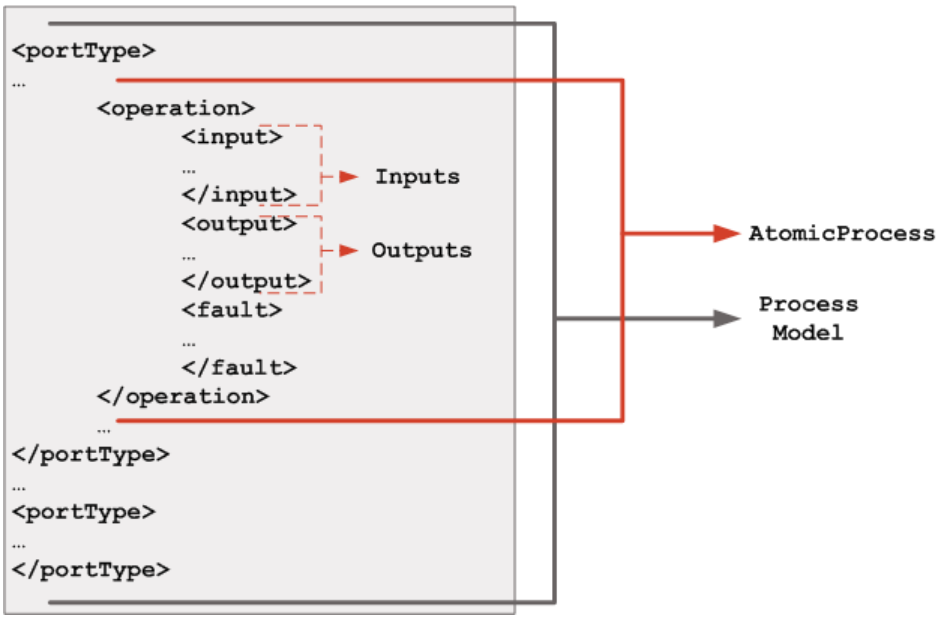

Figure 57. Traduction WSDL en OWL-S (PAOLUCCI 2003).

Nous nous intéressons maintenant à la manière de récupérer les informations contextuelles depuis notre modèle de description de services.

6.3.2.2 RECUPERATION DES INFORMATIONS CONTEXTUELLES

Nous venons de présenter comment les informations WSDL sont obtenues depuis le fichier correspondant. Nous avons toutefois précisé qu'avant d'obtenir ces informations, le fournisseur de services doit spécifier le profil du service qu'il souhaite décrire. Or, pour ce faire, il est nécessaire de parcourir notre modèle ontologique, puis d'en tirer les différents types de profils service définis. Le programme doit donc construire le graphe RDF illustré par la Figure 58. Afin d'améliorer la lisibilité du graphique, nous n'avons pas affiché l'ensemble des propriétés contextuelles définies pour un service de restauration.

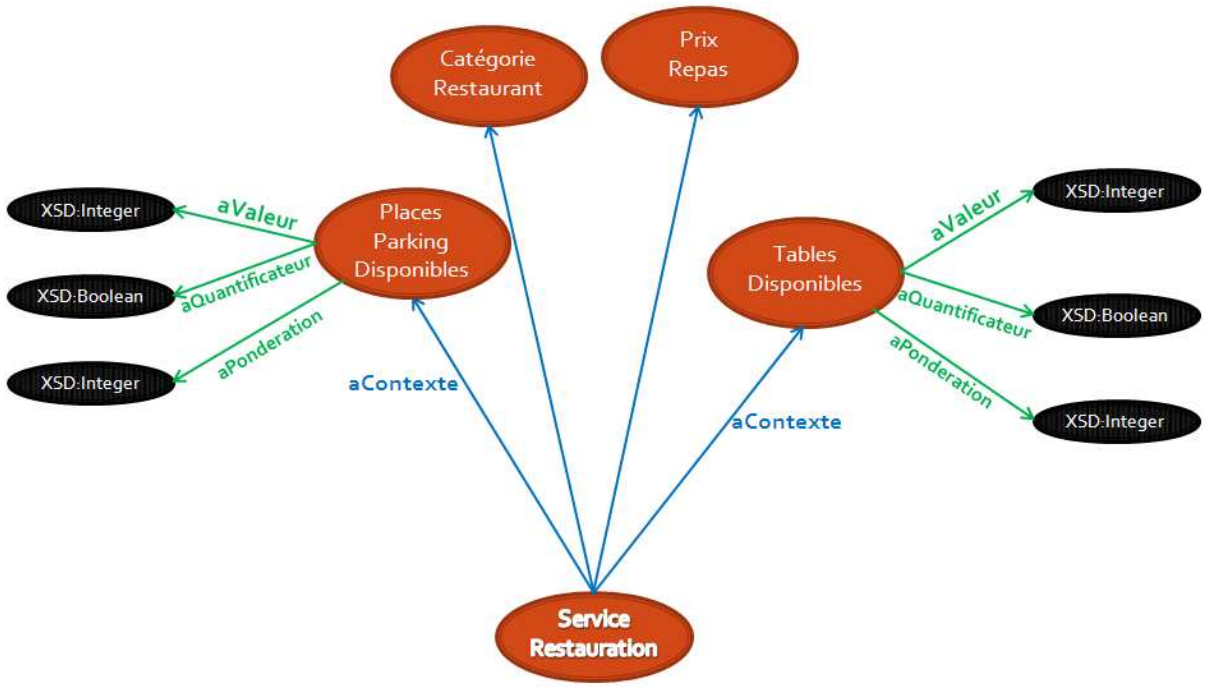

Figure 58. Graphe RDF illustrant les informations contextuelles d'un service de restauration.

Pour manipuler les fichiers ontologie, nous utilisons encore l'API Jena. Nous créons le package OWL qui contient les fichiers: `ManipulerOWL.java` (voir Annexe C. Fichiers de l'interface « Fournisseur de services »), `MODELONT.java`, `PropertyService.java` et `ServiceOWL.java`. La classe `ServiceOWL` repose sur le modèle d'un nœud RDF, tel que défini par l'API Jena. Cette classe est utilisée pour représenter les différents types de services définis par notre modèle. Les propriétés (ou informations contextuelles) associées à chaque type de service sont représentées grâce à la classe `PropertyService`.

Pour parcourir l'ontologie et récupérer les informations contextuelles qui nous intéressent, nous créons la classe `ManipulerOWL`. Elle utilise l'API Jena et son moteur ARQL pour les requêtes SPARQL. SPARQL (PRUD'HOMMEAUX 2008) représente un langage et un ensemble de protocoles de requêtes, permettant l'interrogation de métadonnées, en utilisant des triplets RDF (voir § 2.2.3.2.1). Une requête SPARQL est constituée d'une chaîne de caractères délimitée par des accolades {}.

La requête suivante permet d'obtenir la liste des profils service définis dans notre modèle:

```
     "PREFIX owl: <http://www.w3.org/2002/07/owl#> " +
     "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> "+
     "PREFIX prof: <http://www.daml.org/services/owl-s/1.2/Profile.owl#Profile>"+
     "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"+
     "SELECT ?subClass " +
     "WHERE {" +
     "   ?subClass rdfs:subClassOf prof" . " +
     "   }";
```

Les trois premières lignes précisent l'espace de nom (le *namespace*) pour les préfixes `rdfs`, `prof`, et `rdf`. La clause `SELECT` nomme la variable qui sera renvoyée, ici il s'agit de la variable `subClass` (les variables SPARQL sont toujours précédées par ? (PRUD'HOMMEAUX 2008)). La clause `WHERE` définit la clause RDF que l'on cherche à satisfaire, sous la forme d'un triplet RDF du type « *Sujet, Prédicat, Objet* ». Ici, il s'agit de renvoyer l'ensemble des sous-classes de la classe `Profile`, dont l'URI (`<http://www.daml.org/services/owl-s/1.2/Profile.owl#Profile>`) est associée au préfixe `prof`.

De la même manière, pour obtenir les informations contextuelles associées à un service donné, nous utilisons la requête suivante:

```
     "PREFIX owl: <http://www.w3.org/2002/07/owl#> " +
     "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> "+
     "SELECT ?subClass ?proper ?value " +
     "WHERE {" +
     "   ?subClass rdfs:subClassOf <http://www.daml.org/services/owl-s/1.2/
Service.owl#"+NameService+">; " +
     "   rdfs:subClassOf [ a owl:Restriction; owl:onProperty ?proper; owl:allValuesFrom
?value].}";
```

Ici, nous souhaitons renvoyer l'ensemble des sous-classes de la classe `NameService`, puis pour chacune de ces sous-classes, on renvoie le type de valeur attendu, défini par la restriction OWL présente dans l'ontologie. En pratique, les classes visées par cette requête sont les sous-classes de la classe `ServiceContext`. Chacune de ces classes a trois propriétés: `aValeur`, `aPonderation` et `aQuantificateur`. Ici, ce qui nous intéresse c'est la restriction défini pour la propriété

`aValeur`, qui détermine le type de valeur attendu pour l'information contextuelle (ou la propriété service) considérée (voir § 4.3.2.2). Cette requête SPARQL permet donc de récupérer l'ensemble des informations contextuelles définies pour un contexte de service donné (par exemple `Service_Restaurant_Context`), ainsi que les types de valeurs attendus pour ces informations.

6.3.2.3 CREATION DE L'INTERFACE « FOURNISSEUR DE SERVICE »

Pour créer notre interface, nous utilisons la technologie JSP (*Java Server Pages*) (JSP 2009). Les JSP représentent des pages Web générées dynamiquement, dont la présentation (code HTML) est séparée des traitements (code Java). Au premier appel d'une page JSP, le moteur JSP génère et compile automatiquement une Servlet Java, permettant la génération de la page Web proprement dite (JSP 2009). La Servlet reprend entièrement le code HTML, et exécute le code Java, définis dans la classe Java associée. Pour notre interface, nous utilisons trois Servlets:

- une pour afficher la liste des types de services (fichier `service_jsp.java`) ,
- une pour générer le formulaire permettant de définir les valeurs des différentes propriétés contextuelles définies pour le type de service choisi (fichier `recepservice_jsp.java`)
- une pour enregistrer un fichier OWL à partir des informations contenues dans le formulaire (fichier `CreateOWL.java`)

6.3.2.4 APERÇU DE L'INTERFACE "FOURNISSEUR DE SERVICE"

Ayant présenté les différentes technologies et méthodes utilisées pour mettre en place notre interface, nous nous intéressons maintenant au résultat final.

Le premier écran présente la liste des catégories de services, telles que définies par notre modèle de description de service (Figure 59).

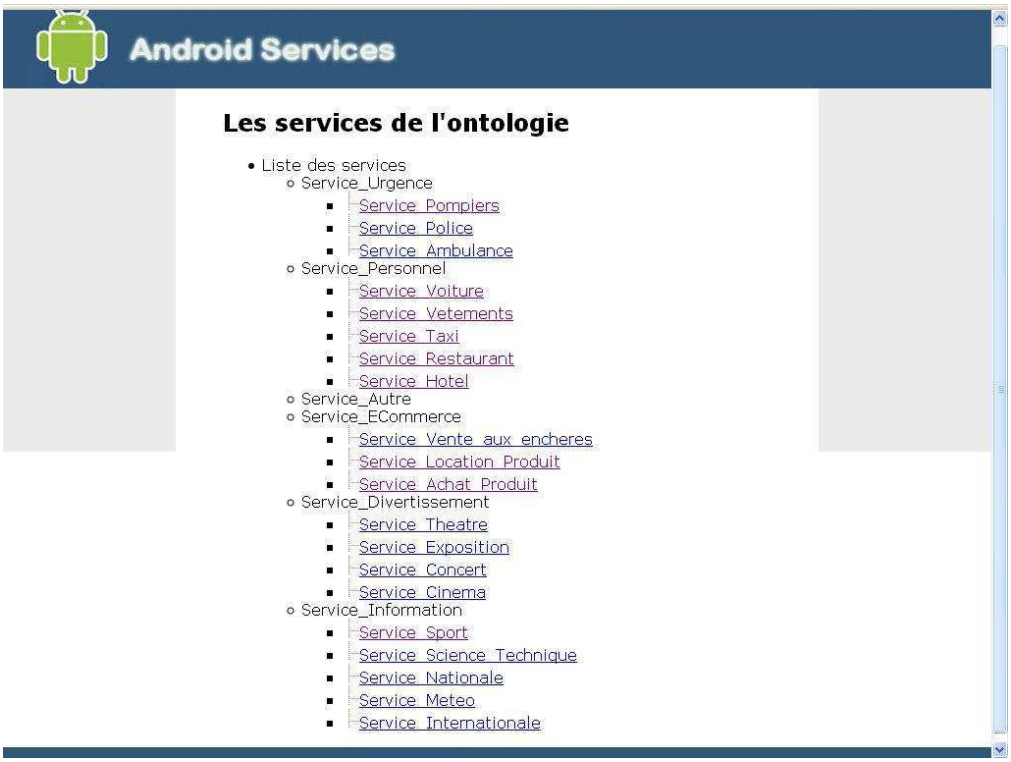

Figure 59. Affichage des catégories de services définies dans le modèle ontologique.

Pour choisir la catégorie de service qu'il souhaite renseigner, le fournisseur de service doit cliquer sur le nom de cette catégorie. Cela a pour effet de générer un formulaire contenant le nom des informations contextuelles définissant le service, avec pour chaque information le type de données attendu, et un champ de saisie. Le fournisseur de service peut ainsi saisir les valeurs des différentes propriétés contextuelles du service. Ce formulaire est aussi utilisé pour spécifier le fichier WSDL associé au service. La Figure 60 illustre le formulaire généré pour un service de type restaurant.

Figure 60. Formulaire permettant de publier un service.

Lorsque le fournisseur de service a terminé la saisie, il clique sur le bouton « *Créer Service* ». Ceci a pour effet de générer un fichier OWL, contenant les informations extraites depuis le fichier WSDL et les propriétés contextuelles du service, instanciées avec les valeurs saisies par le fournisseur de service. Un exemple d'un tel fichier est présenté par les annexes (voir Annexe C. Fichiers de l'interface « Fournisseur de services »).

Dans ce qui suit, nous discutons des limitations de cette approche et des évolutions envisageables.

6.3.2.5 Conclusions concernant le developpement de l'interface « Fournisseur de service »

Pour le moment, les valeurs des différentes propriétés sont saisies manuellement, mais il est aisé d'imaginer une future évolution de l'application, dans laquelle ces valeurs seraient transmises par des capteurs (notamment pour les informations contextuelles dynamiques, telles le nombre de places disponibles dans le parking d'un restaurant).

De plus, en l'état actuel, lors de la génération de la description de service, le programme génère l'ontologie dans son intégralité, en y ajoutant seulement des valeurs pour les propriétés modifiées. Il serait intéressant de ne générer que les classes utilisées dans la description du service, afin de minimiser la taille du fichier final. De cette manière, le temps de traitement d'un tel fichier est réduit. De plus, plus la taille du fichier est réduite, plus le fichier est adapté à être utilisé sur une plate-forme mobile. Il s'agit donc d'optimiser la génération du fichier final, décrivant le service, en vue de son utilisation sur une plate-forme mobile, de type Android™.

Une dernière « limitation » de cette approche est constituée par le besoin d'un serveur, où l'ensemble des descriptions ainsi générées sont stockées et enregistrées. Ceci constitue le registre de services, qui sera interrogé par le « *Moteur Découverte* ».

CHAPITRE 7.

EVALUATION DU PROTOTYPE

Cette partie a pour objectif d'évaluer le fonctionnement du prototype proposé dans cette thèse. Ce prototype est évalué en regard de la principale contrainte pour une application mobile, le temps. Il s'agit ici d'étudier le temps d'exécution d'une requête utilisateur, et ce, en prenant en compte, comme paramètre, le nombre de services à évaluer pour cette requête.

7.1 MESURES DE PERFORMANCE

7.1.1 MESURES DE RAPIDITE

7.1.1.1 ETAPES COMPOSANT L'EXECUTION D'UNE REQUETE

Avant de présenter notre protocole de test, nous rappelons que dans notre approche, les services sont premièrement sélectionnés en fonction de leur catégorie dans la hiérarchie de services définie au paragraphe 4.3.1.3. Un algorithme est ensuite appliqué à l'ensemble des services ainsi sélectionnés, et ce afin de calculer le degré de pertinence de chaque service, par rapport aux critères spécifiés par la requête utilisateur. L'exécution d'une requête utilisateur est donc décomposée selon les étapes suivantes :

- ***Etape 1*** – sélection et chargement des services présents dans la catégorie spécifiée par l'utilisateur dans sa requête.
- ***Etape 2*** – récupération des valeurs de chaque service, pour les propriétés non-fonctionnelles spécifiées par l'utilisateur dans sa requête. Cette étape a pour but de construire le tableau d'ordonnancement illustré par le Tableau 14.
- ***Etape 3*** – calcul du degré de pertinence de chaque service par rapport à la requête de l'utilisateur. Cette étape consiste à appliquer l'algorithme défini au Tableau 15, pour l'ensemble des services sélectionnés durant la première étape.

Le temps total d'exécution d'une requête utilisateur est dès lors la somme des temps correspondants aux étapes listées ci-dessous.

7.1.1.2 PROTOCOLE DE TEST

L'interface « *Utilisateur final* » n'étant pas encore terminée au moment de l'évaluation, nous avons mis en place une interface JSP permettant de créer une requête utilisateur. Une fois la requête créée et validée, nous demandons au programme de nous afficher l'heure à chaque fois qu'il débute une nouvelle étape, parmi celles listées ci-dessus. La différence entre l'heure de fin et l'heure de début d'une étape, nous permet d'obtenir la durée de cette étape. Pour ces tests, nous utilisons un ordinateur portable MacBook Pro, avec un processeur Intel Core 2 Duo, tournant à la fréquence de 2,53 GHz et disposant de 3Gb de mémoire RAM.

Nous étudions ces temps en fonction du nombre de services à traiter pour la requête. En effet, une fois la requête utilisateur validée, le programme cherche à déterminer combien de services permettent de répondre à la requête (de manière partielle ou pas). Notre algorithme de comparaison est appliqué à cet ensemble de services.

7.1.1.3 RESULTATS

Pour chacun de ces cas, nous effectuons plusieurs tests, en calculant la durée de chaque étape. Nous effectuons des tests où le nombre de services présents dans l'annuaire varie. Nous choisissons de tester les cas où l'annuaire contient : un, trois, cing et dix services. Pour les tests comprennant respectivement un et trois services dans l'annuaire, nous effectuons deux séries de tests.

Les résultats ainsi obtenus sont illustrés dans le Tableau 20.

Nombre services	Etape 1	Etape 2	Etape 3	Durée totale
1	00:00:00,162	00:00:00,023	00:00:00,014	00:00:00,199
1	00:00:00,158	00:00:00,005	00:00:00,011	00:00:00,174
3	00:00:00,444	00:00:00,021	00:00:00,018	00:00:00,465
3	00:00:00,461	00:00:00,018	00:00:00,016	00:00:00,479
5	00:00:00,795	00:00:00,025	00:00:00,014	00:00:00,834
10	00:00:01,533	00:00:00,070	00:00:00,045	00:00:01,648

Tableau 20. Premier test – Temps d'exécution d'une requête utilisateur.

A la suite de ce premier test, nous pouvons faire les remarques suivantes :

- La durée du premier test avec un seul service (la première ligne du Tableau 20) est supérieure à la durée des deux autres tests effectués avec un seul service (ligne 2 du Tableau 20). Ceci s'explique par le chargement en mémoire de l'ensemble des librairies nécessaires à notre prototype. Ce premier temps d'exécution est plus long que les autres temps, car ce test représente la phase d'initialisation du prototype. C'est la raison pour laquelle nous avons effectués plusieurs tests dans ce cas de figre. D'ailleurs, c'est pour cette raison que les temps obtenus, pour le deuxième test avec un seul service, sont inférieurs aux temps obtenus lors du premier test.
- La durée d'exécution de l'algorithme issu de notre travail de recherche (Etape 3) semble être constante et ne représente pas la principale partie du temps total d'exécution d'une requête. Pour mieux illustrer ceci, nous calculons la part (en pourcentage) de chaque temps, par rapport à la durée totale (Tableau 21). Nous observons alors que le chargement des services en mémoire occupe le plus de temps.

Nombre services	Etape 1	Etape 2	Etape 3
1	81,41	11,56	7,04
1	90,80	2,87	6,32
3	95,48	4,52	3,87
3	96,24	3,76	3,34
5	95,32	3,00	1,68
10	93,02	4,25	2,73

Tableau 21. Premier test – Répartition (en %) des durées correspondant à chacune des étapes.

Ce premier test permet de mettre en évidence l'efficacité de nottre algorithme. Son temps d'exécution est compatible avec les contraintes de temps des algorithmes d'appareillement d'ontologies (EUZENAT 2007). En effet, dans (EUZENAT 2007), les auteurs présentent une évaluation détaillée de ces différents algorithmes, en termes de pertinence, rapidité et efficacité. Même s'ils ne donnent que quelques mesures de rapidité, ils indiquent le besoin d'avoir des algorithmes rapides notamment lorsqu'il s'agit de répondre à une requête utilisateur. Dans notre cas, les temps d'exécution de notre algorithme de comparaison ne représentent qu'un faible pourcentage du temps total d'exécution de la requête. Cela prouve sa rapidité.

Nous souhaitons toutefois essayer d'améliorer ces durées d'exécution des requêtes. Pour ces tests, les requêtes et l'algorithme de comparaison sont exécutés sur la même machine, ce qui réduit déjà considérablement le temps d'exécution total. En effet, lors de l'implémentation de l'interface mobile, les utilisateurs seront tributaires des caractéristiques de leur connexion réseau (Edge/GPRS/UMTS ou encore WiFi, voire même WiMax). Ces technologies ne possèdent pas les mêmes caractéristiques et il

est fort probable que l'utilisateur aura à attendre plus ou moins longtemps, pendant que l'interface mobile communique avec le registre et le serveur.

Nous voulons donc réduire ces temps d'exécution à leur valeur minimale. Pour ce faire, nous modifions la description des fichiers, de manière à ne contenir que les concepts utilisés par notre prototype. En d'autres termes, nous supprimons de notre modèle de description de services les concepts OWL-S qui ne sont pas nécessaires à notre prototype. Il s'agit de concepts tels `ServiceModel` (nous n'avons pas besoin de modéliser les traitements du service, puisque l'ensemble des services qui nous concernent n'effectuent aucun traitement) ou encore les concepts ayant un lien avec la spécification de règles logiques (`Expression` et `LogicLanguage`). Nous réduisons ainsi la taille du fichier *.owl décrivant chaque service. Dans la première série de tests, ce fichier faisait environ 196kb, alors qu'après réduction il n'en fait plus que 112kb.

Nous relançons donc une nouvelle série de tests, avec les mêmes paramètres que précédemment. Seule la taille des fichiers à traiter a été réduite, afin de déterminer le gain de temps que cela induit. Les résultats de ces tests sont indiqués dans le Tableau 22.

Nombre services	*Etape 1*	*Etape 2*	*Etape 3*	*Durée totale*
1	00:00:00,055	00:00:00,005	00:00:00,007	00:00:00,067
1	00:00:00,056	00:00:00,004	00:00:00,008	00:00:00,068
3	00:00:00,173	00:00:00,017	00:00:00,011	00:00:00,201
3	00:00:00,165	00:00:00,014	00:00:00,011	00:00:00,190
5	00:00:00,295	00:00:00,027	00:00:00,013	00:00:00,335
10	00:00:00,715	00:00:00,054	00:00:00,019	00:00:00,788

Tableau 22. Deuxième test – Temps d'exécution d'une requête utilisateur.

Les durées d'exécution des requêtes sont ici beaucoup plus courtes. La durée d'exécution de l'algorithme de comparaison demeure quasi-constante, par rapport à la première série de tests. La durée de chargement des services est cependant réduite d'environ la moitié.

Nous construisons le Tableau 23 afin d'illustrer la comparaison entre les temps obtenus dans les deux séries de tests. A titre de repère, nous notons qu'entre les deux séries de tests, nous avons réduit la taille des fichiers services d'environ 42,8%, en la diminuant de 84kb.

	Gain Etape 1		*Gain Etape 2*		*Gain Etape 3*		*Gain Total*	
Nombre de services	(en s)	(%)	(en s)	(%)	(en s)	(%)	(en s)	(%)
1	00:00:00,160	74,42	00:00:00,007	58,33	00:00:00,060	89,55	00:00:00,227	77,21
1	00:00:00,106	65,43	00:00:00,019	82,61	00:00:00,006	42,86	00:00:00,131	65,83
3	00:00:00,271	61,04	00:00:00,004	19,05	00:00:00,007	38,89	00:00:00,264	56,77
3	00:00:00,296	64,21	00:00:00,004	22,22	00:00:00,005	31,25	00:00:00,289	60,33
5	00:00:00,492	62,52	00:00:00,006	18,18	00:00:00,003	18,75	00:00:00,501	59,93
10	00:00:00,818	53,36	00:00:00,016	22,86	00:00:00,026	57,78	00:00:00,860	52,18

Tableau 23. Comparaison entre les deux séries de tests.

Nous remarquons d'importantes réductions des temps d'exécution pour l'ensemble des tests composant cette deuxième série. Les réductions les plus considérables sont toutefois atteintes pour

l'Etape 1, autrement dit pour le chargement des descriptions de services (fichiers *.owl). En pourcentage, ces réductions sont toutefois supérieures à la réduction de la taille de chaque fichier. La durée d'exécution de notre algorithme se retrouve aussi améliorée. Cette durée demeure la durée la plus courte entrant dans le calcul du temps d'exécution final.

Nous pouvons donc conclure quant à la rapidité et l'efficacité de notre algorithme. Son temps d'exécution est réduit et ne représente qu'une faible partie (en dessous de 8%) du temps d'exécution total. Nous soulignons le fait que ce sont les temps de chargement des fichiers qui sont les plus importants. Cela représente le temps nécessaire au programme pour récpérer les valeurs des différentes propriétés contextuelles des services. Pour réduire ces temps, nous envisageons une étude approfondie des requêtes SPARQL. La clé réside en effet dans une formulation efficace de ces requêtes. A la manière des requêtes SQL, celles-ci doivent être optimisées pour permettre une réduction supplémentaire de ces temps.

Cela dit, les temps obtenus lors de ces deux séries de tests sont tout à fait corrects, et n'empêchent pas l'utilisation de notre prototype dans un environnement mobile. En réduisant la taille des fichiers, nous arrivons à réduire l'ensemble de ces temps en dessous d'une seconde, ce qui semble adéquat pour un temps d'attente en environnement mobile.

7.1.2 Mesures de pertinence

Les mesures présentées ci-dessus prouvent la rapidité de notre algorithme. Nous voulons cependant noter qu'il ne s'agit pas de son seul avantage. Tel que le remarquent les auteurs de (EUZENAT 2007), la plupart des algorithmes permettant l'appareillement d'ontologies échouent dans la compréhension des concepts, et dès lors ne permettent pas d'obtenir des résultats solides et pertinents. L'intérprétation et la compréhension des concepts démeurent dans la tête du développeur, et il est quasi impossible de programmer un ordinateur afin d'apprendre ces concepts. Notre algorithme pallie à ces manquements, et permet de retourner une liste ordonnée de services pertinents par rapport à la requête de l'utilisateur.

Afin de mesurer la qualité de l'algorithme, nous devons nous intéresser aux coûts et aux bénéfices qu'il implique pour l'utilisateur final. Le renvoi d'un résultat pertinent représente un avantage (ou un bénéfice) pour l'utilisateur, tandis qu'un résultat hors sujet a un certain coût pour l'utilisateur (notamment le temps passé à lire une information non pertinente).

Le bénéfice d'un résultat dépend de sa position sur la liste de résultats renvoyée à l'utilisateur. Les résultats situés au début de la liste sont dès lors très importants, car il est très probable que l'utilisateur les lise ou traite. Un résultat pertinent situé au début d'une telle liste apporte un plus grand bénéfice, qu'un résultat pertinent situé vers la fin de la liste. De la même manière, un résultat non-pertinent placé en haut d'une telle liste représente un coût, lui aussi plus important que si ce même résultat aurait été placé vers la fin de la liste. En outre, plus le nombre de résultats non-pertinents est important et plus ces résultats se retrouvent en tête de liste, moins l'utilisateur aura envie de parcourir la liste dans son ensemble, pour peut-être découvrir d'hypothétiques résultats pertinents. Aussi, il existe un coût associé au non-renvoi d'un résultat pertinent.

Nous souhaitons aussi évaluer le degré de précision de notre algorithme. Cette mesure est utile car elle prend des valeurs comprises dans l'intervalle [0;1], et qui peuvent ainsi être comparées à des mesures similaires pour d'autres algorithmes. La précision est une mesure qui permet de statuer

quant à l'utilité d'une liste de résultats (MAHESH 2001). La précision de l'algorithme est définie de la manière suivante(MAHESH 2001):

$$Précision = \frac{Nombre\ de\ résultats\ pertinents\ retournés}{Nombre\ total\ de\ résultats\ retournés}$$

Or, nous avons conçu notre algorithme de manière à prendre en compte l'ensemble des services présents dans le registre, pour un profil de service donné. Si nous prennons l'exemple d'une requête utilisateur précisant que ce dernier est à la recherche de « restaurants de type brasserie », l'algorithme ne va pas ignorer les services restaurants ayant d'autres catégories que « *Brasserie* ». L'algorithme calcule un score pour chaque service de type « restaurant », et retourne une liste de services classés selon leur score décroissant. Si dans sa requête l'utilisateur a affecté une pondération importante au critère « catégorie du restaurant », il y a de fortes chances que parmi les premiers résultats retournés, il y en ait qui soient du type « *Brasserie* ».

Il est dès lors difficile d'utiliser comme seul indicateur, la mesure de précision telle que définie plus haut. Nous souhaitons appliquer une nouvelle mesure, appelée *précision moyenne*, qui est un indicateur idéal pour les moteurs de recherche (ZHU 2004). La précision moyenne est définie de la manière suivante (MAHESH 2001) :

- Soit N le nombre d'éléments dans la liste de résultats ;
- Soit R[i] le i^{ème} résultat dans la liste de résultats ;
- Soit P[i]=1, si le R[i] est pertinent, sinon P[i]=0 ;
- Soit P le nombre de services du registre, pertinents par rapport à la requête initiale.

$$Précision[i] = \frac{\sum_{k=1..i} P[k]}{i}$$

$$Précision\ moyenne = \sum_{i=1..N} \frac{Précision[i] * P[i]}{P}$$

Pour pouvoir calculer cette valeur, nous créeons plusieurs descriptions de services restaurants, chacune avec des caractéristiques différentes. Elles sont détaillées dans le Tableau 24.

Attribut contextuel	*Service 1*	*Service 2*	*Service 3*	*Service 4*	*Service 5*	*Service 6*	*Service 7*
Catégorie du restaurant	Trad[29].	Rapide	Brasserie	Trad.	Pizza	Gast[30].	A Thème
Style de cuisine	Fr[31].	Europe	Fr.	Europe	Europe	Fr.	Fr.
Places de parking disponibles	4	10	0	6	0	7	15
Tables disponibles	2	2	4	3	6	5	10
Climatisation	OUI	NON	OUI	OUI	NON	OUI	NON
Accès handicapés	OUI	OUI	OUI	OUI	NON	NON	OUI
Prix du repas	20	13	10	35	15	30	14

Tableau 24. Valeurs des attributs contextuels des services créés.

[29] Traditionnel
[30] Gastronomique
[31] Française

Nous créeons ensuite deux requêtes utilisateur différentes, telles qu'illustrées par le Tableau 25 et le Tableau 26.

Attribut contextuel	aQuantificateur	aPonderation	aValeur
CategorieRestaurant	-	5	*Trad.*
StyleCuisine	-	1	Europe
Places_Parking_Disponibles	OUI	5	2
Places_Tables_Disponibles	OUI	4	4
Climatisation	-	3	OUI
AccesHandicapes	-	1	NON
PrixRepas	NON	2	25

Tableau 25. Requête utilisateur TEST1.

Attribut contextuel	aQuantificateur	aPonderation	aValeur
CategorieRestaurant	-	1	*Trad.*
StyleCuisine	-	5	Française
Places_Parking_Disponibles	OUI	1	2
Places_Tables_Disponibles	OUI	5	2
Climatisation	-	1	NON
AccesHandicapes	-	1	NON
PrixRepas	NON	5	15

Tableau 26. Requête utilisateur TEST2.

Nous appliquons finalement notre algorithme, pour chacune des deux requêtes utilisateur. Les scores de chaque service sont listés ci-dessous (Tableau 27). La liste de services retournée par le programme est illustrée par le Tableau 28.

Requête utilisateur	*Service 1 (S1)*	*Service 2 (S2)*	*Service 3 (S3)*	*Service 4 (S4)*	*Service 5 (S5)*	*Service 6 (S6)*	*Service 7 (S7)*
TEST1	-3	-3	4	-1	5	7	9
TEST2	11	1	7	18	20	22	21

Tableau 27. Scores des services pour chacune des deux requêtes test.

Requête utilisateur	1^{er}	$2^{ème}$	$3^{ème}$	$4^{ème}$	$5^{ème}$	$6^{ème}$	$7^{ème}$
TEST1	S7	S6	S5	S3	S4	S2	S1
TEST2	S6	S7	S5	S4	S1	S3	S2

Tableau 28. Liste ordonnée des services retournée pour chaque requête test.

Nous calculons ensuite les valeurs de précision et de précision moyenne pour les deux listes de résultats ainsi obtenues. Pour ce faire, nous devons définir une règle permettant de statuer sur la pertinence d'un résultat donné. Nous choisissons d'appliquer la règle suivante :

$$Un\ résultat\ i\ est\ défini\ comme\ pertinent\ ssi\ SCORE(i) \geq \frac{SCORE_{MAX}}{2}$$

Nous commençons par appliquer cette définition pour calculer les valeurs de précision associées aux deux requêtes, TEST1 et TEST2.

$$Précision_{TEST1} = \frac{Nombre\ de\ résultats\ pertinents\ retournés}{Nombre\ total\ de\ résultats\ retournés} = \frac{3}{7} \approx 0{,}43$$

$$Précision_{TEST2} = \frac{Nombre\ de\ résultats\ pertinents\ retournés}{Nombre\ total\ de\ résultats\ retournés} = \frac{5}{7} \approx 0{,}71$$

Pour un même algorithme, nous obtenons deux précisions différentes. D'où l'intérêt de calculer une précision moyenne. Selon la définition de la précision moyenne donnée plus haut, nous avons les valeurs suivantes :

- N = 7
- $P[i] = 1\ ssi\ SCORE(R[i]) \geq \frac{SCORE_{MAX}}{2}, sinon\ P[i] = 0$
- Nombre de services pertinents par rapport à la requête TEST1 – P_{TEST1} = 3
- Nombre de services pertinents par rapport à la requête TEST2 – P_{TEST2} = 5

Pour le calcul de la précision moyenne, nous devons calculer la précision de chaque résultat. Nous reportons les résultats de nos calculs dans le tableau suivant.

TEST		1er	2ème	3ème	4ème	5ème	6ème	7ème
TEST1	R[i]	S7	S6	S5	S3	S4	S2	S1
	SCORE(R[i])	9	7	5	4	-1	-3	-3
	P[i]	1	1	1	0	0	0	0
	Précision[i]	1	$^2/_2$	$^3/_3$	$^3/_4$	$^3/_5$	$^3/_6$	$^3/_7$
	Précision moyenne	$\sum_{i=1..7} \frac{Précision[i] * P[i]}{3} = \frac{3}{3} = 1$						
TEST2	R[i]	S6	S7	S5	S4	S1	S3	S2
	SCORE(R[i])	22	21	20	18	11	7	1
	P[i]	1	1	1	1	1	0	0
	Précision[i]	1	$^2/_2$	$^3/_3$	$^4/_4$	$^5/_5$	$^5/_6$	$^5/_7$
	Précision moyenne	$\sum_{i=1..7} \frac{Précision[i] * P[i]}{5} = \frac{5}{5} = 1$						

Tableau 29. Calcul de la précision moyenne.

Nous observons que cette fois-ci, les deux valeurs obtenues sont identiques, ce qui est logique étant donné le fait que nous testons le même algorithme. De plus, nous obtenons une valeur de 1, ce qui veut dire que notre algorithme renvoie l'ensemble des résultats pertinents, par rapport à une requête donnée, et présents dans le registre. Cette mesure indique aussi le fait que notre algorithme ordonne parfaitement les résultats, selon leur degré de pertinence.

7.2 Mesures relatives a l'utilisateur final

Jusqu'ici, nous avons évoquées des mesures relatives à l'algorithme de comparaison lui-même. Nous nous intéressons maintenant aux mesures permettant l'évaluation de l'expérience utilisateur. Or, le développement de l'interface « *Utilisateur Final* » n'étant pas encore abouti, nous ne pouvons qu'évoquer ces mesures à titre informatif. Elles seront intégrées à l'évaluation de l'application finale, et font dès lors partie des perspectives applicatives de notre prototype.

Une première mesure concerne les informations demandées à l'utilisateur (EUZENAT 2007). Cette mesure concerne notre application puisque, lors de la création de la requête utilisateur, nous

demandons à l'utilisateur d'entrer des informations. Il s'agit de déterminer si cette quantité d'informations peut être optimisée ou réduite, mais aussi de voir comment cela affècte l'expérience de l'utilisateur.

Il s'agit ensuite d'évaluer le niveau de satisfaction global de l'utilisateur (EUZENAT 2007). Nous devons prendre en compte des indicateurs, tels la rapidité de l'application, la consommation de ressources, la compréhension des résultats, ainsi que de l'application globale, ou encore les niveaux d'utilité et d'usabilité de l'application (B. SENACH 1990). En effet, alors que l'utilité d'une application indique si cette dernière permet à l'utilisateur d'atteindre ses objectifs, l'usabilité d'une application est une mesure à la frontière entre utilité potentielle et utilité réelle (B. SENACH 1990). L'usabilité d'une application ne dépend pas des caractéristiques locales de l'application, mais bien de ses propriétés générales. Pour ce faire, des études doivent être menées auprès d'une population d'utilisateurs tests. Cette population devra être définie en faisant attention à des critères tels que l'âge, le sexe ou encore la catégorie socio-professionnelle de l'utilisateur. Ces critères nous permettront d'identifier le niveau d'acceptation ou de compréhension de l'application, par rapport à différents types d'utilisateurs.

7.3 Conclusion

La Figure 61 liste quelques unes des mesures que nous souhaitons utiliser lors de l'évaluation de l'application. Pour déterminer l'utilité de notre application, nous utilisons des mesures des performances système, telles les mesures de rapidité et de pertinence présentées dans la sous-section 7.1. Pour éstimer l'usabilité de notre application, nous devons estimer la facilité d'apprentissage, la facilité d'utilisation ou encore la qualité de la documentation mise à disposition de l'utilisateur final (mode d'emploi, questions fréquemment posées, etc.) (B. SENACH 1990).

Figure 61. Mesures de l'utilité et de l'usabilité d'une application (B. SENACH 1990).

Ces deux types de mesures (mesures de l'utilité et mesures de l'usabilité) représentent des mesures complémentaires. Les mesures de l'utilité comprennent des mesures des performances, alors que les mesures de l'usabilité concernent des mesures subjectives.

Les différentes mesures mentionnées ci-dessus posent plusieurs problèmes, notamment au niveau de leur validité, de leur fidélité ou encore au niveau de leur précision.

En effet, l'évaluation d'une application est toujours dirigée par le contexte. Le choix de mesures pertinentes pour un « *diagnostic de qualité ergonomique dépend non seulement des objectifs de l'évaluation, mais aussi des caractéristiques de la population et des exigences des tâches* » (B. SENACH 1990). Par exemple, la qualité de l'interface est essentielle pour certaines applications, alors que d'autres ont pour facteur critique le temps d'exécution.

Notre premier défi sera de choisir des mesures judicieuses, en fonction du contexte d'utilisation de notre application. Il s'agit ensuite d'intégrer à ces mesures l'expérience utilisateur de notre application. Notre deuxième défi sera de recueillir et d'analyser des données comportementales, rendant compte de l'utilisation de l'application. Cette étape nous permettra d'identifier les difficultés rencontrées par les utilisateurs, puis de développer des solutions pour les réduire.

CHAPITRE 8.
CONCLUSION GENERALE ET PERSPECTIVES

Dans la vision du Web de demain, les tâches seront automatisées et transparentes pour l'utilisateur. Contrairement aux paradigmes traditionnels, notamment client-serveur, où l'information est centralisée par un serveur, le paradigme de l'informatique ubiquitaire sous-entend une omniprésence de réseaux de communication et d'éléments logiciels et matériels. Dans un tel environnement, les utilisateurs se retrouvent au centre des applications. Munis de terminaux mobiles, ils exécutent des services, subordonnés aux ressources et/ou services disponibles. Chacun de ces services est caractérisé par des besoins fonctionnels et non-fonctionnels, exprimés souvent en termes de fiabilité ou performance. Il est dès lors évident que satisfaire l'ensemble des besoins caractérisant un service, dans un environnement hautement hétérogène et dynamique, représente une tâche difficile.

Pour intégrer les ressources fournies par l'environnement, une des approches moderne consiste à utiliser les sémantiques pour la description de ces ressources. L'usage des sémantiques permet de rendre ces ressources compréhensibles et manipulables par des ordinateurs. Dans la vision du Web sémantique, les ontologies représentent le principal vecteur permettant de représenter et manipuler les annotations sémantiques entourant les ressources et les connaissances. Héritant de l'hétérogénéité du Web sémantique, les ontologies sont donc nombreuses et variées.

Les travaux de recherche présentés ici ont été menés autour d'une problématique qui trouve son origine dans cette hétérogénéité. En effet, les techniques traditionnelles de recherche d'informations ne suffisent plus face à la diversité et au nombre croissant d'informations disponibles aujourd'hui sur le Web. Dans notre approche, nous avons choisi les ontologies afin de modéliser un service en termes de besoins non-fonctionnels. Le modèle de base des services Web permet de répondre aux problèmes d'interopérabilité, mais ne permet pas une découverte automatique de ces services. Ce sont principalement les standards WSDL et UDDI qui sont aujourd'hui utilisés pour définir les fonctionnalités des services, et ce de manière *syntaxique*. Ces descriptions suffisent pour des applications « préconçues », mais ne permettent pas aux ordinateurs de *comprendre* les informations ainsi décrites. C'est l'annotation des services Web traditionnels avec des *sémantiques* qui a permis la création de services Web sémantiques.

Afin d'atteindre le but visé dans ce travail de recherche, nous avons proposé un modèle ontologique pour la description des services Web sémantiques. Sur la base de ce modèle, nous avons ensuite développé une plate-forme pour la recherche et la découverte de services ainsi décrits. Cette plate-forme introduit donc des fonctionnalités qui sont actuellement absentes dans les travaux concernant la description et la découverte de services Web sémantiques. Il s'agit d'abord du modèle décrivant les attributs contextuels du service, mais aussi d'un protocole de découverte de services faisant appel à ces attributs contextuels.

Notre modèle de description de services impose une structure cohérente pour l'ensemble des services. Non seulement ce modèle facilite l'analyse et le traitement des descriptions de services, mais il permet aussi la création d'interfaces utilisateur intuitives. Des interfaces basées sur ce modèle

permettent la création de descriptions de services et de requêtes, profitant à la fois aux fournisseurs de services et aux utilisateurs finaux. L'utilisateur peut affecter différentes pondérations à ses critères de recherche, et notre algorithme intègre ces pondérations dans la recherche du service le plus pertinent.

En plus de permettre la construction d'une telle interface, le modèle de description de service fait le lien avec des ontologies spécifiques à des domaines précis, et permet d'incorporer les termes de ces ontologies dans la description du service. L'utilisateur final obtient donc des résultats plus complets et précis, que c'est le cas pour la recherche à base de mots-clés. En outre, le fournisseur de services a à sa disposition un outil lui permettant de fournir des descriptions de services utilisant un même vocabulaire, sans avoir besoin de traduire chaque description de service.

Enfin, dans la dernière partie de ce manuscrit, nous avons présenté une évaluation de notre prototype, plus particulièrement une évaluation des temps associés à l'exécution d'une requête utilisateur. La discussion qui clôt cette thèse est une opportunité d'identifier les perspectives applicatives et académiques, des travaux de recherche présentés ici.

8.1 PERSPECTIVES

8.1.1 PERSPECTIVES APPLICATIVES

Cette section a pour but de discuter les quelques lacunes du présent prototype, dans le cadre d'éventuels travaux futurs, qui permettraient de les résoudre. En effet, dans les travaux de recherche discutés dans ce manuscrit, nous nous sommes concentrés sur la description des propriétés non-fonctionnelles d'un service et sur la découverte de services intégrant ces propriétés. L'implémentation de notre modèle peut toutefois être améliorée, en intégrant notamment les éléments suivants :

- ***Donner à un fournisseur de services la possibilité d'ajouter une catégorie de services, dans la taxonomie déjà présente*** – en effet, l'interface fournisseur de service permet de créer des descriptions de services conformes à notre modèle, mais ne permet pas d'ajouter une nouvelle catégorie de services à ce modèle.
- ***Stocker les préférences utilisateur dans un profil utilisateur*** – pour l'instant, à chaque nouvelle requête, l'utilisateur doit saisir ses préférences et ses pondérations. Il serait intéressant de pouvoir stocker, dans un profil utilisateur, ces préférences et pondérations, de manière à faciliter la saisie d'une requête. Le profil utilisateur contiendrait des requêtes types, avec des valeurs et des pondérations prédéfinies pour les catégories de services les plus recherchées par l'utilisateur.
- ***Optimiser l'échange de données avec le terminal mobile*** – en effet, lors des évaluations de notre prototype, nous avons identifié la taille des fichiers comme le principal facteur impactant sur la durée d'exécution d'une requête. Plus la taille des fichiers échangés est importante, plus le temps d'exécution est long. Il s'agit donc d'optimiser ces envois et réceptions de fichiers, entre le terminal mobile et la plate-forme fixe. L'idée serait de ne pas transférer des fichiers OWL décrivant entièrement le service, mais de faire des requêtes afin d'obtenir seulement les informations nécessaires (notamment les valeurs des propriétés non-fonctionnelles telles l'adresse, le nom ou le numéro de téléphone). Ceci permettrait une

implémentation beaucoup plus légère et efficace de l'application sur une plate-forme mobile. Il s'agit aussi d'optimiser la syntaxe des requêtes SPARQL.

- ***Evaluer l'expérience utilisateur*** – il s'agit d'évaluer l'interface homme-machine décrite dans ce travail de recherche, à travers les mesures suivantes: établir un diagnostic d'usage de l'application, comparer les avantages et inconvénients de cette application par rapport à des applications similaires, et enfin contrôler l'ergonomie de l'application.

Enfin un dernier travail à réaliser concerne l'automation des requêtes. Il serait intéressant, lorsque l'on affiche un itinéraire d'un point A vers un point B, de pouvoir générer automatiquement une requête pour un service « *Taxi* », par exemple, avec comme paramètres les points de départ et d'arrivée. De cette manière, en visualisant l'itinéraire, l'utilisateur peut lancer une nouvelle recherche de services.

D'autres considérations regardent l'amélioration du registre de services, la fin du développement de l'application mobile, l'amélioration de l'interface serveur, puis l'intégration des retours utilisateurs (définition d'une population test, définition de questionnaires utilisateurs, traitement des retours, amélioration de l'application en conséquence).

8.1.2 PERSPECTIVES SCIENTIFIQUES

Notre approche représente un lien entre deux domaines : l'informatique diffuse et les services Web. Dans ce manuscrit, nous avons expliqué comment les services Web peuvent être modélisés ou décrits afin d'intégrer leurs caractéristiques non-fonctionnelles (ou contextuelles) dans leur découverte. Nous avons proposé une approche permettant de constituer une description de service conforme à notre modèle et ceci à partir de la description WSDL du service. Notre proposition est intéressante car elle permet de profiter de la diversité de services existant actuellement, et n'implique pas un départ de « zéro ».

A court terme, ce travail peut être complété par l'implémentation d'un autre algorithme de comparaison, afin d'évaluer l'efficacité de notre algorithme. A long terme, plus de perspectives peuvent être envisagées. Nous pensons, par exemple, à la définition et à l'implémentation de mécanismes permettant le dialogue ou la collaboration entre services. Une autre perspective concerne l'étude d'autres langages de description des services, plus particulièrement l'approche WSMO, qui gagne en popularité aujourd'hui.

Nous pouvons aussi envisager l'introduction de raisonnements à base de logique floue, qui aurait pour résultat l'amélioration de notre prototype en outil d'aide à la décision, et ce sur la base de requêtes précédentes de l'utilisateur. Si, par exemple, l'application détermine que l'utilisateur a souvent pour habitude de chercher un restaurant aux alentours de midi, l'utilisateur se verra suggérer un restaurant, en fonction des paramètres qu'il définit habituellement dans ses requêtes. La logique floue sera utilisée afin de créer des règles permettant d'interpréter les habitudes de l'utilisateur, par rapport à un contexte donné (position géographique, intervalle de temps, etc.).

Finalement, l'ensemble des contributions présentées ici a été introduit dans le développement de la plate-forme TransportML (ROXIN et al. 2009). Le but de cette plate-forme sera de rendre automatique la découverte de services Web, sur la base du modèle de description proposé dans ce manuscrit. Ce modèle sera aussi utilisé afin de favoriser la collaboration entre les différents services.

ANNEXES

ANNEXE A – MODELE OWL-S ETENDU (EXTRAITS DE L'ONTOLOGIE)

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY time-entry "http://www.isi.edu/~pan/damltime/time-entry.owl#" >
    <!ENTITY Service "http://www.daml.org/services/owl-s/1.2/Service.owl#" >
    <!ENTITY Country "http://www.daml.org/services/owl-s/1.2/Country.owl#" >
    <!ENTITY Profile "http://www.daml.org/services/owl-s/1.2/Profile.owl#" >
    <!ENTITY Process "http://www.daml.org/services/owl-s/1.2/Process.owl#" >
    <!ENTITY Grounding "http://www.daml.org/services/owl-s/1.2/Grounding.owl#" >
    <!ENTITY drsonto040520 "http://cs-www.cs.yale.edu/homes/dvm/daml/drsonto040520.owl#" >
    <!ENTITY ObjectList "http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#" >
    <!ENTITY Expression "http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#" >
    <!ENTITY        ProfileAdditionalParameters        "http://www.daml.org/services/owl-
s/1.2/ProfileAdditionalParameters.owl#" >
    <!ENTITY                                                    Ontology1239957158891
"http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#" >
]>


<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#"
     xml:base="http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:Process="http://www.daml.org/services/owl-s/1.2/Process.owl#"
     xmlns:drsonto040520="http://cs-www.cs.yale.edu/homes/dvm/daml/drsonto040520.owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:time-entry="http://www.isi.edu/~pan/damltime/time-entry.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:ProfileAdditionalParameters="http://www.daml.org/services/owl-
s/1.2/ProfileAdditionalParameters.owl#"
     xmlns:ObjectList="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#"
     xmlns:Profile="http://www.daml.org/services/owl-s/1.2/Profile.owl#"
     xmlns:Service="http://www.daml.org/services/owl-s/1.2/Service.owl#"
     xmlns:Expression="http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#"
     xmlns:Country="http://www.daml.org/services/owl-s/1.2/Country.owl#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:dc="http://purl.org/dc/elements/1.1/"

xmlns:Ontology1239957158891="http://www.semanticweb.org/ontologies/2009/3/Ontology12399571
58891.owl#"
     xmlns:Grounding="http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about=""/>

<!--

///////////////////////////////////////////////////////////////////////////////////////
    //
    // Object properties
    //

///////////////////////////////////////////////////////////////////////////////////////
     -->
```

```
<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#aContexte -->
    <owl:ObjectProperty rdf:about="&Service;aContexte">
        <rdfs:range rdf:resource="&Service;ServiceContext"/>
        <rdfs:domain rdf:resource="&Service;ServiceProfile"/>
    </owl:ObjectProperty>

<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#vend -->
    <owl:ObjectProperty rdf:about="&Service;vend">
        <rdfs:range rdf:resource="&Service;Produit"/>
        <rdfs:domain rdf:resource="&Service;Service_Achat_Produit"/>
    </owl:ObjectProperty>
<!--

///////////////////////////////////////////////////////////////////////////////////////
    //
    // Data properties
    //

///////////////////////////////////////////////////////////////////////////////////////
-->
```

```
<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#aQuantificateur -->
    <owl:DatatypeProperty rdf:about="&Service;aQuantificateur">
        <rdfs:domain rdf:resource="&Service;ServiceContext"/>
        <rdfs:range rdf:resource="&xsd;boolean"/>
    </owl:DatatypeProperty>

<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#aValeur -->
    <owl:DatatypeProperty rdf:about="&Service;aValeur">
        <rdfs:comment xml:lang="fr"
            >Cette propriete permet de definir la valeur d'un element du contexte
service.</rdfs:comment>
        <rdfs:domain rdf:resource="&Service;ServiceContext"/>
    </owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#aAdresse -->
    <owl:DatatypeProperty rdf:about="#aAdresse">
        <rdfs:comment xml:lang="fr"
            >Cette propri&#233;t&#233; permet de d&#233;finir l&#39;adresse d&#39;un
service.</rdfs:comment>
        <rdfs:domain rdf:resource="&Service;ServiceProfile"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>


<!-- http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#aPonderation -
->
    <owl:DatatypeProperty rdf:about="#aPonderation">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:comment xml:lang="fr"
            >Permet de pond&#233;rer les diff&#233;rentes propri&#233;t&#233;s d&#39;un
service.</rdfs:comment>
        <rdfs:domain rdf:resource="&Service;ServiceContext"/>
        <rdfs:range rdf:resource="&xsd;float"/>
    </owl:DatatypeProperty>


<!-- http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#aTelephone -->
    <owl:DatatypeProperty rdf:about="#aTelephone">
```

```
        <rdfs:comment xml:lang="fr"
            >Cette propri&#233;t&#233; permet de d&#233;finir le num&#233;ro de
t&#233;l&#233;phone d&#39;un service.</rdfs:comment>
        <rdfs:domain rdf:resource="&Service;ServiceProfile"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>

<!--
  ///////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //

///////////////////////////////////////////////////////////////////////////////////
     -->

<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#CategorieRestaurant -->
<owl:Class rdf:about="&Service;CategorieRestaurant">
        <rdfs:subClassOf rdf:resource="&Service;Service_Restaurant_Context"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&Service;aValeur"/>
                <owl:allValuesFrom rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <dc:description xml:lang="fr"
            >Definit plusieurs categories de restaurants.</dc:description>
    </owl:Class>

<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#Places_Parking_Disponibles -->
    <owl:Class rdf:about="&Service;Places_Parking_Disponibles">
        <rdfs:subClassOf rdf:resource="&Service;Service_Restaurant_Context"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&Service;aValeur"/>
                <owl:allValuesFrom rdf:resource="&xsd;integer"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <dc:description xml:lang="fr"
            >Indique le nombre de places parking disponibles.</dc:description>
    </owl:Class>


<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#Places_Tables_Disponibles -->
    <owl:Class rdf:about="&Service;Places_Tables_Disponibles">
        <rdfs:subClassOf rdf:resource="&Service;Service_Restaurant_Context"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&Service;aValeur"/>
                <owl:allValuesFrom rdf:resource="&xsd;integer"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <dc:description xml:lang="fr"
            >Indique le nombre de places/tables restantes dans le
restaurant.</dc:description>
    </owl:Class>

<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#ServiceContext -->
    <owl:Class rdf:about="&Service;ServiceContext">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>

<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#Service_Personnel -->
```

```
    <owl:Class rdf:about="&Service;Service_Personnel">
        <rdfs:subClassOf rdf:resource="&Profile;Profile"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&Service;aContexte"/>
                <owl:allValuesFrom rdf:resource="&Service;Service_Personnel_Context"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#Service_Personnel_Context -->
    <owl:Class rdf:about="&Service;Service_Personnel_Context">
        <rdfs:subClassOf rdf:resource="&Service;ServiceContext"/>
    </owl:Class>


<!-- http://www.daml.org/services/owl-s/1.2/Service.owl#Produit -->
    <owl:Class rdf:about="&Service;Produit">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>

<!--
http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#AccesHandicapes -->
    <owl:Class rdf:about="#AccesHandicapes">
        <rdfs:subClassOf rdf:resource="&Service;Service_Restaurant_Context"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&Service;aValeur"/>
                <owl:allValuesFrom rdf:resource="&xsd;boolean"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <owl:incompatibleWith xml:lang="fr"
            >Indique si le restaurant dispose ou non d'un acces
handicapes.</owl:incompatibleWith>
    </owl:Class>

<!--  http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#Climatisation
-->
    <owl:Class rdf:about="#Climatisation">
        <rdfs:subClassOf rdf:resource="&Service;Service_Restaurant_Context"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&Service;aValeur"/>
                <owl:allValuesFrom rdf:resource="&xsd;boolean"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <owl:incompatibleWith xml:lang="fr"
            >Indique si le restaurant dispose ou pas d'une
climatisation.</owl:incompatibleWith>
    </owl:Class>

<!-- http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#PrixRepas -->
    <owl:Class rdf:about="#PrixRepas">
        <rdfs:subClassOf rdf:resource="&Service;Service_Restaurant_Context"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&Service;aValeur"/>
                <owl:allValuesFrom rdf:resource="&xsd;integer"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <owl:incompatibleWith xml:lang="fr"
            >Definit le prix moyen d'un repas pour le restaurant.</owl:incompatibleWith>
```

```
    </owl:Class>
```

ANNEXE B. FICHIERS DU MOTEUR DECOUVERTE

SIMPLEENGINE.JAVA

```
public class SimpleEngine {

    public String textResults = "";

    public String formatTime = "hh:mm:ss:SSS";

    class ServiceScore {
        public Vector<Vector<String>> service;
        public Long score;
        public ServiceScore() {
            service = new Vector<Vector<String>>();
            score = null;
        }
    }
    private OwlLoader loader;
    private Vector<Vector<String>> result;
    public SimpleEngine() {
        result = new Vector<Vector<String>>();
    }
    public void loadEngine(String path, String servicePath) {
        loader = new OwlLoader(path, servicePath);
    }

    public void Compute() {
         System.out.println("Start computing! Compute start time is: " +
Utils.DateTimeNow(formatTime));

        Vector<ServiceScore> results =
algo1Implementation2((getServiceScoreForServiceName("Service_Restaurant")));
        Iterator<ServiceScore> i = results.iterator();
        int k = 0;
        while (i.hasNext()) {
            String txtTemp = "Service " + ++k + " a le score: " + i.next().score + "
<br/>";
            textResults += txtTemp;
        }
        System.out.println("Stop Computing! - Compute end time is : " +
Utils.DateTimeNow(formatTime) );
    }


    public Vector<ServiceScore> getServiceScoreForServiceName(String serviceName) {
        Vector<ServiceScore> vss = new Vector<ServiceScore>();

        ManipulerXML manipXML = new ManipulerXML(Config.BASE_INDEX_FILES);

        System.out.println(serviceName);

        String[] fileList = new String[10];
        fileList = manipXML.getFileListArray(serviceName);

        System.out.println("trouvee : " + fileList.length + " fichiers");

        if (fileList.length == 0) {
            textResults = "There are no registered services!";
        }


        for (int i = 0; i < fileList.length; i++) {
```

```
            System.out.println("File     "+fileList[i]+"     started     to     load     at:
"+Utils.DateTimeNow(formatTime));


            ServiceScore ss = new ServiceScore();

            ManipulerOWL      manipOWL      =      new      ManipulerOWL("file:///"      +
Config.BASE_SERV_ONTOLOGY_PATH + fileList[i]);
            System.out.println("File    "+fileList[i]+"    end    load    at    :    "    +
Utils.DateTimeNow(formatTime));

            List<PropertyService> serviceProps = manipOWL.getPropertyService(serviceName,
manipOWL.getInternalModel()).get(0).getProps();

            System.out.println("File     "+fileList[i]+"     start     query     at     :
"+Utils.DateTimeNow(formatTime));

            for (int j = 0; j < serviceProps.size(); j++) {


                String values[] = new String[2];

                values           =           manipOWL.getValuePropertyService(serviceName,
serviceProps.get(j).getName().toString());

                String propertyName = values[0];
                String propertyValue = values[1];
                Vector<String> vs = new Vector<String>();
                vs.add(propertyName);
                vs.add(propertyValue);
                ss.service.add(vs);
            }
            System.out.println("File    "+fileList[i]+"    terminated    query    at    :
"+Utils.DateTimeNow(formatTime));

            vss.add(ss);
        }

        return vss;

    }

    public Vector<ServiceScore> algo1Implementation2(Vector<ServiceScore> services) {
        System.out.println("Algoritm started at time : "+Utils.DateTimeNow(formatTime));
        Iterator<ServiceScore> iService = services.iterator();
        while (iService.hasNext()) {
            //niveau service
            ServiceScore ss = iService.next();
            Iterator<Vector<String>> iVector = ss.service.iterator();
            Long scoreService = Long.parseLong("0");     //pour retirer le +1 accorder pour
le critere categorie restaurant
            while (iVector.hasNext()) {
                //niveau tableau service
                Vector<String> iServiceLine = iVector.next();
                Iterator<String> iString = iServiceLine.iterator();
                Long scoreLine = Long.parseLong("0");
                while (iString.hasNext()) {
                    //niveau champs
                    String fieldName = iString.next();
                    String fieldValue = iString.next();
                    int i = 0;
```

```
                    //on cherche le critere dans le userCriteria
                    while    (i    <    loader.getUserCriteria().size()    -    1    &&
loader.getUserCriteria().get(i).get(0).compareTo(fieldName) != 0) {
                        i++;

                        if (loader.getUserCriteria().get(i).get(0).compareTo(fieldName) ==
0) {

//*************************************************************************************
                            scoreLine = computePropertyScore(fieldName, fieldValue, i);

//*************************************************************************************
                        }
                    }
                }
                scoreService += scoreLine;
            }
            ss.score = scoreService;
        }
        System.out.println("Algoritm ended at time : "+Utils.DateTimeNow(formatTime));
        return services;
    }


    public    Long    computePropertyScore(String    fieldName,    String    fieldValue,    int
criteriaIndice) {

        if (fieldValue.compareTo("TRUE") == 0 || fieldValue.compareTo("FALSE") == 0) {
            //boolean value
            if   (loader.getUserCriteria().get(criteriaIndice).get(1).compareTo(fieldValue)
== 0) {
                return
Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(3));
            }
        }


        if (isLong(fieldValue)) {

             if ((loader.getUserCriteria().get(criteriaIndice).get(2).compareTo("TRUE") ==
0                   &&                   Long.parseLong(fieldValue)                   /
Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(1)) >= 1) ||
                    loader.getUserCriteria().get(criteriaIndice).get(2).compareTo("FALSE")
==              0              &&              Long.parseLong(fieldValue)              /
Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(1)) <= 1) {

                return  Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(3))
*                           Long.parseLong(fieldValue)                                   /
Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(1));
            } else {
                return                          Long.parseLong("-1")                        *
Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(3))                         *
Long.parseLong(fieldValue)                                                                  /
Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(1));
            }
        } else {
            //ontology case
            if   (loader.getUserCriteria().get(criteriaIndice).get(1).compareTo(fieldValue)
== 0) {
                return
Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(3));
            }
            //for the subclass
```

```
            OntClass ont = loader.getInputModel().getOntClass(fieldValue);
            if (ont != null) {
                Iterator<OntClass> i = ont.listSubClasses(true);
                while (i.hasNext()) {
                    if
(i.next().getURI().compareTo(loader.getUserCriteria().get(criteriaIndice).get(1)) == 0) {
                        return
Long.parseLong(loader.getUserCriteria().get(criteriaIndice).get(3)) / 2;
                    }
                }
            }
            return Long.parseLong("0");
        }
    }

    public static boolean isLong(String st) {
        try {
            Long.parseLong(st);
        } catch (NumberFormatException e) {
            return false;
        }
        return true;
    }

    public Vector<ServiceScore> ResultSetToVectorServiceScore(Iterator it) {
        return null;
    }

    public static void main(String[] args) {
        SimpleEngine se = new SimpleEngine();

        se.loadEngine("file:///"+Config.BASE_USR_ONTOLOGY_FILE,        "file:///"        +
Config.BASE_MAIN_ONTOLOGY_FILE);
        se.Compute();
    }
}
```

ANNEXE C. FICHIERS DE L'INTERFACE « FOURNISSEUR DE SERVICES »

MANIPULEROWL.OWL

```
public class ManipulerOWL {

    private String fileUrl;
    private Model model;

    public ManipulerOWL(){
    }

    public ManipulerOWL(String fileUrl) {
        this.fileUrl = fileUrl;
        try{
        loadModel(fileUrl);
        //System.out.println("Your file is: "+fileUrl);
        } catch(Exception e){
            System.out.println(e);
        }
    }

    public ManipulerOWL(Model model){
        this.model = model;
    }

    public Model getInternalModel(){
        return this.model;
    }

    public String getFileUrl() {
        return fileUrl;
    }

    public void setFileUrl(String fileUrl) {
        this.fileUrl = fileUrl;
    }

    public void loadModel(String storageFile) throws Exception, ClassNotFoundException {

         if (storageFile == null) {
            throw new Exception("Missing Storage File");
        }

        model = ModelFactory.createMemModelMaker().createDefaultModel();
        InputStream in = FileManager.get().open(storageFile);
        //read the model
        model.read(in, null);

        in.close();

    }
    public void loadModel(){
        try {
        loadModel(this.fileUrl);
        } catch (Exception e){
            System.out.println(e);
        }
    }

    public String getXMLAboutService(String serviceType, String serviceName){
        String xmlStr = "";
```

```
        try {
            InputStream in = FileManager.get().open(this.fileUrl);
            Model model = ModelFactory.createMemModelMaker().createDefaultModel();

            model.read(in, null);
            in.close();

            String queryString =
                    "PREFIX owl: <http://www.w3.org/2002/07/owl#>  " +
                    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
                    "PREFIX  prof:  <http://www.daml.org/services/owl-s/1.2/Profile.owl#>  "
+
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
                    "PREFIX  serv:  <http://www.daml.org/services/owl-s/1.2/Service.owl#>  "
+
                    "PREFIX xmlsch: <http://www.w3.org/2001/XMLSchema> " +

                    "SELECT DISTINCT ?serviceContext ?property ?subClass ?valueType " +
                    "WHERE {" +
                    "                        serv:"  +  serviceName  +  "  rdfs:subClassOf
serv:"+serviceType+" ; " +
//                    "        a ?type ; " +
                    "         rdfs:subClassOf [ a owl:Restriction; owl:onProperty ?onProp;
owl:allValuesFrom ?serviceContext].   " +
                    "         ?property rdfs:subClassOf ?serviceContext . " +
                    "                    ?property  rdfs:subClassOf  [  a  owl:Restriction;
owl:onProperty ?onProp1; owl:allValuesFrom ?valueType].   " +
                    "  OPTIONAL {         ?subClass rdfs:subClassOf ?property . }" +

//                    "         ?type2 a ?type3 ." +

                    "      }";


            Query query = QueryFactory.create(queryString);

            // Execute the query and obtain results
            QueryExecution qe = QueryExecutionFactory.create(query, model);
            ResultSet results = qe.execSelect();

            xmlStr = ResultSetFormatter.asXMLString(results);

            qe.close();

        } catch (Exception e) {
            e.printStackTrace();
        }

        return xmlStr;
    }

    /*
    recuperer la liste des services
     */
    public List<ServiceOwl> getService(Model localModel) {

        List<ServiceOwl> servs = new ArrayList<ServiceOwl>();

        try {
            String queryString =
                    "PREFIX owl: <http://www.w3.org/2002/07/owl#>  " +
                    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
                    "PREFIX prof: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>" +
```

```
                "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "PREFIX serv: <http://www.daml.org/services/owl-s/1.2/Service.owl#>" +

                "SELECT ?class " +
                "WHERE {" +
                "      ?class rdfs:subClassOf prof:Profile . " +
                "      }";



        Query query = QueryFactory.create(queryString);

        // Execute the query and obtain results
        QueryExecution qe = QueryExecutionFactory.create(query, localModel);
        ResultSet results = qe.execSelect();

        // Output query results
        qe.close();

        while (results.hasNext()) {
            ServiceOwl ser = new ServiceOwl();
            QuerySolution sol = results.nextSolution();

            for (Object var : results.getResultVars()) {
                //System.out.println(sol.get(var.toString()).toString());
                if (sol.get(var.toString()) != null) {

                    ser.setURI(sol.get(var.toString().toString()));
                    String str[] = ser.getURI().toString().split("#");
                    ser.setName(str[1].toString());

                }

            }

            servs.add(ser);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }

    return servs;
}


/*            recuperer les services a partir d'un autre service         */
public List<ServiceOwl> getService(String service, Model currentModel) {

    List<ServiceOwl> servs = new ArrayList<ServiceOwl>();

    try {

        String queryString =
                "PREFIX owl: <http://www.w3.org/2002/07/owl#>  " +
                "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
                "PREFIX prof: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>" +
                "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "PREFIX serv: <http://www.daml.org/services/owl-s/1.2/Service.owl#>" +
                "SELECT ?subClass  " +
                "WHERE {" +
                "      ?subClass rdfs:subClassOf serv:" + service + ". " +
                "      }";
```

```
            Query query = QueryFactory.create(queryString);

            // Execute the query and obtain results
            QueryExecution qe = QueryExecutionFactory.create(query, currentModel);
            ResultSet results = qe.execSelect();

            // Output query results
            qe.close();

            while (results.hasNext()) {
                ServiceOwl ser = new ServiceOwl();
                QuerySolution sol = results.nextSolution();

                for (Object var : results.getResultVars()) {
                    if (sol.get(var.toString()) != null) {

                        ser.setURI(sol.get(var.toString().toString()));
                        String str[] = ser.getURI().toString().split("#");
                        ser.setName(str[1].toString());
                    }
                }
                servs.add(ser);
            }
        } catch (Exception e) {
        }
        return servs;
    }

    /*          recuperer les proprietets des services       */
    public    List<PropertyService>    getPropertyService(String    ServiceName,    Model
currentModel) {

        List<PropertyService> propers = new ArrayList<PropertyService>();

        try {

            String queryString =
                    "PREFIX owl: <http://www.w3.org/2002/07/owl#>  " +
                    "PREFIX skos: <http://www.w3.org/2004/02/skos/core#> " +
                    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                    "PREFIX                                                                    :
<http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#> " +
                    "SELECT DISTINCT ?subClass ?proper ?value " +
                    "WHERE {" +
                    "              <http://www.daml.org/services/owl-s/1.2/Service.owl#"  +
ServiceName + "> rdfs:subClassOf ?y; " +
                    "          rdfs:subClassOf [ a owl:Restriction; owl:onProperty ?proper;
owl:allValuesFrom ?value].   " +
                    "}";

            Query query = QueryFactory.create(queryString);

            QueryExecution qe = QueryExecutionFactory.create(query, currentModel);
            ResultSet results = qe.execSelect();

            while (results.hasNext()) {


                PropertyService prop = new PropertyService();
                QuerySolution sol = results.nextSolution();
```

```
                for (Object var : results.getResultVars()) {
                    if (sol.get(var.toString()) != null) {

                        if (var.toString().equals("proper")) {
                            prop.setURI(sol.get(var.toString()));
                            String str[] = prop.getURI().toString().split("#");
                            prop.setName(str[1].toString());

                        }
                        if (var.toString().equals("value")) {
                            prop.setType(sol.get(var.toString()).toString());

                            String value = sol.get(var.toString()).toString();
                            prop.setType(value);
                            String                          str1[]                          =
sol.get(var.toString().toString()).toString().split("#");
                            prop.setURIType(sol.get(var.toString()));

                            if (str1[0].equals("http://www.w3.org/2001/XMLSchema")) {

                                String str[] = prop.getURIType().toString().split("#");
                                prop.setType(str[1].toString());
                            } else {

                                String str[] = prop.getURIType().toString().split("#");
                                prop.setType(str[1].toString());
prop.setProps(this.getPorService(prop.getType()));
                            }
                        }
                    }
                }
                propers.add(prop);
            }

            // Output query results

        } catch (Exception e) {
            e.printStackTrace();
        }
        return propers;
    }

    ///     get property recursive
    public List<PropertyService> getPorService(String NameService) {

        List<PropertyService> props = new ArrayList<PropertyService>();

        try {

            String queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>  " +
                    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
                    "PREFIX prof: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>" +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                    "SELECT ?subClass ?proper ?value " +
                    "WHERE {" +
                    "        ?subClass rdfs:subClassOf <http://www.daml.org/services/owl-
s/1.2/Service.owl#" + NameService + ">; " +
                    "       rdfs:subClassOf [ a owl:Restriction; owl:onProperty ?proper;
owl:allValuesFrom ?value].}";


            Query query = QueryFactory.create(queryString);
```

```
            // Execute the query and obtain results
            QueryExecution qe = QueryExecutionFactory.create(query, model);
            ResultSet results = qe.execSelect();

            // Output query results
            qe.close();
            while (results.hasNext()) {
                PropertyService proser = new PropertyService();
                QuerySolution sol = results.nextSolution();

                for (Object var : results.getResultVars()) {
                    if (sol.get(var.toString()) != null) {

                        if (var.toString().equals("subClass")) {
                            proser.setURI(sol.get(var.toString().toString()));
                            String str[] = proser.getURI().toString().split("#");
                            proser.setName(str[1].toString());

                        }
                        if (var.toString().equals("value")) {
                            String                          str1[]                          =
sol.get(var.toString().toString()).toString().split("#");
                            if (str1[0].equals("http://www.w3.org/2001/XMLSchema")) {

                                proser.setURIType(sol.get(var.toString().toString()));
                                String str[] = proser.getURIType().toString().split("#");
                                proser.setType(str[1].toString());
                            } else {

                                proser.setURIType(sol.get(var.toString().toString()));
                                String str[] = proser.getURIType().toString().split("#");
                                proser.setType(str[1].toString());
                            }
                        }

                        proser.setProps(this.getPorServicecollection(proser.getName()));
                    }
                }
                props.add(proser);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return props;
    }

    //returner la collection
    public List<PropertyService> getPorServicecollection(String NameService) {
        List<PropertyService> props = new ArrayList<PropertyService>();
        try {

            String queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>  " +
                    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                    "SELECT ?subClass " +
                    "WHERE {" +
                    "        ?subClass  rdfs:subClassOf  <http://www.daml.org/services/owl-
s/1.2/Service.owl#" + NameService + ">; " +
                    "     }";


            Query query = QueryFactory.create(queryString);
```

```
            // Execute the query and obtain results
            QueryExecution qe = QueryExecutionFactory.create(query, model);
            ResultSet results = qe.execSelect();
            // Output query results
            qe.close();
            while (results.hasNext()) {

                PropertyService proser = new PropertyService();
                QuerySolution sol = results.nextSolution();

                for (Object var : results.getResultVars()) {
                    if (sol.get(var.toString()) != null) {


                        proser.setURI(sol.get(var.toString().toString()));
                        String str[] = proser.getURI().toString().split("#");
                        proser.setName(str[1].toString());
                        //System.out.println(proser.getName());
                        proser.setProps(this.getPorServicecollection(proser.getName()));
                    }
                }
                props.add(proser);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return props;
    }

    //------------------------------------------------------------------------//
//get values for service name in order to calculate the score of the service
    public String[] getValuePropertyService(String nameService, String propertyService) {

        String[] values = new String[2];
        try {

            String queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>  " +
                    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
                    "PREFIX prof: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>" +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                    "PREFIX serv: <http://www.daml.org/services/owl-s/1.2/Service.owl#>"+
                    "SELECT ?type ?value " +
                    "WHERE {" +
                    "      serv:hasvalue" + propertyService + " a ?type;" +
                    "                                  serv:aValeur  ?value.}";


            Query query = QueryFactory.create(queryString);

            // Execute the query and obtain results
            QueryExecution qe = QueryExecutionFactory.create(query, model);
            ResultSet results = qe.execSelect();

            // Output query results
            qe.close();
            while (results.hasNext()) {

                QuerySolution sol = results.nextSolution();

                for (Object res : results.getResultVars()) {
                    if (sol.get(res.toString()) != null) {
```

```
                        Node valueNode = sol.get(res.toString()).asNode();

                        if (res.toString().equals("value")) {

                            values[1] = valueNode.getLiteralValue().toString();

                        } else {
                            values[0] = valueNode.getURI();
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        //return props;
        return values;
    }
}
```

CREATEOWL.JAVA

```
package Servlet;

import OWL.ManipulerOWL;
import OWL.MODELONT;
import OWL.PropertyService;
import WSDL.ManipulerWsdl;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.mem.ModelMem;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.Ontology;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.vocabulary.*;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.io.Reader;
import java.util.*;
import com.hp.hpl.jena.ontology.Individual;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntProperty;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.rdf.model.Model;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.*;

import java.net.URI;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import javax.servlet.ServletContext;
import javax.xml.registry.infomodel.User;
import org.mindswap.pellet.jena.PelletReasonerFactory;


public class CreateOWL extends HttpServlet {


    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        response.setContentType("text/html");
```

```
         PrintWriter out = response.getWriter();

        try {

            String serviceprof= request.getParameter("servicename").toString();
            String inputFileName="Ontologie_Service_Profile_Hierarchy_150509.owl";

            OntModel m = ModelFactory.createOntologyModel();

      InputStream in = CreateOWL.class.getResourceAsStream(inputFileName);

               if (in == null)
              {
                    throw new IllegalArgumentException(
                    "File: " + inputFileName + " not found");
              }

              // read the OWL file
              m.read(in, "");



                        List<PropertyService> props = new ArrayList<PropertyService>();

              ManipulerOWL                  manip                  =                  new
ManipulerOWL(this.getServletContext().getRealPath("/")+"Ontologie_Service_Profile_Hierarch
y_150509.owl");

              props = manip.getPropertyService(serviceprof).get(0).getPorps();
         for (int i = 0; i < props.size(); i++) {
         // if (props.get(i).getSizeProperty()==0){

          if (props.get(i).getType().equals("boolean")){
                OntClass largeFormat = m.getOntClass( props.get(i).getURI().toString());
              boolean varbool= true;
              if(request.getParameter(props.get(i).getName()).toString().equals("oui")){
                varbool= true;
              }

              else{
               varbool= false;
              }

       Iterator it = largeFormat.listDeclaredProperties();
        while (it.hasNext()) {
              OntProperty prop = (OntProperty) it.next();
              if                    (prop.getURI().equals("http://www.daml.org/services/owl-
s/1.2/Service.owl#aValeur"))
              {
                  Individual  ind  =    m.createIndividual("http://www.daml.org/services/owl-
s/1.2/Service.owl#hasvalue"+props.get(i).getName().toString(),largeFormat);
                  ind.addLiteral(prop, varbool);


         }
         }
        }

          if (props.get(i).getType().equals("integer")){

              OntClass largeFormat = m.getOntClass(props.get(i).getURI().toString());
             int                                    nbr                                    =
Integer.parseInt(request.getParameter(props.get(i).getName()).toString());
```

```
   Iterator it = largeFormat.listDeclaredProperties();
        while (it.hasNext()) {
            OntProperty prop = (OntProperty) it.next();
            if                    (prop.getURI().equals("http://www.daml.org/services/owl-
s/1.2/Service.owl#aValeur"))
            {
                Individual  ind  =    m.createIndividual("http://www.daml.org/services/owl-
s/1.2/Service.owl#hasvalue"+props.get(i).getName().toString(),largeFormat);
                ind.addLiteral(prop, nbr);


        }
        }

         }

         if (props.get(i).getType().equals("string")){

              OntClass largeFormat = m.getOntClass( props.get(i).getURI().toString());
            String str = request.getParameter(props.get(i).getName()).toString();
   Iterator it = largeFormat.listDeclaredProperties();
        while (it.hasNext()) {
            OntProperty prop = (OntProperty) it.next();
            if                    (prop.getURI().equals("http://www.daml.org/services/owl-
s/1.2/Service.owl#aValeur"))
            {
                Individual  ind  =    m.createIndividual("http://www.daml.org/services/owl-
s/1.2/Service.owl#hasvalue"+props.get(i).getName().toString(),largeFormat);
                ind.addLiteral(prop, str);
        }
        }

         }
           OntClass     service     =     m.getOntClass("http://www.daml.org/services/owl-
s/1.2/Service.owl#ServiceProfile");

           OntProperty                                  proper                                =
m.getOntProperty("http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#a
Telephone");
            String tele = request.getParameter("telephone").toString();

                Individual  ind  =    m.createIndividual("http://www.daml.org/services/owl-
s/1.2/Service.owl#hasvaluetelephone",service);
                ind.addLiteral(proper, tele);

         OntProperty                                properadrres                              =
m.getOntProperty("http://www.semanticweb.org/ontologies/2009/3/Ontology1239957158891.owl#a
Telephone");
            String adresse = request.getParameter("adresse").toString();

                Individual                                indadrees                             =
m.createIndividual("http://www.daml.org/services/owl-
s/1.2/Service.owl#hasvalueadresse",service);
                indadrees.addLiteral(properadrres, adresse);


    FileWriter     fileWriter   =   new   FileWriter(getServletContext().getRealPath("/")   +
File.separator + "WEB-INF" + File.separator + "text1.owl");

   m.write(fileWriter, "RDF/XML-ABBREV");


     // }
```

```
         }
           out.println("Le fichier est  generée");

        } catch (Exception e) {
            out.println(e.getMessage());
        }
    }
}
```

RECUPSERVICE.JAVA

```
package Servlet;

import WSDL.ManipulerWsdl;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import org.apache.commons.fileupload.*;

import org.apache.commons.fileupload.disk.*;
import org.apache.commons.fileupload.servlet.*;
import java.util.*;
import java.io.*;
public class Recupservice extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
         PrintWriter out = response.getWriter();

          try {

              String nom="";

            /**    l'enregistrement du photo du site   */
     out.println("oui save file");


     boolean isMultipart = ServletFileUpload.isMultipartContent(request);

       FileItemFactory factory = new DiskFileItemFactory();

      ServletFileUpload upload = new ServletFileUpload(factory);

      List  items = upload.parseRequest(request);


Iterator iter = items.iterator();
while (iter.hasNext()) {
    FileItem item = (FileItem) iter.next();

    if (item.isFormField()) {
       String fieldName = item.getFieldName();
if(fieldName.equals("name"))
System.out.println(fieldName);
    } else {
```

```
       File fullFile = new File(item.getName());
       nom=fullFile.getName();

              File                    savedFile                    =                    new
File(this.getServletContext().getRealPath("/")+"service\\", fullFile.getName());

out.println(this.getServletContext().getRealPath("/")+"service\\"+fullFile.getName());

item.write(savedFile);

    }
}
  out.println("oui save file");

   ManipulerWsdl                    manp                    =                    new
ManipulerWsdl(this.getServletContext().getRealPath("/")+"service\\"+nom);
  manp.getWSDLService().getName();


                ServletContext sc = this.getServletContext();
                RequestDispatcher                    rd                    =
sc.getRequestDispatcher("/recepservice?nameserv="+manp.getWSDLService().getName());
                rd.forward(request, response);

        } catch (Exception e) {
            out.println(e.getMessage());
        }
    }
}
```

SERVICEONT.JAVA

```
public class serviceont extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
          response.setContentType("text/html");
         PrintWriter out = response.getWriter();

          try {

              String nom="";

            /**    l'enregistrement du photo du site   */
     out.println("Oui, sauvegarder fichier");


     boolean isMultipart = ServletFileUpload.isMultipartContent(request);

       FileItemFactory factory = new DiskFileItemFactory();

      ServletFileUpload upload = new ServletFileUpload(factory);

      List  items = upload.parseRequest(request);

Iterator iter = items.iterator();
while (iter.hasNext()) {
    FileItem item = (FileItem) iter.next();

    if (item.isFormField()) {
       String fieldName = item.getFieldName();
if(fieldName.equals("name"))
System.out.println(fieldName);
```

```
    } else {

       File fullFile = new File(item.getName());
       nom=fullFile.getName();

             File                    savedFile                    =                    new
File(this.getServletContext().getRealPath("/")+"service\\", fullFile.getName());

out.println(this.getServletContext().getRealPath("/")+"service\\"+fullFile.getName());

item.write(savedFile);
    }
}
  out.println("Oui, sauvegarder fichier");

  InputStream in = serviceont.class.getResourceAsStream(nom);

               if (in == null)
              {
                    throw new IllegalArgumentException(
                    "File: " + nom + " not found");
              }

              // read the OWL file


   ManipulerWsdl                    manp                    =                    new
ManipulerWsdl(this.getServletContext().getRealPath("/")+"service\\"+nom);
 out.println( manp.getWSDLService().getName());

 /*
                 ServletContext sc = this.getServletContext();
                 RequestDispatcher                    rd                    =
sc.getRequestDispatcher("/recepservice?nameserv="+manp.getWSDLService().getName());
                 rd.forward(request, response);*/

        } catch (Exception e) {
            out.println(e.getMessage());
        }

    }
}
```

EXTRAIT D'UN FICHIER OWL GENERE

```
- <Service:CategorieRestaurant rdf:about="http://www.daml.org/services/owl-s/1.2/Service.owl#hasvalueCategorieRestaurant">
    <Service:aValeur rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Restaurant_Rapide</Service:aValeur>
  </Service:CategorieRestaurant>
- <Ontology1239957158891:AccesHandicapes rdf:about="http://www.daml.org/services/owl-s/1.2/Service.owl#hasvalueAccesHandicapes">
    <Service:aValeur rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</Service:aValeur>
  </Ontology1239957158891:AccesHandicapes>

- <Service:StyleCuisine rdf:about="http://www.daml.org/services/owl-s/1.2/Service.owl#hasvalueStyleCuisine">
    <Service:aValeur rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Cuisine_Oceanie</Service:aValeur>
  </Service:StyleCuisine>

- <Service:Places_Parking_Disponibles rdf:about="http://www.daml.org/services/owl-s/1.2/Service.owl#hasvaluePlaces_Parking_Disponibles">
    <Service:aValeur rdf:datatype="http://www.w3.org/2001/XMLSchema#long">3</Service:aValeur>
  </Service:Places_Parking_Disponibles>
```

```
− <Ontology1239957158891:PrixRepas rdf:about="http://www.daml.org/services/owl-s/1.2/Service.owl#hasvaluePrixRepas">
    <Service:aValeur rdf:datatype="http://www.w3.org/2001/XMLSchema#long">23</Service:aValeur>
  </Ontology1239957158891:PrixRepas>
− <Ontology1239957158891:Climatisation rdf:about="http://www.daml.org/services/owl-s/1.2/Service.owl#hasvalueClimatisation">
    <Service:aValeur rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</Service:aValeur>
  </Ontology1239957158891:Climatisation>
```

ANNEXE D. FICHIERS DE L'INTERFACE « *UTILISATEUR FINAL* »

ANDROIDMANIFEST.XML

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
	package="utbm.SEM" android:versionCode="1"
	android:versionName="1.0">
	<application	android:icon="@drawable/icon"	android:label="@string/app_name"
android:debuggable="true">
		<activity android:label="@string/app_name" android:name=".SEM">
			<intent-filter>
				<action android:name="android.intent.action.MAIN" />
				<category android:name="android.intent.category.LAUNCHER" />
			</intent-filter>
		</activity>
		<activity android:name=".activity.ServiceDetail"></activity>
	</application>
	<uses-sdk android:minSdkVersion="2" />

<uses-permission android:name="android.permission.INTERNET"></uses-permission>
</manifest>
```

RES/LAYOUT/DETAILS.XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical" android:layout_width="fill_parent"
	android:layout_height="fill_parent">
	<TextView android:layout_width="fill_parent"
		android:layout_height="wrap_content" android:text="@string/hello" />
	<TextView android:id="@+id/lblStatus" android:layout_width="fill_parent"
		android:layout_height="fill_parent" android:text="... editing ..." />
		    <Button android:id="@+id/cmdCalculate"
	    android:layout_width="wrap_content"
	    android:layout_height="wrap_content"
	    android:text="Calculate"
    />
</LinearLayout>
```

GLOSSAIRE

A

ACAN Ad Hoc Context Aware Network

ADSL Asymmetric Digital Subscriber Line

API Applications Programming Interface

ASCII American Standard Code for Information Interchange

Axiome – Il s'agit soit d'un énoncé dont la véracité est évidente, soit d'un énoncé qui ne peut pas être nié sans avoir une contradiction..

B

BPEL Business Process Execution Language

BPEL4WS Business Process Execution Language for Web Services

C

CAS Context-Aware Service

CASD Context-Aware Service Discovery

CC/PP Composite Capabilities / Preference Profile

CC Context Client

CIS Contextual Information Service

CIDS Context Information Distribution System

CIB Context Information Base

CIDS Context Information Dissemination System

CIS Context Information Source

CM Context Mediator

ConQo Context- and QoS- Aware Service Discovery

Contexte – Le contexte est défini comme l'ensemble des informations pouvant être utilisées pour caractériser la situation d'une entité (personne, endroit, objet, etc.)

CORBA Common Object Request Broker Architecture

COSS Context-aware, Ontolgy-based Semantic Service discovery

CPU Central Processing Unit

CSA Context Service Adapter

D

DARPA Defence Advance Research Projects Agency

DDRD Dynamic Decentralized Resource Discovery

DiffServ Differentiated Service

DMC Decision Making Component

DMTF Distributed Management Task Force

DSCP Differentiated Services Code Point

E

ebXML electronic business eXtensible Markup Language

EE Execution Environment

F

FTP File Transport Protocol

G

GUI Graphical User Interface

GPRS General Packet Radio Service

H

HTTP Hyper Text Transfer Protocol

HCI Human Computer Interface

I

IETF Internet Engineering Task Force

IFLA International Federation of Library Associations

I/O Input/Output

IOPEs Inputs, Outputs, Preconditions, Effects

J

JVM Java Virtual Machine

JPEG Joint Photographic Experts Group

K

KIF Knowledge Interchange Format

L

LAN Local Area Network

M

MAC Media Access Control

MDA Model-Driven Architecture

N

NACE Nomenclature Statistique des Activités Economiques

NAICS North American Industry Classification System

O

OASIS Organization for the Advancement of Structured Information Standards

OGSA Open Grid Service Architecture For Distributed Systems Integration

OMG Object Management Group

Ontologie – Une ontologie représente un ensemble structuré de termes et concepts, en lien avec un champ d'informations donné. Une ontologie est un modèle de données illustrant les concepts d'un domaine de connaissance, ainsi que les relations entre ces concepts. Une ontoogie permet de raisonner à propos des concepts ainsi définis.

OWL Web Ontology Language

OWL-S Web Ontology Language for Services

P

PACL Permitted Access Control List

PDA Personal Digital Assistant

PKI Public Key Infrastructure

PM Policy Manager

P2P Peer-To-Peer ou Pair-à-pair

PURL Persistent Uniform Ressource Locator

Q

QoS Quality of Service

QoC Quality of Context

R

RDF Resource Description Framework

RDF(S)

Réification – En informatique, la réification est le procédé qui consiste à transformer un concept en un objet informatique. Dans le cadre de la gestion des connaissances, la réification permet de transformer l'information en connaissance.

REST Representational State Transfer

RPC Remote-Procedure Call

S

SDD Service Definition Document

SDF Service Deployment Framework

SDK Software Development Kit

Sémantique – En linguistique, la sémantique représente une branche qui étudie les signifiés (signification des mots composés, rapport de sens entre les mots, etc.). En informatique, la sémantique fait référence à des règles formelles, définies afin de permettre aux ordinateurs de comprendre les informations manipulées. En informatique, ce terme est utilisé en opposition à celui de syntaxe (pour les langages de programmation). Entre les deux concepts, il y a la même différence qu'entre les concepts de « fond » et « forme ».

Sensibilité au contexte – Il s'agit d'une propriété d'un système, qui utilise le contexte pour furnir des informations et/ou des services pertinents par rapport à la situation de l'utilisateur.

Service Web – Un service Web est défini comme la réalisation d'une ou plusieurs actions par une entité, pour une autre entité. Un service Web est défini dans un domaine donné. L'entité réalisant les acions du service est appelée fournisseur de service, l'autre entité étant le demandeur de service.

Service Web sémantique – Un service Web sémantique respecte la définition d'un service Web, mais sa représentation peut être interprétée par un ordinateur. Un service Web sémantique a plusieurs niveaux d'abstraction. Un service concret est la réalisation spécifique de certaines actions d'une entité pour une autre. Un service abstrait correspond à un ensemble de services concrets. L'avantage d'un service abstrait c'est qu'il peut faire référence à d'autres services, sans avoir à spécifier l'ensemble des aspects définnissant ces services.

SIC Standard Industrial Classification

SICE Service Invocation Condition Evaluator

SIP Session Initiation Protocol

SLO Service Logic Object

SL Service Layer

SLA Service Level Agreement

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

SOAP Simple Object Access Protocol

SQL Structured Query Language

SPARQL SPARQL Protocol And RDF Query Language

SWRL Semantic Web Rule Language

SWSF Semantic Web Services Framework

T

TINA Telecommunication Information Networking Architecture

TCP Transport Protocol

U

UBL Universal Business Language

UCL URL Class Loader

UDDI Universal Description, Discovery And Integration

UML Unified Modeling Language

UN/CEFACT United Nations Center for Trade Facilitation and Electronic Business

UNSPSC United Nations Standard Products and Services Code

URI Universal Ressource Identifier

URL Uniform Resource Locator

V

VLAN Virtual Local Area Network

VoIP Voice Over Internet Protocol

VPN Virtual Private Network

W

W3C World-Wide Web Consortium

WiFi Wireless Fidelity

WIMAX IEEE 802.16 Standard

WLAN Wireless Local Area Network

WSCI Web Service Choreography Interface

WSCL Web Service Conversation Language

WSCL Web Service Choreography Language

WS-CDL Web Services Choreography Description Language

WSDL Web Service Definition Language

WSDL-S Web Services Semantics

WS-I Web Services Interoperability Organization

WSMO Web Service Modeling Ontology

WSML Web Service Modeling Language

WSMX Web Service eXectution Model

WWW World Wide Web

X

XML Extensible Markup Language

XML-RPC Extensible Markup Language Remote Proceduring Calling Protocol

XSL Extensible Stylesheet Language

XSP Extensible Service Protocol

BIBLIOGRAPHIE

(ABOWD 1999) ABOWD, G.D., DEY, A.K., BROWN, P., DAVIES, N., SMITH, M., STEGGLES, P. «Towards a Better Understanding of Context and Context-Awareness.» *Lecture Notes in Computer Science (LNCS) - Proceedings of the 1st International Syposium on Handheld and Ubiquitous Computing.* Karlsruhe, Allemagne: Springer-Verlag, 1999. 304-307.

(AKKIRAJU 2005) AKKIRAJU, R., et al.,. *Web Service Semantics - WSDL-S.* Vers. 1.0. 5 Novembre 2005. http://www.w3.org/Submission/WSDL-S/ (accès le Novembre 10, 2009).

(ALESSO 2006) ALESSO, H.P, SMITH C.F.,. *Thinking on the Web - Berners-Lee, Gödel and Turing.* Édité par John Wiley & Sons. Hoboken, New Jersey: Wiley Interscience, 2006.

(ALLEMANG 2008) ALLEMANG, D., HENDLER J.,. *Semantic Web for the Working Ontologist - Effective Modeling in RDFS and OWL.* Copyright by Elsevier Inc. Burlington MA 01803: Morgan Kaufmann Publishers, 2008.

(ISO 3166 2009) *Autorité de mise à jour de l'ISO 3166 (ISO 3166/MA) – point focal de l'ISO pour les codes de pays.* 2009. http://www.iso.org/iso/fr/country_codes.htm (accès le Novembre 9, 2009).

(B. SENACH 1990) B., SENACH. *Evaluation ergonomique des interfaces homme-machine: une revue de la littérature.* Rapport de recherche, Institut National de Recherche en Informatique et en Automatique (INRIA), 1990.

(BAADER 2002) BAADER, F., CALVANESE, D., MCGUINNESS, D.L., NARDI, D., PATEL-SCHNEIDER, P., et al. *The Description Logic Handbook: Theory, Implementation and Applications.* Édité par Aachen University of Technology BAADER Franz. Cambridge University Press, 2002.

(BAKHOUYA 2008) BAKHOUYA, M., GABER J. «Approaches for Ubiquitous Computing.» Édité par LABIOD H. *Wireless Ad-hoc and Sensor Networks* (ISTE Hermes Science and Lavoisier), 2008: 111-142.

(BALZER 2004) BALZER, S., LIEBIG T., WAGNER M.,. «Pitfalls of OWL-S: a practical semantic web use case.» *Proceedings of the 2nd International Conference on Service-oriented Computing (ICSOC'04).* New York, NY, USA: ACM, 2004. 289-298.

(BATTLE 2005) BATTLE, S., et al.,. *Semantic Web Services Framework (SWSF) Overview.* 9 Septembre 2005. http://www.w3.org/Submission/SWSF/ (accès le Novembre 10, 2009).

(BECHHOFER 2004) BECHHOFER, S., VAN HARMELEN, F., HENDLE, J., HORROCKS, I., MCGUINNESS, D.L., PATEL-SCHNEIDER, P.F., STEIN, L.A.,. *OWL Web Ontology Language Reference.* Édité par M., SCHREIBER, D., DEAN. 10 Février 2004. http://www.w3.org/TR/owl-ref/ (accès le Novembre 9, 2009).

(BELLWOOD 2004) BELLWOOD, T., et al.,. *UDDI Spec Technical Committee Draft.* Copyright © OASIS Open 2002-2004. 19 10 2004. http://uddi.org/pubs/uddi_v3.htm (accès le Novembre 9, 2009).

(BENTHAM 2009) BENTHAM, J. *La classification des crimes et délits.* Édité par DOUCET J.P. 7 Novembre 2009. http://ledroitcriminel.free.fr/la_science_criminelle/les_sciences_juridiques/la_loi_penale/infraction/bentham_classification_delits.htm (accès le Novembre 10, 2009).

(BERNERS-LEE 1998) BERNERS-LEE, T. «Axioms of Web Architecture.» 1998. http://www.w3.org/DesignIssues/Principles.html (accès le Novembre 9, 2009).

(BERNERS-LEE et al. 2001) BERNERS-LEE, T., et al. «The Semantic Web: a New Form of Web Content that is Meaningful to Computers will unleash a Revolution of new Possibilitiés.» *Scientific American* 284, n° 5 (Mai 2001): 34-43.

(BOAG 2007) BOAG, S., et al. *XQuery 1.0: An XML Query Language.* Vers. 1.0. 23 Janvier 2007. http://www.w3.org/TR/xquery/ (accès le Novembre 10, 2009).

(BRAUN 2008) BRAUN, I., STRUNK A., STOYANOVA G., BUDER B.,. «ConQo – A Context- And QoS-Aware Service Discovery.» *Proceedings of IADIS International Conference on WWW/Internet.* Freiburg, Allemagne, 2008.
BRAY, T., PAOLI J., SPERBERG-MCQUEEN C.M., MALER E., YERGEAU F.,. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* 26 Novembre 2008. http://www.w3.org/TR/xml/ (accès le Novembre 9, 2009).

(BREITMANN 2007) BREITMANN, K.K., CASANOVA, M.A., TRUSZKOWSKI, W.,. *Semantic Web: Concepts, Technologies and Applications.* NASA Monographs in Systems and Software Engineering. Édité par Professor Michael Hinchey. Londres: © Springer-Verlag, 2007.

(BROENS 2004) BROENS, T. «Context-aware, Ontology based, Semantic Service (COSS) Discovery.» Thèse de doctorat, Université de Twente, Enschede, Pays-Bas, 2004, 87.

(CHEN et KOTZ 2000) CHEN, G., et D. KOTZ. *A Survey of Context-Aware Mobile Computing Research.* Rapport Technique, Hanover, NH, USA: Darmouth College, 2000.

(H. CHEN 2004) CHEN, H.,. *Context Broker Architecture (CoBrA).* 15 Juillet 2004. http://cobra.umbc.edu/ (accès le Novembre 10, 2009).

(CHINNICI 2007) CHINNICI, R., MOREAU J.J., RYMAN A., WEERAWARANA S.,. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.* W3C. 26 Juin 2007. http://www.w3.org/TR/wsdl20/ (accès le Novembre 9, 2009).

(CHRISTENSEN 2001) CHRISTENSEN, E., CURBERA F., MEREDITH G., WEERAWARANA S.,. *Web Services Description Language (WSDL) 1.1.* W3C. 15 Mars 2001. http://www.w3.org/TR/wsdl (accès le Novembre 9, 2009).

(CLARK 1999) CLARK, J., DEROSE S.,. *XML Path Language (XPath).* Vers. 1.0. 16 Novembre 1999. http://www.w3.org/TR/xpath (accès le Novembre 10, 2009).

(Classfication de Nice 2009) *Classification internationale des produits et des services aux fins de l'enregistrement international des marques établie en vertu de l'Arrangement de Nice.* 2009. http://www.wipo.int/classifications/nice/fr/ (accès le Novembre 9, 2009).

(Contexte 2009) *Contexte.* Wikipédia. 23 Septembre 2009. http://fr.wikipedia.org/wiki/Contexte (accès le Novembre 10, 2009).

(CUDDY 2005) CUDDY, S., KATCHABAW M., LUTYYa H. «Context-aware service selection based on dynamic and static service attributes.» *Proceedings of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob).* Montréal, Canada, 2005. 13-20.

(D&B 2009) D&B. *Numéro D-U-N-S®.* 2009. http://dbfrance.dnb.com/French/default.htm?Loc=/French/DataBase/duns.htm (accès le Novembre 9, 2009).

(DIP 2004) *Data, Information, and Process Integration with Semantic Web Services.* 9 Juillet 2004. http://dip.semanticweb.org/ (accès le Novembre 10, 2009).

(DE BRUIJN 2008) DE BRUIJN, J. *The Web Service Modeling Language (WSML).* 8 Août 2008. http://www.wsmo.org/wsml/wsml-syntax (accès le Novembre 9, 2009).

(DE BRUIJN et al. 2005) DE BRUIJN, J., et al. *Web Service Modeling Ontology (WSMO).* Édité par H., POLLERES, A., ROMAN, D., LAUSEN. 3 Juin 2005. http://www.w3.org/Submission/WSMO/ (accès le Novembre 10, 2009).

(DEY 2001) DEY, A.K.,. «Understanding and Using Context.» *Personal and Ubiquitous Computing* (Springer-Verlag) 5, n° 1 (2001): 4-7.

(Metadata 2005) *Digital Libraries - Metadata Resources.* 24 Octobre 2005. http://archive.ifla.org/II/metadata.htm (accès le Novembre 9, 2009).

(DIMARZIO 2008) DIMARZIO, J.F. *Android - a Programmer's Guide.* McGraw-Hill, 2008.

(DOLCE 2006) *DOLCE : a Descriptive Ontology for Linguistic and Cognitive Engineering.* 7 Juillet 2006. http://www.loa-cnr.it/DOLCE.html (accès le Novembre 10, 2009).

(Dynamic binding 2009) *Dynamic binding.* 27 Octobre 2009. http://en.wikipedia.org/wiki/Dynamic_binding_%28computer_science%29 (accès le Novembre 10, 2009).

(ebXML 2009) *ebXML Specs.* Copyright © OASIS Open 2006. 2009. http://www.ebxml.org/specs/index.htm (accès le Novembre 9, 2009).

(efurgences 2007) efurgences. *Scores et classification des malades aux urgences.* 2007. http://www.efurgences.net/index.php/scores/85-scores-classification-urgences (accès le Novembre 10, 2009).

(ERL 2005) ERL, Thomas,. *Service-Oriented Architecture: Concepts, Technology, and Design.* Copyright © 2005 Pearson Education, Inc. Vol. I. Prentice Hall PTR, 2005.

(EUZENAT 2007) EUZENAT, J., SHVAIKO P. *Ontology Matching.* Berlin: Springer-Verlag, 2007.

(FIELDING 2000) FIELDING, R.T. «Architectural Styles and the Design of Network-based Software Architectures.» Thèse de doctorat, University of California, Irvine, USA, 2000, 180.

(GABHART 2007) GABHART, K., JOHNSON D. «kXML-RPC Enables Service-Oriented Mobile Computing.» *devx.com.* 13 Juillet 2007. http://www.devx.com/Java/Article/34963 (accès le Novembre 10, 2009).

(GENESERETH 1994) GENESERETH, M.R., FIKES, R.E., et al. *Knowledge Interchange Format (KIF) - Reference Manual.* Vers. 3.0. 7 Décembre 1994. http://logic.stanford.edu/kif/Hypertext/kif-manual.html (accès le Novembre 10, 2009).

(GHAFOUR 2004) GHAFOUR, S.A.,. «Méthodes et outils pour l'intégration des ontologies.» Mémoire de stage de DEA "Documents Multimédia, Images, et Systèmes d'Information Communicants", Laboratoire d'InfoRmatique en Images et Systèmes d'information (LIRIS), Ecole Doctorale "nformatique et Information pour la Société" (EDA 335), 2004, 51.

(GHIDINI 2001) GHIDINI, C., GIUNCHIGLIA F. «Local models semantics, or contextual reasoning = locality + compatibility.» *Artificial Intelligence* (Elsevier Science Publishers Ltd.) 127, n° 2 (Avril 2001): 221-259.

(GIAMPAPA 2005) GIAMPAPA, J., PAOLUCCI M., SRINIVASAN N., VACULIN R., et al. *Translator from WSDL to OWL-S.* 6 Juin 2005. http://www.semwebcentral.org/projects/wsdl2owl-s/ (accès le Novembre 10, 2009).

(GRAY 2001) GRAY, P.D., SALBER D. «Modelling and Using Sensed Context Information in the Design of Interactive Applications.» *Lecture Notes in Computer Science (LNCS) - Proceedings of the 8th International Conference on Engineering for Human-Computer Interaction (IFIP).* Londres, Angleterre: Springer-Verlag, 2001. 317-336.

(GRIMM 2007) GRIMM, S. «Discovery - Identifying Relevant Services.» Chap. 8 dans *Semantic Web Services - Concepts, Technologies and Applications*, de GRIMM S., ABECKER A. éd. par STUDER R. Berlin: Springer-Verlag, 2007.

(GRIMM et al. 2007a) GRIMM, S., et al. «Knowledge Representation and Ontologies - Logic, Ontologies and Semantic Web Languages.» Chap. 3 dans *Semantic Web Services - Concepts, Technologies and Applications*, de R., GRIMM, S., ABECKER, A. éd. par STUDER, 51-105. Berlin: Springer-Verlag, 2007a.

(GUARINO 1997) GUARINO, N.,. «Understanding, Building and Using Ontologies.» *International Journal of Human Computer Studies - Special Issue on using Explicit Ontologies in KBS Development* (Academic Press, Inc.) 46, n° 2-3 (Février-mars 1997): 293-310.

(GUDGIN 2007) GUDGIN, M., HADLEY M., MENDELSON N., MOREAU J.J., NIELSEN H.F., KARMARKAR A., LAFON Y.,. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* Vers. 2ème. W3C. 27 Avril 2007. http://www.w3.org/TR/soap12-part1/ (accès le Novembre 9, 2009).

(HELD 2002) HELD, A., BUCHHOLZ S., SCHILL A.,. «Modeling of Context Information for Pervasive Computing Applications.» *Proceedings of SCI 2002 / ISAS 2002.* Orlando, USA, 2002.

(HENDLER 2001) HENDLER, J. «Agents and the Semantic Web.» *IEEE Intelligent Systems* 16, n° 2 (Mars-avril 2001): 30-37.

(HENRICKSEN et al. 2003) HENRICKSEN, K., et al. «Generating Context Management Infrastructure from High-Level Context Models.» *Proceedings of the 4th International Conference on Mobile Data Management (MDM'03).* Melbourne, Australia, 2003. 1-6.

(HENRICKSEN et INDULSKA 2006) HENRICKSEN, K., et J. INDULSKA. «Developing context-aware pervasive computing applications: Models and approach.» *Pervasive and Mobile Computing* (Elsevier) 2, n° 1 (Février 2006): 37-64.

(HERMAN 2007) HERMAN, I.,. *Web Ontology Language (OWL).* Édité par W3C. 15 Octobre 2007. http://www.w3.org/2004/OWL/ (accès le Novembre 9, 2009).

(HOFER 2003) HOFER, T., SCHWINGER W., PICHLER M., LEONHARTSBERGER G. *Proceedings on the 36th Annual Hawaii International Conference on System Sciences (HICSS'03).* Big Island,Hawaii: IEEE Computer Society, 2003.

(Homographe 2009) *Homographe.* 4 Octobre 2009. http://fr.wikipedia.org/wiki/Homographe (accès le Novembre 5, 2009).

(HORROCKS et al. 2004) HORROCKS, I., et al. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* Édité par W3C. 21 Mai 2004. http://www.w3.org/Submission/SWRL/ (accès le Novembre 9, 2009).

(INDULSKA 2003) INDULSKA, J., ROBINSONA R., RAKOTONIRAINY A., HENRICKSEN K.,. «Experiences in Using CC/PP in Context-Aware Systems.» *Lecture Notes in Computer Science (LNCS) - Proceedings of the 4th International Conference on Mobile Data Management.* Springer-Verlag, 2003. 247-261.

(JSP 2009) *JavaServer Pages.* 5 Novembre 2009. http://fr.wikipedia.org/wiki/JavaServer_Pages (accès le Novembre 10, 2009).

(JSP 2009a) *JavaServer Pages Technology.* 2009a. http://java.sun.com/products/jsp/index.jsp (accès le Novembre 10, 2009).

(Jena 2009) *Jena – A Semantic Web Framework for Java.* 2009. http://jena.sourceforge.net/ (accès le Novembre 10, 2009).

(KAVANTZAS 2004) KAVANTZAS, N., BURDETT D., RITZINGER G., FLETCHER T., LAFON Y.,. *Web Services Choreography Description Language (WS-CDL).* Vers. 1.0. W3C. 11 Décembre 2004. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/ (accès le Novembre 10, 2009).

(KLYNE 2004) KLYNE, G., REYNOLDS, F., WOODROW, C., OHTO, H., HJELM, J., BUTLER, M.H., TRAN, L. *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies.* Vers. 1.0. W3C. 15 Janvier 2004. http://www.w3.org/TR/CCPP-struct-vocab/ (accès le Novembre 10, 2009).

(KORPIPAA 2003) KORPIPAA, P., MANTYJARVI J., KELA J., KERANEN H., MALM E.J.,. «Managing context information in mobile devices.» *IEEE Pervasive Computing* (IEEE Computer Society) 2, n° 3 (Juillet-septembre 2003): 42-51.

(kXML 2005) *kXML.* 4 Décembre 2005. http://kxml.sourceforge.net/index.shtml (accès le Novembre 10, 2009).

(CCMU 2009) «La Classification Clinique des Malades des Urgences modifiée.» *Observatoire des Urgences de Midi-Pyrénées.* 15 Octobre 2009. http://www.oru-mip.fr/docs/ccmu.pdf (accès le Novembre 10, 2009).

(LASSILA 2005) LASSILA, O. «Using the semantic web in mobile and ubiquitous coputing.» *Proceedings of the Working Conference on Industrial Applications of Semantic Web.* Jyvaskyla, Finlande: International Federation of Information Processing (IFIP), 2005.

(LAUSEN 2007) LAUSEN, H., LARA R., POLLERES A., DE BRUIJN J. «Description.» Chap. 7 dans *Semantic Web Services - Concepts, Technologies and Applications*, de GRIMM S., ABECKER A. éd. par STUDER R., 402. Berlin: Springer-Verlag, 2007.

(Pompiers 2009) «Les statistiques des services d'incendie et de secours édition 2009.» *Ministère de l'Intérieur, de l'Outre-Mer et des Collectivités Territoriales.* 6 Août 2009. http://www.interieur.gouv.fr/sections/a_votre_service/statistiques/securite_civile/2008/statistiques-2008/view (accès le Novembre 10, 2009).

(Lingua franca 2009) *Lingua franca.* 14 Octobre 2009. http://fr.wikipedia.org/wiki/Lingua_franca (accès le Novembre 9, 2009).

(NACE 2009) *List of NACE codes.* 9 Novembre 2009. http://ec.europa.eu/competition/mergers/cases/index/nace_all.html (accès le Novembre 10, 2009).

(MAEDCHE et STAAB 2001) MAEDCHE, A., et S. STAAB. «Ontology Learning for the Semantic Web.» Édité par IEEE Educational Activities Department. *IEEE Intelligent Systems* 16, n° 2 (Mars 2001): 72-79.

(MAHESH 2001) MAHESH, K. *Text Retrieval: a Primer.* Rapport de recherche, Oracle Corporation, 2001.

(MARTIN 2004) MARTIN, D., et al. *OWL-S: Semantic Markup for Web Services.* 22 Novembre 2004. http://www.w3.org/Submission/OWL-S/ (accès le Novembre 9, 2009).

(MARTIN 2008) —. *OWL-S: Semantic Markup for Web Services.* Décembre 2008. http://www.ai.sri.com/daml/services/owl-s/1.2/overview/ (accès le Novembre 10, 2009).

(MCCARTHY 1997) MCCARTHY, J., BUVAC S.,. «Formalizing Context (Expanded Notes).» *Working Papers of the AAI Fall Symposium on Context in Knowledge Representation and Natural Language.* California: American Association for Artificial Intelligence (AAAI), 1997. 99-135.

(MCDERMOTT 2004) MCDERMOTT, D.,. «DRS: A Set of Conventions for Representing Logical Languages in RDF.» 12 Janvier 2004. http://www.cs.yale.edu/homes/dvm/daml/DRSguide.pdf (accès le Novembre 10, 2009).

(MCGUINESS 2004) MCGUINESS, D.L., VAN HARMELEN F.,. *OWL Web Ontology Language.* Édité par W3C. 10 Février 2004. http://www.w3.org/TR/owl-features/ (accès le Novembre 9, 2009).

(MEIER 2009) MEIER, R. *Professional Android Application Development.* Indianapolis, Indiana: Wiley Publishing, 2009.

(MITCHELL 2002) MITCHELL, K. «Supporting The Development of Mobile Context-Aware Systems.» Thèse de doctorat, Computer Department, Lancaster University, Lancaster, Angleterre, 2002, 224.

(MOTIK 2005) MOTIK, B., SATTLER, U., STUDER, R.,. «Query Answering for OWL-DL with rules .» *Web Semantics: Science, Services and Agents on the World Wide Web* (Elsevier) 3, n° 1 (Juillet 2005): 41-60.

(MURPHY 2009) MURPHY, M.L. *Beginning Android.* Apress, 2009.

(NAICS 2009) NAICS. *North American Industry Classification System (NAICS).* U.S. Census Bureau. 11 Juin 2009. http://www.census.gov/eos/www/naics/ (accès le Novembre 9, 2009).

(NARAYANAN 2002) NARAYANAN, S., MCILRAITH, S.A.,. «Simulation, verification and automated composition of web services.» Édité par ACM (Association for Computing Machinery). *Proceedings of the 11th international conference on World Wide Web.* Honolulu, Hawaï, 2002. 77-88.

(PAOLUCCI 2003) PAOLUCCI, M., SRINIVASAN N., SYCARA K., NISHIMURA T. «Towards a Semantic Choreography of Web Services: from WSDL to DAML-S.» *Proceedings of the 1st International Conference on Web Services (ICWS).* Las Vegas, NV, USA, 2003.

(PAWAR 2006) PAWAR, P., TOKMAKOFF A.,. «Ontology-Based Context-Aware Service Discovery for Pervasive Environments.» *Proceedings of the 1st IEEE International Workshop on Services Integration in Pervasive Environments (SIPE'2006).* Lyon, France, 2006.

(PURL 2009) *Persistent Uniform Resource Locator.* 19 Avril 2009. http://en.wikipedia.org/wiki/Persistent_Uniform_Resource_Locator (accès le Novembre 10, 2009).

(Protégé 2009) *Protégé.* 2009. http://protege.stanford.edu/ (accès le Novembre 10, 2009).

(PRUD'HOMMEAUX 2008) PRUD'HOMMEAUX, E., SEABONE A. *SPARQL Query Language for RDF.* 15 Janvier 2008. http://www.w3.org/TR/rdf-sparql-query/ (accès le Novembre 10, 2009).

(Réification 2009) *Réification.* 12 Septembre 2009. http://fr.wikipedia.org/wiki/Réification (accès le Novembre 9, 2009).

(ROXIN et al. 2008) ROXIN, A., et et al. «Location Models for Pervasive Road Networks.» (Journal of International Transactions on Systems Science and Applications) 4, n° 2 (Mai 2008): 162-166.

(ROXIN et al. 2008a) —. «Middleware Models for Location-Based Services: a Survey.» *Proceedings of the 2nd International Workshop on Agent-oriented software engineering challenges for Ubiquitous and Pervasive Computing (AUPC).* Sorrento, Italy, 2008a. 35-40.

(ROXIN et al. 2007) —. «Survey of wireless geolocation techniques.» *IEEE Globecom Workshops.* Washington DC, USA, 2007. 1-9.

(ROXIN et al. 2009) —. «TransportML: a Middleware for Location-Based Services Collaboration.» *IEEE Proceedings of the 2nd International Workshop on Service-computing, Context-aware, Location-aware and Positioning techniques (SCLP'09).* Cairo, Egypt., 2009.

(SCHILIT 1994) SCHILIT, B., ADAMS, N., WANT, R.,. «Context-aware computing applications.» *Proceedings of the Workshop on Mobile Computing Systems and Applications.* Santa Cruz, CA, USA: IEEE Computer Society, 1994. 85-90.

(B. SENACH 1990) SENACH, B. *Evaluation ergonomique des interfaces homme-machine: une revue de la littérature.* Rapport de recherche RR-1180, INRIA, 1990.

(Service 2009) *Service.* 6 Octobre 2009. http://fr.wikipedia.org/wiki/Service (accès le Novembre 9, 2009).

(Service 2009a) *Service.* Barry & Associates, Inc. 2009a. http://www.service-architecture.com/web-services/articles/service.html (accès le Novembre 9, 2009).

(SHENG 2005) SHENG, Q.Z., BENATALLAH B.,. «ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development.» *Proceedings of the International Conference on Mobile Business (ICMB).* IEEE Computer Society, 2005. 206-212.

(SOWA 1997) SOWA, J.F. «Principles of Ontology.» *Knowledge Systems, AI Laboratory - Stanford University.* 3 Décembre 1997. http://www-ksl.stanford.edu/onto-std/mailarchive/0136.html (accès le Novembre 9, 2009).

(SPERBERG-MCQUENN 2008) SPERBERG-MCQUENN, C.M., THOMPSON, H.,. *XML Schema.* Édité par W3C. 24 Décembre 2008. http://www.w3.org/XML/Schema (accès le Novembre 9, 2009).

(SIC 2008) *Standard Industrial Classification (SIC) Code List.* 13 Mai 2008. http://www.sec.gov/info/edgar/siccodes.htm (accès le Novembre 10, 2009).
STRANG, T., POPIEN, C.L.,. «A Context Modeling Survey.» *Workshop on Advanced Context Modelling, Reasoning and Management - The Sixth International Conference on Ubiquitous Computing (UbiComp).* 2004.

(SQL 2009) *Structured Query Language (SQL).* 8 Novembre 2009. http://fr.wikipedia.org/wiki/Structured_Query_Language (accès le Novembre 10, 2009).

(Synonymes 2009) *Synonymes.* 9 Novembre 2009. http://fr.wikipedia.org/wiki/Synonymes (accès le Novembre 5, 2009).

(Taxinomie 2009) *Taxinomie.* 2009 Octobre 2009. http://fr.wikipedia.org/wiki/Taxonomie (accès le Novembre 9, 2009).

(kSOAP 2003) *The home of kSOAP.* 25 Août 2003. http://ksoap.objectweb.org/ (accès le Novembre 10, 2009).

(kXML-RPC 2002) *The home of kXML-RPC.* 2002. http://kxmlrpc.objectweb.org/ (accès le Novembre 10, 2009).

(Intelligent Software Agents 2009) *The Intelligent Software Agents Lab.* 2009. http://www.cs.cmu.edu/~softagents/ (accès le Novembre 10, 2009).

(OWL-S Matchmaker 2002) *The OWL-S Matchmaker.* Carnegie Mellon University. 2002. http://www.cs.cmu.edu/~softagents//daml_Mmaker/daml-s_matchmaker.htm (accès le Novembre 10, 2009).

(Thomas Register 2009) *Thomas Register.* © 2009 Wikpedia Foundation. 5 Novembre 2009. http://en.wikipedia.org/wiki/Thomasnet (accès le Novembre 9, 2009).

(UNSPSC 2009) *United Nations Standard Products and Services Code (UNSPSC).* 2009. http://www.unspsc.org/ (accès le Novembre 10, 2009).

(VALLEE 2006) VALLEE, M., RAMPARANY F., VERCOUTER L. *Composition et adaptation contextuelle de services pour la communication ambiante: un état de l'art.* Centre G2I de l'ENSM, 2006.

(UAProf 2001) «WAG UAProf.» *Open Mobile Alliance.* © 2001, Wireless Application Forum, Ltd. 20 Octobre 2001. http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf (accès le Novembre 10, 2009).

(WANG 2004) WANG, X.H., ZHANG D.Q., GU T., PUNG H.K. «Ontology-based context modeling and reasoning using OWL.» *Proceedings of Workshops of the Second IEEE Annual Conference on Pervasive Computing and Communications.* IEEE Computer Society, 2004.

(WEISER 1991) WEISER, M. «The Computer for the Twenty-First Century.» *Scientific American* (Scientific American) 265, n° 3 (1991): 94-104.

(WELTY 2000) WELTY, C.,. «Towards a Semantics for the Web.» *Proceedings of Invited Presentation at the Dagstuhl Symposium on Semantics for the Web.* Dagstuhl: Schloss Dagstuhl, 2000.

(WINER 1999) WINER, D.,. *XML-RPC Specification.* 2004-2009 Scripting News, Inc., 1998-2004 UserLand Software, Inc. 15 Juin 1999. http://www.xmlrpc.com/spec (accès le Novembre 9, 2009).

(ZHU 2004) ZHU, M.,. *Recall, Precision and Average Precision.* Rapport de recherche, Department of Statistics and Actuarial Science, Canada: University of Waterloo, 2004.

(ZHU et al. 2005) ZHU, F., MUTKA M.W., NI L.M.,. «Service Discovery in Pervasive Computing Environments.» *IEEE Pervasive Computing* (IEEE Computer Society) 4, n° 4 (Octobre-décembre 2005): 81-90.

PRODUCTION SCIENTIFIQUE PERSONNELLE

PUBLICATIONS DANS DES JOURNAUX

« LOCATION MODELS FOR PERVASIVE ROAD NETWORKS »

- ROXIN A., WACK M.
- Journal of International Transactions on Systems Science and Applications (ITSSA)
- May 2008
- Vol.4, n° 2, p. 162-166.

"Context-aware mobile computing aims at designing applications that automatically adapt their behaviors to the available location information and the available nearby sensors and devices. This is done in order to fulfill tasks in a way that suits users' current context best. To achieve this, context representation and manipulation are important issues, so as to establish formal context models. In this paper, basic elements of context-aware systems are described with an emphasis on location information representations. Space models for location-based applications are presented. Considered realistic applications concern intelligent vehicles and pervasive road networks."

« SEMANTIC WEB SERVICE DISCOVERY »

- ROXIN A., WACK M.
- *Journal of Web Semantics*
- Identifiant article : JWS-D-09-00140
- Soumis en août 2009.

"This article is a survey of main existing approaches in Semantic Web Services discovery process. The first part of the article presents main definitions regarding Semantic Web Services. Next are discussed main aspects of a discovery framework, notably architectural components and matching algorithms. Main discovery approaches, WSDL-S, OWL-S, WSMO and SWSF are addressed. Finally, key challenges as well as remaining open issues are briefly identified. The scope of this article is to provide the reader with the necessary knowledge about current and future research in semantic-based service discovery."

PUBLICATIONS DANS DES CONFERENCES INTERNATIONALES AVEC COMITE DE LECTURE

TRANSPORTML: A MIDDLEWARE FOR LOCATION-BASED SERVICES COLLABORATION

- ROXIN A., DUMEZ C., COTTIN N., WACK M., GABER J.,
- IEEE 2nd International Workshop on Service-computing, Context-aware, Location-aware and Positioning techniques (SCLP'09)
- December 20-23, 2009
- Cairo, Egypt

"Location-Based Services (LBS) are information and entertainment services, accessible with mobile devices and making use of the position of the mobile device. GNSS is usually used to deliver location information.

Although LBS are now widely used, better results could be obtained by merging information issued by several services. Inter-LBS collaboration is still uncommon although a lot of work is done on service orchestration languages such as WS-BPEL 2.0. One of the obstacles is that LBS are created and updated on the fly. This dynamism makes manual composition scenario update difficult.

To tackle this problem, we introduce TransportML, a middleware for inter-LBS collaboration, in an automatic fashion. TransportML relies on technologies such as semantic Web services and semantic Web. In this platform, services are discovered and collaboration scenarios are elaborated on the fly. As a consequence, the platform is auto-adaptive and composition scenario maintenance is no longer required."

MIDDLEWARE MODELS FOR LOCATION-BASED SERVICES : A SURVEY

- ROXIN A., DUMEZ C., WACK M., GABER J.,
- ICPS Proceedings of the 2nd international workshop on Agent-oriented software engineering challenges for ubiquitous and pervasive computing (AUPC)
- 6-10 July, 2008
- Sorrento, Italy
- ISBN : 978-1-60558-206-1
- p. 35-40.

"Embedded computing systems, sensor networks, LBS pervasive deployment environments, and worldwide computing systems have common characteristics. They are large scale, decentralized and dynamic networks, and needing context-awarness to automatically adapt their behaviour and continue their execution despite network dynamics. Identifying innovative software engineering approaches that take into account all the above mentionned characteristics is a real challenge. This paper focuses on LBS applications and the middleware models required for supporting their operation and characteristics."

MODELING OF COOPERATIVE NAVIGATION IN PERVASIVE E-LEARNING APPLICATIONS USING (MAX, PLUS) ALGEBRA

- NAIT-SIDI-MOH A., ASSOUSSOU D., ROXIN A., WACK M.
- IEEE Globecom Workshops
- 26-30 November, 2007
- Washington DC, USA
- DOI : 10.1109/GLOCOMW.2007.4437808
- ISBN : 978-1-4244-2024-7
- p. 1-7.

"This paper presents a new approach based on (max, plus) algebra to model cooperative navigation in pervasive e- learning applications. This modeling approach is appropriate for modeling these systems that are governed, on the one hand, by events occurrence in a discrete state space, and on the other hand, by synchronization and parallelism phenomena. In addition, this approach enables to verify in a formal way the model consistency."

SURVEY OF WIRELESS GEOLOCATION TECHNIQUES

- ROXIN A., GABER J., WACK M., NAIT-SIDI-MOH A.
- IEEE Globecom Workshops
- 26-30 November, 2007
- Washington DC, USA
- DOI: 10.1109/GLOCOMW.2007.4437809
- ISBN : 978-1-4244-2024-7
- p. 1-9.

- *"Ubiquitous and pervasive networks and applications include a growing number of research themes. In most use cases, applications require location information to interpret their environment and behave accordingly. In this paper, location algorithms and positioning methods that can be used for wireless geolocation are presented."*

LOCATION-BASED MODELS FOR PERVASIVE ROAD NETWORKS

- ROXIN A., WACK M., GABER J.,
- International Conference on Pervasive Services (ICPS'07)
- 15-20 July, 2007
- Istanbul, Turkey
- ISBN : 1-4244-1325-7
- DOI: 10.1109/PERSER.2007.4283954
- p. 443-447.

- *"Context-aware mobile computing aims at designing applications that automatically adapt their behaviors to the available location information and the available nearby sensors and devices. This is done in order to fulfill tasks in a way that suits users' current context best. To achieve this, context representation and manipulation are important issues, so as to establish*

formal context models. In this paper, basic elements of context-aware systems are described with an emphasis on location information representations. Space models for location-based applications are presented. Considered realistic applications concern intelligent vehicles and pervasive road networks."

PUBLICATIONS DANS DES LIVRES

L'INFORMATION GEOGRAPHIQUE

- ROXIN A.,
- *Géopositionnement et mobilités* / ed. par UTBM
- 2009
- ISBN : 978-2-914279-40-6
- p. 57-91.

« Ce chapitre présente les différents types de représentation de l'information géographique, ainsi que les modélisations standardisées de ce type d'information (SVG, GML, GDF ou UML). Nous nous intéressons ensuite aux différents procédés permettant la collecte de ces informations (photogrammétrie, laser scanning). Finalement, nous listons les principales sources d'informations géographiques (sources de référence, sources thématiques, etc.). »

COMMUNICATIONS INTER-VEHICULES

- ROXIN A.
- *Géopositionnement et mobilités* / ed. par UTBM
- 2009
- ISBN : 978-2-914279-40-6
- p. 163-186.

« Ce chapitre traite des principaux standards permettant la communication inter-véhicules. Il s'agit notamment du DSRC (Dedicated Short Range Communications), WAVE (Wireless Air Vehicular Environment) et CALM (Continuous Air Long Medium interface). Les principaux projets européens ayant implementé la communication inter-véhicules sont aussi présentés. »