République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderahmane Mira de Béjaïa

Faculté des Sciences et des Sciences de l'Ingénieur

Département d'Informatique

École Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère

En Informatique

Option

Réseaux et Systèmes Distribués

Thème

DÉCOUVERTE ET COMPOSITION DES SERVICES WEB APPROCHE PAR LES GRAPHES

Présenté par

Nadia TASSOULT

Soutenu devant le jury composé de :

Président	Saâdia TAS	Maitre de conférence, U. A/Mira, Béjaïa, Algérie
Rapporteur	Hamamache KHEDDOUCI	Profeseur, UCBL, Lyon, France
Examinateur	Mohamed BENMOHAMED	Profeseur, U. Mentouri, Constantine, Algérie
Examinateur	Abdallah BOUKERRAM	Maitre de conférence, UFA, Sétif, Algérie

Promotion 2005 – 2006

Résumé

Les services Web sont des applications modulaires indépendantes, auto-déscriptives, qui peuvent être publiées, localisées, et invoquées à travers le Web. De nos jours, plusieurs compagnies et organisations implémentent leurs services métiers sur Internet. Donc, assurer l'efficacité de la sélection et l'intégration inter-organisations des services sur le Web au cours de leurs exécutions est une étape importante pour la provision des services Web. Le problème majeur dans ce cas est de développer des mécanismes permettant de localiser automatiquement le service Web correct répondant aux exigences de l'utilisateur. En particulier, si aucun service Web ne peut satisfaire une requête d'un client, il devrait y avoir une possibilité de combiner un ensemble de services Web existants pour accomplir cette requête. Cependant, les services Web sont créés et mis à jour à la volée. L'analyse et la génération d'un plan de composition manuellement dépasse la capacité humaine. Cette tendance a déclenché un nombre considérable d'efforts de recherche sur la composition de services Web, tant dans le milieu académique que dans le milieu industriel.

Dans ce mémoire, nous nous intéresserons à la composition automatique de services Web. Nous étudions d'abord le contexte de recherche à savoir les services Web, technologies associées, découverte et composition. Puis, nous donnons une vue d'efforts de recherche récents de la composition automatique. Ceci conduit à étudier la possibilité d'appliquer à ce problème une nouvelle approche en se basant sur les graphes. Notre principale contribution est de prouver la faisabilité d'une telle approche à répondre au problème de la composition automatique de services Web, pour ceci nous avons proposé un algorithme de composition basé sur ce formalisme.

Mots clés : Service Web, Découverte automatique, Composition automatique, Graphe.

Abstact

Web services are self-contained, modular business applications that have open, Internet-oriented, standard-based interfaces,which can be published, located, and called across the Web.This is a reason why more and more companies and organizations now implement their core business over Internet. Thus the ability to efficient selection and integration of inter-organizational services on the Web at runtime becomes an important issue to the Web service provision. The general problem is about how to develop mechanisms to automatically locate the correct Web service in order to meet the user's requirements. In some cases, if no single Web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfil the request. However, the Web services are created and updated at the flight. That exceeds the human capacity to analyze them and generate a plan of composition manually. This tendency started a considerable number of research efforts on Web services composition from both the industry and the academia.

In this thesis, we focus on the automated composition of Web services at runtime. In order to achieve our propose, we first study the different related topics : Web services, related technologies, discovery and composition. Then, we give a sight of recent research efforts towards the study of automatic composition. It drives to study the possibility to apply to this problem a new approach basing on the graphs. Our main contribution is to prove the feasibility of such an approach to answer to the problem of the automatic composition of Web services, for this we proposed an algorithm of composition based on this formalism.

Keywords : Web service, Automatic discovery, Automatic composition, Graph.

Dédicaces

A mes très chers parents,
A ceux qui aiment Nadia,
A moi même,
Et à la belle vie...j'espère,
Je dédie ce travail.

Remerciements

NOUS tenons à remercier en premier lieu le Dieu ; tout puissant qui nous a accordé la volonté et le courage pour la réalisation de ce travail.

Si ce mémoire a pu voir le jour, c'est certainement grâce au soutien et l'aide de plusieurs personnes qui m'ont permis d'accomplir ce travail dans des conditions idéales. Je profite de cet espace pour les remercier tous.

Mes vifs remerciements accompagnés de toute ma gratitude s'adressent à mon encadreur Hamamache KHEDDOUCI, professeur à l'*université Claude Bernard Lyon 1*, pour m'avoir proposé ce sujet intéressant, pour ses précieux conseils. Je le remercie surtout pour la confiance qu'il a manifesté à mon égard.

Mes plus sincères remerciements vont également aux membres de jury qui m'ont fait l'honneur de participer à ma soutenance de mémoire. Je souhaite mentionner : M^{me} Saâdia TAS, Maître de conférence à l'*université de Béjaïa*, M^{r} Mohamed BENMOHAMED, Professeur à l'*université de Constantine* et M^{r} Abdallah BOUKERRAM, Maître de conférence à l'*université de Sétif* (UFA).

Merci pour *M^{r}* A. TARI, chef de département d'informatique à l'*Université A/MIRA* de *Béjaïa*, pour les efforts qu'il a fourni pour que notre formation puisse se dérouler. Par l'occasion nous ne pouvons oublier d'exprimer notre gratitude à tous les enseignants qui ont intervenu durant notre formation au sein de l'école doctorale ReSyD2.

J'exprime ma reconnaissance que j'éprouve envers tous ceux qui, de près ou de loin, ont contribué, par leurs savoir faire, leurs conseils, leurs soutient moral ou leurs amitiés à l'aboutissement de ce travail.

Je ne pourrais clôturer ces remerciements sans me retourner vers les êtres qui mes sont le plus chers, qui ont eu un rôle essentiel et continu pendant ma réussite, et qui sans eux aucune réussite n'aurait été possible. J'adresse de tout mon cœur mes remerciements à ma chère mère qui fut toujours mon seul exemple, je lui suis infiniment reconnaissant pour son soutien, son amour et la confiance qu'elle m'a offert tout au long de ces années. Un très grand merci à mon père que Dieu le plus puissant m'aide pour lui rendre un minimum de ce que je lui dois. Je veux leurs dire que leurs soutien a été toujours ma source d'espoir et m'a incité toujours à penser à s'améliorer chaque lendemain. Qu'ils trouvent dans ce travail le fruit de leurs travail.

ReSyD2, Université de Béjaia — *Nadia TASSOULT*
30 Avril, 2007

The computing field is always in need of new cliches.
Alan Perlis

Introduction générale

L' évolution technologique qui a lieu dans l'actualité informatique telles que les bases de données relationnelles, l'amélioration des langages s'exécutant du coté serveur ainsi que la création de langages de script ont participé à reconsidérer la façon dont on peut mettre à disposition les informations sur Internet. Effectivement, par ce biais, il est possible d'apporter une dynamisation dans le retour des informations. La vision par laquelle on percevait le Web a évoluée. Ainsi, nous sommes passés d'une vision dans laquelle il a été constitué d'un ensemble de pages accessibles par mots clés à une vision du Web où celui-ci devient un fournisseur de ressources accessibles par leurs contenus. De même, nous assistons à une révolution de la façon dont les entreprises interagissent avec leurs fournisseurs, partenaires et autres clients.

Durant la dernière décennie, le nombre et la diversité des services a considérablement accrus sur le Web, ce qui a abouti à l'apparition du *e-commerce*. Une initiative industrielle a récemment défini un nouveau paradigme pour le *e-commerce* dans lequel les applications sont encapsulées et présentées sous la forme de services électroniques intégrés : *services Web*. Un service Web (*e-service*) peut être considéré comme un site Web qui ne propose pas seulement de l'information, mais qui autorise éventuellement d'effectuer une action ou un changement dans le monde. Ces services Web représentent un défi informatique mais aussi économique.

La technologie des services Web est devenue la clé du monde d'Internet orienté service, dans une période de temps très courte, en permettant l'interaction des applications inconnues et indépendantes sur l'Internet d'une manière plus simple et normalisée que n'importe quelle autre approche jamais réclamée. Même si les technologies telles que CORBA (*Common Object Request Broker Architecture*) et COM (*Commun Object Model*) présentaient les concepts distribués orientés service, elles ne pourraient jamais devenir le choix universel des entreprises et des réalisateurs. Ces architectures ne sont pas ouvertes entre elles car on ne peut assembler que des composants d'une même architecture. L'interopérabilité entre différents systèmes est devenue une nécessité avec l'avènement du B2C (*Business to Consumer*) où les entreprises

offrent des services à leurs clients à travers des applications Web et du B2B (*Business to Business*) qui repose sur l'échange de services et d'information entre les entreprises. Les services Web sont une solution à ce problème d'interopérabilité pour la collaboration entre les différents systèmes indépendamment de toute plate-forme et de tout langage.

Ce qui rend la notion de *e-service* si attirante est qu'il semble que les e-services soient en mesure de permettre une meilleure automatisation dans la gestion des applications, par exemple en permettant leur découverte dynamique, en facilitant la communication et la négociation, voire en permettant leur composition en des services plus complexes. Le problème majeur est comment développer des mécanismes pour localiser automatiquement le service Web et interagir avec ce dernier pour composer des services plus complexes afin de satisfaire l'utilisateur. En particulier, si aucun service Web simple ne peut satisfaire une requête d'un client, il devrait y avoir une possibilité de combiner un ensemble de services Web existants pour accomplir cette requête. De plus, les informations utilisées sont toujours définies de façon aléatoire par les développeurs, en regardant les possibilités qu'offrent le langage XML *(eXtensible Markup Language)* ainsi que d'autres qui y sont associés aujourd'hui, s'est alors posée la question de savoir s'il ne serait pas possible de munir l'information sur le Web d'une sémantique, qui soit interprétable par machine afin de rendre ces tâches semi-automatiques ou automatiques, c'est ce qui s'appelle le *Web sémantique*.

Dans le cadre du Web sémantique, dont le but est de baser les traitements d'informations du Web sur une définition sémantique de celles-ci, de nouveaux langages pour une description formelle plus riche et précise des e-services ont été proposés. Ces langages, comme par exemple DAML-S *(DARPA*[1] *Agent Markup Language Services)*, ont permis la création de dictionnaires structurés, *les ontologies*, regroupant des définitions de e-services basées sur des concepts définis eux-mêmes dans l'ontologie. Grâce à ces langages et les ontologies associées, de nouvelles approches pour une découverte automatique basée sur la définition des e-services, et plus uniquement sur des mots-clés, ont été définies. Ainsi, la notion *services Web sémantiques* est née.

Les progrès récents dans le domaine des services Web rendent,pratiquement, possible la publication, la localisation et l'invocation des applications à travers le Web. C'est une raison pour laquelle de plus en plus les compagnies et les organismes implémentent maintenant le noyau de leurs affaires au-dessus d'Internet. Ces ressources présentées sous le nom de *services Web* devraient pouvoir être facilement exploitables de manière automatique. Pour cela, diverses initiatives ont vu le jour dans les deux milieux académique et industriel afin de rendre

[1]Defence Advanced Research Projects Agency, http ://www.darpa.mil

cette exploitation possible. Ces efforts fournissent des plates-formes et des langages permettant de découvrir, exécuter et intégrer les services. Par exemple, Universal Description, Discovery, and Integration (*UDDI*), Web Services Description Language (*WSDL*), Simple Object Access Protocol (*SOAP*). Quelques autres initiatives, y compris BPEL4WS *(Business Process Execution Language for Web service)* et DAML-S ServiceModel, se focalisent sur la représentation de la composition de services où le flux de processus et les liaisons entre les services sont connus à priori.

Cependant, ces initiatives ne fournissent pas les moyens pour la composition automatique de services existants. Si la fonctionnalité exigée ne peut pas être réalisée par un service simple existant, la composition de services disponibles pour accomplir la demande doit être spécifiée manuellement. Les services Web sont créés et mis à jour à la volée. L'analyse et la génération d'un plan de composition manuellement dépassent la capacité humaine. Par conséquent, la construction de service Web composite automatique ou semi-automatique est critique. À cette fin, plusieurs approches ont été proposées afin de traiter ce problème. La plupart d'elles sont inspirées des recherches de workflow et des techniques de l'intelligence artificielle.

Dans cette étude, nous allons nous fixer pour but de définir ce que sont les services Web, les technologies qui y sont associées, et les thèmes de recherche des services Web à savoir la découverte et la composition de services. Plus précisément, nous concentrons sur la composition automatique de services Web. L'objectif de ce travail est de proposer un cadre basé sur les graphes pour soutenir le processus de composition.

Plan du mémoire

Pour étudier le problème de la découverte et composition de services Web, nous avons divisé notre mémoire en trois parties. Dans la première partie, nous présentons le contexte de recherche. Nous commençons cette partie en évoquant une définition du service Web. Nous verrons comment les services Web peuvent interagir grâce aux couches standard composant leur architecture. Deux architectures ont été proposées : la première dite de référence est dédiée aux services Web élémentaires, la seconde, associée aux services Web composites, est appelée étendue. Par la suite, nous décrivons les technologies sur lesquelles reposent les services Web et les thèmes de recherche telles que : la découverte, la sélection, la substitution et la sécurité des services Web. Pour finir, nous listant les plates-formes facilitant la mise en oeuvre de service Web.

Nous nous focalisons dans la deuxième partie sur l'étude de la composition. Nous concentrons, tout d'abord, sur la description de la problématique de composition de services Web, puis nous étudions le cycle de vie de leur composition, tant pour les services Web élémentaires que pour les services Web composites. Dans la section suivante, nous soulignons les différentes issues ayant un grand impact sur la composition à savoir la coordination, la transaction et la modélisation de conversation. Nous dépeignons les travaux de recherche principaux liés à la composition de services Web. De plus, nous présentons un tableau comparatif des différents travaux. Nous concluons cette partie en effectuant une synthèse de l'état de l'art et en présentant brièvement la contribution de notre travail concernant la composition automatique de services Web.

La troisième partie établit le fondement de notre travail. Nous montrons dans un premier temps comment représenter l'ensemble des services disponibles sous forme de graphe. La définition formelle du modèle de graphe représentant l'ensemble de services est ensuite décrite. Puis, nous étudions comment découvrir les services Web disponibles ainsi comment les composer, en suivant une méthodologie proposée. Nous proposons un algorithme qui calcule cette composition automatiquement. Des exemples illustrant l'approche clôt la partie sur la proposition.

En fin, le travail fut clôturé par une conclusion à travers laquelle une évolution est faite sur le travail développé et les perspectives à envisager.

Partie 1

Les services Web : concepts et technologies

1.1 Introduction

L'architecture de composants distribués a engendré un développement rapide et évolutif d'applications distribuées et complexes. Au cours de ce développement, on a assisté à la mise en place de trois architectures par composants : COM (*Component Object Model* [24]), java RMI (*Remote Method Invocation*) et CORBA (*Common Object Request Broker Architecture* [64]). La mise en oeuvre de ces trois architectures soulève des difficultés dans le cadre d'une infrastructure ouverte telle que Internet. En effet, ces architectures, bien qu'utilisant un modèle objet distribué, proposent chacune sa propre infrastructure. Ce qui impose une forte liaison entre les services offerts par les composants et leurs clients. Ainsi, on ne peut assembler que des objets CORBA (*ou COM*) entre eux.

Parallèlement après l'avènement du B2C, où les entreprises mettent en ligne leurs services pour leurs consommateurs à travers des applications Web, celles-ci souhaitent augmenter leurs productivités à l'aide du paradigme B2B. Le B2B repose sur l'échange de produits, d'informations et de services entre entreprises. Ceci implique la collaboration avec des systèmes proposés par d'autres concepteurs et par conséquent une maîtrise de l'hétérogénéité. L'interoperabilité est ainsi devenue une nécessité pour l'entreprise dans le monde du B2B. Toutes ces technologies ont pavé la voie à une nouvelle solution pour la collaboration entre les différents systèmes ; *les services Web*. À l'instar des technologies précédentes, les services Web s'appuient sur les réseaux Internet et emploient des standards répandus comme HTTP[1], SMTP[2] et XML[3]. Ainsi, ils bénéficient des avantages du coût, de la simplicité et du support de l'industrie.

[1]Hyper Text Transport Protocol
[2]Simple Mail Transfert Protocol
[3]eXtensible Markup Language

1.2 Les services Web

Le modèle des services Web repose sur une architecture orientée service SOA [14] *(Service Oriented Architecture)*. L'architecture SOA est un modèle abstrait qui définit un système par un ensemble d'agents logiciels distribués qui fonctionnent ensemble afin de réaliser une fonctionnalité globale préalablement définie. Celle-ci fait intervenir trois catégories d'acteurs : les fournisseurs de services (*i.e. les entités responsables du service Web*), les clients qui servent d'intermédiaires aux utilisateurs de services et les annuaires qui offrent aux fournisseurs la capacité de publier leurs services et aux clients le moyen de localiser leurs besoins en terme de services. Il s'agit d'une technologie qui privilégie la logique métier et n'utilise la technique que pour mettre en oeuvre l'architecture. Un service est une notion abstraite pour désigner les fonctionnalités d'un agent logiciel qui implémente une fonctionnalité de service.

Les services Web sont un ensemble de standards permettant aux applications d'interagir à travers le Web indépendamment de toute plate-forme et de tout langage. De manière plus conceptuelle, cela ouvre tous les dialogues possibles entre deux applications. La garantie de cet ensemble de spécifications vient de ce qu'il s'appuie sur les standards éprouvés du Web : (XML, HTTP, etc.). On utilise donc la même recette que ce qui fait le succès de l'Internet, en résumé HTTP et IP[4]. Ils reposent sur une architecture par composants qui utilisent des protocoles Internet comme infrastructure pour la communication entre les différents composants. Les normes des langages et des protocoles qui sont associées aux services Web sont portées, soit par le W3C[5], soit par le consortium OASIS[6].

1.2.1 L'origine des services Web

Les premières implémentations du modèle client-serveur ont posé des questions essentielles parce qu'il implique un fort couplage entre le client et le serveur et qu'il est appuyé sur une logique dans laquelle le serveur est prioritaire. Parallèlement, un modèle plus souple, fournisseur-consommateur, a toujours existé. Ce modèle suppose un couplage faible, lors de l'utilisation, entre le client et le serveur. Le serveur ne connaît pas son client, le client ne fige pas sa relation avec le serveur. L'interaction est fondée sur un contrat défini très précisément par le fournisseur.

[4]Internet Protocol
[5]World Wide Web Consortium, http ://www.w3.org
[6]Organization for the Advancement of Structured Information Standards, http ://www.oasis-open.org/

Face à la demande croissante d'interconnexions entre les applications, il arrive très souvent que chaque application se transforme en producteur d'information et que les solutions mises en oeuvre soient de type '*connexion point à point*'. C'est pourquoi de nouveaux concepts ont émergé : l'EAI-*Enterprise Application Integration*- répond à ce besoin de ne pas tirer des fils entre chaque application et permet de créer un bus d'échange. Pour fonctionner, que ce soit en mode *point à point* ou via un *EAI*, il fallait un standard. C'est précisément ce à quoi souhaitent répondre les spécifications des services Web.

Les services Web répondent à une double demande : celle d'un échange sûr et celle d'un échange ouvert. Il s'agit de donner les moyens à deux applications de s'échanger simplement. L'échange ouvert est permis, puisqu'un serveur ne doit pas nécessairement connaître à l'avance son client pour offrir son service vers l'extérieur [52].

1.2.2 Définition et description de service Web

W3C, le consortium principal des activités du Web, travaille sur le thème de service Web par l'intermédiaire d'un groupe d'activité : *Web service Activity*. Lors des récentes publications [84, 40], ce groupe d'activités a établi une définition du concept de service Web :

> "*A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based massages conveyed by Internet protocols.*"

Selon IBM [47] :

> "*Web Services are self-contained, modular applications, accessible via the Web through open standard languages, which provide a set of functionalities to businesses or individuals.*"

Curbera et al. [28], définit un service Web comme :

> *"A networked application that is able to interact with standard application-to-application Web protocols over well defined interfaces, and which is described using standard functional description languages."*

À la lecture de ces diverses définitions, il apparaît que plusieurs acteurs définissent les services Web par des caractéristiques technologiques distinctes. Celles ci sont toutes utiles pour décrire les services et chacune d'elle exprime une facette de ce qu'il est maintenant convenu d'appeler services Web. Une définition globale pourrait donc être :

> *Une application logicielle, légèrement couplée, à interaction dynamique, identifiée par un URI*[7]*, pouvant interagir avec d'autres composants logiciels et dont les interfaces et les associations ont la capacité d'être publiées, localisées et invoquées via XML et l'utilisation des protocoles Internet communs. Ils sont les bases permettant de construire des systèmes distribués ouverts sur Internet, grâce à leur interface asynchrone utilisant des technologies indépendantes des plates-formes et de leurs composants réutilisables, appelés service.*

Les services Web tentent de répondre à certaines contraintes ou problématiques liées aux besoins de faire transiter des informations par l'intermédiaire d'Internet. Pour commencer, les services Web sont multi langages et multi plates-formes, une interopérabilité due au fait que l'ensemble des protocoles et des langages utilise le formalisme XML.

L'objectif ultime de l'approche *services Web* est de transformer le Web en un dispositif distribué de calcul où les programmes (*services*) peuvent interagir de manière intelligente en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en services plus complexes [22, 61].

À présent, les éléments de base de l'architecture des services Web sont bien définis, même s'ils sont amenés à évoluer [1]. Ces éléments regroupent un langage de définition des interfaces de services, WSDL, une spécification XML qui permet la publication et la localisation des services dans les annuaires, UDDI et un protocole d'échange de messages, SOAP.

Les services Web sont donc plus un phénomène ou un concept voire un contexte, qu'une technologie. Il est évident que les services reposent sur diverses technologies, mais finalement ils sont une volonté commune des organismes, de standards et des utilisateurs de développer des outils permettant une réelle interopérabilité. En général, ils ont les caractéristiques suivantes :

Faiblement couplés : dans le développement de logiciels, le couplage se rapporte typiquement au degré de dépendance entre les composants/modules logiciels. Contrairement aux composants fortement couplés (*tels que DCOM ou CORBA*), les services Web sont auto-

[7] Uniform Resource Identifier

nomes et peuvent fonctionner indépendamment les uns des autres. Il n'est pas nécessaire de connaître la machine, le langage, le système d'exploitation et la panoplie de détails nécessaires à programmer aux deux extrémités du continuum de communication pour qu'une communication puisse avoir lieu. Cela offre une flexibilité qui permettra justement aux entreprises de se sortir du pétrin des protocoles propriétaires et des coûts engendrés par l'intégration que les communications fortement couplées requièrent.

Interaction dynamique : le consommateur de service Web peut localiser et invoquer celui-ci au moment de l'exécution du programme sans avoir à programmer cette habilité à l'avance.

Accessibilité universelle : les services Web peuvent être définis, décrits et découverts à travers le Web qui permet une accessibilité facile. Non seulement les utilisateurs de services Web peuvent localiser les services appropriés, mais les services eux-mêmes peuvent se décrire et s'annoncer de sorte qu'ils soient possibles pour se lier et interagir entre eux.

Langages standards : les services Web sont décrits par des langages XML standards qui ont été considérés comme parties de la technologie Web. Ces standards sont d'une abstraction plus élevée. Bien que les services Web pouvant être implémentés par différents langages de programmation, leurs interfaces sont décrites par des langages XML standards uniformes.

Réutilisable : les service Web, une fois créés, sont réutilisables par le consommateur de service Web. Cette caractéristique signifie que pour une application spécifique, le gestionnaire peut développer un service Web qui effectuera cette tâche pour chacune des interfaces qui sollicitera ou encore mieux, utiliser cette fonction à partir d'un service Web déjà existant et disponible. Les développeurs n'auront plus à reprogrammer une application similaire pour chacun des langages ou des environnements. Ils se serviront seulement de la portion du '*service* dont ils ont besoin.

Les efforts actuels portent en particulier sur la définition de solutions à la composition de services Web pour faciliter le développement d'applications à partir de l'intégration de services existants. l'idée, poursuivie avec les services Web, est de mieux exploiter les technologies de l'Internet en substituant, autant que possible, les humains qui réalisent actuellement un certain nombre de services, par des machines en vue de permettre une découverte et/ou une composition automatique de services sur l'Internet. L'automatisation est donc un concept clé qui doit être présent à chaque étape du processus de conception et de mise en œuvre des services Web.

1.2.3 Architecture

Pour permettre l'échange d'information entre des applications distantes, indépendamment des systèmes d'exploitation et des langages de programmation, les services Web sont composés de couches standard. Nous mettons en relief dans cette section deux types d'architectures. La première dite de référence et traditionnellement utilisée pour les services Web élémentaires. La seconde architecture est plus complète et est fréquemment utilisée lors de la composition de services Web. Elle est appelée architecture étendue.

1.2.3.1 Architecture de référence

Les efforts de recherche et de développement récents autour des services Web ont conduit à un certain nombre de spécifications qui définissent aujourd'hui l'architecture de référence des services Web. Cette architecture (FIG. 1.1) s'articule autour des trois rôles suivants [50] :

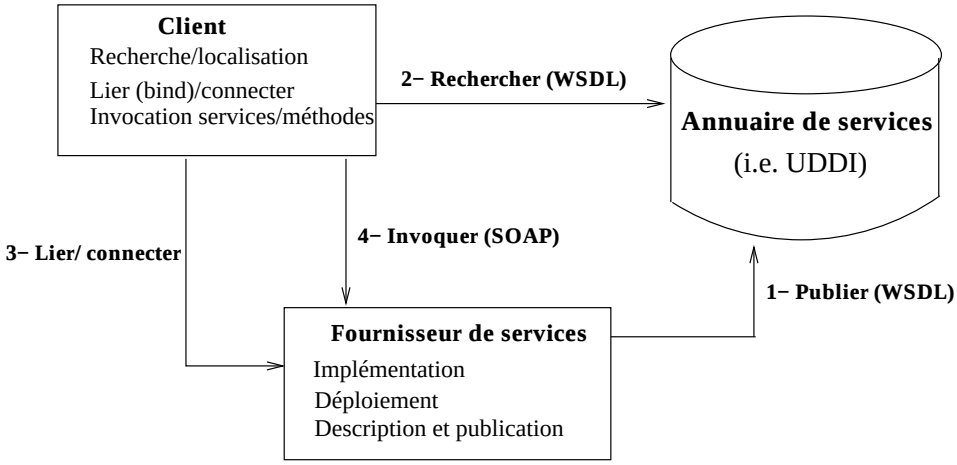

FIG. 1.1 – Architecture de référence des services Web.

- ***Fournisseur de services :*** correspond au propriétaire du service. D'un point de vue technique, il est constitué de la plate-forme d'accueil du service.
- ***Client :*** correspond au demandeur de service. D'un point de vue technique, il est constitué de l'application qui va rechercher et invoquer un service. L'application cliente peut être elle-même un service Web. Le rôle du demandeur de service peut être assuré par un *browser* piloté par une personne ou un programme sans interface utilisateur.
- ***Annuaire de services :*** correspond à un registre de description de services offrant des facilités de publication de services à l'intention des fournisseurs ainsi que des facilités de recherche de services à l'intention des clients.

Les interactions de base entre ces trois rôles incluent les opérations de publication, de recherche et de liens d'opérations [51, 53]. Pour garantir l'interopérabilité des trois opérations, des propositions de standards ont été élaborées pour chaque type d'interactions. Nous citons notamment les standards emergeant suivants :

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

Nous souhaitons conserver ici un haut niveau d'abstraction pour décrire les fonctions essentielles des différents composants, sans entrer dans les détails de leur implémentation abordée dans la section 1.2.5. La description de cette architecture des services Web adopte volontairement un point de vue conceptuel. Cependant, cette architecture n'est pas satisfaisante concernant le protocole de composition de services Web. De plus, il existe de nombreux nouveaux standards émergent (*tels que les standards de sécurité, d'administration et de Web sémantique*) dans le domaine des services Web que cette architecture ne peut pas aisément supporter. Il s'avère donc nécessaire d'étendre l'architecture de base de services Web comme présenté dans la section suivante.

1.2.3.2 Architecture étendue

Cette architecture étendue est aussi appelée *pile des services Web*, du fait qu'elle est constituée de plusieurs couches se superposant les unes aux autres. Elle utilise les couches standards de la première architecture en ajoutant au-dessus d'autres couches plus spécifiques.

La pile des services Web est sujette à de nombreuses discussions. Il est donc assez compréhensible que la pile des services Web ne trouve pas encore de consensus quant à sa définition standardisée. La figure 1.2 est une proposition de vue globale simplifiée de la pile des services Web [51]. Dans un souci d'interopérabilité, les différentes couches s'interfacent avec des standards. Chaque couche de la pile des services Web répond à des préoccupations fonctionnelles différentes telles que la sécurité, la messagerie fiable, les transactions et le routage .

La pile est constituée de plusieurs couches, chaque couche s'appuie sur un standard particulier. On retrouve, au-dessus de la couche de transport (*HTTP, SMTP*), les trois couches formant l'infrastructure de base décrite précédemment.

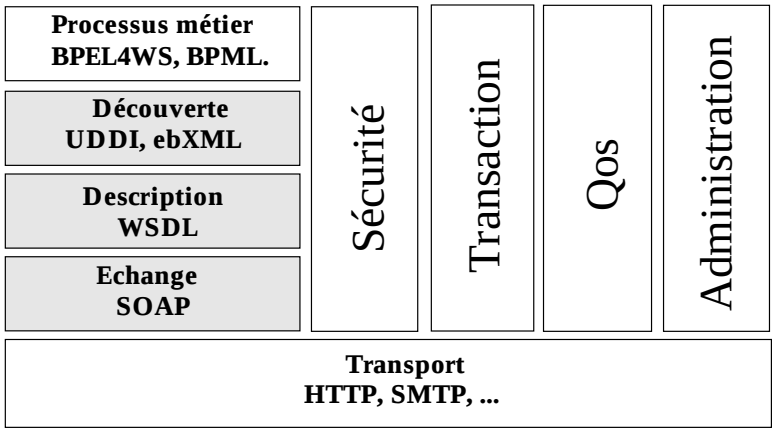

FIG. 1.2 – Architecture en pile (étendue).

Des composants d'infrastructures de type MQSeriees, Corba, RMI et autres, peuvent également être utilisés à ce niveau. Ces couches s'appuient sur les standards émergeants : SOAP comme moyen d'échange des messages, WSDL pour définir l'interface statique d'un service, UDDI un outil de découverte et de localisation.

Dans le contexte de rendre les processus métiers (*business processes*) accessibles à l'intérieur d'une entreprise et au-delà même des frontières d'une entreprise, deux types de couches permettent de compléter l'infrastructure de base :

Les couches dites transversales [27] (sécurité, administration, transactions et QoS[8]) rendent viable l'utilisation effective des services Web dans le monde industriel et dont on retrouve toutes les notions associées aux problématiques de sécurité et celles en rapport avec la QoS. La partie Administration est un peu particulière, car il s'agit de mettre en place des enchaînements de services et par conséquent de créer des services Web composites. Des travaux tentent d'intégrer le Web sémantique dans ces couches transversales en ajoutant une couche verticale représentant le Web sémantique et étant utilisable par les quatre couches horizontales représentant les standards.

Une couche processus métier permet l'utilisation effective des services Web dans le domaine du e-commerce (*e-business*) en permettant leur intégration (*composition*). Elle établit la représentation du processus métier comme un ensemble de services Web. De plus, la description de l'utilisation des différents services Web composant ce processus est disponible par l'intermédiaire de cette couche.

[8]Quality of Service

1.2.4 Scénario général de fonctionnement des services Web

Dans le scénario de fonctionnement normal, un fournisseur de services héberge un module logiciel implémentant un ou plusieurs services Web accessibles via le réseau. Il définit une description de son service et la publie dans un annuaire. Le client utilise les facilités de recherche disponibles au niveau de l'annuaire pour retrouver et sélectionner un service donné. Il examine ensuite la description du service sélectionné pour récupérer les informations nécessaires lui permettant de se connecter au fournisseur et d'interagir avec l'implémentation du service. D'une manière simplifiée, les principales étapes [50] d'exécution des services Web sont :

- ***Découverte de service*** *:* le demandeur du service lance la recherche d'un service correspondant à ses besoins sur un annuaire UDDI qui peut être public ou privé.
- ***Récupération des informations de description de service*** *:* le demandeur du service récupère de l'annuaire UDDI la description de ce service au format WSDL.
- ***Connexion au service Web*** : la communication entre le composant demandeur du service et celui du fournisseur est assurée en phase d'exploitation à travers des *wrappers SOAP* (*listner et proxy*), qui servent d'interfaces entre ces composants et les protocoles de communication de l'infrastructure de déploiement. Le proxy du composant demandeur du service émet une requête SOAP au composant fournisseur du service. Le protocole HTTP véhicule le message SOAP jusqu'au *listner* du fournisseur du service.
- ***Le service Web renvoie sa réponse*** : le service Web du fournisseur renvoie sa réponse au demandeur sous la forme d'un document XML via SOAP.

1.2.5 Présentation des technologies associées aux services Web

Une caractéristique qui a permis un grand succès de la technologie des services Web est qu'elle est construite sur des technologies standards de l'industrie. Dans cette section, nous décrivons ces technologies.

1.2.5.1 XML - *eXtensible Markup Language*

Il a été soumis auprès du W3C [15] en 1998. Historiquement, XML a profité des meilleurs de SGML (*Standard Generalized Markup Language*) et de l'expérience d'utilisation de HTTP pour aboutir à une technologie infiniment plus flexible et plus simple à utiliser.

```
<?xml version="1.0"  encoding="UTF-8" ?>
<xs:schema xmlns : xs = "http://www.w3.org/2001/XMLSchema"
 <xs:element name= "nomelement">
       <xs:annotation>
           <xs:documentation>
               Structure générale d'un document XML
           </xs: documentation>
       </xs:annotation>
       <xs:complexType>
           <xs:sequence>
               <xs:element name="element1"   type="xs:string" />
               <xs:element name="element2">
                ...
           <xs:sequence>
       <xs:complexType>
 </xs:element>
 ...
</xs:schema>
```

TAB. 1.1 – Extrait de code : structure d'un document XML.

L'extrait de code Tab. 1.1 ci-dessus fournit une représentation générale de la structure d'un document XML.

XML est le langage fondateur des services Web permettant d'identifier la structure d'un document. Il offre un nouveau mode de représentation des données utilisables dans des pages HTML. De ce fait, XML est alors flexible et extensible, et est devenu rapidement le standard d'échange de données sur le Web. Il utilise les balises pour indiquer le début et la fin du bloc de données afin de créer une hiérarchie de composants de données associées appelés *élément*.

La section suivante a pour objectif de présenter les principales technologies permettant de structurer les documents XML :

- **La DTD (*Document Type Definition*)** : qui par l'intermédiaire d'une syntaxe particulière, permet de spécifier comment construire le document XML en identifiant la manière dont les balises doivent apparaître dans ce modèle de document. Elle permet de vérifier la syntaxe d'un document.

- **Les Schémas (*XML Schema*)** : la DTD présente de nombreux inconvénients : sa syntaxe est différente de XML et il n'y a pas de notion d'espace de noms. En remplaçant, le consortium W3C a défini la notion de schéma. Les schémas sont une évolution des DTD, qui quant à eux utilisent le formalisme XML. Ils ont été publiés pour fournir une alternative technique au DTD en supportant les espaces de noms de manière à faciliter la conception des vocabulaires XML extensibles et un système de typage plus riche.

Pour pouvoir exploiter les fichiers XML, il existe une galaxie de technologies. Parmi celles-ci nous pouvons citer :

- XPath[9] [87] : pour définir la manière d'adresser des parties d'un document XML.
- XQuery[10] [88] : s'intéresse à la structure logique abstraite d'un document XML.
- XSL[11] [86] : pour définir la présentation de document XML.

1.2.5.2 SOAP - *Simple Object Access Protocol*

SOAP est un protocole au format XML dont les premières versions publiques datent de 1998. Il s'agit d'un standard supporté par le W3C [26]. Le but de SOAP est d'assurer l'interaction entre services Web en transportant les paquets de données encapsulés sous forme de texte structuré au format HTTP. Les implémentations permettant d'utiliser SOAP sont nombreuses, parmi celles-ci on peut cependant noter Axis[12]. Le standard SOAP assure l'interopérabilité entre composants tout en restant indépendant des plates-formes et langages de programmation. Il repose sur deux standards HTTP et XML et respectivement pour la structure des messages et pour le transport. Mais, il n'exclut pas l'utilisation d'autres protocoles de transport (*SMTP, HTTP Extension Framework, etc*). Le protocole SOAP ne passe pas par des ports précis mais utilise le protocole HTTP qui lui permet de traverser les proxies et les pare-feux (*firewalls*), contrairement à d'autres modes de communications telles que les communications via les socket java, RMI, etc.

Ce protocole consiste en trois parties : une enveloppe qui définit un canevas pour décrire le contenu du message et comment le traiter, un jeu de règles de codage à exprimer les cas de types de données définis par l'application et une convention pour représenter des appels de procédure à distance (*RPC's*) et ses réponses.

[9] XML Path
[10] XML Query
[11] eXtensible Stylesheet Language
[12] http ://ws.apache.org/axis/

1.2.5.2.1 Modèle d'échange de messages SOAP

Le protocole de transmission SOAP [85] est fondamentalement un paradigme d'échange à sens unique d'un expéditeur à un récepteur, mais les applications peuvent créer des modèles plus complexes d'interaction par la combinaison de tels échanges à sens unique.

SOAP fournit un modèle de traitement distribué où un document XML (*message SOAP*) est fourni d'un expéditeur à un récepteur final par l'intermédiaire de zéro ou plusieurs nœuds intermédiaires (*SOAP intermediates*). Le chemin suivi par un message SOAP est nommé *message path*. Ce modèle de traitement distribué peut soutenir beaucoup de modèles d'échange de message incluant, mais non limités aux messages à sens unique, des interactions *requête/réponse* et des conversations pair à pair (FIG. 1.3).

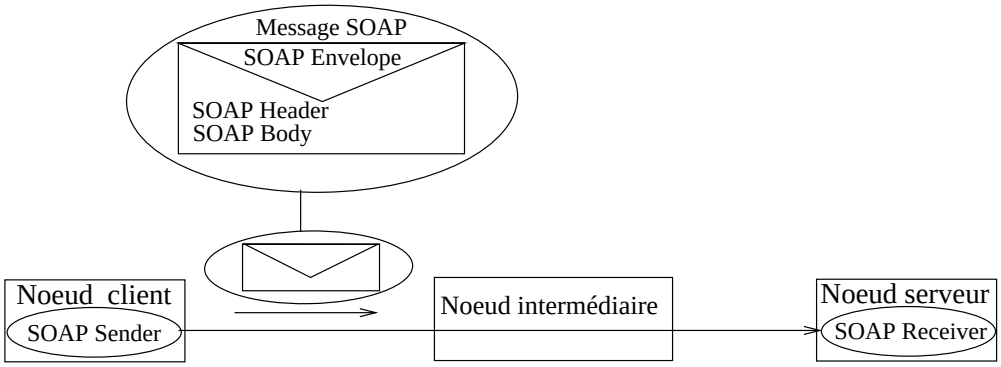

FIG. 1.3 – Structure d'un message SOAP.

Quand un message arrive dans une entité SOAP, elle suit le processus décrit ci-dessous :

1. Identifie toutes les parties du message SOAP destiné à cette entité.
2. Vérifie que toutes les parties obligatoires identifiées dans (1) sont soutenues par l'entité et les traite en conséquence. Si ce n'est pas le cas, elle rejette le message.
3. Si l'entité n'est pas la destination suprême du message, elle enlève alors toutes les parties identifiées dans (1) avant l'expédition du message.

Enveloppe SOAP

Un message SOAP est composé d'un élément obligatoire appelé *SOAP envelope*. L'enveloppe SOAP définit l'espace de nom (*namespace : URI permettant de connaître la provenance de chaque balise*) indiquant la version de SOAP utilisée et un attribut optionnel *encodingStyle* permettant d'identifier les règles d'encodage mises en œuvre pour un message.

L'enveloppe SOAP est constituée d'un en-tête facultatif (*SOAP header*) et d'un corps obligatoire (*SOAP body*). L'extrait de code (Tab. 1.2) suivant présente le squelette d'un message SOAP.

```
<SOAP-Env:Envelope xmlns:SOAP-Env="http://www.w3.org/2001/06/soap-envelope/"
 SOAP-env:encodingStyle= "http://schemas.xmlsoap.org/soap/encodingStyle/ " />
   < SOAP-Env:Header>
      <authentication xmlns=http://myDomain/app" >
                  ...
      </authentication>
      <mes:Message  xmlns: mes="URL"
           SOAP-Env: actor="http://www.info.uqam.ca/application
           SOAP-Env: mustUnderstand="1" >
           Ce message est pour toi et tu dois le prendre en compte!
      </mes:Message>
   </ SOAP-Env:Header>
   < SOAP-Env:Body>
        ...
   </ SOAP-Env:Body>
</ SOAP-Env:Envelope>
```

TAB. 1.2 – Extrait de code : squelette d'un message SOAP.

En-tête SOAP

L'élément *Header* est optionnel mais s'il est présent, il vient immédiatement après l'élément *Envelope*. Il contient les informations à traiter par les intermédiaires du message lors de sa transmission. Par exemple, on peut y ajouter des informations sur la transaction en cours, l'authentification de l'expéditeur, etc. L'en-tête peut utiliser les attributs *mustUnderstand* et/ou *SOAP actor* pour déterminer comment le destinataire d'un message doit le traiter.

- L'attribut ***mustUnderstand*** peut être utilisé pour indiquer si une entrée d'en-tête est obligatoire ou facultative pour être traitée par le destinataire. Il peut prendre la valeur '1' ou '0'. L'absence de cet attribut est équivalente à sa présence avec la valeur '0'.
- L'attribut ***SOAP actor*** peut être utilisé pour indiquer le destinataire d'un élément d'en-tête. La valeur de l'attribut *SOAP actor* est une URI. L'omission de l'attribut actor indique que le destinataire est le destinataire ultime du message SOAP.

Corps SOAP

Contient toutes les données applicatives à transmettre au destinataire final. Il est utilisé pour effectuer des appels de procédures à distance ou transporter le rapport des erreurs (*SOAP fault*). Le corps doit fournir le nom de la méthode invoquée par une requête, ou le nom de la méthode pour générer la réponse. Il doit aussi, fournir l'espace de nom correspondant au nom du service.

Les règles d'encodage

Les règles d'encodage fournissent un mécanisme permettant d'encoder en XML de manière standard les données échangées par les applications. Cet encodage est fondé sur les Schémas XML W3C. Pour faire référence au Schémas XML, la valeur de l'attribut *encodingStyle* doit être 'http ://schemas.xmlsoap.org/soap/encosding/'. Les règles d'encodage SOAP précisent en particulier comment encoder les types de base (*string, integer, etc*) et les types composés.

Modes de transport de messages SOAP

Les messages SOAP peuvent être véhiculés soit en mode RPC[13] soit en mode messagerie. L'approche orientée message exige de définir le format de message, de composer le message en utilisant le format avant de l'envoyer et d'extraire cette donnée à sa réception côté destinataire. Tandis que l'approche RPC réalise toutes ces opérations d'une manière semi-automatique parce que la méthode appelée détermine le format RPC et les standards d'encodage SOAP, pour les objets, déterminent les paramètres qui vont être encodés.

SOAP RPC est la partie la plus connue de la spécification. En effet, la plupart des applications utilisant SOAP le font pour des appels de procédures à distance, car SOAP offre un mécanisme simple et portable à cette intention.

1.2.5.3 WSDL - *Web Services Description Language*

WSDL est un langage au format XML dont la version 1.1 [25] a été soumise en tant que note auprès du W3C en mars 2001. Il permet de décrire, en particulier, les interfaces des services Web. Ces descriptions sont des documents XML. WSDL repose sur SAOP, HTTP et MIME[14] comme mécanisme d'invocation d'objet distant et sur les schémas XML comme

[13]Remote Procedure call
[14]Multipurpose Internet Mail Extensions

système de types. Il permet, par extension, l'utilisation d'autres langages de définition de types.

Le document WSDL définit les services comme des ensembles de points finaux d'accès aux réseaux, aussi appelés *ports*, à travers lesquels on effectue l'échange de messages. Les opérations fournies par le service et les messages échangés sont décrits d'une manière abstraite, un protocole réseau concret et un format de message leurs sont associés pour définir un point final (FiG. 1.4).

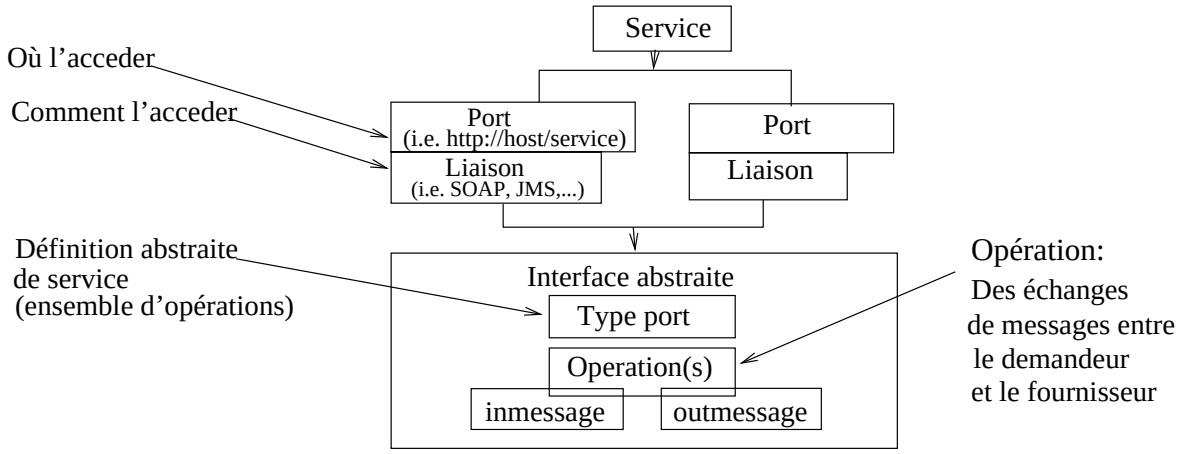

FIG. 1.4 – Structure d'un WSDL.

Les opérations dans le fichier WSDL peuvent être orientées document ou orientées RPC. Lorsque le fichier WSDL identifie une simple opération du protocole SOAP comme orienté document, les messages requête (*input*) et réponse (*output*) spécifiés pour cette opération contiennent des document XML ; tandis que les opérations orientées RPC ont des messages *input* contenant les paramètres d'entrée et des messages *output* contenant les résultats.

WSDL décrit un service Web en deux étapes fondamentales : une abstraite et une concrète. La définition abstraite de points finaux et de messages est séparée de leur déploiement réseau concret ou des liaisons de format de données. Ceci permet la réutilisation de la description abstraite de données en cours d'échange et des types de port qui sont des ensembles abstraits d'opérations. Le protocole concret et les spécifications de format de données pour un type de port particulier constituent une liaison réutilisable. Un port est défini en associant une adresse réseau avec une liaison réutilisable (FIG. 1.5). Enfin, la réunion de points finaux concrets en un point final abstrait forme un service. Outre la description des opérations d'un service, WSDL permet également de fournir l'adresse du fournisseur de service.

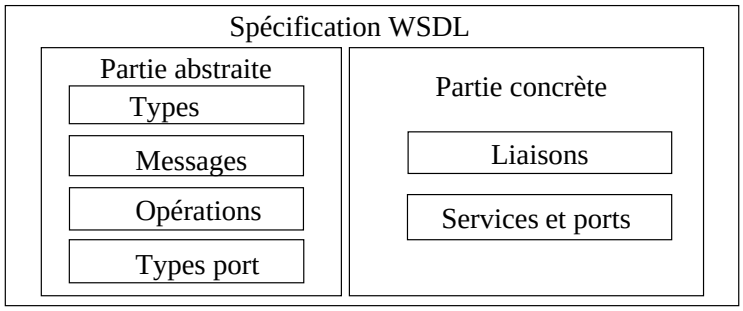

FIG. 1.5 – Spécification d'un service Web avec WSDL.

1.2.5.3.1 Structure d'un document WSDL

Un document WSDL apparaît tout simplement comme une série de définitions. L'élément racine du document s'appelle <*definitions*> qui peut en encapsuler d'autres. La grammaire d'un document WSDL se présente sous forme d'un schéma XML.

Entête du document WSDL

Spécifie le nom du service et des espaces de noms utilisés dans la suite du document, ici *'xsd'* pour l'utilisation de Schéma XML et 'soap' pour une liaison SOAP (Tab. 1.3).

```
<definitions
    name="Servicename"
    targetNamespace="http://www.info.uqam.ca/ ServiceDePrix.wsdl"
    xmlns=http://schemas.xmlsoap.org/wsdl/
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:ns=" http://www.info.uqam.ca/ServiceDePrix.wsdl"
    xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
```

TAB. 1.3 – Extrait de code : entête du document WSDL.

Définition des types de données utilisés

L'élément <*types*> décrit les types de données appliquées aux messages échangés (Tab. 1.4). Ces messages comportent les paramètres bien typés d'invocation de service (*ou de la réponse*).

```
<types>
    <xsd:schema>
        <xsd:element name="nouveau type">
            <xsd:sequence>
                <xsd:element name="attribut1"  type="xsd:string"/>
                <xsd:element name="attribut2"  type="xsd:float"/>
            </xsd:sequence>
        </xsd:element>
    </xsd:schema>
</types>
```

TAB. 1.4 – Extrait de code : structure de l'élément <types> du document WSDL.

Définition des messages échangés

Les messages échangés sont définis dans des éléments <*message*> (Tab. 1.5). Un message peut être simple ou peut contenir des parties spécifiées par le sous-élément <*part*>. Chaque partie est associée à un type à partir d'un système de type utilisant un attribut de typage.

```
<message name="nmtoken">
    <part name="nmtoken"  type="xsd:string"/>
</message>
```

TAB. 1.5 – Extrait de code : définition de l'élément <message> du document WSDL.

Définition des types de port

Les opérations offertes par un service sont définies dans l'élément <*portType*>. Ces opérations prennent comme paramètre entrée/sortie les messages modélisés. Elles peuvent être de nature :

- *Unidirectionnelle (One-way)* : le point final du service reçoit un message <*input*> mais ne renvoie jamais de réponse. Ce type d'opération définit un message d'entrée, mais pas de message de sortie ni d'erreurs.
- *Requête/réponse (Request-response)* : on communique en mode synchrone. Le point final reçoit un message <*input*> et retourne un message corrélé <*output*> ou un ou plusieurs messages de faute <*fault*>.
- *Sollicitation/réponse (Solicit-response)* : identique à l'opération *requête/réponse* à la seule différence que les éléments <*input*> et <*output*> sont inversés.
- *Notification :* le point final envoie un message de notification <*output*>, sans besoin de réponse du client (Tab. 1.6).

```
<portType name="typeport">
<!- One way-->
<operation name="nomop ">
<input message="msgRequete"/>
</operation>
<!- Request-Response-->
<operation name=" nomop ">
<input message="msgRequete "/>
<output message="msgReponse "/>
<fault message="MsgErreur"/>
</operation>
```

```
<!- Sollicit-response-->
<operation name="nomop">
<outputmessage="msgSollicite"/>
<input message="msgRequete "/>
</operation>
<!- Notification-->
<operation name="nomop">
<output message="msgReponse"/>
</operation>

</portType>
```

TAB. 1.6 – Extrait de code : définition de l'élément <portType> du document WSDL.

Définition des liaisons

Une fois les opérations définies, on les associe à un protocole et à un encodage. Cette association s'appelle une *liaison* et est par conséquent définie dans un élément <*binding*>. Dans une liaison, l'attribut *name* définit un nom unique pour chaque liaison employée dans le document WSDL. L'attribut *type* permet à la liaison de faire référence au *portType* qu'elle relie. La spécification WSDL définit les liaisons suivantes : *SOAP ; HTTP GET et POST ; MIME*. Des liaisons complémentaires peuvent être créées, si c'est nécessaire (Tab. 1.7).

```
<binding name="maliaison"  type=" TypePort">
   <soap:binding style="rpc/document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="nomop">
            <soap:operation  soapAction="uri" />
                <input>
                    <soap:body use="encoded"  encodingStyle="http://..."/>
                </input>
                <output>
                    <soap:body use="encoded"  encodingStyle="http://.../"/>
                </output>
        </operation>
    </soap:binding>
</binding>
```

TAB. 1.7 – Extrait de code : définition de l'élément <binding> du document WSDL.

Association à une URL

La dernière étape est d'associer cette liaison à une implémentation particulière, ce que l'on fait dans la balise <*service*>. Un service peut avoir plusieurs points d'accès. Chacun de ces différents points d'accès est défini dans un élément <*port*> (Tab. 1.8).

```
<service name="nomservice">
    <port name="nomPort"  binding="maliaison">
              <soap:address  location="http://www.devoteam.com/nomservice"/>
    </port>
</service>
```

TAB. 1.8 – Extrait de code : définition de l'élément <service> du document WSDL.

1.2.5.4 UDDI - *Universal Description, Discovery and Integration*

Contrairement à SOAP et WSDL, UDDI n'est pas supporté par le W3C mais par le consortium OASIS [82]. C'est un standard qui est né d'une initiative commune à un certain nombre de grands acteurs du monde de l'informatique, dont notamment *Microsoft, IBM et Intel*. L'objectif primaire d'UDDI est la spécification d'un canevas pour décrire et découvrir des services Web. En fait, le but initial du consortium a été de soutenir des répertoires mondiaux où chacun pourrait publier des descriptions de services et où chacun pourrait interroger ces répertoires pour trouver les services d'intérêt. L'idée était un UBR[15] pourrait contenir chaque service déployé dans le monde.

Le cœur du projet UDDI est son annuaire contenant des données techniques et administratives sur les fournisseurs de services, les services hébergés et les protocoles mis en œuvre par ces services. Le noyau d'UDDI travaille avec la notion de registre d'affaire '*business registry*', qui peut être invoqué comme un service Web. Ce registre peut être employé au niveau d'affaires pour vérifier si un partenaire donné a des interfaces concernant un service Web particulier, pour trouver une telle compagnie dans une industrie donnée avec un type donné de service, et pour localiser des informations sur la façon dont un partenaire a exposé un service Web afin d'étudier les détails techniques priés pour interroger ce service. Grâce à un jeu d'API[16] XML basé sur SOAP, on peut interagir avec le registre au moment de la conception et de l'exécution des services Web pour obtenir des références aux services d'intérêt, de manière à ce que ces services puissent être invoqués et utilisés.

[15] Universal Business Registry

[16] Application Programming Interfaces

Dans ce cadre, un serveur UDDI est utilisé. On rencontre un grand nombre d'implémentations de serveurs UDDI, que ce soient des produits fournis par des sociétés telles que Microsoft ou d'autres qui sont des implementations gratuites tel que jUDDI[17]. Il est important de savoir que ces serveurs ne contiennent pas l'adresse des services Web mais l'adresse des fichiers WSDL les décrivant. L'annuaire UDDI est consultable sous plusieurs facettes :

- ***Pages blanches*** **:** elles comportent les informations sur les entreprises telles que : les noms des entreprises, les détails de leurs contacts et les services fournis. Elles comprennent également une liste des identifiants grâce auxquels une entreprise peut être repérée.
- ***Pages jaunes*** **:** elles sont composées d'informations s'appuyant sur des standards de classification industrielle normalisée permettant de répertorier des entreprises et des services en fonction de taxonomies. Les pages jaunes comprennent la description, au format WSDL, des services Web déployés par les entreprises. Une entreprise peut disposer de plusieurs entrées dans l'annuaire pour l'ensemble des divers services qu'elle propose.
- ***Pages vertes*** **:** contiennent des informations de liaison sur les services. Elles fournissent des descriptions techniques se basant sur les URI d'accès, les protocoles utilisés, etc.

1.2.5.4.1 Modèles d'information UDDI

UDDI [27] consiste en un schéma XML pour des messages SAOP, et une description des spécifications d'API UDDI. Ensemble, ceux-ci forment un modèle d'information de base et un cadre d'interaction fournissant la capacité de publier l'information concernant une liste de services Web. Le standard schéma XML a été choisi en raison de sa capacité de représenter des types de données riches aussi bien que sa capacité de décrire et valider facilement l'information basant sur les modèles d'information représentés dans les schémas.

Le schéma XML d'UDDI définit quatre types noyaux d'information permettant d'utiliser les services Web. Ceux-ci sont : informations métiers, informations services, informations de liaison et informations sur les spécifications de services (FIG. 1.6).

[17]Le site de jUDDI, http ://ws.apache.org/juddi/

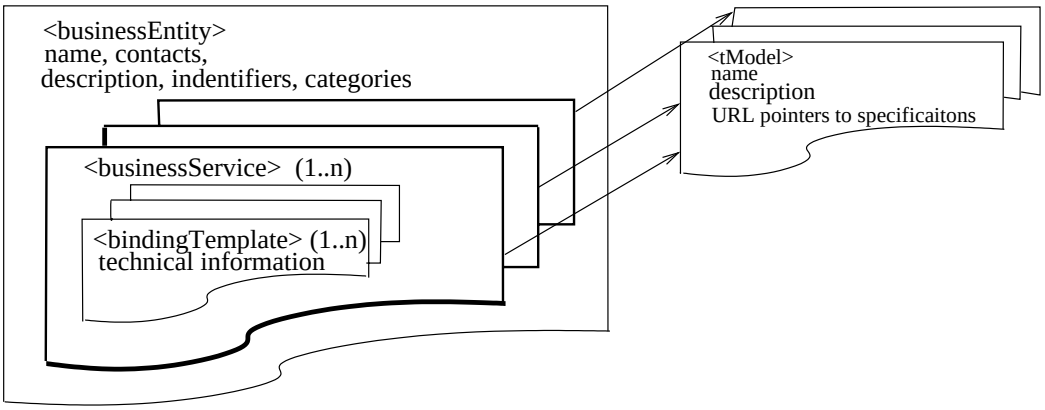

FIG. 1.6 – Structures de données de base UDDI.

Informations sur le métier de l'entreprise - businessEntity

Les services Web publiés et les types d'affaires traités par une entreprise sont contenus dans une structure appelée "*businessEntity*" et correspondent notamment aux pages blanches. L'élément *businessEntity* peut contenir un ou plusieurs éléments *businessService* pour chacune des familles de services offerts par l'entreprise. Il s'agit des informations décrivant le fournisseur du service tels que des noms, des adresses, des contacts et d'autres détails administratifs.

Informations sur les services métiers - businessService

Les informations contenues dans l'élément "*businessService*" correspondent aux pages jaunes. Il s'agit des informations de description des services Web référencés tels que le nom du service, sa description, son code. Une entreprise peut enregistrer plusieurs services. Cette structure est un récipient descriptif utilisé pour grouper une série de services Web relatifs liés à un processus métier. Un service métier comprend des informations descriptives sur une famille de services.

Informations de liaison - bindingTemplate

Les détails techniques sur la façon dont un service est accédé sont fournis par un ou plusieurs structures de liaison "*bindingTemplate*". Elles servent également de pages vertes de l'annuaire UDDI. La liaison associe l'entrée de service Web à l'URI exacte identifiant son emplacement. Il s'agit des informations indiquant une adresse d'un point d'accès, ainsi que des références aux protocoles exacts mis en œuvre. Un service peut avoir plus d'un modèle de liaison.

Informations techniques - tModel

Les "*tModels*" fournissent la capacité de décrire la conformité à des spécifications techniques.

Une définition détaillée du protocole utilisé pour accéder au service, ainsi que les informations permettant de connaître les normes auxquelles le service est conforme sont fournies afin d'avoir une description suffisamment précise du service. Dans de nombreux cas, le *tModel* référence un fichier WSDL qui décrit l'interface SOAP du service Web.

L'extrait de code (Tab. 1.9) suivant représente les structures de données UDDI décrites sous forme de schéma XML.

```
<businessEntity authorizedName="..."businessKey="..."operator="...">
    <name> ... </name>
    <description xml:lang="fr">
         Nous fournissons des solutions et services
    </description>
    <address>     </address>
    </contact>      </contacts>
    <discoveryURLs>   </discoveryURLs>
    <businessService    businessKey="..."  serviceKey="...">
        <name>  Page acuueil  </name>
        <description xml:lang="fr">   Bienvenue    </description>
        <bindingTemplates>
            <bindingTemplate    bindingKey="... "  serviceKey=" ...">
                <accessPoint URLType="http">...</accessPoint>
                </tModelInstanceDetails>
                    <tModelInstanceInfo tModelKey="..."/>
                </tModelInstanceDetails>
            </bindingTemplate>
        </bindingTemplates>
    </businessServices>
</businessEntity>
```

Tab. 1.9 – Extrait de code : structures de données UDDI.

1.2.5.5 Relation entre éléments WSDL et le contenu de l'annuaire UDDI

UDDI et WSDL sont complémentaires dans l'implémentation des services Web. UDDI fournit une méthode de publication et de découverte de description de service. Les entités

de données UDDI fournissent un support pour définir aussi bien les informations métiers et celles de service. WSDL fait une distinction entre les messages et les ports. Il n'est pas obligatoire de fournir des informations sur le port dans un fichier WSDL. Ils sont séparés des implémentations, un fichier WSDL contient exclusivement des informations abstraites sur l'interface et ne fournit aucune donnée d'implémentation concrète. Ceci permet à plusieurs systèmes d'écrire des implémentations d'une même interface. Une interface de service est décrite par un document WSDL. Elle contient la définition de service qui sera utilisée pour implémenter un ou plusieurs services. Le document implémentation de service contient une description de service qui implémente une interface. Au moins un des éléments import va contenir une référence au document interface de service. Une description de service WSDL complète est une combinaison d'une interface de service et d'un document d'implémentation. Lors de la publication d'une description WSDL, une interface de service est publiée en tant que *tModel* avant qu'une implémentation de service ne soit publiée en tant que *businessService* avec un ou plusieurs *bindingTemplates*.

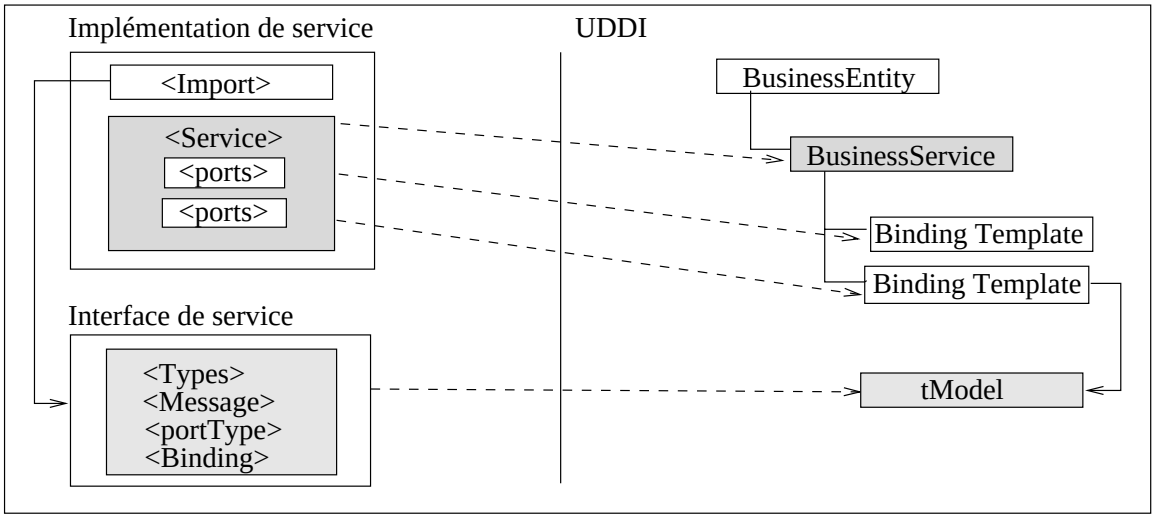

FIG. 1.7 – Correspondance entre éléments WSDL et les structures de données UDDI.

1.2.6 Publication et découverte des descriptions de services

1.2.6.1 Publication de services

La publication de services Web inclut la production et la publication des descriptions de services [54].

1.2.6.1.1 Production des descriptions de services

La description de service peut être générée, codée à la main ou composée à partir des définitions d'interfaces de services existantes. Les réalisateurs peuvent coder à la main une description entière de service, incluant l'entrée d'UDDI. Il existe des outils pour générer des parties du WSDL et potentiellement des parties de l'entrée d'UDDI à partir des modèles de programmation et de déploiement des exécutables de services Web.

1.2.6.1.2 Publication des descriptions de services

Une description de service peut être publiée dans de multiples registres de services en utilisant des mécanismes variés fournissant ainsi des possibilités variables d'utilisation des services en mode dynamique.

Le cas le plus simple est la publication directe où un fournisseur de service envoie la description de service directement au demandeur de service. Ceci peut être accompli en utilisant un attachement *E-mail* ou même une distribution d'un *CD-ROM*. Une publication directe peut se produire après que deux partenaires ont convenu aux conditions de faire des *e-Business* à travers le Web, ou après un payement des frais d'accès au service par le demandeur. Dans ce cas, le demandeur de service peut maintenir une copie locale de la description de service. En revanche, la publication automatique utilisant un annuaire UDDI définit un simple mécanisme HTTP GET pour rechercher des descriptions de services Web en utilisant une URL donnée. L'emplacement de l'annuaire UDDI est généralement défini par l'administrateur du système. Les fournisseurs de services Web présentent diverses options pour déployer les registres UDDI. Trois grandes catégories de déploiement sont mises en œuvre : déploiement public, inter-entreprise et intra-entreprise.

Pour les déploiements publics, un groupe d'entreprises regroupant notamment *Microsoft, IBM et SAP* sont engagées à mettre en place un nœud public "*UBR*" pour les services Web, conforme aux spécifications énoncées par le consortium UDDI.org, dans le but d'être découvert par de nouveaux partenaires potentiels ou utilisateurs de services. UBR est un registre UDDI public répliqué dans de nombreuses entreprises. Il sert à la fois de ressources pour les services Web présents sur Internet et de test pour les développeurs.

Dans les deux autres catégories, un registre privé est déployé par une entreprise ; un contrôle plus étroit sur les types d'informations enregistrées est alors possible. Ces registres privés peuvent être dédiés à une seule entreprise ou à des groupes de partenaires commerciaux. UDDI définit aussi des protocoles pour la réplication entre les registres et pour la fédération des approbations concernant plusieurs déploiements. Il existe plusieurs types de nœuds privés d'UDDI pouvant être utilisés selon la portée du domaine des services Web devant être publiés.

- ***Nœud UDDI pour une application entreprise interne :*** les services Web utilisés au sein d'une compagnie pour l'intégration des applications interne peuvent être publiés sur un nœud UDDI de ce genre. La portée de ce nœud UDDI peut être une application simple, départementale ou de corporation. Ces nœuds UDDI se trouvent derrière le pare-feu et permettent lors de la publication de service plus de contrôle sur leur enregistrement, leur condition d'accessibilité, de disponibilité et de publication.

- ***Nœud portail UDDI :*** les compagnies publiant des services Web pour des partenaires externes, afin qu'ils puissent les disposer, peuvent utiliser un nœud portail UDDI. Un nœud portail UDDI fonctionne à l'extérieur du pare-feu du fournisseur de service ou entre les pare-feux. Ce genre de nœud privé UDDI contient seulement des descriptions de services qu'une compagnie souhaite fournir aux demandeurs de service à partir des partenaires externes. Des mécanismes d'accès sélectifs selon le profil des utilisateurs sont mis en œuvre. Ceci permet à des compagnies de garder le contrôle de leurs descriptions de services, accéder au nœud UDDI et à la qualité de service pour les nœuds UDDI.

- ***Nœud catalogue du partenaire avec UDDI :*** les services Web utilisés par une compagnie particulière peuvent être publiés sur un nœud catalogue du partenaire UDDI derrière les pare-feux. Ce genre de nœud privé UDDI contient seulement des descriptions de services Web approuvés, examinés et validés à partir des partenaires légitimes. Le contexte métier pour ces services Web peut être visé à des demandeurs spécifiques.

- ***Nœud E-Marketplace UDDI :*** les services Web, pour lesquels le fournisseur de service prévoit concurrencer pour métiers avec d'autres services Web, doivent être publiés sur un nœud e-marché UDDI ou un nœud opérateur UDDI. Ces nœuds sont accueillis par un organisme de normalisation d'industrie ou consortium et contiennent des descriptions de services des entreprises dans une industrie particulière. Ces services peuvent être exigés pour supporter des normes spécifiques, interfaces ou types de données. Les nœuds E-Marketplace UDDI fournissent généralement quelques filtrages pour des entrées illégitimes et des qualités garanties de service.

1.2.6.2 Découverte de services

La découverte de services Web inclut l'acquisition des descriptions de services et leur utilisation.

1.2.6.2.1 Acquisition des descriptions de services

Comme la publication des descriptions de services Web, l'acquisition de ces descriptions dépend du niveau de description de service dans l'annuaire. Les demandeurs de services peuvent rechercher des services Web pendant deux phases différentes de cycle de vie d'une application, pendant la phase de conception et au cours de l'exécution. Au temps de conception, les demandeurs de services cherchent des descriptions de services Web par le type d'interface qu'ils supportent. Au temps d'exécution, les demandeurs de services cherchent un service Web en se basant sur la façon dont il communique ou sur la qualité de service annoncée.

Avec l'approche de publication directe, le demandeur de service met en cache la description du service pendant la conception pour son usage au temps d'exécution. La description du service peut être représentée statiquement par un programme, stockée dans un fichier ou un simple annuaire local de description de service.

Les mécanismes vues ont besoin de supporter un mécanisme de requête fournissant une recherche par type d'interface, information de protocoles de liaisons, propriétés (*telles que des paramètres de QoS*), taxonomie du service, information de métier, etc.

1.2.6.2.2 Utilisation des descriptions de services

Après l'acquisition de la description de service, le demandeur de service peut l'invoquer. Il utilise la description de service pour générer des requêtes SOAP ou des proxy dans un langage de programmation spécifique. Cette génération peut être faite pendant la conception ou au cours de l'exécution. Divers outils peuvent être utilisés pour générer des liens à partir des documents WSDL. Ces liens présentent une API au programme applicatif et encapsulent les détails du message XML.

1.2.6.3 Les APIs UDDI

Le modèle de programmation UDDI définit deux famille d'APIs permettant l'accès et la manipulation de l'annuaire UDDI : L'API d'interrogation et celle de publication. Ces APIs acceptent les messages XML encapsulés dans des enveloppes SOAP. À l'aide de ces APIs, les programmeurs peuvent développer des applications en *java, c++* ou tout autre langage afin d'accéder à l'annuaire et publier leurs services, rechercher les services publiés par d'autres ou supprimer les leurs.

1.2.6.3.1 API de publication

L'API de publication a pour objectif premier de sauvegarder et de supprimer les structures de données par l'UDDI. Les diverses primitives de cette API servent aux fournisseurs de services pour publier et supprimer la publication ; dans l'annuaire UDDI ; des informations les concernant. Les programmeurs peuvent employer l'API de publication pour créer des interfaces riches pour des outils permettant une interaction directe avec un annuaire UDDI. Le principe clé de fonctionnement de l'API de publication doit permettre seulement aux individus autorisés à publier ou changer l'information dans un annuaire UDDI. Chacune des implementations individuelles d'un annuaire UDDI distribué maintient une liste unique des parties autorisées pour des données de *businessEntity* ou de *tModel* qui ont été créées par un individu particulier. Les changements et les suppressions sont autorisés seulement si une requête de changement (*par l'intermédiaire d'un appel d'API*) est faite par le même individu qui a créé l'information effectuée.

L'API de publication se compose de quatre fonctions *'save_xx'* et de quatre fonctions *'delete_xx'*, une pour chacune des quatre structures de données principales d'UDDI (*businessEntity, businessService, bindingTemplate, tModel*). Une fois autorisée, une partie individuelle peut enregistrer tout nombre d'ensembles d'information de *businessEntity* ou de *tModel* et peut changer l'information précédemment publiée.

1.2.6.3.2 API d'interrogation

L'API [81] d'interrogation est encore divisible en deux parties - une partie utilisée pour

la construction des programmes permettant de rechercher et passer en revue une information se trouvant dans un annuaire UDDI, et une partie différente qui serait utile au cas où des invocations de services Web éprouvent des échecs. Une API d'interrogation se compose de deux types d'appels permettant à un programme de localiser rapidement des entreprises candidates, des services Web et des spécifications. L'API *'find_xx'* fournit au visiteur une large vue sur les données enregistrées basant sur une variété de critères de recherche. Alternativement, les copies à jour d'une structure particulière (par exemple *businessEntity, businessService, bindingTemplate, le tModel*) peuvent être recherchées par l'intermédiaire d'un appel direct. Ces appels directs s'appellent API *'get_xx'*.

1.3 Technologies du Web sémantique pour les services Web

Les services Web sémantiques [33, 61] se situent à la convergence de deux domaines de recherche importants qui concernent les technologies de l'Internet : le *Web sémantique* et les *services Web*. Le Web sémantique s'intéresse principalement aux informations statiques disponibles sur le Web et propose de munir l'information sur le Web d'une sémantique afin de la rendre manipulable par des composants logiciels actifs (*services, agent, etc*). Les services Web, quant à eux, ont pour préoccupation première l'interopérabilité entre applications via le Web en vue de rendre le Web plus dynamique. Il semble donc nécessaire de tendre vers des services intelligibles pour des machines : c'est le concept de service Web sémantique. La figure 1.8 définit quel est le positionnement précis des services Web sémantiques par rapport aux domaines cités précédemment.

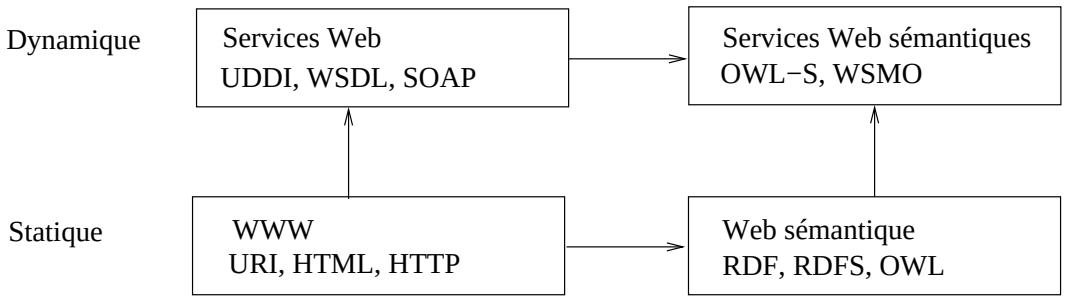

FIG. 1.8 – Positionnement des services Web sémantiques.

Une définition, des services Web sémantique, qui semble être la plus consensuelle est celle de ***Li et Horrocks*** [57] :

> "*Semantic Web Services aim to describe and implement Web services so has to make them more accessible to automated agents. Here, ontologies can be used to describe services so that agents (both human and automated) can advertise and discover services according to a semantic specification of functionality (as well as other parameters such as cost, security, etc.).*"

On peut donc en ressortir qu'on utilise les ontologies pour décrire les services, et grâce à cela, la découverte, la composition et l'interrogation de services pourront être automatisées. Il est peut-être utile de définir d'abord ce que sont les significations exactes, dans le domaine des services Web, des termes déjà introduit : *"ontologie", "sémantique", "interopérabilité".*

> ***Ontologie :*** on trouve de nombreuses définitions différentes, nous allons utiliser celle définie par Tom Gruber [39] : "*An ontology is a description (like a formal description of a program) of the concepts and the relationships that can exist for an agent or a community of agents.*"

> ***Sémantique :*** *en informatique, tout langage est représenté par une syntaxe précise que l'on appelle également sémantique. Pour le Web sémantique, on parlera plutôt d'un formalisme logique permettant de décrire des ontologies [46].*

> ***Interopérabilité :*** *Notion cruciale dans le monde d'Internet, qui a pour objectif de permettre à plusieurs systèmes logiciels, de même nature ou hétérogènes, de communiquer et de coopérer afin de minimiser les coûts de leur intégration. Elle repose sur des normes techniques qui définissent des exigences accompagnées de recommandations permettant à deux systèmes qui satisfont aux exigences de dialoguer et d'évoluer tant qu'ils respectent la norme définissant leurs interfaces.*

En d'autres termes, une ontologie est une définition des concepts de base, de leurs propriétés et de leurs relations utilisés pour décrire un domaine de connaissance et des règles exprimant les liens sémantiques sur ces concepts. Une ontologie est un modèle conceptuel d'un domaine spécifique, à titre d'exemples : *Finance, Tourisme, Transport, Médecine, etc.* Par l'utilisation des ontologies, le Web sémantique cherche à atteindre une interopérabilité d'un point de vue sémantique.

Le besoin d'automatisation du processus de conception et de mise en oeuvre des services Web rejoint les préoccupations à l'origine du Web sémantique, à savoir comment décrire formellement les connaissances de manière à les rendre exploitables par des machines. En conséquence, les technologies et les outils développés dans le contexte du Web sémantique peuvent certainement compléter la technologie des services Web en vue d'apporter des réponses crédibles au problème de l'automatisation.

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

- Processus de description et de publication de services.
- Découverte de services.
- Sélection de services.
- Composition de services.
- Administration de services.

Pour permettre une utilisation effective des services Web dans les domaines dont les exigences vont au-delà de la capacité d'interactions simples via des protocoles standards, par exemple, dans le domaine du *e-business*, cette utilisation est motivée par les possibilités de coopération et de coordination entre des entreprises. Le challenge est alors d'être capable de spécifier et de mettre en oeuvre des processus métiers intra ou inter-entreprises. Ceci pose donc fondamentalement un problème d'intégration fonctionnelle des activités d'entreprises. Nous introduisons dans la section suivante la problématique de l'intégration.

1.4 Problématique de l'intégration

La nécessité de rendre les entreprises performantes et réactives à leur environnement, les conduit de plus en plus à migrer vers une organisation orientée processus. Cela revient en conséquence à définir des objectifs communs aux activités d'une entreprise : ce but peut être efficacement atteint via la spécification des processus métiers (*opérationnels*). Il s'agit donc de coordonner les flux de processus entre leurs différentes fonctions.

Selon le consortium WfMC[18], un processus métier est un ensemble d'une ou plusieurs activités liées qui réalisent collectivement un objectif d'entreprise, normalement dans le contexte d'une structure organisationnelle définissant les rôles fonctionnels et les associations. Plus concrètement, au niveau organisationnel, un processus métier est une structure logique indépendante des frontières établies par la hiérarchie de l'entreprise, qui exprime un regroupement d'activités ou de sous-processus réalisés par les différents départements (*ou services*) de l'entreprise. En d'autres termes, plutôt que de décrire les fonctions des services de l'entreprise en termes d'activités non nécessairement liées, on identifie un ensemble d'activités, issues de fonctions diverses, mais résumées autour de la réalisation d'un objectif commun.

Cette problématique, de coordonner des activités issues de différentes fonctions autour d'un objectif commun, interpelle directement la notion de travail de groupe supporté par un système de gestion de workflow (*WfMS*). Un *WfMS* permet la définition, la création et la gestion de l'exécution des workflows qui automatisent (*tous ou en partie*) les processus métiers, ces derniers étant, dans les cas les plus complexes, des processus métiers inter-entreprises. Le consortium WfMC propose la définition suivante d'un workflow :

> "*un système workflow définit, gère et réalise des procédures en exécutant des programmes dont l'ordre d'exécution est prédéfini dans une représentation informatique de la logique de ces procédures - les workflows.*"

Un système de gestion de workflow est aux processus opérationnels ce qu'un système de gestion de base de données est aux données [54]. La technologie workflow s'appuie beaucoup, en effet, sur la technique de modélisation puisqu'un workflow se doit implanter un modèle en répondant, en particulier, aux questions suivantes [51] :

- Quelles sont les activités à réaliser ?
- Quelles sont les compétences nécessaires pour réaliser ces activités ?
- Quand faut-il réaliser ces activités ?
- Quels sont les outils et les informations nécessaires à la réalisation de ces activités ?

Des divergences de vues sur le rôle et le contenu des couches hautes de la pile (i.e., *les relations entre services Web, processus métiers et workflows*) apparaissent clairement dans la littérature. Ce point est important car il interpelle directement les problèmes d'intégration de processus d'entreprises, i.e. une intégration effectuée à un haut niveau d'abstraction en s'appuyant sur la *sémantique* des services. Ce type d'intégration constitue un des apports

[18]Workflow Management Coalition, http ://www. wfmc.org

les plus prometteurs de l'approche *services Web*. Contrairement aux approches classiques d'intégration (*i.e., les EAI qui sont des applications propriétaires*), les services Web proposent une approche flexible pour l'intégration de systèmes hétérogènes en s'appuyant sur un modèle d'intégration basé sur un couplage faible des composants (*peer-to-peer*) et en exploitant de manière intensive les standards du Web.

L'intégration des services Web peut être établie comme une composition de services entre eux. D'après ***Fensel*** [33], les services peuvent interagir de manière intelligente en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes. Nous étudions le problème de composition lors des sections suivantes.

1.5 Thèmes de recherche actuels

Les sujets actuels de recherche dans le domaine des services Web sont nombreux. Un nombre considérable d'études tournent autour de la découverte de services et ses sujets rattachés comme la sélection,la substitution et la composition ainsi que la sécurité.

Dans cette section nous allons décrire les travaux les plus pertinents.

1.5.1 Sélection de services Web

Avec la sélection de services Web, on cherche à choisir le meilleur fournisseur d'un service Web, étant donné un ensemble de fournisseurs de ce service. Il y a une proposition de base sur la qualité de service (QoS[19]) dans le modèle de ***Ran*** [69]. La plate-forme de cette proposition est un modèle qui peut coexister avec les registres UDDI. Les registres actuels peuvent fournir des services aux clients pour qui la qualité de service n'est pas importante. Les registres basés sur ce modèle peuvent servir aux applications qui ont besoin de qualité de service. Il y a quatre rôles dans ce modèle (FIG. 1.9).

- Fournisseur de services Web.
- Consommateur de services Web.
- Certificateur de la qualité de service.
- Et le nouvel registre.

[19] Qualité of Service

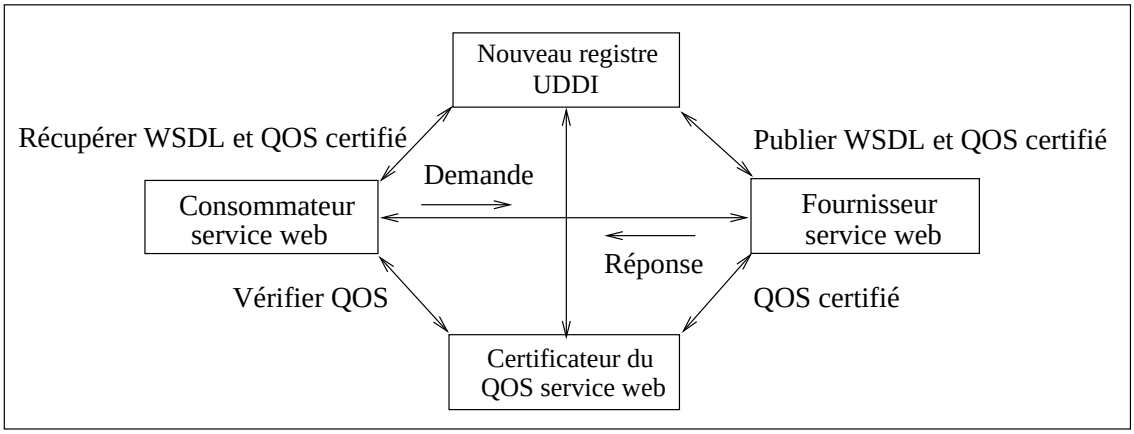

FIG. 1.9 – Nouveau modèle de registre et découverte de services .

Le nouveau registre est différent du modèle actuel UDDI en ayant information sur la description fonctionnelle du service Web et sur la qualité de service enregistré. Le fournisseur du service Web doit d'abord communiquer sa QoS, revendique au certificateur de QoS. Le certificateur vérifie les revendications et certifie ou pas la revendication. Le résultat est envoyé au fournisseur avec des informations d'identification de certification. Ces informations sont aussi enregistrées dans le dépôt du certificateur identifiées par 'certification Id'. Une fois la certification QoS est envoyée par le certificateur, le fournisseur s'inscrit dans le nouveau registre UDDI tant avec la description fonctionnelle du service qu'avec sa qualité certifiée associée aux informations de service. Le registre UDDI communique avec le certificateur pour vérifier l'existence de la certification. Après la vérification couronnée de succès, le registre enregistre alors le service dans son dépôt.

Un consommateur d'un service Web désire certain besoin fonctionnel et de qualité de service, comme "le temps de réponse non plus grand que 2 millisecondes avec le coût moins de 10 euros par invocation". La consultation pourrait être faite selon la description fonctionnelle du service Web désirable, avec la qualité exigée de service attribuée comme des contraintes de consultation. Il cherche dans le registre UDDI un service Web avec la fonctionnalité exigée comme d'habitude. S'il y avait des services Web multiples dans le registre UDDI avec des fonctionnalités semblables, alors la qualité de service mettrait en application une recherche plus détaillée. La recherche rendrait un service Web offrant la fonctionnalité exigée avec le jeu désirable de qualité de service. S'il n'y a aucun service Web avec ces qualités, on donne des réactions au consommateur. Le consommateur peut alors réduire ses contraintes de qualité de service. Une fois qu'un service Web est trouvé, le fichier WSDL et les informations QoS certifiées sont obtenus par le consommateur, il peut vérifier les revendications de QoS avec le certificateur utilisant l'identificateur de certification. Une fois qu'il est satisfait, il peut invoquer le service Web selon le modèle actuel.

1.5.2 Découverte de services Web

La découverte de services Web permet la connexion automatique à des services sans intervention humaine. Dans le contexte d'une application qui a besoin d'exécuter une fonctionnalité implémentée comme un service Web par plusieurs fournisseurs, la découverte fait référence au processus de recherche des services Web implémentant la fonctionnalité souhaitée. Les registres UDDI sont des entités qui servent d'appui à la découverte de services Web pour les applications client. UDDI définit un protocole d'interrogation et de mise à jour d'un annuaire d'informations sur les services Web. Cette approche par annuaire est intéressante lorsque les informations sont stockées dans des emplacements bien connus. Une fois que l'annuaire est localisé, il reste à l'interroger pour obtenir les informations souhaitées. Dans ce sujet, des composants fonctionnels de l'architecture servent à localiser des services Web sur un réseau et à déterminer sa disponibilité. Il existe trois travaux pertinents : *OWL-S, WSDA et WS-Discovery.* Ils seront décrits dans les sections suivantes.

1.5.2.1 OWL-S *Ontology Web Language-Service*

OWL-S [58] est un langage de description de services issu des travaux du programme du DAML. Il utilise la logique de description et les ontologies définies par OWL [59], langage de définition d'ontologies, pour fournir des descriptions de services dans lequel la sémantique est incluse. Les anciennes versions d'OWL-S s'appelaient DAML-S, qui lui-même est basé sur DAML[20]+OIL[21] [44].

Les intérêts liés à l'utilisation de OWL-S sont que ce langage inclut la sémantique et contient des fonctions indispensables à la mise en œuvre de services Web : *la description, la recherche et l'invocation de service.* Il définit un ensemble de classes et de propriétés afin de constituer l'ontologie. OWL-S est composé de trois parties principales décrites ci-dessous : *Profil de service, Modèle de service, Accès au service.* Les deux premiers profils sont des représentations abstraites tandis que l'accès au service se situe à un niveau concret de spécification. La figure 1.10 présente les différentes ontologies qui caractérisent OWL-S.

[20]DARPA Agent Markup Language, http ://www.daml.org/
[21]Ontology Inference Layer, http ://www.w3.org/TR/owl-ref/

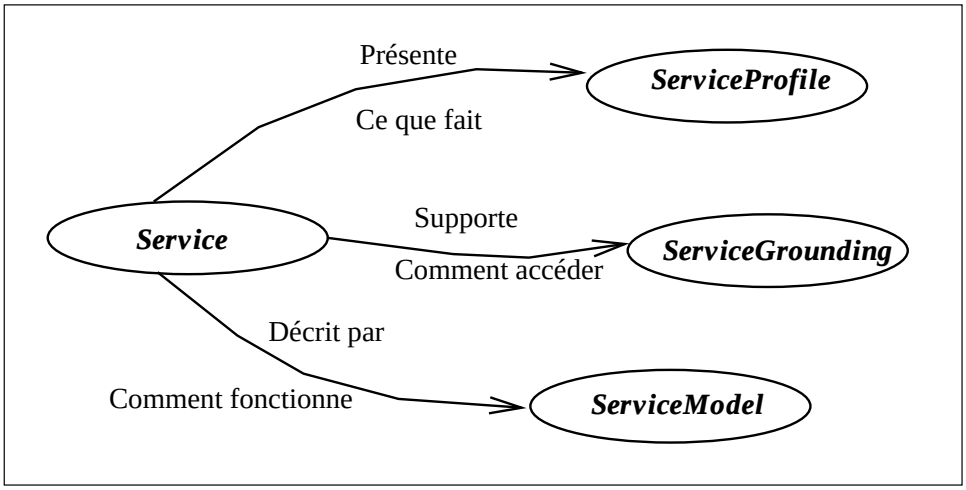

FIG. 1.10 – Représentation des différents éléments de OWL-S.

1. ***Profil de service (ServiceProfile)*** : ce profil donne une description de haut niveau du service et de son fournisseur en décrivant non seulement les services fournis, mais également des pré-conditions à la fourniture de ce service. Il est utilisé lors de la publication et la recherche d'un service. Il inclut trois types d'informations décrites au format RDF[22] : la description, le comportement fonctionnel et les attributs non fonctionnels.

   - *La description* : doit être compréhensible par les humains. Elle est composée de la classe 'acteur' (*actor*), dont les propriétés sont les suivantes : adresse physique, URL et d'autres renseignements divers (*nom, téléphone, adresse électronique, etc.*).
   - *Le comportement fonctionnel* : permet de rendre public les opérations qu'effectuent le service, en décrivant les paramètres d'entrée et de sortie de la méthode.
   - *Les attributs fonctionnels :* apportent des informations supplémentaires concernant le service. Ces informations peuvent être soit des pré-conditions à l'accès du service, soit des données relatives à la qualité de service (*temps de réponse, coût du service, etc.*).

2. ***Modèle de service (ServiceModel) :*** présente le fonctionnement du service en décrivant dans le détail et de manière relativement abstraite les opérations à effectuer pour y accéder. Le modèle de service est également écrit en RDF. Il s'agit du modèle de processus qui représente le fonctionnement du service en terme de processus internes. Il permet également d'effectuer un contrôle poussé du déroulement du service. Les composants principaux du ce modèle sont :

[22]Resource Description Framework, http ://www.w3.org/RDF/

- *L'ontologie de processus (process ontology) :* décrit un service en termes de paramètres d'entrée, de sortie, de pré-conditions et post-condition. Elle peut être utilisée afin de supporter l'invocation et la composition automatique de services Web.
- *L'ontologie de contrôle de processus (process cotrol ontology)* : décrit chaque processus comme étant un état, en prenant en compte son activation, son exécution et sa terminaison.

3. ***Accès au service (ServiceGrounding)*** : permet de décrire dans le détail la manière d'accéder à un service. Tout type abstrait déclaré dans le *Service Model* s'y verra attribuer une manière non ambiguë d'échanger l'information. C'est dans cette partie que le protocole à utiliser, les formats de messages et le type de transport sont spécifiés. La spécification concrète des messages est effectuée par l'intermédiaire de WSDL.

OWL-S permet de réaliser les deux tâches suivantes :

- ***Découverte automatique de services Web :*** implique une localisation automatique des services Web qui fournissent un service particulier et satisfaisant les contraintes demandées. OWL-S permet d'exprimer et de résoudre des requêtes avec un contenu sémantique. L'information nécessaire pour la découverte de service Web peut être spécifiée comme un langage sémantique interprétable par ordinateur aux sites du service Web et l'annuaire de service où les ontologies peuvent être utilisés pour la localisation automatique des services. Ainsi, OWL-S doit fournir les annonces déclaratives des propriétés et des possibilités de service pouvant être utilisées pour une découverte automatique

- ***Invocation automatique de services Web*** : implique l'exécution automatique de service Web identifié par le programme d'application. L'exécution d'un service Web peut être considérée comme une collection d'appels de fonction. OWL-S fournit un API déclaratif et interprétable par ordinateur pour exécuter ces appels de fonction. Un agent logiciel devrait pouvoir interpréter le langage pour comprendre quelle entrée est nécessaire, quelle information sera retournée et comment exécuter le service automatiquement. Ainsi, OWL-S fournit un ensemble d'APIs pour que l'invocation soit automatique.

OWL-S doit cependant être soutenu par le langage WSDL afin de pallier au manque de gestion d'échanges de messages, ce qui permettra par exemple d'utiliser SOAP pour échanger des messages XML. OWL-S pourra alors être réservé à une description abstraite et sémantique des services. De plus, son utilisation des logiques de descriptions pour modéliser les services permet une grande puissance d'expression, que ne possèdent pas les autres systèmes.

La figure 1.11 permet de présenter la relation entre OWL-S et WSDL.

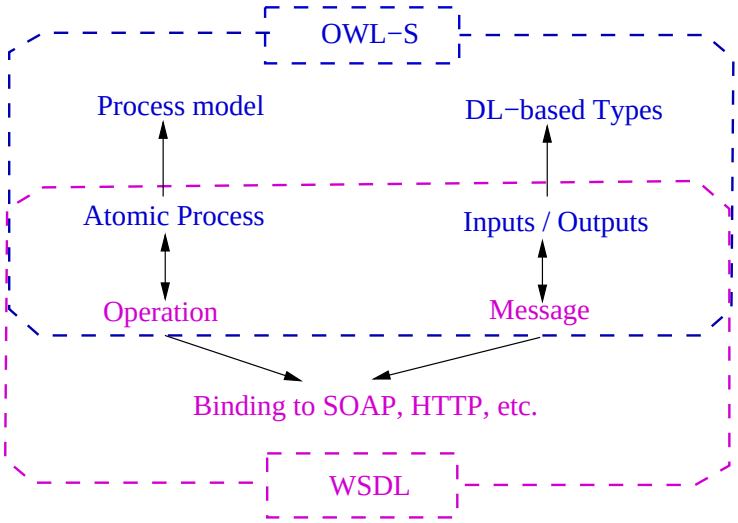

FIG. 1.11 – Relation entre OWL-S et WSDL.

1.5.2.2 WS-Discovery *Web Services Dynamic Discovery*

WS-Discovery est une spécification développée par les représentants de *Microsoft, Bea Systems, Intel et Canon* [7]. Cette spécification définie des protocoles multi-diffusion (*multicast*) pour annoncer et découvrir dynamiquement les services Web.

Au lieu de s'enregistrer dans un registre connu, les services Web qui souhaitent être découverts, annoncent explicitement leur arrivée et leur départ sur réseau. Lorsqu'un service Web se connecte au réseau, il annonce son arrivée en émettant un message *"Hello"*. En quittant le réseau, le service Web doit envoyé un message "*Bye*" au réseau. Ce message informe les autres services du réseau que le service Web concerné n'est plus disponible.

Pour compléter cette approche élémentaire de l'arrivée et de départ des services, WS-Discovery définit deux messages, "*Probe*" et "*Resolve*", pour localiser les services Web. Ces messages ne sont émis qu'en mode multi-diffusion. Afin de limiter le trafic réseau et d'optimiser le processus de découverte, WS-Discovery propose une entité appelée *proxy de découverte*. L'objectif de cette entité est de maintenir l'information sur les fournisseurs de services Web. Un tel *proxy* remplace le besoin d'envoyer des messages en multi-diffusion. Les messages "*Probe*" sont envoyés en mono-diffusion (*unicast*) vers le *proxy*.

Par défaut, un nouveau client suppose qu'aucun *proxy de découverte* (DP) n'est disponible, et en écoutant à ces annonces, les clients peuvent détecter les nouveaux services Web

disponibles. Pour chercher un service Web par type, le client envoie un message *"Probe"* au groupe multicast. Pour localiser un service par nom, le client envoie un message *"Resolve"* au même groupe multicast. À la différence d'un client, un service Web envoie toujours les messages *"Hello"* et *"Bye"* en multi-diffusion, et répond toujours à *"Probe"* et *"Resolve"* avec *"Probe Match"* et *"Resolve Match"* en mono-diffusion, respectivement. Il n'a pas besoin de reconnaître explicitement et/ou dépister la disponibilité d'un *proxy de découverte*.

Cependant, si un ou plusieurs *DP* sont disponible, ils annoncent leur arrivée en envoyant un message *"Hello"* en mono-diffusion avec un *proxy de découverte* de type bien connu en réponse à toute *"Probe"* ou *"Resolve"*. Les services Web recevant ce message peuvent tirer parti des services de *proxy* et ne sont plus obligés d'utiliser les protocoles de multi-diffusion. Avec l'écoute à ces annonces, les clients détectent qu'un ou plusieurs *DP* sont disponibles, et pour des recherches suivantes, les clients utilisent un protocole spécifique à ce *proxy de découverte* pour communiquer avec un ou plusieurs de ces *proxy*. De tel protocole définisse les messages de recherche que les clients envoient directement au *proxy de découverte* plutôt qu'à un groupe multicast. Si ces *DP* sont insensibles, ou s'ils envoient un *"Bye"*, les clients retournent à employer les messages indiqués ci-dessus.

1.5.2.3 WSDA - *Web Service Discovery Architecture*

WSDA [89] est une architecture ouverte pour la découverte de services Web. Elle incorpore plusieurs standards comme XML, XML Schéma, SOAP, WSDL et XQuery. Au temps d'exécution, les applications peuvent employer cette architecture pour découvrir les services. Celle-ci définit des services, interfaces, opérations et des protocoles de liaison. Le plus important de cette architecture est la proposition d'une étendue au registre UDDI pour que cela offre quatre interfaces : l'interface *Presenter, Consumer, MinQuery et XQuery*.

- ***Presenter*** : fournit au client la possibilité de chercher la description la plus récente du service Web. Cette interface a un identificateur qui peut être une URI ou une URL et un mécanisme de recherche *http verb="GET"*.
- ***Consumer :*** permet aux fournisseurs de publier un ensemble de tuples en utilisant l'opération *publish*. Un tuple peut inclure un contenu arbitraire tels que : la description de service ou de la qualité de service, la localisation d'une copie, etc. Il a comme attribut un *content link* (*URI ou URL*), un *type* -le sort de contenu à publier, un *context* -décrit le pourquoi de la publication du contenu ou comment il peut être utilisé, quatre *soft state time stamps* et deux éléments facultatifs : *metadata* et *content*. La déclaration de

la méthode *publish* est : *(TS4, TS5) publish(XML tupleset)*.

- ***MinQuery*** : permet de consulter l'ensemble de services Web publiés dans le registre UDDI. Cette interface a deux opérations *XML getTuples()* et *XML getLinks()*.
- ***XQuery*** : implémente un ensemble de requêtes plus puissantes sur les services Web publiés telle que : une requête de selection sur les attributs (*context, type, content*). Ces requêtes sont exprimées dans le standard XQuery [88]. La déclaration de la méthode *query* est : *XML query(XQuery query)*.

Outre les fonctionnalités offerte à travers les quatre interfaces, la caractéristique la plus importante de WSDA est de pouvoir exprimer des requêtes en utilisant le langage XQuery. Ceci est beaucoup plus puissant que le type des consultations permises par l'API d'UDDI.

1.5.3 Composition de services Web

Depuis ces dernières années, le Web devient la plate-forme par laquelle beaucoup d'entreprises communiquent avec leurs partenaires et proposent leurs services aux clients. Par conséquent, les services Web, qui sont des applications indépendantes pouvant être publiées, localisées et accédées à travers le Web [91], sont très utilisés. Les exemples de services Web sont des applications pour *réserver un hôtel, acheter un ticket d'avion, visiter un monument, etc*, à travers le Web. Individuellement, chaque service fournit une fonctionnalité limitée. Ainsi, apparaît le besoin de composer des services Web afin de construire un service composite (*ou composé*) proposant une fonctionnalité de plus haut niveau d'abstraction.

Étant donné un ensemble de services Web disponibles et une requête du client, le problème de la composition est de construire un nouveau service Web composite, donc spécifier comment coordonner les services Web disponibles pour réaliser la requête du client. Une telle spécification peut être obtenue soit automatiquement, c-à-d, trouver un algorithme qui réalise cette composition, ou manuellement par un être humain. De plus, comment traiter la coordination de plusieurs composants de services Web et vérifier le flux de contrôle et le flux de données, afin de garantir l'exécution correcte du service Web composite.

La composition de services Web est fondée sur deux activités différentes [62] : *l'orchestration* et *la chorégraphie*. L'orchestration de services fait référence à l'activité de création de processus, exécutables ou non, qui utilisent des services Web. La chorégraphie se concentre sur les séquences de messages échangés entre différents acteurs (*typiquement l'échange de messages entre des services Web*) plutôt qu'un processus spécifique exécuté par un acteur en particu-

lier. En effet, l'industrie se concentre beaucoup plus sur l'orchestration et la spécification des services Web composites, exprimées dans des langages. Actuellement il existe différents standards permettant de réaliser l'orchestration et la chorégraphie, tels que BPEL4WS *(Business Process Execution Language for Web Services)*, WSCI *(Web Service Choreography Interface)* et BPML [49] *(Business Process Management Language)*.

L'objectif de la composition de services est de créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services existants, composés ou non en vue d'apporter une valeur ajoutée. Étant donnée une spécification de haut niveau des objectifs d'une tâche particulière, la composition de services implique la capacité de sélectionner, de composer et de faire interopérer des services existants [51]. Dans la suite, nous nous intéresserons au problème de composition pour lequel, on peut relever dans la littérature, les problèmes sous-jacents suivants :

- Comment découvrir l'ensemble des services Web à utiliser dans la composition ;
- Comment publier les composants qui constituent la partie d'une composition ;
- Comment définir la qualité de service Web composite en termes de composants des services par rapport à la requête du client ;
- Comment mettre la sécurité et les politiques d'authentification du service Web composite basé sur ceux des composants.

1.5.3.1 Substitution de services Web

La composition de services Web implique des nouvelles questions, à savoir la compatibilité et la substitution. Comme dans la situation suivante : un service Web S_1 qui constitue la partie d'un service Web composé S_c devient non disponible. Afin de continuer à exploiter le service S_c, un autre service S_2 qui offre les mêmes fonctionnalités que S_1 doit être trouvé, ainsi que le service Web S_c obtenu en substituant S_1 avec S_2 se comporte comme S_c. Il y a donc besoin de la substitution d'un service Web par un autre dans les cas suivants :

- Pendant l'exécution, le service Web échoue ou ne peut pas être atteint.
- Il y a un nouveau service Web qui fournit un service meilleur, en terme de paramètres de qualité de service.
- Une nouvelle version d'un service Web choisi, offerte par le même fournisseur est disponible.

Avec la substitution, le but est d'être capable de substituer un service Web, si c'est nécessaire, sans affecter le comportement observable du processus coopératif. L'approche présentée dans [2] est basée sur l'idée que les spécifications de services Web doivent être classées pour définir les relations entre eux en termes de compatibilité. Pour établir la compatibilité, il compare les signatures des méthodes et les spécifications des méthodes. Cette approche exige que la spécification d'une méthode soit la pré-condition et la post-condition.

1.5.4 Sécurité de services Web

Le problème principal des technologies de services Web réside dans l'absence de standards de sécurité. La version actuelle du protocole SOAP, définie par *IBM et Microsoft*, ne prend absolument pas en compte ces aspects. SOAP définit le format standard des messages qui transiteront entre les partenaires commerciaux via HTTP. L'utilisation du langage XML pour établir cette structure de messages permet de passer sans encombre les pare-feux des entreprises. Il reste toutefois possible d'assurer un minimum de sécurité avec les standards actuels du Web. Le protocole HTTP-S[23] [13], par exemple, assure une sécurité de base *point à point*. En revanche, il n'existe aucun moyen de garantir la sécurité de *bout en bout* des échanges commerciaux. Les techniques standard de sécurisation des applications Web sont applicables seules ou combinées afin d'offrir des services Web sûrs et fiables. Ces techniques s'inspirent d'une riche expérience et sont tout à fait efficaces. Cependant, elles n'offrent pas des solutions intégrées au sein de l'architecture des services Web. À mesure que le service Web XML augmente en complexité (*nécessité de traverser des frontières sécurisées, présence sur plusieurs systèmes ou entreprises, etc.*), les programmeurs sont contraints de bâtir des solutions personnalisées qui, bien qu'efficaces, n'offrent pas une véritable interopérabilité.

Pour répondre à ces besoins et accroître l'interopérabilité entre les services Web, deux principales organisations de sécurité tentent de fournir une architecture sécuritaire pour les services Web. Ils s'agit de SAML (*Security assertion markup Language*) [83] d'OASIS et de WS-Security (*Web Services Security Language*) [75] développé par IBM. SAML vise à créer un standard de propagation de *bout en bout* du contexte de sécurité de services Web. WS-Security s'appuie sur le mécanisme d'extensibilité de la norme SOAP pour proposer des fonctionnalités de sécurité optimales, directement intégrées dans les services Web. Il a été dernièrement présenté à OASIS qui a pris le standard sous son aile. OASIS hébergeant les deux standards, certains observateurs anticipaient une confrontation entre les promoteurs de l'un et de l'autre standard.

[23] Secure HTTP

1.5.4.1 WS-Security *Web Services Security*

Cette spécification a été soumise par *Microsoft, IBM et Verisign*. WS-Security couvre les fonctionnalités suivantes : l'authentification, la gestion de l'intégrité et le cryptage des échanges. Il précise en outre la manière de décrire les droits et les conditions d'utilisation associés à un service Web.

Ces fonctionnalités ne constituent pas en soi une solution de sécurisation complète ; le langage WS-Security est un bloc fonctionnel qui s'utilise conjointement avec l'infrastructure de services Web et d'autres concepts de sécurité ou protocoles en général, tels que Kerberos [63], SSL (*Secure Sockets Layer*) [34] ou PKI (*public-key infrastructure*) [73] pour répondre à diverses exigences en matière de sécurisation des applications. La spécification intègre notamment les spécifications mises au point par le W3C autour du chiffrement et de la signature des documents XML - Signature Digital XML (*XML Digital Signature*) [80], l'encodage XML (*XML Encryption*) [30] et les certificats X.509 [45]. La spécification se compose de trois documents :

1. ***SOAP Message Security 1.0 core*** : décrit les extensions apportées aux messages SOAP afin d'assurer l'intégrité et la confidentialité des messages. Cette spécification ne définit pas un type de sécurité spécifique mais se veut capable de prendre en compte un large panel de modèles de sécurité et de technologies d'encodage.
2. ***UsernameToken Profile*** : décrit comment un utilisateur de service Web peut fournir un "*UsernameToken*" afin de s'identifier (*en utilisant un nom d'utilisateur et éventuellement un mot de passe*) auprès d'un producteur de service Web.
3. ***X.509 Certificate Token Profile*** : décrit comment utiliser les certificats *X.509* avec la spécification "*SOAP Message Security 1.0 core*". Un certificat *X.509* peut être utilisé pour valider une clé publique servant à authentifier un message (*éventuellement encodé*) supportant WS-Security.

WS-Security se décline en nombreuses sous-spécifications : WS-Trust, WS-Secure, WS-Policy, WS-Privacy, WS-Federation et WS-Authorization.

WS-Policy, WS-Trust et WS-Privacy : décrivent la procédure de sécurité, ainsi que les parties impliquées dans cette procédure et la manière dont un service Web est tenu de la prendre en compte avant de s'exécuter.

WS-Secure Conversation, WS-Federation et WS-Authorization : se chargent de définir comment est authentifié un message.

Destinée à intégrer d'autres modules en cas de besoins, la spécification WS-Security se veut avant tout évolutive. Une principale volonté exprimée par les trois acteurs dans leur communiqué commun : concevoir un cadre qui puisse s'adapter à n'importe quelle technologie de sécurité.

1.6 Outils disponibles pour la mise en œuvre de service Web

La base de service Web étant étudiée, cette section est consacrée aux différents outils afin de mettre en œuvre ce concept.

1.6.1 Plates-formes commerciales

Le concept service Web atteint aujourd'hui de nombreux acteurs du monde informatique. Les compagnies d'infrastructure de *e-Business* commencent à annoncer des plates-formes pour supporter quelques niveaux d'automatisation de ce dernier. Des exemples de tels produits incluent *Hewlett-Packard e-parlent* ; une plate-forme de description ; d'enregistrement et de découverte automatique pour les e-services, les outils de *Microsoft .NET* et *BizTalk*, *Oracle* , les cadre d'application d'*IBM* pour les *e-Business* et *Sun's Open Network Environment*. Nous nous sommes restreints aux éditeurs les plus reconnus : *Microsoft, IBM et Sun*

Visual studio .NET de Microsoft[24]

Gère toutes les technologies de base du développements de services Web : SOAP, XML, WSDL, UDDI. Plus particulièrement, cet outil facilite le déploiement des fichiers WSDL qui sont accessibles à partir d'une URL désignée spécialement par l'outil. Visual studio n'est supporté que par la plate-forme *.NET*.

[24]http ://msdn.microsoft.com/vstudio/productinf/trial/default.aspx

Websphere Studio Application Developper de IBM[25]

Intègre un outil de génération automatique des documents WSDL. Le registre UDDI est supporté par *Websphere Studio Application Developper* depuis sa version 5. L'orchestration de service est rendue possible par l'intégration du langage WSFL dans cette plate-forme.

Sun ONE developper Studio de Sun[26]

Ce logiciel est supporté par la plate-forme développée par *Sun* : *J2EE*. *Sun ONE* propose le mécanisme de génération croisée de WSDL et Java en déployant le fichier WSDL à partir d'une classe java et inversement. Les protocoles de communication sont assurés par SOAP 1.1 et une version java de SOAP. L'orchestration des services Web est assurée par ebXML[27].

La communauté de logiciels libres *Apaches* participe aussi à la mise en œuvre des services Web en proposant plusieurs outils. Nous en présentons deux : *Apache-SOAP* et *AXIS.*

Apache-SOAP[28]

C'est le premier projet de la fondation Apache concernant les services Web. Il fonctionne sur toutes les plates-formes supportant le langage Java. Apache-SOAP représenterait uniquement une implémentation de SOAP et ne supporte pas WSDL : sa génération automatique n'est pas possible, ainsi son développement doit être manuel. Il est remplacé par une nouvelle architecture afin d'être plus flexible et plus performant : *Axis*

Axis - Apache eXtensible Interaction System[29]

Axis succède à *Apache-SOAP* dans le cadre des projets de services Web. L'amélioration la plus conséquente apportée à *Axis* est la génération automatique de fichiers WSDL. Ce mécanisme est réversible : à partir d'une classe java (*réciproquement un document WSDL*) *Axis* génère un fichier WSDL (*réciproquement des fichiers java*) [3].

Il y a également quelques plates-formes proposées par le milieu académique. [77] est une plate-forme d'invocation et de matching de services soutenue par des agents logiciels. [20] est une plate-forme de composition de services basée sur l'infrastructure *workflow* de *HP*.

[25] http ://www-3.ibms.com/software/awdtools/studioappdev/
[26] http ://wwws.sun.com/software/sundev/jde/index.html
[27] electronic business using eXtensible Markup Language, http ://www.ebxml.org
[28] http ://ws.apache.org/soap/
[29] http ://ws.apache.org/axis/

1.7 Conclusion

Cette partie établit une étude du fondement de notre travail : les services Web. Dans un premier temps, les concepts de base des services Web ont été étudiés. Nous avons évoqué une définition du service Web en se basant sur les travaux du W3C. Nous avons ainsi étudié deux architectures : la première dite de référence est dédiée aux services Web élémentaires, la seconde, associée aux services Web composites, est appelée étendue. Cette architecture repose sur différents standards. Nous mettons en relief les trois principales : WSDL, décrivant les services Web fournis, UDDI publiant les descriptions des services Web et SOPA permettant d'établir un protocole de communication entre les entités fournisseur et demandeur de service. Les technologies de Web sémantique pour les services Web ont été passé en revue.

Parallèlement à l'évolution des recherches sur les services Web, nous assistons à un regain d'intérêt autour des sujets rattachés au service Web tels que la sélection, la découverte et la composition. Tout d'abord, nous avons mis l'accent sur la problématique de l'intégration qui peut être établie comme une composition de services entre eux, en soulignant les problèmes liés à la combinaison de services lors de l'intégration. Puis nous avons décrit les travaux les plus pertinents.

Un défit important qu'il reste à entreprendre est celui de la sécurité. c'est un problème permanent et en constante évolution de l'industrie informatique qui ne peut se régler complètement. Cependant, pour que les services Web jouissent d'une implantation massive auprès des entreprises, le niveau de sécurité minimum et intrinsèque des services Web, devra se développer considérablement. Plusieurs technologie et protocoles sont déjà à l'etude et/ou utilisées afin de sécuriser les services Web. Afin de mettre en œuvre le concept de service Web les éditeurs de logiciels fournissent différents outils.

Nous traiterons, dans la deuxième partie de notre mémoire, le problème de la composition de services. D'abord, nous nous discutons la situation actuelle du service orienté calcul en particulier la composition de services Web, puis nous nous concentrons sur la description de la composition de services Web. Ainsi, nous soulignons les différents mécanismes transactionnels pour les services Web ayant un grand impact sur la composition. Nous exposons également les différents travaux de la littérature concernant la composition de services Web.

Partie 2

Approches de composition de services Web

2.1 Introduction

Les services sont conçus pour l'implémentation des processus métiers dans un environnement de calcul distribué en se basant sur un ensemble de standards émergeant. Le SOC[1] comme une nouvelle approche vers le calcul distribué, a préparé le terrain pour des changements dans la conception, l'architecture, la livraison et l'utilisation des applications logicielles. Dans ce paradigme, les entités distribuées sont implémentées comme des services. Ceux ci sont simplement des composants logiciels autonomes, indépendants des plates-formes en vue de fournir l'interopérabilité et la collaboration dans des environnements hétérogènes. La conception et l'implémentation de telle infrastructure pouvant être réalisées en utilisant les services Web.

La nature neutre des plates-formes des services Web crée l'opportunité de la construction des services composites en combinant des services élémentaires ou complexes déjà existants, probablement offert par différentes entreprises. Par exemple, un service de planification de voyage peut être développé en combinant plusieurs services élémentaires tels que *la réservation d'hôtel*, l*'achat d'un billet d'avion*, *location de voiture*, etc. Le processus de développement de ces services complexes s'appelle *la composition de service*. Il nécessite entre autres la spécification des activités qui doivent être exécutées, comment ces activités sont reliées, comment l'information est utilisée, quelles exceptions pouvant se produire, quelles sont les ressources impliquées, etc. Cette tendance a déclenché un nombre considérable d'efforts de recherche sur la composition de services Web dans les deux domaines académique et industriel.

[1] *Service Oriented Computing*

Un des secteurs importants qui tire bénéfice de l'application du paradigme de SOC est le secteur *e-Business*. Le SOC fournit principalement l'infrastructure requise pour intégrer des applications *e-Business*. En effet, il permet aux organismes d'implémenter leurs services métiers sur Internet en utilisant des langages et des protocoles standards. De tels standards permettent aux développeurs d'accéder aux applications déployées en se basant sur ce qu'elles font, plutôt que sur la façon dont elles le font, ou la façon dont elles ont été implémentées.

2.2 Le calcul orienté service

Le calcul orienté service SOC [67, 1] est un paradigme de calcul qui vise à réaliser des applications distribuées au sein (*ou inter*) d'une organisation par la réutilisation des services existants comme étant des blocs de base. Le SOC implique les couches : services, fonctionnalités et rôles, comme décrit par l'architecture orientée service étendue (*SOA*) représentée dans la figure 2.1.

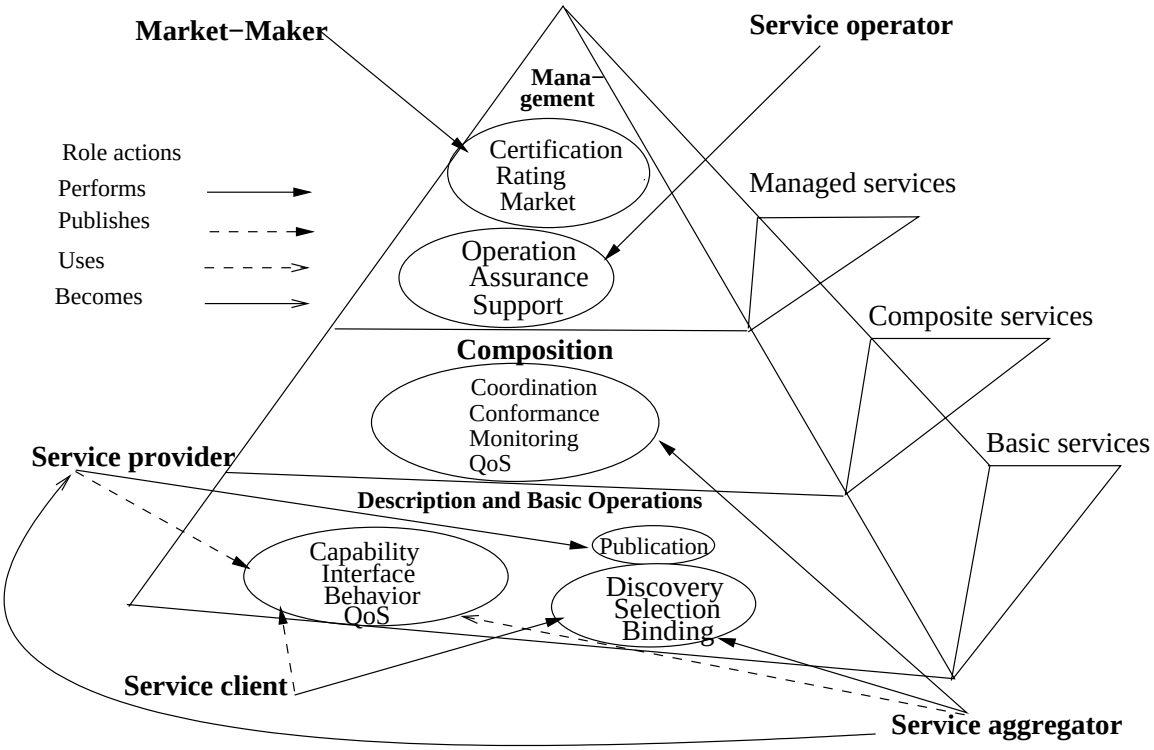

FIG. 2.1 – Architecture orientée service- étendue : couches services, fonctionnalités et rôles.

Les services de base, leurs descriptions et les opérations de base (*publication, découverte, sélection et liaison*) qui utilisent ces descriptions constituent la base de SOA. La couche supérieure dans la pyramide de SOA fournit un support additionnel requis pour la gestion de composition de services. La couche de composition de services contient les rôles et les fonctionnalités nécessaires pour la consolidation de multiples services dans un seul service composite. Les services composites résultants peuvent être utilisés par les agrégateurs de services comme des composants (*services de base*) dans d'autres compositions de services ou par les clients comme des applications. Ainsi, Les agrégateurs de services deviennent des fournisseurs de services, en publiant les descriptions des services composites qu'ils créent. Ils développent des spécifications permettant au service composite d'exécuter des fonctions incluant :

- ***Coordination :*** contrôle l'exécution des services composants, gère le flux de données entre eux et le résultat des services composants (*en spécifiant le processus workflow et utilisant le moteur du workflow pour contrôler le service au temps d'exécution*).
- ***Monitoring :*** souscrit les événements ou l'information produites par les services composants, et publie les événements composites de haut niveau (*par le filtrage et la corrélation des événements composants*).
- ***Conformance :*** assure l'intégrité du service composite par le matching de ses types de paramètres avec ceux de ses composants, impose des contraintes sur les services composants et effectue la fusion de données des activités.
- ***Qualité de Service (QoS) de la composition :*** agrége et relie la QoS des composants pour dériver la QoS composite, y compris le coût global, performance, sécurité, authentification, intégrité, fiabilité et disponibilité du service composite.

L'application du SOC dans le Web est manifestée par les services Web. Les interactions entre services Web se produisent typiquement comme des appels SOAP contenant des données XML. Les clients et les fournisseurs de services utilisent les technologies standards des services Web pour effectuer les opérations de base de SOA représentées sur la figure 2.1. Les agrégateurs de services peuvent utiliser BPEL4WS [48] pour créer de nouveaux services Web en définissant la composition correspondante des interfaces et des processus internes des services existants. Les notions de la réutilisation et de l'extensibilité sont les clés du paradigme SOC : ces notions définissent comment des services plus complexes ou plus adaptés aux besoins du client peuvent être établis, à partir d'autres services existants, en minimisant le temps de développement et de ressources.

2.3 Composition de services Web

La réponse à certaines requêtes de clients doit faire appel à plusieurs services Web. Ce processus, que nous appelons composition de services permet aux services d'interagir de manière intelligente en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en services plus complexes. La composition de services est adressée quand une requête du client ne peut pas être satisfaite par un seul service disponible, mais par un service composite obtenu en combinant d'autres services disponibles. Un service composite peut être considéré comme une sorte de client par rapport à ces composants, puisqu'il les invoque indirectement. On distingue deux types principaux de composition de services Web :

2.3.1 Composition statique

Il s'agit d'un type de composition dans lequel les services Web à composer sont pré-calculés avant qu'un client ne fasse une requête du service composite. Ce type de composition peut être appliqué dans des environnements '*stables*' où les services Web participants sont toujours disponibles et où le comportement du service composite est le même pour tous les clients.

2.3.2 Composition dynamique

Dans ce type de composition, les services Web à composer sont déterminés lors de l'exécution de la requête d'un client. Ils peuvent être déterminés selon les besoins, les contraintes de chaque client, la disponibilité des services Web, etc. La composition dynamique apparaît la plus intéressante. D'une part, elle promet d'être capable de faire face à un environnement très dynamique dans lequel des services apparaissent et disparaissent rapidement. D'autre part, elle permet de mieux satisfaire les besoins de chaque client. La plupart des travaux actuels se concentrent sur la composition dynamique des services Web.

2.3.3 Cycle de vie de la composition de services

Avant de développer concrètement un service Web, le cycle de vie de ce service doit être connu. Tout d'abord nous rappelons le déroulement de la création d'un service Web

élémentaire (*aussi dit de base ou atomique*). Ensuite, nous décrivons le cycle de vie d'une composition de services.

2.3.3.1 Service Web élémentaire

Pour créer un service Web, il convient premièrement de définir quel type de fonction sera fourni aux organisations l'utilisant. La seconde étape est l'implémentation du service. Le langage de programmation est entièrement au choix de développeur. Une fois le service est implémenté, l'étape suivante est spécifique au service Web. Ce stade est le déploiement du service. Le document généré par le déploiement est un document WSDL décrivant le service. Il permet de déployer concrètement le service Web. Le service étant alors accessible, le fournisseur de ce service doit tout d'abord s'enregistrer sur un annuaire choisi. Puis sur ce même annuaire, le service doit être publié, c'est-à-dire rendu public afin qu'il soit utilisé.

2.3.3.2 Service Web composite

La plupart du temps qu'il soit dans le milieu industriel ou académique, la concentration est mise sur le développement des spécifications qui capturent les interactions entre les services. Cependant, peut d'attention est prêtée au cycle de vie de la composition de services [70, 92, 37].

J. Yang et **M.P. Papazoglou** dans [92] ont développé un cadre pour le développement de composition de services. Ce cadre préconise une approche par étapes selon les spécifications de composition de services. Les activités de cette approche par étapes sont collectivement désignées sous le nom du cycle de vie de la composition de services. L'idée derrière l'approche par étapes est de commencer par une définition abstraite et de la rendre progressivement concrète de sorte que nous puissions produire des processus exécutables de ces spécifications abstraites. Dans cette approche quatre larges phases sont distinguées : la définition, la programmation, la construction et l'exécution de la composition, comme illustré dans la figure 2.2.

Le système débute dans la phase de définition *"Définir une composition abstraite"* avec un service composite abstrait, qui spécifie les activités constitutives d'un service composite et les contraintes sous lesquelles elles fonctionnent. Les compositions abstraites ne fournissent pas toutes les informations nécessaires pour le processus métier (*comment les activités sont structurées ? et quels sont les services utilisés ?*).

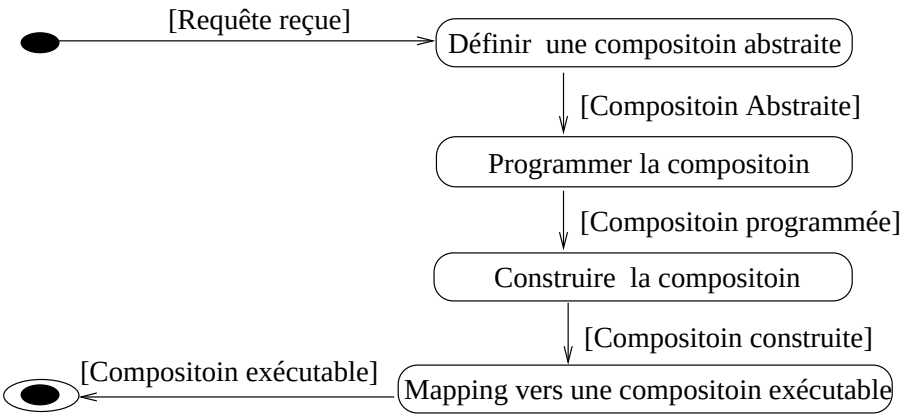

FIG. 2.2 – Cycle de vie de la composition de services.

Dans la phase de programmation *"Programmer la composition"*, le système de composition de services détermine comment et quand les services fonctionneront et les prépare pour l'exécution. Son but principal est de rendre la définition développée dans la phase précédente concrète par la corrélation et la synchronisation des messages pour exprimer les dépendances de données. Pendant cette phase, le système peut produire des programmes alternatifs de composition et les présente au développeur pour la sélection.

Après, le système de composition de services se poursuit avec la phase de construction *"Construire la composition"*, pour construire une composition non ambiguë des services concrets à partir de l'ensemble de services constituant souhaitables ou potentiellement disponibles. Similaire à la phase de programmation, le système peut produire des combinaisons de constructions alternatives (*changement dans la qualité ou le prix*) à partir desquelles le développeur d'application peut choisir.

Pour finir, pendant la phase d'exécution "*Mapping vers une composition exécutable"*, le système de composition de services prépare les services composites construits pour l'exécution. La spécification résultante est convertie en un langage exécutable standard d'orchestration de services Web (*BPEL4WS*).

2.4 Mécanismes transactionnels pour les services Web

Un service Web peut être vu comme une application distribuée, s'appuyant sur d'autres services (*ou applications*) hébergés par des organisations différentes et indépendantes les unes des autres, ne partageant pas les mêmes règles de gestion. Ainsi, le choix d'un service entrant

dans une composition peut être effectué dynamiquement au moment de l'exécution.

L'exécution des services nécessite de s'appuyer sur des transactions distribuées, complexes, souvent de longue durée et qui éventuellement peuvent mettre en œuvre des mécanismes de compensation. De très nombreux modèles de transactions ont déjà été définis notamment dans le domaine des système distribués [1, 66]. Aucun de ces modèles ne répond parfaitement et complètement aux besoins de services composites. De plus, chaque service composant s'appuie sur un système de gestion de transactions choisi ou conçu pour le composant, considéré individuellement. Lorsqu'un service est intégré comme composant, il est fort probable que sont système de gestion de transactions ne répond pas aux besoins de la composition vue comme tout. Par conséquent, la composition de services Web est une tâche très complexe et challenge [9, 10]. Avant d'effectuer la composition de services, certains aspects pour permettre cette composition doivent être considérés. Nous discutons dans la section suivante les différentes issues ayant un grand impact sur la composition à savoir : *coordination, transaction, contexte et modélisation de conversation.*

2.4.1 Coordination

L'infrastructure de base des services Web présentée dans la section 1.2.3.1 satisfait des interactions simples et des invocations de méthodes entre les entités en interaction. Lors de la composition de services, il est probable que l'interaction exige la coordination des séquences des opérations, pour assurer l'exactitude et la consistance. Dans le but de modéliser ces abstractions et simplifier le développement des services Web, différents efforts de standardisation, tels que WS-Coordination [18] par IBM ou WS-CF par Sun [16] ont été pris.

2.4.2 Transaction

Pour ajouter de la garantie aux interactions, il est d'importance considérable d'ajouter des protocoles de transaction au cadre de coordination [66] afin de fournir des transactions à courte durée, appelées transactions atomiques, et aussi bien des activités metiers longues. Dans ce cas, il n'est pas toujours possible d'assurer les propriétés ACID (*atomicity, consistency, integrity, and durability*) des transactions du service Web. IBM a proposé un standard important appelé *WS-Transaction* [19, 35, 36], construit au dessus du *WS-Coordination*, définit des protocoles pour les transactions centralisées et pair-à-pair, dont toutes les deux exigent de la coordination.

2.4.3 Contexte

Dans la littérature, le terme contexte est largement utilisé et prend par fois des sens différents selon les travaux. En termes de services Web, un contexte peut être vu comme l'information utilisée par le service Web pour ajuster l'exécution et le rendement afin de fournir au client un comportement personnalisé et adapté à ses besoins. Le contexte est extensible avec de nouveaux types d'information à tout moment, sans changements dans l'infrastructure de base. Il peut contenir des informations telles que le nom du consommateur, l'adresse, la location courante et les ressources utilisées, y compris le matériel et le logiciel. Par exemple, le standard *WS-Context* [17], proposé par Sun, permet la spécification et la gestion du contexte.

2.4.4 Modélisation de conversation

Le modèle de conversation facilite la découverte automatique des services et des liens, l'analyse et la validation du modèle de composition de services, la génération de squelette de composition ainsi que la génération du modèle de conversation. Beaucoup de travaux relatifs à la modélisation des conversations dans les services Web ont été présentés, incluant : la spécification WSCL, WSCI, et aussi WS-Coordination et WS-Transaction, bien qu'ils ne fournissent pas des moyens pour modéliser les conversations métiers, mais proposent plutôt des conversations spécifiques pouvant être utilisées pour coordonner les parties en interaction [8].

2.4.4.1 WSCL - *Web Service Conversation Language*

WSCL [5] permet de définir les interfaces abstraites des services Web, c-à-d les conversations au niveau métier soutenus par un service Web. Il spécifie les documents XML à échanger et leur ordre. WSCL peut être employé conjointement avec d'autres langages de description de service comme WSDL pour fournir des informations sur les protocoles de liaisons ou pour spécifier les interfaces abstraites soutenues par les services concrets. WSDL spécifie comment envoyer des messages à un service, mais sans définir l'ordre dans lequel ils vont d'être envoyés. Tandis que, WSCL définit les séquences d'échange de documents entre services Web. Il y a quatre éléments principaux selon la spécification de WSCL.

Description des types de documents

Référence les types (*schémas*) de documents qui peuvent être échangés ou reçus au cours d'une conversation. Ces schémas sont des documents XML séparés et référencés par leur URL dans l'élément XMLDocumentType de la spécification de conversation.

Interactions

Modélisent les actions des conversations sous la forme d'échanges de documents entre deux participants. Les types suivants d'interactions sont actuellement définis dans WSCL : Receive et Send (*one-way interactions*), ReceiveSend et SendReceive (*two-way interactions*) et Empty.

1. *One-Way Interactions :* représente un message simple à sens unique envoyé ou reçu par un participant. On distingue deux sous-types pour ce type d'interactions : *Receive et Send*. *Send* représente un document envoyé par un participant. *Receive* représente un document reçu.
2. *Two-Way Interactions :* peut être soit un *SendReceive* ou *ReceiveSend*, selon si le participant envoie un message pour lequel il obtient une réponse (*SendReceive*) ou répond à une demande qu'il reçoit (*ReceiveSend*).
3. *Empty :* aucun document n'est échangé, le rôle de ce type d'interactions est de modéliser le début et la fin d'une conversation.

Transitions

spécifiant la relation d'ordre entre les interactions. Une conversation peut passée d'une interaction à l'autre suivant les séquences définies. Une transition spécifie une interaction source, une interaction destination, et optionnellement, un type de document de l'interaction source. Si l'interaction source spécifie plus d'un document a échangé et le type de ce dernier influe sur les prochaines interactions possibles il est nécessaire de spécifier une condition sur la transition.

Conversations

Les conversations listent toutes les interactions et les transitions qui forment une conversation. Elles définissent aussi des informations additionnelles sur la conversation, y compris le nom de la conversation et les interactions de début et de fin. Elles spécifient également l'ordre des opérations, c-à-d les séquences possibles dans lesquelles les documents peuvent être échangés.

Un avantage principal de WSCL est qu'il prépare le terrain pour la création des outils permettront aux développeurs de services de décharger l'infrastructure des tâches reliées à la conversation. Le fournisseur de service peut fournir des définitions de conversation directement aux utilisateurs du service en les enregistrant comme tModels d'UDDI. Ensemble, UDDI, WSDL et WSCL permettent aux développeurs d'implémenter des services Web capables de s'engager spontanément dans des interactions dynamiques et complexes inter entreprises.

2.4.4.2 WSCI - *Web Service Choreography Interface*

WSCI [6] est un langage de description d'interface, basé sur XML, décrivant comment les opérations de service Web peuvent être chorégraphies dans un contexte d'échange de messages. WSCI fonctionne en conjonction avec WSDL pour construire l'interface dynamique du service Web en réutilisant les opérations définies pour l'interface statique. Il décrit également le comportement observable de service Web en termes de dépendances temporelles et logiques entre les messages échangés, à l'aide de contrôle des séquences échangées de messages, la corrélation, gestion d'exception et des transactions. Les concepts suivants font partie du modèle fondamental :

Interface

WSCI vise à décrire le comportement externe observable d'un service Web dans un tel échange de message. Il mappe cette description à la notion d'interface. Un service Web peut exposer des interfaces multiples pour soutenir les différents scénarios. Dans ce sens, chaque interface décrit comment un service Web est perçu pour se comporter dans le contexte d'un scénario particulier. De ce fait, permettant au client de comprendre entièrement comment agir avec le service dans un tel scénario.

Activités et leurs chorégraphies

WSCI décrit le comportement d'un service Web en termes d'activités chorégraphies. Les activités peuvent être atomiques ou complexes, c-à-d périodiquement composé d'autres activités. Les activités atomiques traitant des messages correspondant à l'exécution des opérations définies dans des langages statiques de définition de service tel que WSDL. Les activités complexes se composent récursivement d'autres activités. Chaque activité complexe décrit la chorégraphie des activités qui la composent. WSCI supporte la définition des types suivants de chorégraphies :

- *Exécution séquentielle :* les activités doivent être exécutées dans l'ordre séquentiel.
- *Exécution parallèle :* toutes les activités doivent être exécutées, mais elles peuvent être exécutées dans n'importe quel ordre.
- *boucle :* les activités sont exécutées plusieurs fois en se basant sur l'évaluation d'une condition ou d'une expression. WSCI support : for-each, while et repeat-until.
- *Exécution conditionnelle :* un ensemble d'activités sont exécutées en se basant sur l'évaluation des conditions (*switch*) ou sur l'occurrence d'un événement (*choice*).

Processus

Un processus est une portion de comportement libellée par un nom pouvant être réutilisé par référence à ce dernier. Ainsi, le processus est l'unité de réutilisation de WSCI. Un processus est utilisé (*ou réutilisé*) par instanciation. Il peut être instancié par la réception d'un ou plusieurs messages déclenchant la chorégraphie décrite par ce processus. L'instanciation peut également se faire par appel ou reproduction du processus.

Propriétés

Elles sont justes un objet de modélisation pour WSCI et peuvent être employées quand la description du comportement externe du service exige de référencier une valeur dans la définition d'interface. Elles sont équivalentes aux variables dans les langages de programmation. En particulier, elles peuvent être employées pour éviter, dans la définition d'interface, les références explicites aux messages abstraits décrits dans WSDL. Le concept des propriétés fournit une couche d'abstraction dans WSCI. Par conséquent, elles peuvent être employées comme :

- Une abstraction du message ;
- Une référence à une valeur établie par un message à la réception ;
- Une référence à une valeur établie par le service lui-même lors de l'instanciation.

Pour ces raisons, aucune algèbre précise pour les propriétés n'a été spécifiée.

Contexte

Le contexte décrit l'environnement dans lequel un ensemble d'activités est exécuté. Chaque définition de contexte concerne un ensemble particulier d'activités et chaque activité est définie dans exactement une seule définition de contexte. Il décrit l'environnement en termes de :

- L'ensemble de déclarations disponibles aux activités ;
- Les propriétés transactionnelles associées à l'exécution des activités ;
- L'ensemble d'événements exceptionnels qui peuvent surgir pendant l'exécution des activités et le comportement exceptionnel déclenché par ces événements.

Corrélations de messages

Dans WSCI, une conversation représente un échange de message entre deux services ou plus dans un scénario particulier. Le concept de la corrélation décrit comment les conversations sont structurées et les propriétés qui doivent être échangées. C'est le mécanisme par lequel un message reçu par le service est associé à une conversation particulière. Chaque message appartenant à une conversation doit être désigné par une instance particulière de corrélation.

Comportements exceptionnels

La déclaration du comportement exceptionnel fait partie de la définition du contexte. Ainsi, WSCI permet d'associer un ensemble d'actions à exécuter en réponse à une exception. Les comportements exceptionnels qui peuvent se produire sont les suivants :

- Réception d'un message particulier exceptionnel dans ce contexte.
- Occurrence d'un *timeout*.
- Occurrence de faute qui pourrait correspondre à la réception d'un message de faute WSDL ou à la génération d'une faute par le service lui même.

L'occurrence d'une exception cause l'arrêt du contexte courant après que les activités liées à l'exception aient été exécutées. Ainsi, le comportement défini dans le contexte parent est repris.

Comportements transactionnels

Un contexte peut être associé à une transaction. La transaction décrit les propriétés transactionnelles des activités qui sont exécutées dans ce contexte. Fondamentalement, la présence de la transaction affirme que l'ensemble d'activités est exécuté d'une façon tout ou rien. Une transaction est soit atomique soit ouverte-incluse, c-à-d composé d'autres transactions.

Modèle global

Le modèle global est décrit par une collection d'interfaces ainsi que de liens entre les opérations des services communicants décrites par WSDL. Les services échangeront des messages à travers ces liens.

2.4.5 Gestion d'exécution

Lors de la composition, l'exécution du service composite doit être considérée. Nous distinguons entre l'exécution centralisée et distribuée du service Web composite. L'exécution centralisée est similaire au paradigme de client/serveur. Dans ce cas, le serveur est le programmateur central qui contrôle l'exécution des composants du service Web composite. Le paradigme distribué en revanche s'attend à ce que les services Web participants partagent leurs contextes d'exécution. Chacun des serveurs exécutant un service Web a son propre coordonnateur, qui doit collaborer avec les coordonnateurs des autres serveurs, pour garantir une exécution correcte et ordonnée des services. Une forme hybride des paradigmes distribués et centralisés peut être un coordonnateur contrôlant non seulement un, mais un ensemble de services Web.

2.4.6 Infrastructure

La découverte du service dans UDDI est limitée aux propriétés fonctionnelles seulement. Cependant, les services ont différentes propriétés (*décrites en termes d'attributs*), y compris les propriétés fonctionnelles et non fonctionnelles. Lors de la découverte de services, les propriétés non fonctionnelles du service doivent être matchées avec les contraintes des fournisseurs de services. Ce problème de traitement de contraintes de service est formellement défini en tant que la résolution des conflits possibles entre les fournisseurs et les demandeurs de service et s'assure que le service créé par la composition est valide. Ces contraintes peuvent être vues comme des contraintes imposées sur les attributs de service. Afin de décrire avec précision les services Web et faciliter leur découverte, les propriétés non fonctionnelles devant être prisent en compte, telles que les contraintes reliées à la qualité de service QoS. Dans ce contexte, des travaux de recherche [8, 69] suggèrent un autre modèle pour la découverte de services Web en étendant le modèle classique des services Web.

L'approche adoptée par ***Ran*** [69] (*voir section 1.5.1*) étend le modèle du service Web existant en ajoutant une autre entité appelée "*Certificateur QoS*". Le processus de publication, de la recherche et de liaison du service Web demeure le même. Mais il s'ajoute le rôle du *Certificateur QoS* pour approuver la QoS réclamée par le fournisseur de service avant qu'il soit enregistrer dans l'annuaire UDDI. Pour réaliser l'extension proposée du modèle de service Web, il est nécessaire d'étendre la structure de données d'UDDI par un élément additionnel appelé "*qualityInformation*". La structure fournit la description de différents aspects de QoS tels que :

- QoS relié au temps d'exécution, tels que : la performance (*temps de réponse, latence*), fiabilité, disponibilité, flexibilité, traitement d'exception, etc.
- QoS relié au support de transaction, tels que les standards supportés, coût et perfection.
- QoS relative à la sécurité, par exemple l'authentification, autorisation, confidentialité, cryptage de données et la non répudiation.

Indépendamment du modèle conventionnel de service Web, l'idée d'une infrastructure de service Web basée sur une technologie pair à pair (*P2P*) est très commune dans la littérature. Les systèmes *P2P* apportent plus de disponibilité, de scalabilité et d'extensibilité.

2.5 Travaux relatifs à la composition de services Web

De nos jours, plusieurs compagnies et organisations implémentent leurs services métiers sur Internet. Donc, assurer l'efficacité de leurs services hétérogènes au cours de l'exécution est une étape importante vers le développement des applications de services Web. En particulier, si aucun service Web ne peut satisfaire une requête d'un client, on essaye alors de composer de nouveaux services en combinant un ensemble de services Web existants. Des efforts considérables ont été fournis et des recherches très soutenues sur la composition de services Web dans le domaine académique et industriel. Plusieurs initiatives ont été prises dans le but de fournir des plates-formes et des langages qui offrent l'intégration des systèmes hétérogènes, tels que WSCI, WSCL, WSFL, BPEL4WS et DAML-S ServiceModel qui se focalisent sur la représentation des compositions de services où le flux des processus et les liaisons entre les services sont connus à priori. Nous étudions dans cette section les deux langages de composition de services Web les plus utilisés : *WSFL, BEL4WS*.

2.5.1 WSFL - *Web Service Flow Language*

Ce langage créé par HP [56] permet d'intégrer la composition de services Web comme un processus métier statique. Il propose deux styles de composition :

1. ***Modèle de flux (Flow Model) :*** il s'agit de la description de la succession d'étapes afin d'obtenir l'objet attendu. Ce modèle est aussi appelé processus métier.
2. ***Modèle global (Global Model) :*** il s'agit des interactions entre deux services, appelées aussi contrat.

Selon le type de composition, le document WSFL comporte comme élément premier le nom de type (*Flow Model ou Global Model*). Les services Web intervenant dans le processus sont connus par l'intermédiaire de l'élément *serviceProvider*. Le processus est constitué de différentes étapes appelées activités (*activity*). Pour chacune de ces activités, nous devons faire apparaître quel service à mettre en jeu (*performedby*), quelle opération est permise (*operation*) et par quel type de port (*portType*). L'élément *controllink* permet d'établir les conditions et les liens hiérarchiques entre les différentes activités constituant le processus.

2.5.2 BPEL4WS - *Business Process Execution Language for Web Services*

BPEL4WS [48] est un langage de composition permettant l'implémentation de service Web. Il a été proposé par *Microsoft, IBM, BEA* et d'autres grands acteurs du monde de l'informatique et de l'Internet. Ce langage est en fait une convergence entre le langage *XLANG* [76] de Microsoft et le langage *WSFL* 1.0 d'IBM, qui est également au format XML. BPEL4WS permettant la coordination des services Web dans un flux de processus. En d'autres termes, il s'agit d'un moteur d'orchestration. Il constitue une couche supérieure au langage de description WSDL. Il utilise, en effet, WSDL pour définir les opérations de services Web élémentaires à appeler et pour présenter le processus métier comme un nouveau service Web. Trois éléments permettant à BPEL4WS de gérer le flux de processus :

1. ***Les transactions (Exception handling and Transaction) :*** sont principalement utiles dans le contexte de BPEL4WS pour gérer les erreurs et les appels d'autres services si le service appelé est indisponible ou défaillant, etc.

2. ***Les partenaires (Roles and Partners) :*** ce sont les différents services Web invoqués dans le processus. Ils ont un rôle spécifique dans un processus donné. Chaque partenaire est décrit par son nom, son rôle (*myRole*) et son rôle dans le processus (*partnerRole, optionnel*).
3. ***Les espaces de stockage (Persistance and Containers) :*** ils permettent la transmission des données. Le flux de processus BPEL4MS permet que ces données soient consistantes à travers les messages échangés entre les services Web. Un message peut prendre les types suivants : l'appel (*invoque*), la réponse (*reply*) et l'attente (*receive*).

Malgré tous ces efforts, la composition de services Web reste encore une tâche très complexe en raison de l'augmentation du nombre de services Web disponibles ce qui entraîne un alourdissement de la recherche de ces services dans l'annuaire de publication devenu de plus en plus volumineux. De plus, les services Web peuvent être créés et mis à jour continuellement, donc le système de composition doit être capable de détecter ces mis à jour durant son exécution. Enfin, il n'existe pas un seul langage de définition et d'évaluation des services Web car ils peuvent être développés par des organisations différentes qui utilisent des modèles de conception et de description différents. Par conséquent, la construction de service Web composite automatique ou semi-automatique est critique au succès des applications de service Web. Actuellement, plusieurs équipes de recherche à travers le monde s'intéressent aux services Web et à leur composition. Plusieurs approches ont vu le jour et la plupart d'entre elles sont inspirées par les recherches dans le domaine des *workflow* et les techniques de l'intelligence artificielle (*IA*).

Pour la première, un service composite est semblable à un workflow [20] où il est spécifié le flux des tâches d'un travail. La définition d'un service composite inclut un ensemble de services atomiques avec le contrôle de flux de données entre les services. Dans la seconde, des méthodes de composition automatiques sont nécessaires pour générer le plan automatiquement et la plupart d'entre elles sont en relation avec un planning IA et la démonstration déductive de théorème. L'hypothèse générale de telles méthodes est que chaque service Web peut être spécifié par ses pré-conditions et ses effets (*ou post-condition*). L'état pré-requit pour l'exécution de service est la pré-condition et le nouveau état généré après l'exécution c'est l'effet. Un exemple typique est un service pour s'authentifier à un site Web. Les informations en entrée sont le *"nom utilisateur"* et le *"mot de passe"*, la sortie est un message de confirmation. Après l'exécution, le site Web change d'état de *"not logged in"* à *"logged in"* qui restera jusqu'à l'invocation du service *"log out"*. Si l'utilisateur peut spécifier les pré-conditions et les effets exigés par le service composite, un plan ou un processus est généré automatiquement par les preuves de théorèmes logiques ou les planning IA sans connaissance du workflow prédéfini.

Dans ce qui suit, nous allons se focaliser sur les méthodes basées sur les workflow et IA Planning. Les méthodes peuvent être entièrement automatique ou semi-automatique.

2.5.3 Composition de services Web par la technique de Workflow

Un workflow statique signifie que le demandeur doit établir un modèle de processus abstrait avant de commencer la composition. Ce processus abstrait inclut un ensemble de tâches et leurs dépendances de données. Chaque tâche contient une clause de la requête qui est utilisée pour chercher le bon service Web atomique afin d'accomplir cette tâche. Dans ce cas, seulement la sélection et les liens au service Web atomique sont faites par un programme automatiquement. Cependant, la composition dynamique crée le modèle de processus et sélectionne les services Web atomiques automatiquement. Cela exige au demandeur de spécifier plusieurs contraintes, y compris les dépendances atomiques, ses préférences et autres.

EFlow [20, 21] est une plate-forme pour la spécification, construction et gestion des services composites développée par HP. EFlow s'exécute au dessus des plates-formes e-services (*ESPs*), comme HP e-speak [23] ou Sun Jini [4]. Ces plates-formes permettent le développement, le déploiement et la livraison sécurisée des e-services aux entreprises et clients. Les fournisseurs de services peuvent enregistrer les descriptions des services qu'ils offrent et peuvent contrôler et gérer l'exécution de services. Les services peuvent être découverts par les consommateurs en spécifiant des critères de recherche. L'invocation de services est limitée aux utilisateurs autorisés.

Dans EFlow, le service composite est décrit comme un schéma de processus qui compose des services de base ou composites. Les services composites sont modélisés par un graphe définissant les flux d'invocations de services semblable aux diagrammes d'activité d'UML[2], qui est également traduit en une structure XML. Le graphe est créé manuellement mais il peut être mis à jour dynamiquement. Il contient des noeuds services, décisions et événements. Les noeuds services représentent l'invocation d'un service atomique ou composite, les noeuds décisions spécifient les règles de contrôle du flux d'exécution, et les noeuds événements permettent aux processus du service d'envoyer et recevoir plusieurs types d'événements. Les arcs dans le graphe représentent les dépendances d'exécution entre les noeuds. Bien que le graphe doit être spécifié manuellement, EFlow fournit l'automatisation pour lier les noeuds avec les services concrets. La définition d'un nœud service contient une méthode de recherche permettant de retourner une référence à un service spécifique. En particulier, la méthode de recherche est

[2]Unified Modelling Langage

exécutée à chaque fois qu'un nœud du service est activé car la disponibilité de services peut changer très fréquemment dans un environnement très dynamique. Le moteur EFlow notifie les accomplissements des méthodes et détermine alors le prochain nœud de service à exécuter. Il est relié au ESP par l'intermédiaire d'un adaptateur EFlow pour obtenir les notifications toutes les fois que l'environnement de service change, c-à-d, quand les définitions de services changent ou de nouveaux fournisseurs et services sont enregistrés dans ESP.

EFlow implémente plusieurs caractéristiques, comme la découverte automatique de service, sélection automatique de conversation, nœuds multi-service et nœuds génériques. Il spécifie des règles de sélection de services, permettant d'analyser les exigences des utilisateurs pour créer plusieurs paramètres d'entrée, afin de découvrir les services appropriés. Si plus qu'un service est trouvé, le meilleur matching est pris. La sélection de conversation permet à l'utilisateur de choisir la conversation dans un nœud de service au temps d'exécution, qui peut être utile quand les interfaces des services découverts sont inconnues. EFlow alors peut choisir une conversation appropriée à partir du dépôt de conversation. Les nœuds multi-services sont utiles, quand il est nécessaire d'appeler de multiples instances parallèles du même service.

Si un service composite a été défini, ceci peut être parfait pour plusieurs utilisateurs, mais certains peuvent avoir besoin de fonctionnalités additionnelles. Pour faire face à ces besoins, EFlow ajoute la notion de création dynamique de services en incluant des nœuds génériques de services. Un utilisateur peut choisir des services additionnels à partir d'un dépôt de services, qui sont alors passés au nœud générique afin de le configurer. Par conséquent, les nœuds multi-services permettent l'activation de plusieurs exemples du même nœud de service, tandis que les nœuds génériques permettent la sélection automatique de différents nœuds de service.

Yang et Papazoglou [91] proposent une approche pour la planification, la définition et l' exécution de la composition en se basant sur le concept du composant Web dans le but de rendre la composition de services flexible et extensible. Ce concept regroupe un ensemble de services préexistant (*de base ou composite*) et présente leurs interfaces et opérations d'une façon cohérente et unifiée sous forme de définitions de classes. Ces composants sont publiés en tant que des services Web et peuvent être utilisés par des applications Web.

Pour découvrir un composant Web, il est nécessaire de définir ses propriétés (*messages*) et opérations dans WSDL. Une fois qu'une classe composant Web est définie, elle peut être réutilisée, spécialisée et étendue. Le même principe s'applique aux activités de la composition de services en représentant le service composite comme un service Web spécial contient un ensemble de constructions et une logique de composition. Cette dernière permet de défi-

nir comment les services composants peuvent être combinés, synchronisés et coordonnés. Un composant Web spécifie la logique de composition en terme des constructions de dépendances de messages et de type de composition. Ces constructions sont privées à un composant Web. Le composant est spécifié dans le langage *SCSL (Service Composition Spécification Langage)* et enregistré dans une bibliothèque commune.

La classe de construction de service Web est une classe abstraite utilisée pour créer des définitions à partir des spécifications WSDL. Cette classe de construction produira une classe composant Web pour un service Web enregistré et défini dans WSDL, c-à-d les messages, l'interface et l'exécution de service. Ce composant Web est spécifié sous deux formes : une définition de classe et des spécifications XML en termes de SCSL. On distingue deux parties dans SCSL : l'interface du service composite spécifiée dans sa partie définition (*defn*) et la construction de la logique de composition spécifiée dans sa partie construction (*construct*). La partie construction de SCSL se compose de : le type de composition, les séries d'activités et des constructions de gestion de messages.

Les auteurs ont aussi développé le langage *SCPL (Service Composition Planning Language)*, pour représenter un service composite, en spécifiant des relations entre les composants Web, ou l'ordre d'exécution (*séquentiel ou concurrent*) de composants Web dans la composition. Les spécifications résultantes produiront plus tard une structure d'exécution de service sous forme d'un graphe d'exécution de composition de services (*SCEG*). Lors de l'exécuté, SCEG invoque les services correspondants et les coordonne. SCSL, SCPL et SCEG travaillent ensemble pour créer des compositions de services.

2.5.4 Composition de services Web en utilisant AI planning

Plusieurs efforts de recherche abordent le problème de la composition de services Web via les techniques de planning développées dans le domaine de l'intelligence artificielle (*AI planning*). En général, un problème de planification peut être décrit par un cinq-tuple [S, S_0, G, A, Γ], où S est l'ensemble de tous les états possibles du domaine, $S_0 \subset S$ indique l'état initial de ce domaine, $G \subset S$ représente l'état final du monde que le système de planification veut atteindre, A est l'ensemble d'actions que le planificateur doit exécuter pour transiter d'un état à un autre et la relation de transition $\Gamma \subseteq S \times A \times S$ définie les pré-conditions et les effets pour l'exécution de chaque action. Dans les services Web, S_0 et G sont l'état initial et l'état final spécifiés par le demandeur du service Web, A est l'ensemble des services Web disponibles.

DAML-S (*appelé aussi OWL-S dans les versions les plus récentes*) est le seul langage de service Web qui fait directement la correspondance avec les planning IA. Le changement de l'état produit par l'exécution du service est spécifié à travers les propriétés, les pré-conditions et les effets définis par DAML-S ServiceProfile. Les pré-conditions sont des conditions logiques qui devraient être satisfaites avant l'existence du service demandé. Les effets sont le résultat de l'exécution correcte d'un service.

Les méthodes de composition de services Web basée sur les planning IA ont été classées en plusieurs catégories : *calcul de situation, techniques de preuves de théorèmes, planning basé sur des règles* et autres.

2.5.4.1 Calcul de situation

McIlraith et al. [60, 61, 62] présentent une technologie d'agents pour la composition automatique de services Web. Cette technologie d'agent est basée sur des procédures génériques réutilisables, la personnalisation des contraintes du demandeur de service et l'exploitation du langage sémantique de service Web. Elle est réalisée en utilisant le langage de premier ordre du calcul de situation (*Situation Calculus*) et une version étendue du langage de programmation d'agent *ConGolog* [38].

ConGolog est un langage de programmation logique à niveau élevé développé à l'université de *Toronto*. Sa primaire utilisation était pour les programme de robot et pour soutenir la planification à niveau élevé des tâches de robot. *ConGolog* est construit sur le calcul de situation (*un langage logique pour raisonner sur les actions et l'échange*). Dans le calcul de situation, le monde est conçu comme un arbre des situations, commençant à une situation initiale S_0 et évoluant à une nouvelle situation par l'exécution d'une action a. Ainsi, une situation S est une histoire d'actions effectuées à partir de S_0.

Les services Web sont représentés comme des actions du calcul de situation , cette technologie fonctionne comme suit : la requête de client est présentée au système exprimé sous forme de procédure générique ConGolog en associant ses contraintes et ses préférences. Cette requête ne peut pas être exécutée directement mais exécutée par un agent en exploitant les ontologies OWL-S des services Web, l'agent instancie automatiquement les spécifications avec les contraintes des services Web dans cette ontologie, qui présente une spécification partielle du comportement du service composite désiré. Chaque situation dans l'arbre présente une configuration instantanée du service à chaque point de ces exécutions.

Le langage sémantique des services Web crée une base de connaissance distribuée *KB (distributed knowledge base)*. Ceci fournit des moyens aux agents pour peupler leurs KBs locales de sorte qu'ils puissent raisonner sur des services Web afin d'effectuer la découverte, l'exécution et la composition automatique de services. La base de connaissance de l'agent fournit un codage logique des pré-conditions et des effets des actions du service Web dans le langage de calcul de situation.

Les auteurs définissent une classe indépendante du domaine des services, *service*, divisée en deux sous-classes, *PrimitiveService* et *ComplexService*. Dans le contexte de Web, un service primitif est un programme individuel exécutable sur le Web, qui n'appel pas un autre service Web. Il n'y a aucune interaction continue entre l'utilisateur et un service primitif. L'utilisateur ou l'agent appel le service et le service renvoie une réponse. En revanche, un service complexe se compose de multiples services primitifs, souvent exigeant une interaction ou conversation entre l'utilisateur et les services, de sorte que l'utilisateur puisse prendre des décisions.

Chaque service Web est conçu comme *PrimitiveAction* ou *ComplexAction*. Les actions primitives alternativement sont conçues en tant qu'actions de changement de l'état du monde, en tant qu'actions de regroupement information qui changent l'état de connaissance de l'agent ou en tant que certaine combinaison des deux. Les actions complexes se composent d'actions primitives ou d'autres actions complexes en utilisant des langages de programmation procédurales et des constructions de langages modélisant les processus métiers tels que *Sequence, Parallel, If-then-Else, While* et d'autre.

2.5.4.2 Techniques de preuves de théorèmes

Rao et al. [71, 72] introduisent une méthode pour la composition automatique de services Web sémantiques en utilisant les preuves de théorèmes de la logique linéaire (LL *Linear Logic theorem proving*). La méthode utilise le langage Daml-S de services Web sémantique pour la présentation externe des services Web, tandis que, intérieurement, les services sont présentés formellement par des axiomes extra logique et des preuves dans LL. Ainsi, le modèle de processus pour un service composite peut être produit directement de la preuve. Les auteurs proposent une architecture de système où le traducteur de Daml-S, les preuves de théorèmes de LL et un raisonneur sémantique peuvent fonctionner ensemble pour accomplir la tâche.

L'idée principale de la méthode est comme suit : en donnant un ensemble de services Web disponibles et un ensemble d'attributs fonctionnels et non fonctionnels, la méthode trouve une

composition de services atomiques, qui répond aux besoins d'utilisateur. Les auteurs utilisent la LL propositionnelle permettant de décrire les attributs fonctionnels et non fonctionnels des services Web. La perfection du fragment de LL s'assure que si une solution composable existe alors elle va être trouvée. Cette approche permet de raisonner avec les types à partir de la spécification de service en utilisant le langage du Web sémantique. Ils introduisent un ensemble de règles de sous typage qui définissent un flux de données valide pour les services composites. Les relations de sous typage entre les classes ou les propriétés sont définies dans l'ontologie de domaine. Les règles de sous typage sont définies en tant qu'implication logiques.

Le processus de composition commence d'abord, par la traduction des descriptions sémantiques des services Web existants (*sous la forme de Daml-S ServiceProfile*) en des axiomes extra logiques de LL et la demande d'utilisateur d'un service composite est spécifiée sous forme de séquences LL à prouver. Le raisonneur sémantique, analyse les relations de sous-type des classes et des propriétés dans l'ontologie de domaine et envoie les relations comme axiomes de LL au démonstrateur de théorème de LL (*LL theorem prover*) à l'aide d'un adaptateur. Ce dernier vérifie si la demande peut être satisfaite par la composition de services atomiques existants. Si la séquence correspondante au service composite demandé a été montré et la preuve est générée, alors une présentation de processus de calcul est extraite à partir de la preuve directement. La dernière étape est la construction des modèles de flux, le processus est traduit à DAML-S ServiceModel ou à BPEL4WS sur la demande. Le raisonneur sémantique est exploité comme composant auxiliaire pour détendre le processus de matching des services tout en choisissant et reliant des services atomiques. En conclusion, le résultat est présenté à l'utilisateur par une interface graphique.

2.5.4.3 Planning basé sur des règles

SWORD [68] est un jeu d'outils qui permet aux développeurs de services de composer rapidement des services Web de base pour réaliser de nouveaux services Web composés. Dans SWORD, un service est représenté par une règle qui exprime que pour une certaine entrée donnée, le service est capable de produire une sortie particulière. Un système expert basé sur des règles est alors utilisé pour déterminer automatiquement si le service composite désiré peut être réalisé en utilisant les services existants. Si oui, cette dérivation est employée pour construire un plan d'exécution. L'idée principale de SWORD est comme suit :

1. Différents services sont définis en termes de leurs entrées et sorties dans un modèle universel (*Entité Relation*). Etant donné les entrées et les sorties du service, une règle

est alors définie (*comme dans les systèmes experts à base de règles*) spécifiant quelles sorties pouvant être obtenues par un service ayant reçu de telles entrées.

2. Quand un développeur souhaite créer et déployer un service composite, il spécifie les entrées et les sorties du service composite dans ce modèle et soumette le à SWORD.
3. SWORD détermine à l'aide d'un moteur de règle si le service composite peut être réalisé en utilisant les services existants. Si oui, SWORD génère un plan de composition.
4. Le développeur peut alors voir le plan produit et s'il est approprié, demande qu'une représentation persistante du plan soit produite. Cette représentation contient la séquence des services qui doivent être invoqués pour obtenir les sorties de service composite à partir de ses entrées. Chaque règle lancée correspond à une invocation de service.

2.5.4.4 Autres méthodes de AI planning

SHOP2 [90] est un système de planification HTN (*Hierarchical Task Network*) indépendant de domaine, qui crée des plans par décomposition de tâche. Il décompose une tâche en des sous tâches primitives plus petites qui peuvent être directement exécutées dans l'ordre planifié par SHOP2. La base de connaissance de SHOP2 contient des opérateurs et des méthodes. Chaque opérateur est une description des besoins permettant d'accomplir une certaine tâche primitive, et chaque méthode indique comment décomposer une tâche composée en des sous tâches partiellement ordonnées. Un problème de planification dans SHOP2 est un triple *(S, T, D)*, où S est l'état initial, T est une liste de tâches de but et D est une description de domaine. SHOP2 retournera un plan *P=(p1 p2...pn)*, une séquence d'opérateurs instanciés qui réaliseront T à partir de S dans D.

Un traducteur de Daml-S au SHOP2, traduit une collection de définitions de processus de Daml-S *'M'* pour un groupe de services Web comme suit :

- Chaque processus atomique de M avec des effets est codé comme opérateur SHOP2 qui simule les effets de changement de monde de service Web en changeant son état local par l'intermédiaire de l'opérateur. Par conséquent, un tel service Web n'est pas exécuté pendant le processus de planification.
- Chaque processus atomique de M avec paramètres de sortie est codé en un opérateur SHOP2 dont la pré-condition inclue un appel à un service de collection d'information. De cette façon, ce dernier est exécuté pendant le processus de planification.
- Chaque définition de processus composé de M est codée en tant qu'une ou plusieurs méthodes SHOP2 indiquant comment décomposer une tâche HTN.

Le processus composé T de M deviendra une tâche pour SHOP2. La décomposition recursive de SHOP2 de cette tâche en des instances d'opérateurs correspondra directement à la décomposition recursive de processus composé en des processus atomiques. Un traducteur de SHOP2 au Daml-S, traduira le plan en format Daml-S qui peut être exécuté directement.

Sirin et al. [78] présentent un prototype semi-automatique qui guide l'utilisateur dans la composition de services Web. Ce prototype a deux composants de base : un compositeur et un moteur de déduction. Le moteur de déduction est un raisonneur OWL développé en *PROLOG*. Il stocke les informations concernant les services connus dans sa base de connaissance KB et a la possibilité de trouver les services compatibles (*matching services*). Le compositeur est l'interface utilisateur qui manipule la communication entre l'opérateur humain et le moteur.

Le compositeur permet à l'utilisateur de créer un workflow de services en présentant les choix disponibles à chaque étape. L'utilisateur commence le processus de composition en choisissant un des services enregistrés. Une requête est envoyée au moteur de déduction pour rechercher les informations sur les entrées "*inputs*" du service, et pour chacune des entrées, une nouvelle requête est exécutée pour obtenir la liste des services possibles qui peuvent fournir les données appropriées pour cette entrée. Le compositeur également présente les différentes classes de service disponibles dans le système et filtre les résultats en se basant sur les contraintes que l'utilisateur peut spécifier sur les attributs. Il présente seulement les services dont les sorties *"outputs"* pourraient être passés au service choisi comme entrées *"inputs"*.

Le matching de deux services est établi en utilisant l'information se trouvant dans les profils de services. Les descriptions des paramètres dans le profil permettent de définir deux types différents de matching entre les services, un matching exact et un matching générique. Un matching exact est défini entre deux paramètres restreints à la même classe OWL. Les services qui assurent une sortie d'un matching exact sont préférés pour être dans la composition et vont être exposés au sommet de la liste. Un matching entre les services dont le paramètre de sortie '*output*' est une sous-classe du paramètre d'entrée '*input*' de l'autre service s'appelle un matching générique. Les matchings génériques sont mis à la fin de la liste puisqu'ils sont moins probablement à être choisis pour la composition.

Le choix des services Web possibles est basé sur les attributs fonctionnels et non fonctionnels. Si plus qu'un matching est trouvé, les noms de services peuvent ne pas être assez mnémoniques afin que l'utilisateur puisse savoir ce qu'ils font. Le système filtre les services en se basant sur les attributs non fonctionnels du service telle que la localisation qui sera utile pour déterminer le service le plus approprié pour la tâche courante. Ces attributs sont présen-

tés à l'utilisateur et les contraintes passées pour ces propriétés sont traduites en des requêtes et envoyées au moteur de déduction. Le moteur applique simplement la nouvelle requête au résultat précédent en excluant les services qui ne satisfont pas les contraintes courantes.

2.5.5 Autres techniques de composition de services Web

Hamadi et Benatallah [41] proposent une algèbre basée sur les réseaux de Pétri, pour modéliser le flux de contrôle, comme constituant nécessaire du processus de composition de services Web fiables. Cette algèbre est assez expressive pour capturer la sémantique des combinaisons complexes de services Web. Ainsi, n'importe quel service exprimé en utilisant les constructions d'algèbre peut être traduit en une représentation de réseau de Pétri.

Les réseaux de Pétri sont des techniques de modélisation du processus bien fondées ayant une sémantique formelle. Ils ont été utilisés pour modéliser et analyser plusieurs types de processus comprenant des protocoles, des systèmes de fabrication et des processus metiers. Un réseau de Pétri est un graphe orienté, connecté et biparti dans lequel chaque nœud est une place ou une transition. Les jetons occupent des places. Une transition est permise s'il y a au moins un jeton dans chaque place relié à une transition. Au lancement de n'importe quelle transition permise un jeton est enlevé de chaque place d'entrée "*input place*" et déposé dans chaque place de sortie "*output place*".

Le comportement de service Web est fondé principalement sur un ensemble d'opérations partiellement ordonnées. Par conséquent, il est directement mappé en un réseau de Pétri. Les opérations sont modélisées par des transitions et l'état du service est modélisé par des places. Les arcs entre les places et les transitions sont employées pour spécifier les relations causales. Un service Web est défini formellement comme tuple S=(NameS, Desc, Loc, URL, CS, SN)

- NameS est le nom du service, utilisé en tant que son identificateur unique ;
- Desc est la description du service fourni. Elle récapitule ce que le service offre ;
- Loc est le serveur hébergeant le service ;
- URL est l'invocation du service Web ;
- CS est l'ensemble de ses services composants. Si le CS=NameS alors S est un service de base. Autrement, S est un service composite ;
- SN=(P, T, W, i, o, ℓ) est le réseau de service modélisant le comportement dynamique du service (*un réseau de Place/Transition marquée*) où :

- P est un ensemble fini de places ;
- T est un ensemble fini de transitions représentant les opérations du service ;
- $W \subseteq (P \times T) \bigcup (T \times P)$ est un ensemble d'arcs orientés (relation de flux) ;
- i est une place d'entrée avec ${}^\bullet i = \{x \in P \bigcup T | (x, i) \in W\} = \Phi$;
- o est une place de sortie avec $o^\bullet = \{x \in P \bigcup T | (o, x) \in W\} = \Phi$;
- $\ell : T \longrightarrow A \bigcup \{\tau\}$ est une fonction où A est un ensemble de noms d'opérations et $\tau \notin A$ et dénote une opération silencieuse.

Les opérations silencieuses sont des transitions qui ne peuvent pas être observées. Elles sont employées pour distinguer le comportement externe et interne du service. Les transitions correspondant à ces aux opérations sont représentées graphiquement par des rectangles noirs.

Le réseau de Pétri, qui représente le comportement d'un service, contient une place d'entrée i (*une place sans arcs entrants*), pour absorber l'information, et une place de sortie o (*une place sans arcs sortants*), pour émettre l'information. La place i est considérée comme marquage initial de service S (*seulement i contient un jeton*). L'exécution de S commence quand un jeton est dans la place i et se termine quand le jeton atteint la place o. Ceci facilitera la définition des opérateurs de composition et d'analyser aussi bien que vérifier certaines propriétés (*par exemple, interblocage, accessibilité, etc.*). La sémantique formelle des opérateurs de composition est exprimée en termes de réseaux de Pétri par un *mapping* direct de chaque opérateur à une construction de réseau de Pétri. Des constructions typiques sont spécifiées pour le flux de contrôle : *sequence, alternative, iteration et arbitrary sequence*. Ainsi que d'autres opérateurs plus élaborés, à savoir *parallel with communication, discriminator, selection et refinement*.

Le réseau de service d'un service Web composite S, obtenu en utilisant les constructions définies ci-dessus, contient une place d'entrée i et une place de sortie o. Le concepteur produit un service Web composite en combinant un ensemble de services Web existants en utilisant les opérateurs d'algèbre. Les opérateurs algébriques satisfont quelques propriétés habituelles, telles que la commutativité, l'associativité et la reflexivitéles. Ces propriétés pouvant être utilisées pour transformer et optimiser les expressions des services Web en garantissant la même sémantique des expressions initiales.

Benatallah dans [11] présente le système *SELF-SERV* (*compoSing Web accessibLe inFormation and buSiness sERVices*), permettant de composer les services Web existants et de les exécuter dans un environnement dynamique pair-à-pair. Le système fournit des outils pour spécifier les services composites par des statecharts, des règles de conversion de données et des politiques de sélection de fournisseur.

SELF-SERV fournit un langage déclarative pour les services composites basé sur les *statecharts* [42] : un formalisme émergeant, largement répandu dans le secteur des systèmes réactifs comme étant un standard de modélisation. Statecharts supportent l'expression des dépendances de flux de contrôle en tant que des branchements, fusionnements, concurrences, etc. Ils fournissent également un modèle implicite pour exprimer les dépendances de flux de données et un modèle d'exécution de service pair à pair, par lequel la responsabilité de coordonner l'exécution d'un service composite est distribuée à travers plusieurs composants logiciels pair appelés *coordonnateurs*. Les coordonnateurs sont attachés à chaque service impliqué. Ils initient, contrôlent et gèrent l'état du service composite auquel ils sont reliés.

SELF-SERV définit trois types de services : services élémentaires, composites et communautés de services. Les communautés de services peuvent être vues comme récipients de services alternatifs. Elles fournissent les descriptions des services désirés sans se rapporter à n'importe quel fournisseur. Les fournisseurs peuvent s'inscrire à n'importe quelle communauté d'intérêt pour offrir leurs services. Ils peuvent quitter ces communautés à n'importe quel moment. À la reception d'une demande d'exécution d'une opération, la communauté choisit un de ses membres courants et lui délègue la demande en se basant sur une *politique de sélection*. Cette dernière peut être basée sur la fiabilité de service Web ou n'importe quel algorithme en se basant sur des paramètres tel que le profil de client.

Afin de s'assurer que tous les services fournissent une interface uniforme, chaque service dans *SELF-SERV* est encapsulé par un composant logiciel accueilli par son fournisseur appelé *wrapper*. Chaque *wrapper* d'un service agit en tant que son point d'entrée en manipulant les demandes d'exécutions des opérations fournies par le service. Les opérations d'un service composite sont exprimées en tant qu'une composition d'opérations d'autres services Web en utilisant les *statecharts*.

Un *statechart* se compose des états et des transitions. Les transitions sont libellées par des règles *ECA (Event Condition Action)*. Quand une transition est lancée, sa partie action est exécutée et son état cible est inscrit. Les états peuvent être de base ou composés. L'état de base correspond à l'exécution d'un service, soit élémentaire ou composite. En conséquence,

chaque état de base est marqué avec une invocation de service. Quand l'état est inscrit, cette invocation est exécutée. L'état est normalement quitté quand l'exécution induite par cette invocation est accomplie. Si l'état a des transitions sortantes marquées avec des événements, une occurrence de n'importe lequel de ces événements cause l'état d'être quitté et l'exécution produite est annulée. L'état composé contient un ou plusieurs *statecharts*. On distingue deux types d'états composés : *OR* et *AND*. Un état *OR* contient un *statechart* simple, alors qu'un état *AND* contient plusieurs *statecharts* qui sont prévus pour être exécutés en concurrence. Quand un état composé est inscrit, son état initial devient actif. L'exécution d'un état composé est considérée être accomplie quand elle atteint (tous) l'état(s) final(aux).

Chaque état "*ST*" apparaissant dans la spécification de service composite est représenté par un coordonnateur d'état. Ce dernier, détermine (*i*) quand un état dans un *statechart* est inscrit ? (*ii*) ce qui devrait être fait après que l'état soit inscrit ?, (*iii*) quand l'état est quitté ? et (*iν*) que devrait être fait après que l'état soit quitté ?. La connaissance requise par un coordonnateur, afin de répondre à ces questions au temps d'exécution, est statiquement extraite à partir du *statechart* décrivant les opérations de service composite. Le comportement d'un coordonnateur d'état peut donc être capturé par deux ensembles : (*i*) un ensemble de pré-conditions tel que l'état est inscrit quand une de ces pré-conditions est vérifiée et (*ii*) un ensemble d'actions post-traitement (*postprocessing*) indiquant quels coordonnateurs doivent être notifiés quand un état est quitté. À la fin d'exécution d'un service, le coordonnateur de cet état évalue la partie condition de chaque entrée apparaissant dans sa table post-traitement. Pour chaque entrée dont la condition est évaluée à vraie, il envoie un message de notification au coordonnateur de l'état référencé dans cette entrée.

L'exécution de service composite est orchestré par des échanges de messages *"control-flow notifications"* pair-à-pair entre les *coordonnateurs* des états et par des échanges de messages *"service invocations/completions"* entre les *coordonnateurs* et les *wrappers*. Pour une transition menant à un état *ST* et marquée avec une condition *C*, une action *"[C]/notify(ST)"* est ajoutée dans la table post-traitement. La table des actions post-traitement d'un état *"ST"* est l'union des actions associées aux transitions sortantes de *"ST"*. La table pré-conditions d'un état est produite par la détermination, pour chacune des transitions entrantes de l'état, quelles sont les conditions qui devraient être vérifiées pour que cette transition soit prise.

Hashemian et Mavaddat [43] présentent une approche pour la découverte et composition des services Web basée sur un outil de modélisation spécifique appelé *les automates d'interface* (*IA*) [29]. Ils ont effectué un mapping, à partir des description OWL-S des services Web, vers des automates d'interface.

L'automate d'interface est un modèle à base d'état, semblable aux automates d'états finis, pour représenter le comportement des composants logiciels. L'*IA* est équipé d'un concept de composition et d'un formalisme par lesquels la composition de deux composants peut être calculée. Il expose les entrées et les sorties d'un composant et l'ordre temporel des actions qu'il effectue. Cette représentation exprime que le service Web produit la sortie $\boldsymbol{o}$ si et seulement si l'entrée $\boldsymbol{i}$ requise est donnée. Cette dépendance est formellement représentée par $i \longrightarrow o$.

Les auteurs utilisent un dépôt local contenant des informations sur l'ensemble des services Web existants. Basé sur les propriétés exposées par l'automate d'interface de chaque service Web ws, trois informations sont stockées dans le dépôt : son ensemble d'entrées (A^I_{ws}), son ensemble de sorties (A^O_{ws}) et des informations de dépendances entre les différentes entrées et sorties du service Web (K_{ws}). Par conséquent, chaque ws est formellement défini comme $ws = (A^I_{ws}, A^O_{ws}, K_{ws})$. Noter que K_{ws} est un ensemble d'ensembles de dépendances. Puisque chaque service Web peut exécuté différents scénarios et avoir différents chemins d'exécution, chaque ensemble dans K_{ws} contient les dépendances d'entrée-sortie dans un scénario simple ou chemin d'exécution. Pour chaque service Web, chaque entrée et sortie doit apparaître dans au moins une relation de dépendance d'entrée-sortie dans K_{ws} .

Le problème est converti en un problème général de graphe. Ce graphe est défini comme suit : $G = (V, E)$. L'ensemble V de nœuds représente les entrées et les sorties qui apparaissent dans au moins un A^I_{ws} ou A^O_{ws}. Un arc est orienté du nœud v_x au nœud v_y ($v_x, v_y \in V$), si et seulement s'il y a une dépendance $v_x \longrightarrow v_y$ dans au moins un ensemble de dépendance dans au moins une k_{ws_i}. Chaque arc dans E est un ensemble contenant tous les services Web ayant cette dépendance dans une de leurs ensembles de dépendances. Le problème, en termes de graphe, est comme suit : en donnant un graphe de dépendance $G = (V, E)$ et une requête $ws_q = (A^I_{ws_q}, A^O_{ws_q}, K_{ws_q})$, y a-t-il un sous-graphe qui couvre tous les arcs $v_i \longrightarrow v_o$ dans k_{ws_q} ?

Pour vérifier si toutes les dépendances sont satisfaites par le graphe de dépendance, l'algorithme BFS (*Breadth First Search*) est exécuté sur chaque dépendance $i \longrightarrow o$, et renvoie l'ensemble des nœuds accessibles à partir du nœud i. Si un chemin de i à o est retourné pour chaque dépendance spécifiée, le problème a une réponse. Pour chaque dépendance, l'algorithme BFS renvoie le plus court chemin si plus q'un chemin existe.

Le service demandé est construit par la composition de services Web impliqués dans les chemins retournés. Les ensembles de dépendances dans k_{ws_q} sont vérifiés une à une. Pour un ensemble particulier de dépendances, s'il y a seulement une dépendance et le chemin retourné est de longueur un, ceci signifie qu'il y a au moins un service Web dans le dépôt satisfaisant

cette dépendance et de ce fait on n'aura pas besoin d'une composition. Autrement, si la longueur du chemin est plus de un, le IA est utilisé pour composer les services Web apparaissant comme des étiquettes sur les arcs pour satisfaire la dépendance. S'il y a plus d'une dépendance dans l'ensemble, pour chacune nous faisons la composition ci-dessus, s'il y a lieu. les compositions alors résultantes de chaque dépendance sont exécutées séquentiellement de telle manière que l'exécution ne viole aucune dépendance dans l'ensemble. Pour les dépendances ayant la même action de leur côté gauche, leurs compostions résultantes sont exécutées en parallèle. Les compositions résultantes de chaque ensemble $k^i_{ws_q}$ dans k_{ws_q} sont composées conditionnellement pour représenter différents chemins d'exécution selon la demande.

Benbernou et al. [12] proposent un modèle basé sur un graphe orienté pour modéliser la composition de services Web construite sur l'algorithme *Bcov*, initialement défini pour la découverte automatique de services Web. *Bcov* est une approche repose sur une propriété classique des hypergraphes [31], qui en fait un algorithme itératif qui, à chaque itération, génère un certain nombre de candidats pour ne conserver que les plus petits (*au sens de l'inclusion ensembliste*) à la fin de l'itération. Cette phase de génération de candidats peut être optimisée par l'ajout de d'autres propriétés [74].

L'idée principale de *Bcov* est : en donnant une requête Q et un ensemble de services S, le problème de la découverte consiste à trouver un sous-ensemble de S appelé *"couverture"* de Q contient autant que possible d'information commune avec Q et le moins possible d'information supplémentaire par rapport à Q. Les services et la requête sont décrits sous forme d'une conjonction d'atomes (*dans le formalisme de logique de description*). Les services sont découverts par la construction d'un hypergraphe $(\Sigma; \Gamma)$ où Σ est un ensemble de sommets correspondants aux descriptions de services et chaque atome dans la requête devient une arrête de Γ dans l'hypergraphe. L'algorithme calcule les transversaux minimaux avec les meilleurs coût dans l'hypergraphe en recherchant une meilleure couverture de services réduisant au minimum la conjonction des services qui matche le maximum des atomes dans la requête (*pour plus de détails voir [55, 65]*).

Ce cadre permet de composer les services en fournissant deux approches. La première utilise tous les transversaux minimaux fournis par algorithme *Bcov* et essaye de trouver le meilleur en utilisant un rapport de couverture et un graphe local orienté. Le raisonnement de base de cette approche est donné par les étapes suivantes :

- Pour chaque transversal minimal fourni par algorithme *Bcov*, un graphe orienté est construit où chaque nœud correspond à un service et les arcs sont les rapports entre les entrées et les sorties des services (*du transversal*).
- Après avoir découvert les services atomiques, qui sont des nœuds isolés dans le graphe, les services du transversal minimal sont ordonnés en respectant les atomes de la requête.
- Une fois les services atomiques sont enlevés du transversal, en essaye de trouver dans le graphe orienté, un chemin (*composition linéaire*) ou un sous graphe (*composition non linéaire*) entre le premier et le dernier service (*du l'ensemble ordonné*) en tenant compte de tous les services intermédiaires.
- Calculer le coût de la récupération en considérant seulement les services composés en respectant la requête.

Cependant, la seconde emploie seulement le transversal minimal avec le meilleur coût (*sans utilisé tous les transversaux minimaux comme dans l'algorithme précédent*) et essaye de trouver le plus court chemin entre tous les services ordonnés (*en respectant la requête*) dans un graphe global. La composition dans ce cas est basée sur l'algorithme *Dijkstra* [79], où pour chaque deux services consécutifs dans le meilleur transversal minimal fourni par algorithme *Bcov* (*ordonné et sans considéré les services atomiques*), le plus court chemin est calculé en se basant sur un graphe orienté global maintenu par le fournisseur des services Web. À la fin du processus, nous obtenons un sous graphe qui est considéré comme la solution de la requête.

2.6 Conclusion sur les travaux de composition de services Web

La composition de services Web commence progressivement à être traitée dans la littérature. Cette partie a visé de donner une vue sur les progrès récents dans les techniques de composition automatique de services Web. Nous avons mis en relief plusieurs travaux distincts ; dont l'objet est la composition ; sont les suivants : les recherches de workflow, planning IA et d'autres. Afin de mettre en œuvre ces dimensions de composition, les auteurs utilisent principalement différents outils de modélisation de services. Dans la section 2.5, la composition automatique de services est traitée plus en détail. BPEL4WS et WSFL ont été présentés comme modèles manuels de composition. Contrairement à la description manuelle de la composition de services, beaucoup d'efforts ont été conduits dans la direction de la composition automatique de services. Le tableau 2.1 résume les caractéristiques principales des différentes approches de composition de services Web discutées précédemment.

	Modélisation de QoS	Genre de composition	Type d' orchestration	Stratégie de composition	Sélection de conversation	Support de transaction
[20, 21]	Non	Séquentielle Concurrente	Médiateur	Automatique	Dynamique	Non
[91]	Non	Séquentielle Concurrente	Médiateur	Manuelle	Non	Non
[60, 61, 62]	Non	Séquentielle Concurrente	Médiateur	Automatique	Non	Non
[90]	Non	Séquentielle Concurrente	-	Automatique	Non	Non
[68]	Non	Séquentielle Concurrente	-	Automatique	Non	Non
[78]	Non	Séquentielle Concurrente	Médiateur	Semi-automatique	Non	Non
[71, 72]	Non	Séquentielle Concurrente	Médiateur	Automatique	Non	Non
[11]	Non	Séquentielle Concurrente	Peer to peer	Automatique	Non	Non
[41]	Non	Séquentielle Concurrente	Médiateur	Automatique	Non	Non
[43]	Non	Séquentielle Concurrente	Médiateur	Automatique	Non	Non
[12]	Non	Séquentielle Concurrente	-	Automatique	Non	Non

TAB. 2.1 – Modèles de composition de services Web.

Les méthodes workflow sont la plupart du temps employés dans la situation où la requête a déjà défini le modèle de processus, mais un programme automatique est exigé pour trouver les services atomiques pour satisfaire les besoins. Les méthodes de planification de AI sont employées quand le demandeur n'a aucun modèle de processus mais a un ensemble de contraintes et de préférences. Par conséquent, le modèle de processus peut être produit automatiquement par le programme.

L'idée de la composition semi-automatique de services de ***Serin et al.*** est intéressante, parce que il est très difficile de capturer le comportement dans le détail suffisant et composer les services d'une manière complètement automatique. Bien que la méthode proposée soit simple, elle indique la tendance que le planificateur automatique et l'être humain peuvent travailler ensemble pour générer le service composite.

À la différence des autres méthodes qui utilisent les attributs non fonctionnels seulement pour filtrer le plan généré, ***Rao et al.*** considèrent les attributs non fonctionnels directement dans le processus de preuve de théorème. Les attributs fonctionnels et non fonctionnels du service sont traduits en des propositions dans des axiomes logiques. La distinction entre les attributs fonctionnels et non fonctionnels est possible par les règles de déduction de LL.

Parmi les travaux étudiés, *SELF-SERV* est la seule approche qui propose une architecture pair à pair pour la composition de services. Cependant, l'approche médiateur et par conséquent l'architecture centralisée peut résulter un goulot en invoquant des services.

Les deux approches proposées dans [12] posent deux problèmes. Pour la première, il se pose le problème où ne pouvons pas trouver un chemin de service S_i au S_j. Ce problème surgit du fait que le graphe est local, où la composition est limitée seulement aux services du transversal. Pour remédier au problème, une deuxième approche basée sur un graphe orienté global maintenu par le fournisseur des services Web est proposée. Cependant, d'autres services pouvant être introduits dans la composition autres que les services fournis par algorithme de *Bcov*. À la fin du calcul, nous obtenons a sous-graphe global satisfaisant la requête en composant des services qui ne sont pas fournis par *Bcov*.

Bien que les différentes méthodes fournissent des niveaux différents d'automatisation de la composition de services, nous ne pouvons pas dire que l'automatisation la plus élevée est la meilleure. Puisque l'environnement de service Web est fortement complexe et il n'est pas faisable de produire le tous d'une manière automatique.

Le lecteur notera également qu'il y a encore beaucoup d'effort de développement a faire jusqu'à une plate-forme de composition de services qui fournira tous les caractéristiques énumérés ci-dessous. Nous pensons que l'état actuel des efforts de développement peut être vu comme une première tentative vers la création des plates-formes de composition de services, à cause du manque des caractéristiques additionnelles au sujet d'extension du QoS dans WSDL. La plus part des travaux réalisés n'ont pas su intégrer la QoS. Les résultats de recherches dans le secteur de composition de services montrent que la technologie courante de service Web est limitée puisqu'elle est purement basée sur des standards, tels que SOAP, WSDL ou UDDI.

À notre avis, la plate-forme *EFlow*, est très prometteuse ; même si elle manque d'une description sémantique, et ainsi, il serait également intéressant à suivre le progrès qui sera accompli dans le secteur des services Web sémantiques, à moins que la composition de services fonctionne sans description sémantique de service.

La section suivante établit une synthèse de l'état de l'art.

2.7 Synthèse de l'état de l'art

L'utilisation des services Web est appelée à changer radicalement les procédés de communication intra et inter organisations du fait de la répartition et des traitement des services via le Web. Les services Web constituent un moyen pour les entreprises de s'ouvrir aux autres organisations d'une manière standardisée. Les échanges d'information évoluent vers un plus grand dynamisme. Le développement de services Web devant être standardisé afin de faciliter leur interopérabilité, des consortiums travaillent sur les outils à employer. À ce jour, les plus importants concernant ce concept sont au nombre de deux : le W3C et OASIS. Ces regroupements publient régulièrement de nouvelles normalisations permettant de faire évoluer rapidement les services Web.

Afin que ce concept standardisé soit applicable, les services Web possèdent une architecture bien définie. Cette dernière est composée des couches standards rendant possible l'interopérabilité, objectif premier de l'intégration des services Web. Des technologies basées sur XML sont à ce jour utilisées dans l'intention de permettre un protocole de publication et d'accès universel (*tels que WSDL et DAML-S*). Par la suite, des technologies de référencement des services Web (*tels UDDI et ebXML*) deviennent alors le premier point de rencontre des organisations. Ces répertoires de services Web peuvent être aussi définis en interne, afin d'organiser les services appartenant à la même entreprise. Les technologies étudiées dans notre travail sont celles jugées par les consortiums comme étant les plus largement utilisées. Afin de facilités la mise en œuvre des différents composants de service Web, il est préférable de disposer d'outils de développement. En effet, la tâche est simplifiée par ces logiciels.

Les services Web ont connu un véritable engouement ces dernières années. En plus des défis scientifiques tels que la découverte et la composition de services, des impératifs tels que la sécurité lors de l'accès aux services sont requis afin de permettre l'adoption des services Web par le grand public et le monde de l'industrie. De nombreux travaux émergent alors sur ces thèmes, notamment sur la composition de services Web. Le manque de mécanismes de sécurité constitue le principal frein à l'émergence de cette nouvelle technologie d'intégration. Différents standards complémentaires sont en cours d'élaboration pour combler cette lacune et sécuriser les échanges.

Dans la troisième partie de notre travail concernant la proposition, nous tentons d'établir une représentation graphique des services Web disponibles sur le Web afin de l'utiliser dans la composition de services Web. Ainsi, nous proposons un algorithme pour soutenir le processus de leur composition automatique.

Partie 3

Un algorithme pour la composition automatique de services Web

3.1 Introduction

L'étude des travaux existants a montré qu'à ce jour différentes dimensions de composition de services Web font l'objet d'étude. Cependant, nous estimons que l'objet d'utilisation d'une structure représentant les services Web pouvant être composés n'est pas assez considéré dans le processus de composition. En effet non seulement peu de travaux étudiés utilisent un modèle de graphe, mais aussi ne fournissent pas une structure globale représentant l'ensemble des services disponibles. C'est pourquoi notre approche est proposée sous la forme d'une description d'un modèle dans le contexte de la composition de services Web. Dans cette partie, nous soulignons tout d'abord que notre travail se base sur une structure générale représentant l'ensemble des services Web disponibles, modélisés sous forme de graphe orienté. Dans un second temps, nous précisons que la base de notre travail est les services Web découverts en se basant sur la requête de client et leur composition afin de satisfaire les besoins de client. Ainsi, et pour soutenir le processus de composition automatique de services Web répondant à la requête, nous proposons un algorithme qui consiste à découvrir une combinaison de services couvrant la requête de client et à vérifier l'existence d'une composition de services Web. La sortie de cet algorithme retourne le sous graphe, composé de l'ensembles de services, satisfaisant la requête de client. Ceci permet de définir et à partir des chemins retournés l'ordre d'exécution des services impliqués dans la composition. Nous illustrons à partir des exemples, l'utilisation des services Web dans une composition reposant sur notre approche.

Présentation

3.2 Principe

L'objectif de l'algorithme est de donner une exécution cohérente d'une requête d'un client qui ne peut pas être satisfaite par un seul service Web, mais possible par un ensemble de services Web disponibles découverts en se basant sur la requête de client et le graphe orienté représentant les services Web disponibles. L'idée derrière cet algorithme est d'exploiter les algorithmes de graphes pour découvrir une combinaison de services satisfaisant la requête de client, ainsi les composer en définissant l'ordre d'exécution. Pour montrer l'utilité et l'applicabilité de notre proposition, nous avons besoin de quelques définitions préalables.

3.3 Définitions de base

Définition 1. ***(Service Web)*** *un service Web S est défini comme quatre-tuple (I, O, OP, QoS) où* $I = \{I_k/k > 0\}$ *un ensemble d'entrées de service,* $O = \{O_l/l > 0\}$ *l'ensemble de sorties de service,* $OP = \{OP_j/j > 0\}$ *l'ensemble d'opérations fournie par le service et QoS la qualité de service garantie par ce dernier. Chaque* OP_j *est définie formellement sous forme d'un quatre-tuple* $OP_j = \{in, out,$ précond, postcond$\}$ *avec : in* $\in I$*, out* $\in O$*,* précond et postcond *sont la pré-condition et post-condition nécessaires pour l'exécution d'une opération.*

Définition 2. ***(Classe)*** *une classe est définie formellement sous forme d'un ensemble de tuples* $Cn = \{F_k/k > 0\}$ *un ensemble d'opérations :* $F_k = (OP_j, S_i)$ *où* OP_j *est une opération et* S_i *est le nom de service fournissant cette dernière.*

Définition 3. ***(Graphe orienté de services Web)*** *soit G un graphe orienté défini par* (V, E) *où, V est un ensemble de paramètres d'entrées et sorties des services Web définis et E un ensemble d'arcs orientés, de tel arc est dénoté par* $(V_1 \rightarrow V_2)$ *signifie qu'il existe au moins un service* S_i *peut fournir une opération ayant* V_1 *comme entrée et* V_2 *comme sortie.*

Définition 4. ***(Composition sémantique)*** *deux services* S_i *et* S_j *pouvant être composés sémantiquement ensemble et dénoté par* $S_i \triangleright S_j$ *ssi* $\exists I_k \in I_{S_i}, \exists O_l \in$

$O_{S_j}|I_k \subseteq O_l$ où I_{S_i} et O_{S_j} sont les ensembles d'entrées et sorties de service respectivement et $\subseteq$ dénote un symbole de matching sémantique.

La définition signifie que deux services peuvent se composer si et seulement si les entrées (*pas nécessairement toutes*) d'un service match sémantiquement les sorties de l'autre.

Notre contribution s'articule autour des points suivants :

- Comment établir une structure générale représentant l'ensemble des services Web disponibles.
- Comment rechercher un service Web répondant à un ensemble de critères.
- Comment composer l'ensemble des services Web découverts en définissant l'ordre de leurs exécutions pour répondre à la requête de client.

3.4 Représentation formelle des services Web comme un graphe orienté

Le service Web est essentiellement une collection d'opérations décrites par une interface et fournies à un client. Le minimum d'information exigée pour décrire les fonctionnalités de service est sa signature, composée des noms d'opérations et ses paramètres d'entrée et de sortie. Ces informations peuvent être extraites à partir des spécifications OWL-S des services Web. À partir de $ServiceProfile$ on extrait les paramètres d'entrée et sortie (*inputs/outputs*) de chaque service ainsi que la qualité de service garantie par ce dernier. Les différentes opérations fournies par le service, ses paramètres *in/out* et les *pré-conditions/post-conditions* nécessaire pour leurs exécutions sont fournies par le $ServiceModel$. Le $ServiceGrouding$ sert de support pour la réalisation des liens concret et l'invocation des différents services retournés par l'algorithme de composition proposé.

Pour répondre au problème de composition en réutilisant les services existant (*les différentes opérations fournies par ces derniers*) on propose de traiter les opérations de services comme des concepts partagés et regroupés dans des classes au lieu de les traiter en terme de fonctionnalités particulières. L'idée est de pouvoir représenter les différentes signatures d'opérations comme des signatures abstraites simples requises pour une fonctionnalité donnée. Ainsi, les opérations abstraites résultantes vont être regroupées, basant sur leurs fonctionnalités, en un ensemble de classes C_n de tel sort que chaque classe C_n contient toutes les opérations qui partagent les mêmes paramètres d'entrées/sorties (*fournissent les mêmes fonctionnalités*)

indépendamment de service ou des pré-conditions et post-conditions. Ces classes servent de base pour le processus de composition, qui sera traité en termes des signatures abstraites.

Afin de concevoir un mécanisme par lequel nous pouvons répondre au problème de composition, nous stockons ces informations concernant les services Web sous forme d'une construction de graphe. Ce graphe est défini comme suit :

Un graphe orienté $G = (V, E)$ contient l'information sur les services Web existants dans l'annuaire. L'ensemble V des noeuds représente les *inputs*/*outputs* appartenant au moins à I_{s_i} ou O_{s_i}. Un arc est orienté du noeud V_x vers le noeud V_y dans le graphe ($V_x, V_y \in V$), si et seulement s'il existe au moins une $OP_j \in C_n$ ayant V_x comme paramètre *in* et V_y comme paramètre *out*. Chaque arc dans E est une classe contenant tous les opérations ayant le même paramètre *input*/*output*.

En se basant sur ce graphe, on pourra construire un algorithme qui sert à découvrir l'ensemble des opérations définies par la requête (*par conséquent les services les fournissant*), ainsi de définir la composition automatique de l'ensemble de services Web retourné qui pourra satisfaire la requête présentée et ce en respectant les contraintes tels que fournies par le client, en explorant le sous graphe résultant.

3.5 Algorithme de composition automatique de services Web

Nous proposons un algorithme qui consiste à vérifier l'existence d'une composition de services Web et à construire le programme de composition correspondant suivant une méthodologie proposée. L'algorithme retourne une réponse positive s'il existe une composition qui satisfait la requête du client tout en respectant ses préférences, ainsi que l'ensemble des services Web impliqués dans cette composition, négative sinon.

D'abord nous décrivons une méthodologie de composition de services composée de trois étapes, à savoir un processus de vérification de composition, un processus abstrait de composition et un processus de liaison.

3.5.1 Une méthodologie de composition de services

Notre objectif est de présenter une méthodologie conceptuelle supportant la composition de services existants. Elle consiste à appliquer les processus de base suivants : un processus de vérification de la faisabilité de la composition de services en explorant le graphe de services Web et la requête de client, un processus abstrait pour la construction de la composition par la réutilisation des services existants et un processus de liaison pour convertir la description de la composition résultante dans un langage de composition de service Web, pour le déploiement et l'invocation. Ces processus sont indépendants de l'exécution concrète.

Nous essayons de fournir une façon efficace pour le développement plus rapide des applications Web par la composition de services, où la réutilisation de ceux ci peut être exploitée, et en essayant aussi d'exploiter les langages de composition de services Web tel que BPEL4WS. Ici, nous nous concentrons sur le processus de vérification et celui de la description du processus abstrait de composition, une étape critique afin de traiter la composition de services.

Pour soulager le processus de composition, cet algorithme soumis à un ensemble de critères pris dans l'ordre de priorité comme suit :

- On suppose que dans la phase de vérification de composition que le chemin le plus court et le plus convenable pour minimiser les services qui vont être impliqués dans cette composition. Dans cette phase les pré-conditions et post-conditions ne sont pas prisent en compte.
- Dans le deuxième processus, les pré-conditions et post-conditions interviennent pour choisir une opération OP_j à partir des classes impliquées sur les chemins retournés.
- S'il existe plusieurs opérations OP_j satisfaisant la contrainte pré-condition/post-condition, on choisit la plus convenable en terme de QoS (*la qualité de service garantie par le service fournissant cette opération*).
- On suppose aussi qu'il existe au moins une opération $OP_j \in C_n$ satisfaisant la contrainte *pré-condition/post-condition* lors de choix.

Ainsi, nous résolvons le problème de composition en trois étapes :

1. Vérifier la faisabilité de la composition en explorant les algorithmes de graphe à savoir un algorithme de plus court chemin.
2. Construire un processus abstrait de composition en sélectionnant les différentes opérations (*découvrir les services Web les fournissant*) qui peuvent potentiellement participer

dans la composition en se basant sur les chemins retournés par l'étape une.

3. Convertir ce processus dans un langage de composition de services Web pour l'invocation et l'exécution.

Le reste de cette section décrit chaque processus en présentant son but.

3.5.1.1 Processus de vérification de composition

La première étape dans la résolution de problème est de vérifier d'abord l'existence d'une telle composition avant de lancer le processus de découverte des différents services pouvant être composés pour répondre à une requête donnée (*construire le programme de composition abstrait*). Ce processus est chargé de vérifier la faisabilité de la composition en se basant sur le graphe générale de services Web et la requête fournie par le client. Dans cette phase les pré-conditions et post-conditions n'ont pas d'effet.

Nous supposons que la requête est donnée par : $S_Q = (OP_{S_Q})$ où OP_{S_Q} l'ensemble d'opérations de la requête. Chaque $OP_k \in OP_{S_Q}$ est définie formellement sous forme d'un quatre-tuple $OP_k = \{in,\ out,\ précond,\ postcond\}$ avec : *in* le paramètre d'entrée, *out* le paramètre de sortie et *précond et postcond* sont les pré-conditions et post-conditions nécessaire pour l'exécution de l'opération.

Le problème, en termes de graphe orienté de services Web, est comme suit : On donnons le graphe orienté de services Web $G = (V, E)$ et la requête de client $S_Q = (OP_{S_Q})$, Y a t-il un sous graphe de G qui couvre tous les opérations $OP_k \in OP_{S_Q}$?

Pour ce faire, pour chaque opération $OP_k \in OP_{S_Q}$ on cherche à trouver un chemin sur le graphe G, qui peut satisfaire OP_k, partant de nœud *in* (*paramètre d'entrée de* OP_k) au nœud *out* (*paramètre de sortie de* OP_k). On doit retourner pour chaque OP_k un chemin pour que la composition soit possible.

Une manière optimale pour vérifier si toutes les opérations sont satisfaites par le graphe orienté de services Web est d'exécuter un algorithme de plus court chemin (*pour toute opération* OP_k) sur le nœud *in*, le paramètre d'entrée de OP_k, vers le nœud *out*, son paramètre de sortie dont le but et de minimiser le nombre de services intervenant dans une composition. Ainsi, pour soulager le processus.

Le problème de plus court chemin est défini comme suit :

Définition 5. ***(Problème de plus court chemin)*** *Soit $G = (V, E)$ un graphe et $e \in E$, on associé à chaque arc un poids $l(e) >= 0$ (dans notre cas nous assumons que tous les arcs ont un poids égal 1 appelé longueur du l'arc u). Trouver un chemin élémentaire de V_i à V_j tel que la longueur $l(\mu) = \sum_{e \in \mu} l(e)$ soit minimale.*

Il existe beaucoup d'algorithmes traitant le problème du plus court chemin, nous utilisons celui développé par *Dijkstra* (*pour plus de détails sur l'algorithme voir* [79]), nous le dénotons par *Dijkstra* (G, s), qui donnant un graphe G trouve tous les plus courts chemins entre le sommet s et tout autre sommet dans le graphe. Et *pluscourtchemin*(G, s, t) utilise *Dijkstra* pour trouver le plus court chemin de s à t.

Pour chaque opération OP_k, cet algorithme renvoie l'ensemble de nœuds accessible du nœud *in*. Si un chemin du nœud *in* au nœud *out* existe dans le graphe, nous pouvons arrêter l'algorithme une fois que nous notons que le nœud *out* de OP_k est accessible du nœud *in*. Autrement, l'algorithme termine et renvoie un ensemble ne comprenant pas le nœud *out*, qui signifie qu'il n'y a aucun chemin dans le graphe qui peut répondre à cette opération. Pour chaque opération, l'algorithme renvoie le plus court chemin si plus qu'un chemin existe.

En maintenant les nœuds visités, l'algorithme peut renvoyer le chemin répondant à l'opération OP_k si un tel chemin existe. En outre, en connaissant le chemin nous pouvons extraire l'ensemble de sous graphes d'où chaque arc est choisi. Si un chemin est retourné pour chaque opération $OP_k \in OP_{S_Q}$, le problème a une réponse.

Au début, nous ignorerons les pré-conditions, les post-conditions et la qualité de service (*coût, temps réponse, etc.*) bien que nous les tenons en compte dans le processus suivant car elles jouant un rôle important dans la sélection de l'opération la plus appropriée pour l'exécution parmi les différentes opérations candidates.

3.5.1.2 Processus de composition

Ce processus sert à créer la composition de services par la réutilisation de ceux existants en se basant sur le sous graphe fourni par le premier processus et également la requête S_Q en récupérant les noms des classes impliquées sur les chemins retournés.

Après avoir trouver quels chemins dans le graphe orienté de services Web sont d'intérêt, nous devons trouver une composition satisfaisant la requête donnée. Dans cette phase, la pré-condition, la post-condition de chaque opération ainsi que la qualité de service interviennent ; dans l'ordre de priorité prédéfini ; pour sélectionner les opérations et les services les plus appropriés, en les préparant pour la composition, ce en respectant cette fois ci les contraintes de client.

Nous traitons d'abord, l'ensemble des opérations dans OP_{S_Q} une par une. Pour une opération particulière OP_k, si un chemin de longueur un est retourné, ceci signifie qu'il y a au moins un service Web qui peut satisfaire cette opération et nous n'avons besoin d'aucune composition. Autrement, si un chemin de longueur supérieure à un est retourné, nous avons besoin de composer les services Web apparaissant dans le chemin ensemble, pour satisfaire l'opération.

Pour ce faire, on procède à la récupération des noms des classes impliquées sur les arcs composant le chemin retourné dont le but est de retourner le(s) couple(s) (*nom opération, nom de service*) satisfaisant l'opération $OP_k \in OP_{S_Q}$ en respectant les préférences de client.

1. ***Si le chemin retourné est de langueur un***

   - Il existe au moins une opération $OP_j \in C_n$ (*la classe impliquée sur le chemin*) satisfaisant l'opération $OP_k \in OP_{S_Q}$ sans composition.
   - Choisir une opération $OP_j \in C_n$ en respectant la pré-condition et post-condition de $OP_k \in S_Q$ tel que : $précond(OP_j) = précond(OP_k)$ et $postcond(OP_j) = postcond(OP_k)$
   - S'il existe plusieurs opérations ayant la même pré-condition et post-condition, on choisit l'opération appartenant au service garantissant la meilleure qualité de service (QoS).
   - Retourner le couple (*nom opération, nom service*) choisi à partir de la classe impliquée sur le chemin retourné pour satisfaire cette opération OP_k.

2. ***Si le chemin est de longueur supérieure de deux***

   - Il n'existe pas une opération $OP_j \in C_n$ satisfaisant $OP_k \in OP_{S_Q}$ mais un ensemble d'opérations appartenant aux différentes classes impliquées sur le chemin retourné pour cette OP_k.
   - Pour satisfaire la pré-condition et post-condition de OP_k On procède comme suit :

- Choisir une opération $OP_j \in C_p$ la première classe sur le chemin tel que : $précond(OP_j) = précond(OP_k)$.
- Choisir une opération $OP_j \in C_d$ la dernière classe sur le chemin tel que : $postcond(OP_j) = postcond(OP_k)$.
- Pour les classes intermédiaires C_n sur le chemin (*entre C_p la première classe et C_d la dernière classe*), les opérations OP_j vont être choisies à partir de chacune des C_n en respectant la contrainte suivante : $précond(OP_j) \in C_n = postcond(OP_m) \in C_{précéd}$ la classe qui précède C_n sur le chemin à partir de la quelle on a déjà choisi OP_m.
- S'il existe plusieurs opérations OP_j candidates ayant les mêmes pré-conditions et post-conditions, on choisit l'opération appartenant au service garantissant la meilleure qualité de service (QoS).
- Retourner les couples (*nom opération, nom service*) choisis à partir des classes impliquées sur le chemin retourné pour satisfaire cette opération OP_k.

3.5.1.3 Processus de liaison

Une fois qu'une composition est créée après les deux processus mentionnés, nous employons le processus de liaison pour traduire la composition résultante en des descriptions exécutables pour l'invocation en utilisant un langage de composition de services Web tel que BPEL4WS pour permettre aux utilisateurs et aux programmes externes de l'exécuter.

Il consiste à l'invocation des différents services retournés pour chaque $OP_k \in OP_{S_Q}$. Dans cette phase, on utilise OWL-S ServiceGrouding des services choisis pour établir les liens (*binding*) avec les services et pour récupérer les spécifications techniques des opérations choisies.

Tous ces processus combinés fournissent une méthodologie et une manière efficace pour développer des applications rapidement par la composition et la réutilisation des services existants et même adaptées aux besoins du client.

3.5.2 Algorithme de composition

La méthodologie présentée dans la section précédente est récapitulée sous forme d'un algorithme donné par l'`algorithme` 1.

Algorithme 1 DCoServ : Découvert et Composition de Services Web

Entrée(s) : Un graphe $G = (S, E), S_Q$ la requête de client

Sortie(s) : Le sous graphe couvrant la requête et l'ensemble de services à composer

1: requête-refusée := faux ; *% Phase 1 : vérification de l'existence de composition*
2: **Tantque** ($OP_{S_Q} \neq \phi \wedge \neg$(*requête-refusée*)) **Faire**
3: Extraire une opération OP_k ;
4: Appliquer $pluscourtchemin(G, in, out)$; *% entre le paramètre entrée/sortie de* OP_k.
5: **Si** $\neg(pluscourtchemin(G, in, out))$ **Alors**
6: requête-refusée = vraie ;
7: Exit ;
8: **Sinon**
9: Garder le chemin ;
10: **Fin Si**
11: **Fin Tantque**
12: **Si** requête-refusée = vraie **Alors**
13: La requête du client ne peut pas être satisfaite ou la composition n'existe pas ;
14: **Sinon**
15: **Pour tout** ($OP_k \in OP_{S_Q}$) **Faire**
16: **Si** le chemin retourné est de longueur 1 **Alors**
17: Extraire la classe C_n impliquée sur l'arc ; *% Phase 2 : processus de composition*
18: Choisir $OP_j \in C_n$ tel que : *précond/postcond*(OP_j)=*précond/postcond*(OP_k) ;
19: **Sinon**
20: Extraire les noms des classes C_n impliquées sur l'arc ;
21: Choisir $OP_j \in C_p$ tel que :*précond*(OP_j)=*précond*(OP_k) ; *%* C_p, C_d *sont respectivement la première et la dernière classe sur le chemin.*
22: Choisir $OP_j \in C_d$ tel que :$postcond(OP_j) = postcond(OP_k)$;
23: **Pour tout** $C_n \in]C_p, C_d[$ **Faire**
24: Choisir $OP_j \in C_n$ tel que : $précond(OP_j) \in C_n = postcond(OP_m) \in C_{précéd}$; *%* OP_m *est choisie à partir* $C_{\text{précéd}}$, *la classe qui précède* C_n *sur le chemin.*
25: **Fin Pour**
26: **Fin Si**
27: **Si** exist plusieurs OP_j ayant la même *précond/postcond* **Alors**
28: Choisir $OP_j \in S_i$ garantissant la meilleure QoS ;
29: **Fin Si**
30: Retourner les couples (OP_j, S_i) satisfaisant l'opération OP_k ;
31: **Fin Pour**
32: **Fin Si**
33: Fournir le sous graphe satisfaisant la requête ;

3.5.2.1 Principe de fonctionnement de l'algorithme

Le client présente sa requête sous forme d'une suite d'opérations qu'il désire exécuter, l'algorithme parcourt le graphe orienté de services Web à la recherche d'un chemin qui peut satisfaire l'opération en cours en lançant un algorithme de plus cours chemin pour toute opération de S_Q, si un tel chemin existe, il passe à l'opération suivante jusqu'à ce qu'il épuise la suite d'opérations présentée.

L'algorithme contrôle la satisfaction de la requête, cela en vérifiant les opérations constituant l'ensemble OP_{S_Q} : Si pour toute opération présentée dans la requête un chemin est retourné, alors la requête peut être satisfaite ou la composition est possible.

Si c'est le cas, l'algorithme commence à construire le programme de composition en suivant la méthodologie décrite. Il retourne un sous graphe couvrant la requête, ainsi que l'ensemble des services Web impliqués dans cette composition, en se basant sur les chemins retournés pour chaque opération. Dans le cas contraire une réponse négative est retournée.

3.5.3 Ordre d'exécution des services

En général, deux services Web peuvent se composer dans une des deux manières suivantes :

- Ils peuvent être exécutés un après l'autre séquentiellement. L'exécution séquentielle de deux services Web S_i et S_j est représentée par $S_i \cdot S_j$ ou simplement S_iS_j.
- Ils peuvent être exécutés en parallèle. Ceci signifie qu'aucun ordre spécifique n'est défini dans l'exécution des services Web. L'exécution parallèle de deux services Web S_i et S_j est formellement représentée par $S_i \| S_j$.

Pour chaque opération $OP_k \in OP_{S_Q}$ un ou un ensemble de services est retourné, il reste que composer ces services pour satisfaire la requête. Si un service est retourné pour une opération OP_k, elle peut être satisfaite par au moins un service sans composition. Donc, il peut être exécuté en parallèle. Autrement, Pour chaque OP_k les services retournés devant être composés ensemble pour satisfaire cette dernière. Les compositions résultantes pour chaque opération alors sont exécutées séquentiellement de telle manière que l'exécution ne viole aucune dépendance dans l'ensemble des services retourné pour cette dernière. La seule exception est quand il y a quelques opérations dans l'ensemble OP_{S_Q} dont les suites de services retournées

ayant la même action de leur côté gauche (*commençants par la même action*). Ceci signifie qu'une action ou plus doivent être exécutées bien après l'action commune, et donc, l'exécution parallèle est la seule manière pour que la demande peut être satisfaite.

Puisque tous les opérateurs de composition sont des opérations binaires et parce que chacune des opérations de composition n'expose pas une construction autre que les constructions soutenues par OWL-S, le service Web résultant a les mêmes constructions. Donc, on peut construire le flux de contrôle et de données du service composite basant sur les paramètres *in/out* des opérations choisies et leurs ordres d'exécution définis par le chemin. Ceci signifie que nous pouvons produire un modèle de processus OWL-S pour le nouveau service composite et l'offrir comme nouveau service Web.

3.6 Déroulement de l'algorithme sur des exemples

Dans cette section nous fournissons des exemples illustrant comment l'algorithme proposé dans la section précédente fonctionne.

Soient un ensemble de services Web noté par S_i représentés par le graphe orienté $G = (V, E)$, et $S_Q = (OP_{S_Q})$ une requête simple qui pourrait être posée par un utilisateur. La figure 3.1 présente un simple graphe orienté de services Web : S_1, S_2, S_3, S_4, S_5, S_6, S_7 et S_8. Chaque S_i est défini comme suit : $S_i = (I_{S_i}, O_{S_i}, OP_{S_i}, QoS_{s_i})$ (Tab. 3.1). Les différentes opérations fournies par les services Web sont classées selon leurs fonctionnalités, telles que : les opérations ayant le même paramètre *in/out* appartenant à la même classe (Tab. 3.2).

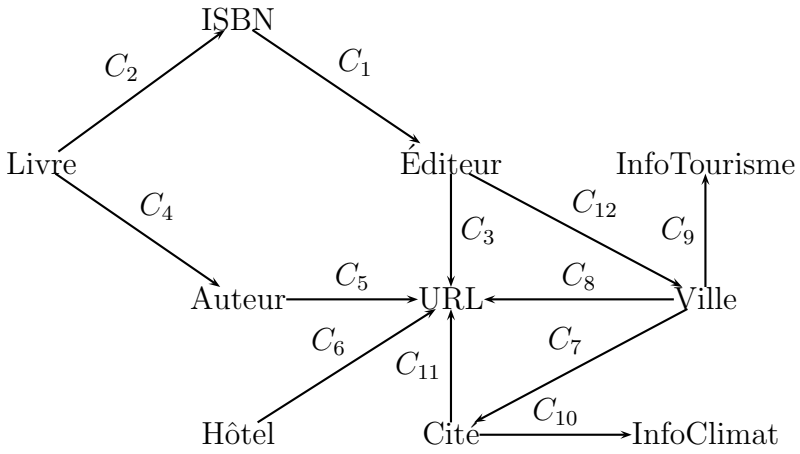

FIG. 3.1 – Un simple graphe orienté de services Web. Les nœuds représentent les paramètres *inputs/outputs* des services Web et les arcs les classes.

$S_1 = (I_{S_1}, O_{S_1}, OP_{S_1}, QoS_1)$, où :
- $I_{S_1} = \{Cité,\ Ville\}$
- $O_{S_1} = \{InfoClimat,\ URL\}$
- $OP_{S_1} = \{OP_1, OP_2\}$
  - $OP_1 = \{Cité,\ InfoClimat,\ w,\ v\ \}$
  - $OP_2 = \{Ville,\ URL,\ p,\ q\}$

$S_2 = (I_{S_2}, O_{S_2}, OP_{S_2}, QoS_4)$, où :
- $I_{S_2} = \{Cité,\ Éditeur,\ Ville\}$
- $O_{S_2} = \{InfoClimat,\ InfoTourisme,\ Ville\}$
- $OP_{S_2} = \{OP_1, OP_2, OP_3\}$
  - $OP_1 = \{Ville,\ InfoTourisme,\ p,\ q\}$
  - $OP_2 = \{Cité,InfoClimat,\ w,\ z\}$
  - $OP_3 = \{Éditeur,\ Ville,\ k,\ l\}$

$S_3 = (I_{S_3}, O_{S_3}, OP_{S_3}, QoS_1)$, où :
- $I_{S_3} = \{Cité,\ Hôtel,\ Ville\}$
- $O_{S_3} = \{InfoTourisme,\ URL\}$
- $OP_{S_3} = \{OP_1, OP_2, OP_3\}$
  - $OP_1 = \{Hôtel,\ URL,\ s,\ r\ \}$
  - $OP_2 = \{Ville,\ InfoTourisme,\ x,\ y\}$
  - $OP_3 = \{Cité,\ URL,\ p,\ q\}$

$S_4 = (I_{S_4}, O_{S_4}, OP_{S_4}, QoS_5)$, où :
- $I_{S_4} = \{Éditeur,\ Livre\}$
- $O_{S_4} = \{URL,\ Auteur\}$
- $OP_{S_4} = \{OP_1, OP_2\}$
  - $OP_1 = \{Éditeur,\ URL,\ o,\ q\}$
  - $OP_2 = \{Livre,\ Auteur,\ i,\ l\}$

$S_5 = (I_{S_5}, O_{S_5}, OP_{S_5}, QoS_2)$, où :
- $I_{S_5} = \{ISBN,\ Auteur\}$
- $O_{S_5} = \{Éditeur,\ URL\}$
- $OP_{S_5} = \{OP_1, OP_2\}$
  - $OP_1 = \{ISBN,\ Éditeur,\ m,\ o\}$
  - $OP_2 = \{Auteur,\ URL,\ m,\ n\}$

$S_6 = (I_{S_6}, O_{S_6}, OP_{S_6}, QoS_4)$, où :
- $I_{S_6} = \{ISBN,\ Éditeur\}$
- $O_{S_6} = \{Livre,\ URL\}$
- $OP_{S_6} = \{OP_1, OP_2\}$
  - $OP_1 = \{ISBN,\ Livre,\ m,\ o\}$
  - $OP_2 = \{Éditeur,\ URL,\ o,\ n\}$

$S_7 = (I_{S_7}, O_{S_7}, OP_{S_7}, QoS_5)$, où :
- $I_{S_7} = \{Hôtel,\ Ville\}$
- $O_{S_7} = \{Cité,\ URL\}$
- $OP_{S_7} = \{OP_1, OP_2\}$
  - $OP_1 = \{Ville,Cité,\ u,\ w\ \}$
  - $OP_2 = \{Hôtel,\ URL,\ s,\ t\ \}$

$S_8 = (I_{S_8}, O_{S_8}, OP_{S_8}, QoS_3)$, où :
- $I_{S_8} = \{Éditeur,\ Livre\}$
- $O_{S_8} = \{Auteur,\ Ville\}$
- $OP_{S_8} = \{OP_1, OP_2\}$
  - $OP_1 = \{\ Livre,\ Auteur,\ i,\ j\}$
  - $OP_2 = \{Éditeur,\ Ville,\ k,\ l\}$

Tab. 3.1 – Liste des services Web disponibles.

N° de classe	Les différentes fonctionnalités	Description de l'opération
C_1	(OP_1, S_5)	$OP_1 = \{$ISBN, Éditeur, m, o$\}$
C_2	(OP_1, S_6)	$OP_1 = \{$ISBN, Livre, m, o$\}$
C_3	(OP_1, S_4)	$OP_1 = \{$Éditeur, URL, o, q$\}$
	(OP_2, S_6)	$OP_2 = \{$Éditeur, URL, o, n$\}$
C_4	(OP_1, S_8)	$OP_1 = \{$Livre, Auteur, i, j$\}$
	(OP_2, S_4)	$OP_2 = \{$Livre, Auteur, i, l$\}$
C_5	(OP_2, S_5)	$OP_2 = \{$Auteur, URL, m, n$\}$
C_6	(OP_1, S_3)	$OP_1 = \{$Hôtel, URL, s, r$\}$
	(OP_2, S_7)	$OP_2 = \{$Hôtel, URL, s, t$\}$
C_7	(OP_1, S_7)	$OP_1 = \{$Ville, Cité, u, w$\}$
C_8	(OP_2, S_1)	$OP_2 = \{$Ville, URL, p, q$\}$
C_9	(OP_1, S_2)	$OP_1 = \{$Ville, InfoTourisme, x, y$\}$
	(OP_2, S_3)	$OP_2 = \{$Ville, InfoTourisme, x, y$\}$
C_{10}	(OP_1, S_1)	$OP_1 = \{$Cité, InfoClimat, w, v$\}$
	(OP_2, S_2)	$OP_2 = \{$Cité, InfoClimat, w, z$\}$
C_{11}	(OP_2, S_8)	$OP_2 = \{$Éditeur, Ville, k, l$\}$
	(OP_3, S_2)	$OP_2 = \{$Éditeur, Ville, k, l$\}$
C_{12}	(OP_3, S_3)	$OP_3 = \{$Cité, URL, p, q$\}$

TAB. 3.2 – Liste des classes résultantes.

Pour la QoS dans notre exemple, on suppose que $QoS_1 > QoS_2 > QoS_3 > QoS_4 > QoS_5$. Le symbole ">" signifie "meilleure".

Le but maintenant, est de trouver un chemin ou un sous graphe pour chaque opération de la requête dans le graphe G. Pour ce faire, en applique l'algorithme *DCoServ*. L'algorithme parcourt l'ensemble des classes à la recherche d'un service Web qui peut exécuter l'opération en cours, si un chemin existe, il passe à l'opération suivante jusqu'à ce qu'il épuise la suite des opérations présentée. Il contrôle la satisfaction de la requête, cela en vérifiant l'ensemble des opérations de la requête : Si pour toute opération un chemin est retourné, alors la requête peut être satisfaite ou la composition est possible.

Si c'est le cas, l'algorithme commence la construction de programme de composition en retournant pour chaque opération ; à partir des classés apparaissant sur les chemins retournés ; les noms des services et les opérations satisfaisant l'opération en cours. À la fin du calcul, nous obtenons le sous graphe global satisfaisant la requête et les services à composer.

Exemple 1

$S_Q = (OP_1, OP_2)$, tels que :

$\begin{cases} \bullet\ OP_1 = \{Ville,\ InfoClimat,\ u,\ v\} \\ \bullet\ OP_2 = \{Hôtel,\ URL,\ s,\ r\} \end{cases}$

Phase1 : vérification de la faisabilité de la composition

- Pour OP_1, un chemin $Ville \overset{C_7}{\rightarrow} Cité \overset{C_{10}}{\rightarrow} InfoClimat$ est retourné.
- Pour OP_2, un chemin $Hôtel \overset{C_6}{\rightarrow} URL$ est retourné.

Puisque pour chaque opération un chemin et retourné alors la composition est faisable.

Phase2 : construction de programme de composition

Pour $OP_1 = \{$Ville, InfoClimat, u, v$\}$

$\begin{cases} \bullet \text{ Le chemin } Ville \overset{C_7}{\rightarrow} Cité \overset{C_{10}}{\rightarrow} InfoClimat \text{ est de longueur deux.} \\ \bullet \text{ Extraire les classes } C_7,\ C_{10} \text{ impliquées sur le chemin} \\ \bullet\ C_7 = \{(OP_1,\ S_7)\}, \text{ où : } OP_1 = \{Ville,\ Cité,\ u,\ w\} \\ \bullet\ C_{10} = \{(OP_1,\ S_1),\ (OP_2,\ S_2)\}, \text{ où :} \\ \text{- } OP_1 = \{Cité,\ InfoClimat,\ w,\ v\} \\ \text{- } OP_2 = \{Cité,\ InfoClimat,\ w,\ z\} \\ \bullet \text{ Choisir de la première classe } C_7 \text{ l'opération : } (OP_1,\ S_7). \\ \bullet \text{ Choisir de la dernière classe } C_{10} \text{ l'opération : } (OP_1,\ S_1) \text{ parce que} \\ post(OP_1) \in S_1 = post(OP_1) \in Q \end{cases}$

Pour $OP_2 = \{$Hôtel, URL, s, r$\}$

$\begin{cases} \bullet \text{ Le chemin } Hôtel \overset{C_6}{\rightarrow} URL \text{ est de longueur un.} \\ \bullet \text{ La classe } C_6 \text{ impliquée sur l'arc contient deux services fournissant la même} \\ \text{opération avec des pré-conditions et post-conditions différentes.} \\ \bullet\ C_6 = \{(OP_1,\ S_3),\ (OP_2,\ S_7)\}, \text{ où :} \\ \text{- } OP_1 = \{Hôtel,\ URL,\ s,\ r\} \\ \text{- } OP_2 = \{Hôtel,\ URL,\ s,\ t\} \\ \bullet \text{ Choisir } (OP_1,\ S_3) \text{ parce qu'elle a la même pré-condition et post-condition.} \end{cases}$

L'opération de Q	Le chemin retourné	Le couple (OP_j, S_i)
$OP_1 = \{Ville,\ InfoClimat,\ u,\ v\}$	$Ville \overset{C_7}{\rightarrow} Cité \overset{C_{10}}{\longrightarrow} InfoClimat$	$(OP_1, S_7), (OP_1, S_1)$
$OP_2 = \{Hôtel,\ URL,\ s,\ r\}$	$Hôtel \overset{C_6}{\rightarrow} URL$	(OP_1, S_3)

TAB. 3.3 – Résultat des chemins retournés par l'algorithme.

Le tableau ci-dessus (Tab. 3.3) récapitule les résultats fournis par l'algorithme. Ainsi, le sous graphe résultant et qui répond à la requête est donné par la figure 3.2.

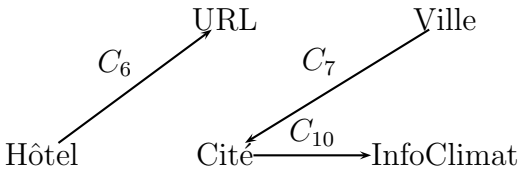

FIG. 3.2 – Le sous graphe couvrant la requête fourni par l'algorithme.

Ordre d'exécution

- Pour $OP_1 = \{Ville,\ InfoClimat,\ u,\ v\}$, les services S_7, S_1 sont retournés ; on doit les composer. Ceci signifie qu'une exécution séquentielle est exigée.
- Pour $OP_2 = \{Hôtel,\ URL,\ w,\ z\}$, un service S_3 est retourné ; on peut répondre à cette opération sans composition de services. Ce service peut être exécuté en parallèle.

Exemple 2

$S_Q = (OP_1, OP_2, OP_3)$, tels que :

$$\begin{cases} \bullet\ OP_1 = \{ISBN,\ URL,\ m,\ n\} \\ \bullet\ OP_2 = \{Livre,\ Auteur,\ i,\ j\} \\ \bullet\ OP_3 = \{Éditeur,\ Ville,\ k,\ l\} \end{cases}$$

Phase1 : vérification de la faisabilité de la composition

- Pour OP_1, il existe deux chemin : $ISBN \overset{C_1}{\rightarrow} Éditeur \overset{C_3}{\rightarrow} URL$, $ISBN \overset{C_2}{\rightarrow} Livre \overset{C_4}{\rightarrow} Auteur \overset{C_5}{\rightarrow} URL$. Le premier chemin est retourné comme le plus court.
- Pour OP_2, un chemin $Livre \overset{C_4}{\rightarrow} Auteur$ est retourné.
- Pour OP_3, un chemin $Éditeur \overset{C_{11}}{\longrightarrow} Ville$ est retourné.

Puisque pour chaque opération un chemin et retourné alors la composition est faisable.

Phase2 : construction de programme de composition

Pour $OP_1 = \{ISBN,\ URL,\ m,\ n\}$

- Le chemin $ISBN \overset{C_1}{\rightarrow} \acute{E}diteur \overset{C_3}{\rightarrow} URL$ est de longueur deux.
- Extraire les classes C_1, C_3 impliquées sur le chemin
- $C_1 = \{(OP_1,\ S_5)\}$, où : $OP_1 = \{ISBN,\ \acute{E}diteur,\ m,\ o\}$
- $C_3 = \{(OP_1,\ S_4),\ (OP_2,\ S_6)\}$, où :
  - $OP_1 = \{\acute{E}diteur,\ URL,\ w,\ v\}$
  - $OP_2 = \{\acute{E}diteur,\ URL,\ o,\ n\}$
- Choisir de la première classe C_1 l'opération : $(OP_1,\ S_5)$.
- Choisir de la dernière classe C_3 l'opération : $(OP_2,\ S_6)$ parce que $post(OP_2) \in S_6 = post(OP_1) \in Q$

Pour $OP_2 = \{Livre,\ Auteur,\ i,\ j\}$

- Le chemin $Livre \overset{C_4}{\rightarrow} Auteur$ est de longueur un.
- La classe C_4 impliquée sur l'arc contient deux services fournissant la même opération avec des pré-conditions et post-conditions différentes.
- $C_4 = \{(OP_1,\ S_8),\ (OP_2,\ S_4)\}$, où :
  - $OP_1 = \{Livre,\ Auteur,\ i,\ j\}$
  - $OP_2 = \{Livre,\ Auteur,\ i,\ l\}$
- Choisir $(OP_1,\ S_8)$ parce qu'elle a la même pré-condition et post-condition.

Pour $OP_3 = \{$Éditeur, Ville, k, l $\}$

- Le chemin $\acute{E}diteur \overset{C_{11}}{\longrightarrow} Ville$ est de longueur un.
- La classe C_{11} impliquée sur l'arc contient deux services fournissant la même opération avec la même pré-condition et post-condition .
- $C_{11} = \{(OP_2,\ S_8),\ (OP_3,\ S_2)\}$, où :
  - $OP_1 = \{\acute{E}diteur,\ Ville,\ k,\ l\}$
  - $OP_2 = \{\acute{E}diteur,\ Ville,\ k,\ l\}$
- Choisir celle fournie par le service garantissant la meilleure QoS : $(OP_2,\ S_8)$

Le tableau suivant (Tab. 3.4) récapitule les résultats fournis par l'algorithme.

L'opération de Q	Le chemin retourné	Le couple (OP_j, S_i)
$OP_1 = \{ISBN,\ URL,\ m,\ n\}$	$ISBN \overset{C_1}{\rightarrow} Éditeur \overset{C_3}{\rightarrow} URL$	(OP_1, S_5), (OP_2, S_6)
$OP_2 = \{Livre,\ Auteur,\ i,\ j\}$	$Livre \overset{C_4}{\rightarrow} Auteur$	(OP_1, S_8)
$OP_3 = \{Éditeur,\ Ville,\ l,\ k\}$	$Éditeur \overset{C_{11}}{\rightarrow} Ville$	(OP_2, S_8)

TAB. 3.4 – Résultat des chemins retournés par l'algorithme.

Le sous graphe résultant et qui répond à la requête est donné par la figure 3.3.

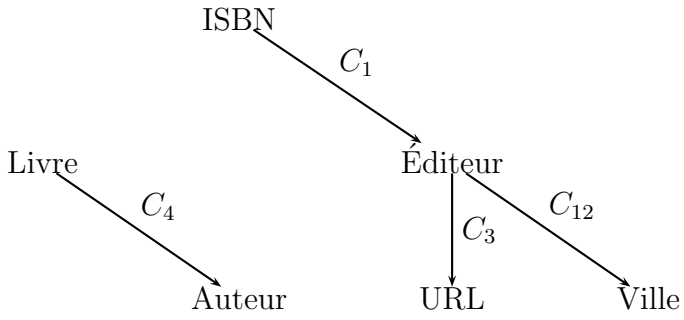

FIG. 3.3 – Le sous graphe couvrant la requête fourni par l'algorithme.

Ordre d'exécution

- Pour $OP_1 = \{ISBN,\ URL,\ m,\ n\}$, les services S_5, S_6 sont retournés ; on doit les composer. Ceci signifie qu'une exécution séquentielle est exigée.
- Pour $OP_2 = \{Livre,\ Auteur,\ i,\ j\}$, un service S_8 est retourné ; on peut répondre à cette opération sans composition de services. Ce service peut être exécuté en parallèle.
- Pour $OP_3 = \{Éditeur,\ Ville,\ l,\ k\}$, un service S_8 est retourné ; on peut répondre à cette opération sans composition de services. Ce service peut être exécuté en parallèle.

Contre exemple

$S_Q = (OP_1, OP_2)$, tels que :

$$\begin{cases} \bullet\ OP_1 = \{Livre,\ Éditeur,\ i,\ j\} \\ \bullet\ OP_2 = \{Éditeur,\ URL,\ k,\ l\} \end{cases}$$

Phase1 : vérification de la faisabilité de la composition

Pour $OP_1 = \{Livre,\ Éditeur,\ i,\ j\}$, il n'existe pas un chemin sur le graphe qui peut répondre à cette opération. L'algorithme ne peut pas satisfaire cette opération, il met *requête-refusée :=true*. Donc, il continu pas la liste des opérations restantes en sortant de la boucle. Par conséquent, c'est inutile de construire le programme de composition car une partie de la requête ne peut pas être satisfaite. Une réponse composition impossible est renvoyée.

3.7 Conclusion

Nous avons présenté dans cette partie comment représenter les services Web disponibles sous forme d'une structure de graphe orienté après avoir classer les différentes opérations fournies par ceux ci en un ensemble de classe en se basant sur leurs fonctionnalités. Dans le but de répondre au problème de composition, pour satisfaire une requête donnée, nous avons proposé une méthodologie pour assister le processus de composition composée de trois processus. Le processus de vérification sert à vérifier l'existence d'une telle composition en explorant les algorithmes de graphes tel que l'algorithme de plus court chemin, Dijkstra. Ceci réduira le nombre de services à composer pour satisfaire la requête.

Si la composition est possible, le processus de composition abstrait est chargé de découvrir la combinaison de services satisfaisant la requête donnée et en respectant les préférences de client. En effet, la découverte des services est maintenue en se basant sur les chemins retournés et les noms des classes impliquées sur les arcs. Le processus de liaison permet de convertir la composition résultante dans une description exécutable pour le déploiement et l'invocation.

Conclusion et Perspectives

Le concept services Web est le nouveau cliché à la mode émanant du monde informatique. Il se répand depuis l'an 2000. Il est difficile de donner une définition stricte de ce que sont les services Web. Les services Web et les protocoles qui y sont associés sont en mouvance constante et n'en ont encore qu'à leurs premières élaborations et implantations par les différents acteurs de la scène informatiques et les entreprises. De plus, malgré l'intérêt croissant pour le phénomène, il n'existe pas encore de définition universelle de ce que sont les service Web. Bien que tous les acteurs majeurs de l'industrie informatique soient partis prenante de cette technologie, ils confrontent sur le terrain de la mise en marché et des organismes de standardisations. Cette confrontation marketing est l'explication derrière l'absence d'une définition unanime de l'industrie. Cependant, il ne faut pas présumer que l'absence de définition commune soit l'indice d'absence d'une technologie normative. En effet, malgré la guerre marketing que se livre les fabricants logiciels et les guerres de territoire que se livrent les organismes de standardisations, plusieurs technologies sont suffisamment normalisées pour permettre aux entreprises de tirer profits des services Web dés aujourd'hui.

Selon ***Engelen*** and ***Gallivan*** [32], SOAP n'est pas une technologie concurrentielle aux systèmes à composants tels que CORBA et DCOM, mais complète plutôt ces technologies. CORBA, DCOM et Enterprise Java permettent le partage de ressource dans une seule organisation tandis que la technologie SOAP à pour objectifs de permettre le partage des ressources entre les différents organismes probablement placés derrière des pare-feux. SOAP est beaucoup semblables au *Object Management Group's CORBA General Inter-ORB Protocol* (GIOP de OMG [64]). Malgré que tous les deux supportent la communication indépendante de protocole dans un environnement hétérogène, CORBA manque d'une base XML et le moins d'être une adoption universel ce qui rend SOAP un meilleur choix pour l'usage dans le cadre des services Web. Le paradigme service Web atteint ce que les standards de calcul d'objet distribué tel que CORBA ont exclu, en essayant de fournir des composants logiciels plus flexibles et moins structurés.

Les recherches au tour des services Web s'étendent sur beaucoup de thèmes intéressants concernant, en particulier, la composition, la découverte, la coordination et la vérification. La construction des services Web nécessite l'intégration de différents systèmes hétérogènes. Ces systèmes sont différents non seulement sous le point de vue technologique, mais également pour le contenu de l'information qu'ils exportent. La meilleure façon pour résoudre le dilemme d'intégration avec une solution simple et interoperable, est d'établir des *middlewares* basées sur des courtiers en utilisant des protocoles d'Internet tels que HTTP et XML comme solution pour rassembler les données qui sont l'essence des services Web.

De plus, nous assistons également à une panoplie de technologies de composition de services, plusieurs initiatives ont été conduites dans le but de fournir des plates-formes et des langages qui permettent l'intégration des systèmes hétérogènes. Cette tendance a déclenché un nombre considérable d'efforts de recherche sur la composition de services Web dans les deux domaines académique et industriel. Il devient ainsi nécessaire de gérer l'exécution et la coordination des différents composants de service Web et vérifier le contrôle du flux des données, afin de garantir l'exécution correcte du service Web composite. De ce fait, des protocoles ont a été mis en œuvre.

Conscients des points faibles des technologies de services Web, les organismes de standardisation sont au travail. De leur côté, les éditeurs spécialistes de la sécurité, tels VeriSign, Entrust, Netegrity, ou encore, RSA Security ont soumis au W3C, ou au consortium Oasis, des propositions de standards tels que SAML (*Security assertion markup language*) et XKMS (*XML key management specification*). Ces propositions de standards sont également soutenues par IBM, Iona, Microsoft et webMethods. Elles ont été prises en compte dans le programme de travail du W3C.

Dans ce mémoire, nous nous somme intéressés à l'intégration des services Web qui peut être traitée comme la composition de ces derniers. Nous avons décelé les problèmes engendrés lors de l'intégration des services. Par conséquent, l'intégration de la qualité de service s'impose.

La composition de services est très délicate. Beaucoup de protocoles supportant la composition de services Web ont été proposés. Ces protocoles se focalisent essentiellement sur la représentation des compositions de services où le flux des processus et les liaisons entre les services sont connus a priori. D'un autre coté, beaucoup de travaux ont essayé de traiter le problème de la composition automatique de services Web et la sélection des services impliqués lors de l'exécution en se basant sur les propriétés fonctionnelles. Par contre, la recherche sur l'intégration de la QoS lors de la découverte et la sélection est restreinte.

Par conséquent, nous avons proposé d'établir une structure générale représentant les services Web disponibles et les relations qui peuvent exister entre eux en termes de leurs paramètres d'entrées et sorties ; c-à-d s'ils peuvent être composés. Nous avons choisi de les représenter sous forme de graphe afin de bénéficier des algorithmes des graphes existant lors de la selection et la découverte de services Web tels que les algorithmes de plus court chemin. En se basant sur le graphe orienté de services Web construit et la requête de client, nous explorons l'algorithme de composition de services Web proposé pour retourner une combinaison de services pouvant participer dans la composition satisfaisant la requête. L'algorithme de plus court chemin permet d'explorer le graphe pour sélectionner et découvrir les services Web répondant à la requête en retournant le plus court chemin si plus qu'un chemin existe. De ce fait, nous réduisons le nombre de services retournés. Lors de la construction de programme de composition, nous tenons en compte de vérifier si une telle composition existe.

Perspectives

Comme il n'existe pas de formalisme standard sur le problème de la composition de services Web alors les perspectives ne peuvent être que nombreuses. Tout d'abord, étudier la complexité et la complétude de l'algorithme proposé.

De même, il serait intéréssant d'étendre notre recherche sur le processus de sélection et découverte automatique des services Web en supportant la QoS en tant qu'élément de la requête où des coûts spécifiques tels que le temps et l'argent sont importants afin d'assurer les contraintes de préférences du client. Ceci aidera à poser des requêtes plus précises et fournir des résultats plus précis. Ainsi, Comment définir la qualité du service Web composite en termes de composants de service par rapport à la requête du client.

Nous aimerions également étudier le problème des entrées/sorties composés si un ensemble d'entrées est imposé pour que le service fournit une ou plusieurs sorties.

La composition de services Web implique aussi de nouvelles questions, à savoir la substitution. Ce problème, peut être traité en exploitant cette notion de classe vue quelle contient toutes les services assurant la même fonctionnalité.

À long terme, il serait également pertinent de pouvoir étudier comment mettre la sécurité et la politique d'authentification du service composite basé sur ceux des composants.

Glossaire

Agent	Un agent est un programme qui agit au nom d'une personne ou d'une organisation.
BPEL4WS	Business Process Execution Language for Web Services, est un langage de chorégraphie pour les services Web.
Couplage (fort,faible)	Utilisé pour caractériser la relation de deux applications, couplage faible signifie une interdépendance faible...
DAML+OIL	Langage permettant de définir une ontologie pour un domaine particulier.
DAML-S	DARPA Agent Markup Language Services, langage de description sémantique de services Web. Il permet la description, la recherche, la sélection et l'exécution d'un service Web particulier mais aussi la composition de services entre eux. DAML-S est en fait une ontologie.
DTD	Document Type Definition, document permettant de définir la structure d'un fichier XML.
EAI	Enterprise Application Integration, solutions du marché pour intégrer les systèmes d'information.
Format d'échange	Modèle des informations échangées par deux partenaires dans le contexte d'une coopération.
HTTP	Hyper Text Transport Protocol, protocole de base de l'Internet.
Interopérabilité	Capacité de deux applications distinctes, et éventuellement hétérogènes et distantes, à coopérer.
Middleware	Logiciel d'intermédiation pour permettre une interopérabilité applicative.

MIME	Multipurpose Internet Mail Extensions, est un format qui a été conçu pour transférer par e-mail des données codées qui ne sont pas du texte (*images...*).
OIL	Ontology Inference Layer, langage de représentation des connaissances.
Ontologie	C'est un catalogue sémantique, dont les descriptions sont à la fois concises, non ambiguës, et qui se doit d'être exploitable par un logiciel (*description formelle*) comme par un opérateur humain (*description littéraire*).
OWL	Web Ontology Language, langage inspiré de DAML et OIL.
OWL-S	OWL Service, nouveau nom donné à DAML-S.
P2P	Peer to Peer, qui contrairement au modèle client-serveur, est une liaison poste à poste.
Proxy	Web Ordinateur qui s'intercale entre deux réseaux et qui permet principalement d'enregistrer dans un cache les pages Web couramment utilisées pour ensuite les délivrer sans qu'il soit nécessaire de se connecter sur le serveur initial.
RDF	Resource Description Framework, modèle et description de syntaxe en vue de l'utilisation de méta données sur le Web. Son objectif est de faciliter le traitement automatisé des informations contenues sur le Web en permettant leur description sans ambiguïté.
Service Web	Application disponible sur le Web et accessible par une interface standardisée. Elle peut interagir avec d'autres services Web indépendamment du système d'exploitation et des langages de programmation utilisés.
SMTP	Simple Mail Transfert Protocol, protocole gérant l'envois des mails entre différents serveurs.
SOA	Service Oriented Architecture, architecture de service.

SOAP	Simple Object Access Protocol, protocole de communication inter applicatif, au dessus de HTTP, comportant un ensemble de règles pour structurer des messages XML et invoquer un service Web.
UDDI	Universal Description Discovery and Integration, norme permettant de créer et de retrouver des services Web. C'est un annuaire en ligne référençant un ensemble de services Web disponibles.
URI	Uniform Resource Identifier, permet d'identifier un point de contenu sur le Web, que ce soit un fichier texte, audio ou vidéo. Une URI peut être une URN ou URL. Une URI est une superclasse abstraite.
URL	Uniform Resource locator l'une de sous catégorié d'URI.
Workflow	Application qui permet de séquencer des tâches suivant un modèle qui définit en particulier comment ces tâches sont synchronisées.
WSDL	Web Service Description Language, langage basé sur XML permettant la description de l'interface d'un service Web.
XQuery	Langage pour interroger et manipuler les données d'un document XML.
XSL	eXtensible Style Language, est un langage dérivé de XML permettant la mise en forme par feuille de style des données XML.

Bibliographie

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web services . Concepts, architectures and applications. *Berlin, Springer-Verlag*, 2004.

[2] V. De Antonellis, M. Melchiori, B. Pernici, and P. Plebani. A methodology for e-service substitutability in a virtual district environment. *Advanced Information Systems Engineering, 15th International Conference (CAiSE 2003), Klagenfurt, Austria*, Juin 2003.

[3] Project Web Services Axis Apache.Org. Axis user's guide. 16 Novembre 2004. http ://ws. Apache.Org/axis/.

[4] K. Arnold and al. The Jini specification. *Addison-Wesley*, Juin 1999.

[5] A. Banerji, C. Bartolini, D. Beringer, and al. Web services conversation language (WSCL) 1.0 . *W3C Note*, 14 Mars 2002. http ://www.w3.org/TR/2002/NOTE-wscl10-20020314/.

[6] BEA, Intalio, Sap, and Sun. Web service choreography interface (WSCI) 1.0. *W3C Note*, 8 Août 2002. http ://www.w3.org/TR/2002/NOTE-wsci-20020808.

[7] J. Beatty and al. Web services dynamic discovery (WS-Discovery). *Microsoft Corporation Inc., BEA Systems, Canon and Intel Specification*, Avril 2005.

[8] B. Benatallah, V. Atluri, and al. On automating Web services discovery. *Springer-Verlag*, 6 Février 2004.

[9] B. Benatallah, M. Dumas, M-C. Fauvet, F.A. Rabhi, and Q.Z. Sheng. Towards patterns of Web services composition. *Technical report, University of New South Wales (2001) UNSWCSE -TR-0111*, Novembre 2001.

[10] B. Benatallah, M. Dumas, M-C. Fauvet, F.A. Rabhi, and Q.Z. Sheng. Overview of somepatterns for architecting and managing composite Web services. *ACM SIGecom Exchanges*, 3(3) :9–16, Août 2002.

[11] B. Benatallah, M. Dumas, Q.Z. Sheng, and A.H.H. Ngu. Declarative composition and peer-to-peer provisioning of dynamic Web services. *In Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 2002.

[12] S. Benbernou, D. Benslimane, M-S. Hacid, H. Kheddouci, and A.Tari. A graph-based approach for Web services composition. *In Proceedings of the 3rd International Conference on Web Information Systems and Technologies (WEBIST 2007), Barcelona, Spain*, 3-6 Mars 2007.

[13] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol - HTTP/1.0. *RFC 1945 (Informational)*, Mai 1996.

[14] D. Booth, H. Haas, F. McCabe, and E. Newcomer. Web services architecture. *W3C Working Draft*, 8 Août 2003. http ://www.w3.org/TR/2003/WD-ws-arch-20030808/.

[15] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (XML) 1.0 (second edition). *W3C Recommendation*, 6 Octobre 2000. http ://www.w3.org/TR/2000/REC-xml-20001006.

[16] D. Bunting, M.C.O. Hurley, M. Little, J. Mischkinsky, E. Newcomer, J. Webber, and K. Swenson. Web services coordination framework (WS-CF) version 1.0. *Sun, Oracle, Iona and Novell Specification*, 28 Juillet 2003. http ://developers.sun.com/techtopics/webservices/wscaf/wscf.pdf.

[17] D. Bunting, M.C.O. Hurley, M. Little, J. Mischkinsky, E. Newcomer, J. Webber, and K. Swenson. Web services context (WS-Context) Ver1.0. *Sun, Oracle, Iona et Novell Specification*, 28 Juillet 2003. http ://developers.sun.com/techtopics/webservices/wscaf/wsctx.pdf.

[18] F. Cabrera and al. Web services coordination (WS-Coordination). *IBM, Microsoft and BEA Systems Specification*, 2002. ftp ://www6.software.ibm.com/software/developer/library/ws-coordination.pdf.

[19] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, and S. Thatte. Web services transaction (WS-Transaction). *IBM, Microsoft and BEA Systems Specification*, 2001. http ://www-106.ibm.com/developerworks/webservices/library/ws-transpec/.

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

[21] F. Casati, M. Sayal, and M.C. Shan. Developing e-services for composing e-services. *In Proceedings of 13th International Conference on Advanced Information Systems Engineering (CAiSE), Springer Verlag, Interlaken, Switzerland*, Juin 2001.

[22] F. Casati and M-C. Shan. Models and languages for describing and discovering e-services. *In ACM SIGMOD, Santa Barbara, USA*, 2001.

[23] Hewlett-Packard Corporation. E-Speak architecture specification version beta2.2. Décembre 1999. http ://www.e-speak.net/library/pdfs/E-speakArch.pdf.

[24] Microsoft Corporation. The component object model specification. *Draft Version 0.9*, Octobre 1995.

[25] F. Curbera, E. Christensen, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. *W3C Note*, 2001. http ://www.w3.org/TR/2001/NOTEwsdl-20010315.

[26] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web services : An introduction to SOAP, WSDL, and UDDI. *IEEE Internet computing*, 6(2) :86–93, 2002.

[27] F. Curbera, D. Ehnebuske, and D. Rogers. Using WSDL in a UDDI registry. *Version 1.07 UDDI Best Practice*, 21 Mai 2002.

[28] F. Curbera, W. Nagy, and S. Weerawaran. Web services. why and how. *Workshop on Object-Oriented Web Services OOPSLA Tampa, Florida, USA*, 2001.

[29] L. de Alfaro and T.A. Henzinger. Interface automata. *In Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM Press*, pages 109–120, 2001.

[30] D. Eastlake and J. Reagle. XML encryption syntax and processing. *W3C Recommendation*, Décembre 2002. http ://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[31] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6) :1278–1304, 1995.

[32] R.V. Engelen and K. Gallivan. The gSOAP toolkit for Web services and peer-to-peer computing networks. *In Proceeding of CCGRID, Berlin, Germany*, pages 128–135, 2002.

[33] D. Fensel, C. Bussler, and A. Maedche. Semantic Web enabled Web services. *In International Semantic Web Conference, Sardinia, Italy*, pages 1–2, 2002.

[34] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol version 3.0. 18 Novembre 1996. http ://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00.

[35] T. Freund and T. Storey. Transactions in the world of Web services, part 1. An overview of WS-Transaction and WS-Coordination. *IBM Specification*, 2002. http ://www-106.ibm.com/developerworks/ webservices/library/ws-wstx1/.

[36] T. Freund and T. Storey. Transactions in the world of Web services, part 2. An overviewof WS-Transaction and WS-Coordination. *IBM Specification*, 2002. http ://www-106.ibm.com/developerworks/webservices/library/ws-wstx2/.

[37] G. Gardarin. XML des bases de données aux services Web. *Dunod*, 2002.

[38] G. De Giacomo, Y. Lespérance, and H. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *AIJ, 121(1-2)*, 121(1-2) :109–169, 2000.

[39] T. Gruber. What is an ontology ? *Summary statement of Gruber's defintion of ontology*, 5 Novembre 2004. http ://www-ksl.standford.edu/kst/what-is-an-ontology.html.

[40] H. Haas. Designing the architecture for web services . *In 12th International World Wide Web Conference's W3C track (WWW2003), Hongrie, Budapest*, 22 Mai 2003. http ://www.w3.org/2003/talks/0521-hh-wsa/.

[41] R. Hamadi and B. Benatallah. A petri net-based model for Web service composition. *In Proceedings of the 14th Australasian Database Conference on Database Technologies*, pages 191–200, 2003.

[42] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4) :293–333, Octobre 1996.

[43] S.V. Hashemian and F. Mavaddat. A graph-based approach to Web services composition. *In Proceedings of the 2005 Symposium on Applications and the Internet (SAINT'05), 0-7695-2262-9/05 IEEE*, 2005.

[44] I. Horrocks, P.F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+OIL : An ontology language for the semantic Web. *In Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 792–797, 2002.

[45] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure certificate and CRL profile. *RFC 2459 (Proposed Standard), Obsoleted by RFC 3280*, Janvier 1999.

[46] M. Philippe Hubert. *Services Web pour le Web Sémantique : mise en oeuvre des Web services sémantiques dans les différentes technologies existantes (ou émergentes) actuelles.* PhD thesis, Centre régional de lorraine, Nancy, 10 Décembre 2004.

[47] IBM. developerWorks : Web services . Web services (r)evolution : part1. 2004. http ://www-106.ibm.com/developerworks/webservices/.

[48] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business process execution language for Web services (BPEL4WS) version 1.1. 2004. http ://www-106.ibm.com/developerworks/library/ws-bpel/.

[49] Business Process Modelling Initiative. Business process modeling language. 24 Juin 2002. http ://www.bpmi.org.

[50] H. Kadima and V. Monfort. Les Web services technique, démarches et outils XML, WSDL, SOAP, UDDI, Rosetta, UML . *Dunod, Paris, ISBN 2 10 0065580*, 2003.

[51] P. Kellert and F. Toumani. Les Web services sémantiques. *Rapport final d'activité spécifique 32 CNRS/STIC, Web sémantique, J. Charlet , P. Laublet and C. Reynaud (Eds)*, pages 93–110, Décembre 2003. http ://rtp-doc.enssib.fr/basedoc/rapports :ASWSRapportsActiviteV3-2003.pdf.

[52] F. Koehl. Web services : définition et retours de réalisation. 22 Janvier 2003. http ://www.atica.pm.gouv.fr/upload/documents/Advese.pdf.

[53] S. Kouadri and B. Hirsbrunner. A context-based service discovery and composition framework for wireless environments. *In Proceedings of the IASTED International Conference on Wireless and Optical Networks (WOC'2003), Banff, Alberta, Canada*, Juillet 2003.

[54] H. Kreger. Web services conceptual architecture (WSCA 1.0). *IBM Software Group*, page 283, Mai 2001.

[55] A. Leger, C. Rey, F. Toumani, B. Benatallah, and M-S. Hacid. On automating Web services discovery. *VLDB Journal, 14(1) :84-96*, 2005.

[56] F. Leymann. Web service flow language (WSFL 1.0). *IBM Document*, 2001. http ://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[57] L. Li and I. Horrocks. A software framework for matchmaking based on semantic Web technology. *In 12th International Conference on World Wide Web (WWW03)*, 2003.

[58] D. Martin. OWL-S : Semantic markup for Web services. *The OWL Services Coalition*, 27 Décembre 2003. http ://www.daml.org/services/owl-s/1.0/owl-s.html.

[59] D.L. McGuinness and F. van Harmelen. OWL Web ontology language overview. *W3C Candidate Recommendation*, Août 2003. http ://www.w3.org/TR/owl-features/.

[60] S. McIlraith and T. C. Son. Adapting Golog for composition of semantic Web services. *In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France*, Avril 2002.

[61] S. McIlraith, T.C. Son, and H. Zeng. Semantic Web services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2) :46–53, 2001.

[62] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of Web services. *In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA*, pages 77–88, Mai 2002.

[63] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos network authentication service (v5). *RFC 4120 (Proposed Standard), Updated by RFC 4537*, Juillet 2005.

[64] Object Management Group (OMG). The common object request broker : Architecture and specification (CORBA). *OMG Technical Document*, 12 Décembre 2004. http :// www.omg.org.

[65] H-Y Paik, C. Rey, B. Benatallah, M-S Hacid, and F. Toumani. Towards semantic - driven, flexible and scalable framework for peering and querying e-catalog communities. *Information Systems International*, 2005.

[66] M. Papazoglou. Web services and business transactions. *World Wide Web : Internet and Web Information System, n° 13*, pages 49–91, 2003.

[67] M.P. Papazoglou and D. Georgakopoulos. Service-oriented computing (special issue). *Communication of the ACM, pguest editors*, 46(10) :25–28, 26 Octobre 2003.

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

[69] S. Ran. A model for Web services discovery with QoS. *ACM SIGecom Exchanges, ACM Press*, 4(1) :1–10, 2003. http ://doi.acm.org/10.1145/844357.844360.

[70] J. Rao. *Semantic Web Service Composition via Logic-based Program Synthesis*. PhD thesis, 2004 :121, ISBN 82-471-6463-9, N-7491 Trondheim, Norway, 2004.

[71] J. Rao, P. Küngas, and M. Matskin. Application of Linear Logic to Web service composition. *In Proceedings of the 1st International Conference on Web Services, Las Vegas, USA*, Juin 2003.

[72] J. Rao, P. Küngas, and M. Matskin. Logic-based Web service composition : from service description to process model. *In Proceedings of the 2004 IEEE International Conference on Web Services (ICWS 2004), San Diego, California, USA*, 6-9 Juillet 2004.

[73] J. Rao and X. Su. Toward the composition of semantic Web services. *In Proceedings of the 2nd International Workshop on Grid and Cooperative (GCC'2003), Springer-Verlag, Shanghai, China*, 3033 of LNCS, Décembre 2003.

[74] C. Rey, F. Toumani, M-S. Hacid, and A. Leger. An algorithm and a prototype for the dynamic discovery of e-services. *Report, LIMOS, Clemont-Ferrand, France*, 2003. http ://www.isima.fr/-rey/.

[75] J. Rosenberg and D.L. Remy. Securing Web services with WS-Security. *Sams Publishing*, 2004.

[76] T. Satish. XLANG. Web services for business process design. *Microsoft Document*, 2001. http ://www.gotdotnet.com/team/xml wsspecs/xlang-c/default.hm.

[77] S. Shrivastava, L. Bellissard, D. Fliot, and al. A workflow and agent based platform for service provisioning. *In Proceedings of the 4th IEEE/OMG International Enterprise Distributed Object Computing Conference (EDOC 2000), Makuhari, Japan, IEEE Computer Society Press*, Septembre 2000.

[78] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of Web services using semantic descriptions. *In Proceedings of Web Services : Modeling, Architecture and Infrastructure Workshop in conjunction with ICEIS2003*, 2002.

[79] S. Skiena. Dijkstra's algorithm. 6.1.1. *Implementing Discrete Mathematics : Combinatorics and Graph Theory with Mathematica*, pages 225–227, 1990.

[80] D. Solo, D. Eastlake, and J. Reagle. XML signature syntax and processing. *W3C Recommendation*, Février 2002. http ://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.

[81] UDDI.org. UDDI technical white paper. 6 Septembre 2000. http ://www.uddi.org/pubs/Iru UDDI Technical White Paper.pdf.

[82] UDDI.org. Universal description, discovery and integration. *Rapport, OASIS UDDI Specification Technical Commite*, Mars 2002. http ://www.oasisopen.org/cover/uddi.html.

[83] M. Verma. Xml security : Ensure portable trust with SAML. Mars 2004. http ://www-128.ibm.com/developerworks/xml/library/x-seclay4/samlovw.

[84] World Wide Web Consortium (W3C). Web services glossary. *W3C Working Draft*, 14 Mai 2003. http ://www.w3.org/TR/2003/WD-ws-gloss-20030514/webservice.

[85] World Wide Web Consortium (W3C). SOAP version 1.2 part 0 : Primer. *W3C Recommendation*, 24 Juin 2003. http ://www.w3.org/TR/soap12-part0/58.

[86] World Wide Web Consortium (W3C). The extensible stylesheet language family (XSL) version 1.1. *W3C Recommendation*, 05 Décembre 2006. http ://www.w3.org/TR/2006/REC-xsl11-20061205/.

[87] World Wide Web Consortium (W3C). XML path language (XPath) version 2.0. *W3C Recommendation*, 23 Janvier 2007. http ://www.w3.org/TR/2007/REC-xpath20-20070123/.

[88] World Wide Web Consortium (W3C). Xquery 1.0 : An XML query language. *W3C Recommendation*, 23 Janvier 2007. http ://www.w3.org/TR/2007/REC-xquery-20070123/.

[89] H. Wolfgang. The Web service discovery architecture. *In Proceedings of the 2002 IEEE/ACM Supercomputing Conference, Baltimore, USA, IEEE Computer Society Press*, pages 1–15, Novembre 2002.

[90] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic Web services composition using SHOP2. *In Proceedings of the Workshop on Planning for Web Services, Trento, Italy*, Juin 2003.

[91] J. Yang and M. P. Papazoglou. Web component : A substracte for Web service reuse and composition. *In Proceedings of the 14th International Conference for Advanced Information Systems Engineering (CAiSE), Toronto, Canada*, LNCS 2348, Mai 2002.

[92] J. Yang and M.P. Papazoglou. Service component for managing service composition life-cycle. *Information Systems, Elsevier*, 29, Issue 2 :97–125, 2004. http ://maximus.uvt.nl/sigsoc/pub/Yang-Papazoglou-Service components for managing the life-cycle of service compositions.pdf.