République Algérienne
Démocratique et Populaire

Ministère de l'Enseignement Supérieure et
de la Recherche Scientifique

Université Djillali Liabès de Sidi-Bel-Abbès

Faculté des Sciences Exactes
Département d'Informatique

Implémentation d'un Algorithme de Découverte de Services Web dans le Contexte des Réseaux Sociaux

Présenté par :

DJOUDI Youcef

Encadré par :

Mme BELKHODJA L.

Juin 2018

Remerciements

Nous souhaitons présenter nos profonds remerciements à **Madame BELKHODJA L.** d'avoir accepté d'être la directrice de ce travail et pour ses interactions rapides, ses précieux conseils, d'avoir veiller sur le bon avancement des travaux et enfin pour la confiance qu'elle nous a accordé depuis le début de ce travail.

Notre grande gratitude va à **M. LEMLOUMA Tayeb**, chercheur au laboratoire IRISA (France) et ancien membre du W3C pour son suivi proche, continu et intense et l'énorme temps qu'il nous a consacré.

Un grand merci à **M. FAHCI Mahmoud** pour son temps, d'avoir partagé ses connaissances et pour nous avoir guidé au moment qu'il fallait. C'est aussi, pour nous, l'occasion de remercier très fortement **tous nos professeurs** d'avoir accompli leurs mission depuis le début de cette formation.

Enfin, à toutes les personnes qui se sont impliquées, de près ou de loin, à l'accomplissement de ce travail.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Introduction Générale

Sommaire

1.1 Introduction

Depuis son apparition, le Web a évolué pour englober diverses sources d'information accessible mondialement. Les organisations de tous les spectres ont déjà déplacé leurs opérations principales sur le Web, ce qui a entraîné une croissance rapide de différentes applications Web. Par conséquent, les services Web sont devenus un des facteurs technologiques le plus significatif. La croissance incessante du nombre de services Web (WS) rend leur découverte de plus en plus difficile. Par exemple, en utilisant les registres de découverte standard, tels que avec les standards UDDI et ebXML, nous sommes rapidement confrontés à leurs propres limites inhérentes car ils ne décrivent que l'aspect fonctionnel de chaque WS et non son rapport aux autres. Il devient donc primordial de gérer les interactions des services et leurs évolution (exemple, collaborations, substitutions, compétitions, composition, etc.) avec

la situation actuelle du Web qui a tant changé depuis sa naissance. En l'état actuel, le Web n'a pas connu seulement une explosion de services mais aussi un changement de comportement des utilisateurs qui se sont transformés de «simples consommateurs d'information» à des «participants actifs» dans ce monde fortement connecté où les données deviennent de plus en plus ouvertes (open data), connectées (linked data), massif (big data) et changeantes.

Capturer les relations entre les WS lorsqu'ils interagissent les uns avec les autres peut être utile de plusieurs façons et dans de nombreux domaine d'applications. Dans notre travail, nous nous intéressons à des nouveaux modèles qui émergent et qui essaient de capturer de telles relations en se basant sur le paradigme des réseaux sociaux. Aujourd'hui, les services Web représentent un domaine de recherche très important. Cette grande importance a donné naissance à des nouvelles méthodes de découverte mais aussi d'autres interactions telles que la composition de services ou le remplacement en cas de pannes. Dans ce travail, nous allons nous focaliser principalement sur les problèmes de la découverte des services Web et la sélection des services Web selon les besoins des utilisateurs ainsi comment tirer profit des connaissances générés des réseaux sociaux pour faciliter la création ou recommandation des services par ou pour des utilisateurs finaux. Notons que ces modèles peuvent aussi servir dans l'ingénierie des services tels que le développement et la composition de nouveaux services ou d'applications distribuées qui emploient ces services d'une manière indirecte et en temps réel.

1.2 Problématique

Bien que le Web et les services Web représentent un thème solide qui a été largement étudié dans des travaux de recherche et d'implémentation. La combinaison du paradigme des réseaux sociaux avec les interactions des services Web entre eux est très récente. En effet, comme nous allons voir, le premier travail qui propose d'utiliser de tel paradigme et l'appliquer aux interactions entre service date de moins de dix ans.

Dans ce domaine, il y a de nombreux standards, protocoles à maîtriser en plus de l'obligation de suivre l'évolution du Web et sa situation à l'heure actuelle. En effet, l'évolution des standards et des implémentations confronte tout chercheur ou développeur au fait que de nombreux outils deviennent ou deviendront obsolètes ou difficile à intégrer en particulier dans un thème récent. On cite par exemple les changements d'adoption des grandes compagnies (exemple Microsoft et IBM) vis à vis du standards UDDI (l'annuaire des services) ou le changement très fréquents des API des grands réseaux sociaux conventionnels (exemple Facebook et Twitter) pour de multiples raisons (sécurité, vie privées, évolution, nouvelles fonctionnalités, etc.)

En plus de ces difficultés techniques auquel il est important des les mentionner et qu'il faut résoudre, la principale problématique peut se traduire par la question générale suivante : «Comment tirer profit du modèle des réseaux sociaux pour optimiser la gestion des services Web et comment peut on exploiter les connaissances ou flux générés des les réseaux sociaux pour faciliter la création (la découverte et/ou recommandation) des services par et/ou pour les utilisateurs finaux ?». Il est important de rappeler que le terme «utilisateur final» peut désigner un utilisateur lambda qui évolue dans un réseau social donnée mais peut également désigner un développeur ou un concepteur d'application basés sur les services ou les recommandation de services. Ce dernier point est encore une tendance actuelle qui qualifie la situation actuelle du Web et du développement de services. En effet, avec l'avènement et l'exposition des dispositifs hétérogènes (telle que les objets connectés), les services Web représentent désormais les candidats idéaux pour des applications simples (dans le sens de la lourdeur du code informatique), compatibles, interopérables et réutilisables de n'importe quel terminal, objet, système d'exploitation, ou depuis n'importe quel type de connexion.

1.3 Contribution

Dans ce travail, nous allons nous focaliser sur le problème de la découverte des services Web et la sélection de ces services avec des besoins d'utilisateurs finaux ainsi que la perpétuelle expansion des services Web et leur nature versatile. Par ce travail, nous souhaitons contribuer à l'enrichissement des quelques investigations menées jusqu'à maintenant concernant ce nouveau domaine qui est le mariage entre réseaux sociaux et services Web. Notre retour d'expérience peut servir à mettre à jour les connaissances en ce qui concerne la situation actuelle du Web. De plus, les outils sélectionnés, après de nombreux tests parfois non concluants, peuvent être adoptés pour une expérimentation/évaluation d'approches visant à développer des services avec une dimension de réseaux sociaux. Notre simulation de modèle proposé devrait permettre la mise en échelle des services et la personnalisation de plateformes que les développeurs souhaiteraient mettre en place.

Chapitre 2

Contexte

Sommaire

2.1 Introduction

Dans ce chapitre, nous allons situer le contexte de notre travail autours des services Web et les réseaux sociaux au sens large du terme. Comme mentionné en introduction, le thème de notre travail est large et implique de nombreux standards et technologies. Certains concepts sont très matures comme les services Web et les architectures SOA. Cependant, d'autres concepts sont relativement plus récents notamment en ce qui concerne les réseaux sociaux et surtout la génération de ce dernier paradigme en incluant les modèles sociaux classiques qui impliquant des utilisateurs humains (modèles classiques) mais aussi les modèles intra-services qui impliquent les interactions entre services Web eux mêmes sans négliger l'implication d'utilisateurs «finaux» au sens présenté dans l'introduction.

2.2 Les services Web

Dans cette section, nous allons décrire les services Web en citant leurs intérêt, en les situant dans une architecture générale et en présentant les principales applications avec la notion de qualité de service.

2.2.1 Description

L'activité de standardisation relative aux services Web du consortium W3C [24] définit un service Web comme une application ou un composant logiciel qui vérifie les propriétés suivantes :

— Il est identifié par un URI (une généralisation de URL).
— Ses interfaces et ses liens (appelés aussi *binding*) peuvent être décrits avec le format XML.
— Sa définition peut être découverte par d'autres services Web.
— Il peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant les protocoles d'Internet.

La définition exacte du terme *service Web* est encore à ce jour fortement discutée. Les différentes définitions proposées s'accordent au moins sur l'idée qu'un service Web est un nouveau type de composant logiciel ayant la capacité de publier ses fonctions sur Internet sous forme de services, de rendre ces services facilement invocables et de les mettre à disposition par l'intermédiaire de protocoles Internet standardisés tels que le protocole HTTP qui est largement utilisé par les navigateurs Internet ou tout autre protocole comme ceux basés sur le format XML.

2.2.2 L'intérêt des services Web

Les services Web comportent de nombreux avantages, ils sont utilisables à distance via n'importe quel type de plateforme notamment avec l'avènement des objets connectés et de l'Internet des objets (IoT). Par ailleurs, le Web

est devenu le moyen préféré des développeurs car les librairies des navigateurs sont disponibles, modulables et facilement intégrables (à bas coût) dans des objets non conventionnels tels que les montres et les lunettes connectées. Par conséquent, les services Web peuvent servir au développement d'applications interopérables et distribuées et sont accessibles depuis n'importe quel type de clients. Les services Web appartiennent à des applications capables de collaborer entre elles de manière transparente pour l'utilisateur final.

2.2.3 Les application des services Web

Les technologies des services Web peuvent être appliquées à toutes sortes d'applications auxquelles elles offrent des avantages considérables en comparaison aux anciennes API propriétaires, aux implémentations spécifiques à une plateforme donnée et à quelques autres restrictions classiques que l'on peut rencontrer (multiplateformes, multi-langages, disponible sur Internet avec une information actualisée disponible en temps réel, etc.). Les applications des services Web sont multiples, autant dans les domaines du Business to Consumer (B2C), Business to Business (B2B) que dans des domaines de gestion, par exemple pour la gestion de stock, gestion commerciale, etc.

* **B2C (Business to Consumer)** : qualifie une application, un site Internet destiné au grand public.
* **B2B (Business to Business)** : qualifie une application, un site Internet destiné au commerce de professionnel à professionnel.

Les services Web peuvent être utiles dans la plupart des scénarios applicatifs lorsque la communication peut être établie sur un modèle à double sens (bidirectionnel) avec des séries de requêtes suivie par des réponses. C'est néanmoins loin d'être aussi limitatif. Beaucoup d'autres modèles peuvent avoir recours aux services Web en toute transparence pour l'application de l'usager. Les entreprises qui mettent à disposition leurs services Web permettent aux développeurs intéressés par ses fonctionnalités de les réutiliser sans avoir à les reprogrammer de nouveau d'où un gain énorme coût et en compatibilité. Le principe des services Web permet d'avoir un partage des fonctionnalités et facilite grandement le développement logicielle et d'une manière plus général

le développement de services.

2.2.4 L'architecture générale des services Web

Jusqu'ici, l'accès via Internet à une ressource applicative donnée ou à une base de données s'effectuait par l'envoi d'une requête en s'appuyant sur des langages conventionnels tels que le PHP ou le JSP. Il s'agissait donc d'un dialogue entre une couche de présentation reposant sur le format HTML (échangé avec le protocole HTTP) et des applications installées sur un serveur distant. Avec les services Web, un dialogue est désormais instauré entre les applications qui peuvent être installées sur des machines distantes, et ceci grâce à des standards basés sur l'écosystème XML. En effet, afin de dialoguer via Internet, ces applications doivent «parler» le même langage. Ce dernier est souvent basé sur les standards XML.

Dans la Figure 2.1, l'architecture de référence des services Web se base sur les trois concepts suivants :

— Le fournisseur de service : c'est le propriétaire du service.
— Le client (ou le consommateur de service) : c'est un demandeur de service. D'un point de vue technique, il est constitué par l'application qui va rechercher et invoquer un service.
— L'annuaire des services : c'est un registre de descriptions de services offrant des facilités de publication de services à l'intention des fournisseurs ainsi que des facilités de recherche de services à l'intention des clients.

Les interactions de base qui existent entre ces trois éléments sont principalement les opérations de : publication, de recherche, d'invocation et de lien ou de liaison (appelé aussi *Bind*).

Pour bien comprendre le fonctionnement de cette architecture, nous expliquons ci-dessous le rôle de chacun des éléments précédents :

— Le fournisseur de service créé le service Web, puis publie son interface ainsi que les informations d'accès au service, dans un annuaire de services

Web.

— L'annuaire de service rend disponible l'interface du service ainsi que ses informations d'accès pour n'importe quel demandeur potentiel de service.

— Le consommateur de service accède à l'annuaire de service pour effectuer une recherche afin de trouver les services désirés. Ensuite, il se lie au fournisseur pour invoquer le service.

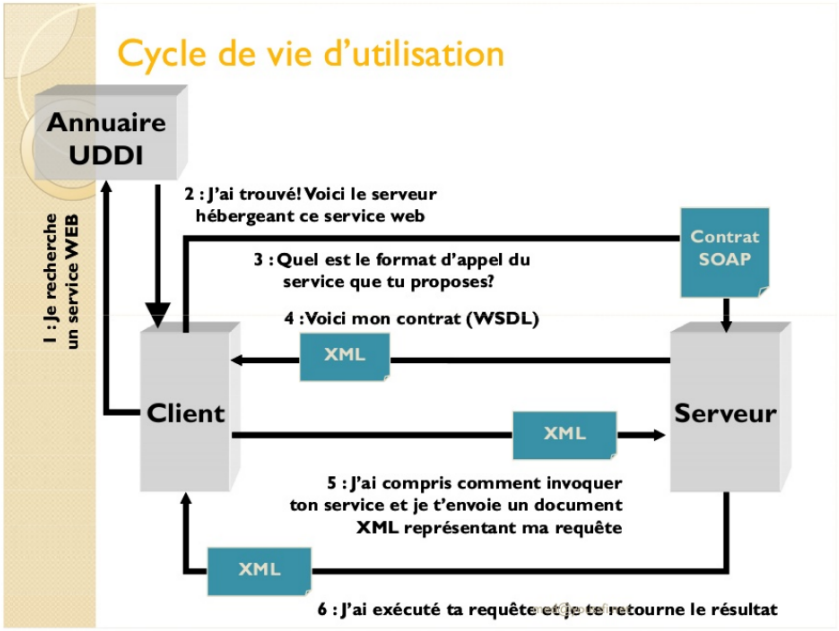

FIGURE 2.1 – Cycle de vie de l'utilisation des services Web

2.2.5 Qualité de service : QoS

Chaque service est identifié par une interface et cette interface comporte des propriétés fonctionnelles et éventuellement des propriétés non-fonctionnelles permettant à un consommateur de trouver le service qui convient à ses besoins. La qualité de service (QoS) représente les propriétés non-fonctionnelles d'un service. Ces propriétés correspondent aux capacités d'un service à réaliser ses fonctionnalités en termes de temps de réponse, de disponibilité, etc. Cette QoS comporte donc plusieurs dimensions telles que le temps de réponse, la réputation, le prix, etc. Plusieurs catégorisations des attributs de QoS sont

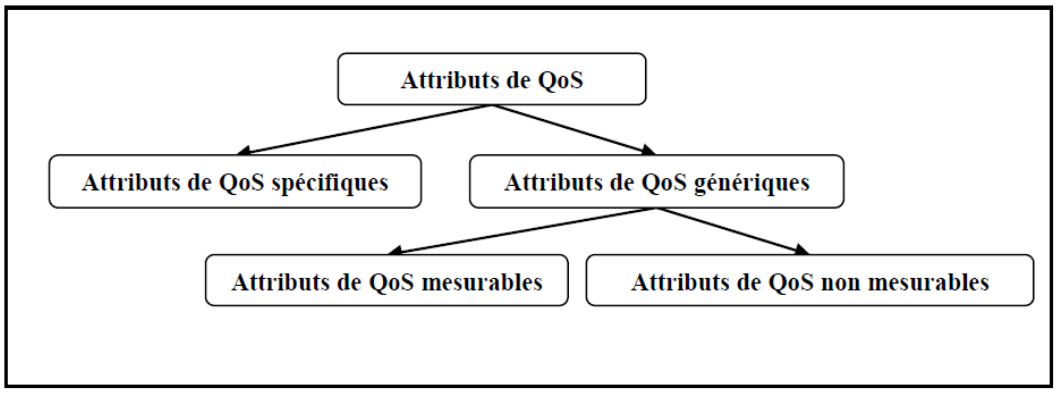

FIGURE 2.2 – Classification des attributs de qualité de service

proposées. Ces attributs peuvent être regroupés en qualitatifs et quantitatifs, génériques et spécifiques, etc. La Figure 2.2 montre une possible catégorisation des attributs de qualité.

2.3 Les principales technologies de développement de service Web

Dans cette section, nous allons discuter les principaux standards et normes impliqués dans la conception et le développement des services Web. Une caractéristique qui a permis un grand succès de la technologie des services Web est qu'elle est construite sur des technologies standard ouvertes impliquants des scientifiques mais aussi l'industrie. Par la suite, nous allons donc présenter les principales technologies.

2.3.1 eXtensible Markup Language (XML)

Le langage eXtensible Markup Language (XML) est un standard promulgué par le consortium W3C qui est l'organisme chargé de standardiser de nombreuses technologies relatives à l'évolution du Web. On retrouve dans le format XML une généralisation des idées contenues dans le format HTML. Dans le format HTML, on n'utilise les balises que pour décrire l'aspect

graphique (présentation) que doit revêtir la page dans un navigateur Web. Dans XML, l'accent est mis sur les types de données et parfois leur sémantique. Les balises permettent d'associer toute sorte d'informations au fil du texte. XML a été conçu pour des documents arbitrairement complexes (notamment pour leur structure arborescente et hiérarchique) tout en s'appuyant sur cinq grands principes simples et clairs :

— La lisibilité à la fois pour les machines et pour les utilisateurs.
— La définition sans ambiguïté du contenu d'un document.
— La définition sans ambiguïté de la structure d'un document.
— La séparation entre documents et relations qui peuvent exister entre les documents.
— La séparation entre la structure du document et la présentation du document.

Le contenu d'un document XML est décrit par une succession d'éléments qui sont en fait des blocs de texte encadrés par des paires de balises ouvrantes («>») et fermantes («<»). Ce sont les «unités de contenu». Ces éléments sont liés entre eux par une organisation hiérarchique : certain éléments apparaissant imbriqués dans d'autres. Le format XML et la définition des types de donnés utilisé dans un document XML (par le biais de DTD : *Document Type Definition*) assurent une séparation effective entre le contenu et de la structure de document.

2.3.2 Les espaces de noms XML (namespaces)

Les espaces de noms XML (appelés aussi *namespaces*) est une *recommandation* du W3C qui été rapidement adopté après la spécification XML 0.1. Il est à noter que dans le consortium W3C, en général, les propositions commencent dans un état de *draft* (document brouillon) et finissent par devenir dans l'état *recommandation* après que les propositions de standardisation deviennent matures ce qui passe auparavant par de nombreuses et longues discussions impliquant des groupes académiques et industriels. La recommandation des espaces de noms vise donc à résoudre les problèmes d'ambiguïté éventuelle des balises dans un document XML

donné. Les namespaces permettent ainsi de résoudre le problème éventuel des différentes interprétations possible d'un même document XML par des applications différentes. En s'appuyant sur le dispositif des URI (utilisé comme identifiant d'un espace de nom donné), on arrive à assurer l'unicité du sens des balises. C'est exactement comme préciser le dictionnaire à utiliser pour un «mot» identique mais qui peut avoir deux sens différent selon le dictionnaire choisi. Les balises et les attributs XML sont alors dotés d'une interprétation spécifique sans ambiguë.

2.3.3 Les schémas XML

La recommandation appelée *XML schéma* a été adoptée après de longues discussions dans les comités techniques du W3C. Elle représente un réel tour de force et une innovation dans l'utilisation de XML rompant clairement avec l'usage original de la publication des documents.

XML schéma précise comment représenter en XML la structure des données en générale – ce qu'on a l'habitude d'appeler les métadonnées dans le monde des bases de données relationnelles (description des tables, des colonnes, de leurs types, etc.). XML schéma est adapté à la description des structures de données des documents XML, des bases de données relationnelles, mais aussi à celle des modèles objet des langages de programmation orientés objet comme JAVA et C++. Un schéma XML définit, d'une part, l'imbrication des éléments entre eux, ce qui s'apparente aux DTD, et d'autre part, le type des éléments et de leurs attributs. L'information fournie par le schéma est donc plus riche que celle dans le DTD. On peut donc conclure que pour des document XML simples l'usage des DTD est suffisant. Cependant, pour un usage plus propre, complexe et professionnel l'usage des schémas XML s'impose.

2.4 Le protocol *Simple Object Access Protocol* (SOAP)

Dans cette section, nous nous intéressons à l'aspect protocolaire qui va garantir que les échanges et interactions impliquant les services Web se dérouleront d'une manière standard et normalisée.

2.4.1 Présentation

Le protocol *Simple Object Access Protocol* (SOAP) [27] est un protocole adopté par le consortium W3C dans le but de favoriser l'échange d'information d'une manière ouverte et interopérable. Si l'on voulait traduire le terme SOAP en français cela donnerait : Protocole Simple d'Accès aux Objets. En effet, le protocole SOAP consiste à faire circuler de donnés encodées généralement en XML, via le protocole HTTP avec le port 80 par défaut (ou en utilisant un autre protocole). Ce choix facilite grandement les communications car le XML est un langage standard et le port utilisé est le port 80 (le même utilisé pour toute navigation Internet sur le Web) ce qui ne pose donc pas de problème pour les firewalls (pare-feu) car il n'est quasiment jamais bloqué contrairement à d'autres protocoles qui emploient d'autre ports ou qui change dynamiquement le port (exemple, le protocole RTP pour les streams audio et vidéo).

Le protocole SOAP a été crée en septembre 1998, avec la version 0.9, par trois grand entreprises :

Microsoft, UserLand et DevelopMentor. IBM n'a participé au protocole SAOP qu'à partir de la version 1.1 en avril 2000. C'est cette même année que SOAP a été soumis au W3C. Depuis septembre 2000, SAOP 1.1 est en refonte complète pour donner jour à la version 1.2 avec un groupe de travail de plus de 40 entreprises. Dedans, on trouve évidemment Microsoft, IBM mais aussi HP, Sun, INTEL, etc.

2.4.2 Caractéristiques

Afin de simplifier la compréhension de SOAP, nous proposons de récapituler les principales caractéristiques du protocole qui sont les suivantes :

- SOAP permet une normalisation des échanges de données. Les données sont encodées en XML et échangées par des appels de procédures à distance en utilisant un protocole standard de communication tel que le HTTP, SMPT, ou POP.
- SOAP est simple, extensible et permet le diagnostic facile des erreurs.
- Les messages SOAP sont unidirectionnel : dans le sens Requête → Réponse.
- Le protocole fonctionne de manière synchrone et asynchrone.
- Le protocole est indépendant de la plateforme et du langage utilisé par l'application.
- Il n'est pas perturbé par un pare-feu.

2.4.3 Fonctionnement

Puisque le fonctionnement de SOAP est basé sur le modèle requête/réponse ce qui engendre au moins un client et un serveur, la meilleure manière de décrire le fonctionnement du protocole est de voir ce qui se passe du côté client ainsi que du côté serveur.

Coté client : le client envoie des messages au serveur correspondant sous forme des requêtes SOAP-XML enveloppées (encapsulées) dans des requêtes http classiques. De même, les réponses du serveur sont des réponses transportées avec le protocole HTTP qui fournit des réponses sous forme SOAP-XML. Dans le processus client, l'appel d'un service est converti en une requête SOAP qui est ensuite enveloppée dans une requête HTTP.

Coté serveur : c'est légèrement plus complexe comaparant au processus du client. En effet, le serveur requiert un processus dit *listener* qui est en fait un processus tournant continuellement sur la machine du serveur en attente des

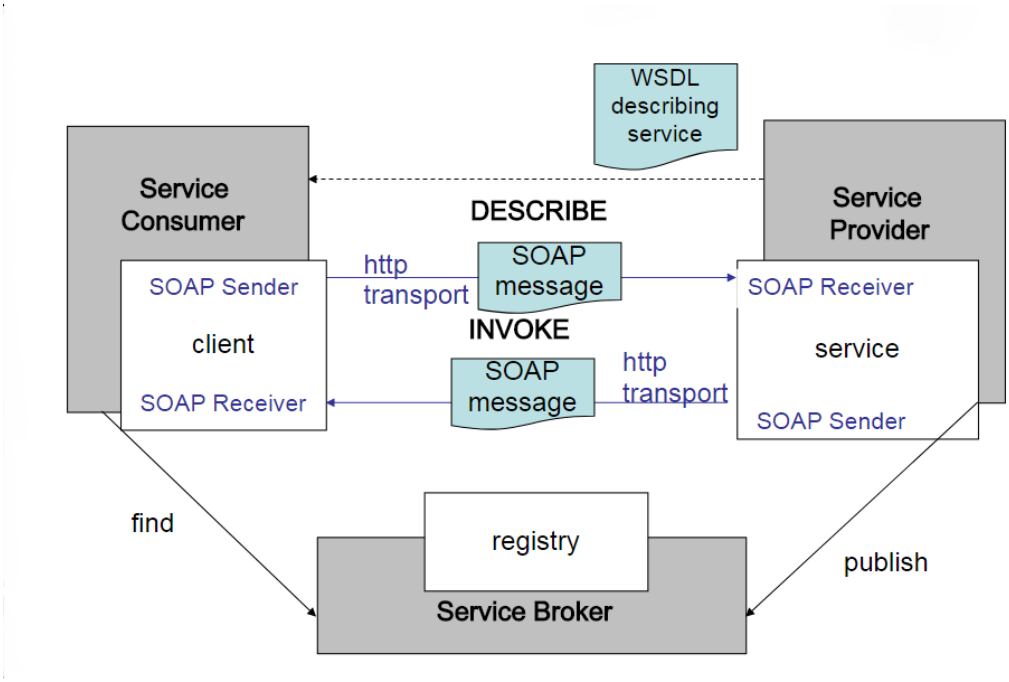

FIGURE 2.3 – Traitement d'un message SOAP

connexions clientes. Le listener est souvent implémenté au travers d'un *servlet* qui a pour tâche d'extraire le message XML-SOPA de la requête http reçue, de le désérialiser c'est à dire d'extraire et de séparer le *nom de la méthode* et les *paramètres* fournis puis invoquer la méthode du service en conséquence. C'est exactement comme si, dans un code de programmation classique, on identifie la fonction à appeler et on luit fournit les arguments nécessaires à cette fonction. Le résultat de la méthode est alors sérialisé, encodé en HTTP et renvoyé au demandeur.

2.5 La technologie *Web Service Description Language* (WSDL)

Nous allons, dans cette section, nous intéresser à la description des services Web à l'aide de la technologie la plus utilisée pour cela. Il s'agit de la norme *Web Service Description Language* (WSDL).

2.5.1 Définitions générale

Le WSDL est un langage dérivé de XML permettant de fournir les spécifications nécessaires à l'utilisation d'un service Web en décrivant les méthodes, les paramètres et les résultats retournés.

Le WSDL est aussi l'équivalent de *Interface Definition Language* (IDL) pour la programmation distribuée avec les architectures de type *Common Object Request Broker Architecture* (CORBA). Ce langage permet de décrire de façon précise les services Web, en incluant des détails tels que les protocoles, les serveurs, les ports utilisés, les opérations pouvant être effectuées, et les formats des messages d'entrée et de sortie.

Dans la littérature, Il y a eu d'autres tentatives de propositions de langages pour résoudre les problématiques liées à la définition des services Web. Microsoft a d'abord proposé *Service Definition Language* (SDL) avec une implémentation fournie dans leur Toolkit SOAP, puis IBM a proposé *Network Accessible Service Specification Language* (NASSL), dont une implémentation est fournie dans l'environnement SOAP4J. Microsoft modifia sa première spécification et proposa *SOAP Contract Language* (SCL). Enfin, les différents acteurs se sont accordés sur la norme WSDL.

2.5.2 Structure d'un document WSDL

Un fichier WSDL contient principalement sept éléments (voir Figure 2.4) que nous détaillons comme suit :

— **Types** : fournit la définition des types de données utilisés pour décrire les messages échangés.
— **Messages** : représente une définition abstraire (noms et types) des données en cours de transmission.
— **PortType** : décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou des erreurs.

— **Binding** : spécifie une liaison entre un <portType> et un protocole concret (exemple : SOAP, HTTP, etc.).

— **Service** : indique les adresses de port de chaque liaison.

— **Port** : représente un point d'accès de services défini par une adresse réseau et une liaison.

— **Opération** : c'est la description d'une action exposée dans le port.

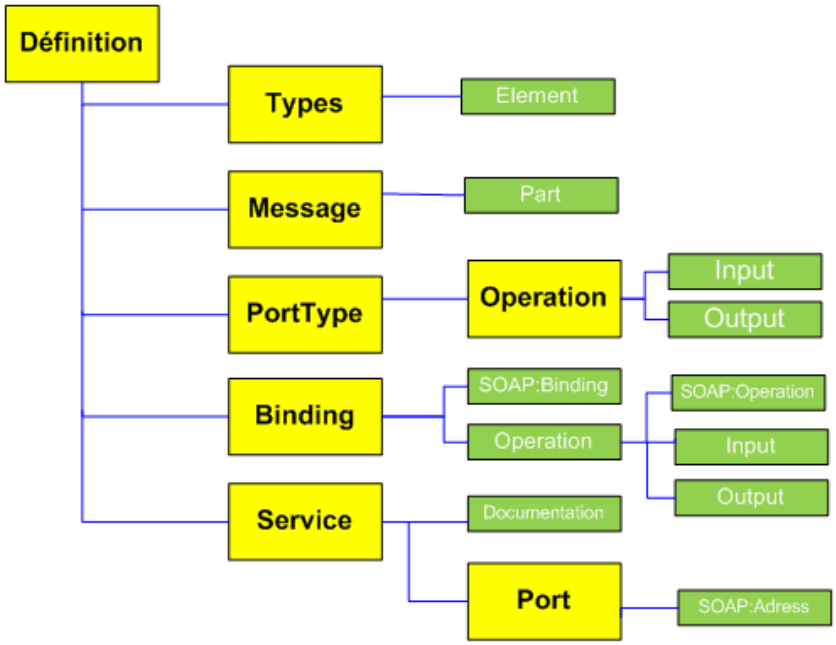

FIGURE 2.4 – La structure générale d'un document WSDL

Afin de faciliter la compréhension de la structure, nous donnons dans la suite un exemple de service Web décrit en WSDL (Listing 2.1). Cet exemple est extrait du dépôt (repository) des services issue des travaux du projet ASSAM [1] que nous avons analysé durant les travaux de ce mémoire. Ce jeu de données (dataset) a été également utilisé par le premier travail scientifique, auquel nous nous appuyons fortement dans ce mémoire, qui a introduit pour la première fois, en 2011, la notion de l'utilisation du concept des réseaux sociaux pour la découverte de service Web (travail de recherche de Pr. Zakaria Maamar [14]).

Le service du dataset ASSAM (dataset étudié dans ce mémoire) et présenté dans Listing 2.1, est le service *Utility* (service numéro 2 de la catégorie information (*news*)). Ce service extrait des news commerciales dans plusieurs

catégories. Il est dédié à être facilement intégrable dans des applications ou des sites en ligne. Comme nous pouvons le constater, ce services référence plusieurs namesapces (neuf en tout, lignes 4-12, Listing 2.1), par exemple le namespace *tm* de valeur *http ://microsoft.com/wsdl/mime/textMatching/*.

Les types utilisés dans ce service sont représentés par 10 éléments employant 4 types complexes ou composites (lignes 16-29, Listing 2.1). Prenons l'exemple du type complexe *BusinessShortNews*, ligne 25, Listing 2.1. Comme le montre Listing 2.2, ce type comporte plusieurs variables comme la date de l'information, le titre du news, et la source de l'information (Listing 2.1, lignes 5, 7, et 10 respectivement).

Le service comporte aussi 24 messages différents. Etant donné la taille du document, nous avons délibérément présenté seulement 2 messages (lignes 33 et 36, Listing 2.1) qui sont *GetIndustryNewsHttpGetIn* et *GetIndustryNewsHttpGetOut*. Notons que contenu du premier message a un type simple (chaîne de caractère), tandis que le deuxième message comporte un contenu de type complexe nommé *ArrayOfBusinessShortNews* (ce type est cité dans la balise «Types», ligne 25, Listing 2.1). Trois «PortType» ont été utilisés dans ce service (lignes 40, 41 et 51). Le deuxième PortType, appelé «BusinessNewsHttpGet», est détaillé dans notre listing. Comme nous pouvons le voir, il comporte 4 opérations dont la première «GetNewsCategory» est détaillée dans les lignes 43-45. La partie «Binding» du service étudié, comporte 3 bindings, dont «BusinessNewsHttpGet» est détaillé (ligne 54). On voit, en effet, l'utilisation du protocole HTTP (avec la méthode GET) avec 4 opérations possibles dont la première (GetNewsCategory, ligne 56) est détaillée en termes de Input (sous forme d'URL) et de Output (sous frome de réponse incluse dans le corps de la réponse HTTP). Le récapitulatif du service comporte trois «port» dont le premier (ligne 72) détient comme point de liaison la valeur *http ://glkev.Webs.innerhost.com/glkev_ws/businessnews.asmx*, comme on peut le voir dans la ligne 73.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <definitions
3 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
4 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://www.myasptools.com/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.myasptools.com/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <!-- PARTIE Types : -->
  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.myasptools.com/">
      <s:element name="GetNewsCategory">...</s:element>
      <s:element name="GetNewsCategoryResponse">...</s:element>
      <s:complexType name="ArrayOfNewsCategory">...</s:complexType>
      <s:complexType name="NewsCategory">...</s:complexType>
      <s:element name="GetLicNewsCategory">...</s:element>
      <s:element name="GetLicNewsCategoryResponse">...</s:element>
      <s:element name="GetIndustryNews">...</s:element>
      <s:element name="GetIndustryNewsResponse">...</s:element>
      <s:complexType name="ArrayOfBusinessShortNews">...</s:complexType>
      <s:complexType name="BusinessShortNews">...</s:complexType>
      <s:element name="GetLicIndustryNews">...</s:element>
      <s:element name="GetLicIndustryNewsResponse">...</s:element>
      <s:element name="ArrayOfNewsCategory" nillable="true" type="s0:ArrayOfNewsCategory" />
      <s:element name="ArrayOfBusinessShortNews" nillable="true"
      type="s0:ArrayOfBusinessShortNews" />
    </s:schema>
  </types>
  <!-- PARTIE Messages : -->
  <message name="GetIndustryNewsHttpGetIn">
    <part name="industry" type="s:string" />
  </message>
  <message name="GetIndustryNewsHttpGetOut">
    <part name="Body" element="s0:ArrayOfBusinessShortNews" />
  </message>
  <!-- PARTIE PortType : -->
  <portType name="BusinessNewsSoap">...</portType>
  <portType name="BusinessNewsHttpGet">
    <operation name="GetNewsCategory">
      <documentation>This method retrives a list of news categories.</documentation>
      <input message="s0:GetNewsCategoryHttpGetIn" />
      <output message="s0:GetNewsCategoryHttpGetOut" />
    </operation>
    <operation name="GetLicNewsCategory">...</operation>
    <operation name="GetIndustryNews">...</operation>
```

```
49      <operation name="GetLicIndustryNews">...</operation>
50    </portType>
51    <portType name="BusinessNewsHttpPost">...</portType>
52    <!-- PARTIE Binding : -->
53    <binding name="BusinessNewsSoap" type="s0:BusinessNewsSoap">...</binding>
54    <binding name="BusinessNewsHttpGet" type="s0:BusinessNewsHttpGet">
55      <http:binding verb="GET" />
56      <operation name="GetNewsCategory">
57        <http:operation location="/GetNewsCategory" />
58        <input>
59          <http:urlEncoded />
60        </input>
61        <output>
62          <mime:mimeXml part="Body" />
63        </output>
64      </operation>
65      <operation name="GetLicNewsCategory">...</operation>
66      <operation name="GetIndustryNews">...</operation>
67      <operation name="GetLicIndustryNews">...</operation>
68    </binding>
69    <binding name="BusinessNewsHttpPost" type="s0:BusinessNewsHttpPost">...</binding>
70    <!-- PARTIE Service : -->
71    <service name="BusinessNews">
72      <port name="BusinessNewsSoap" binding="s0:BusinessNewsSoap">
73        <soap:address location="http://glkev.Webs.innerhost.com/glkev_ws/businessnews.asmx" />
74      </port>
75      <port name="BusinessNewsHttpGet" binding="s0:BusinessNewsHttpGet">...</port>
76      <port name="BusinessNewsHttpPost" binding="s0:BusinessNewsHttpPost">...</port>
77    </service>
78
79  </definitions>
```

Listing 2.1 – Exemple de description de service

```
1       <s:complexType name="BusinessShortNews">
2         <s:sequence>
3           <s:element minOccurs="0" maxOccurs="1" name="Type" type="s:string" />
4           <s:element minOccurs="0" maxOccurs="1" name="Time" type="s:string" />
5           <s:element minOccurs="0" maxOccurs="1" name="Url" type="s:string" />
6           <s:element minOccurs="0" maxOccurs="1" name="Headline" type="s:string" />
7           <s:element minOccurs="0" maxOccurs="1" name="DocumentUrl" type="s:string" />
8           <s:element minOccurs="0" maxOccurs="1" name="Cluster" type="s:string" />
9           <s:element minOccurs="0" maxOccurs="1" name="Source" type="s:string" />
10        </s:sequence>
```

```
11      </s:complexType>
```

Listing 2.2 – Type composite "BusinessShortNews"

2.6 Universal Description Discovery and Integration (UDDI)

Dans cette section, nous présentons l'aspect recensement de services ou comment peut on gérer et stocker et l'apparition des services avec des outils et normes d'annuaire.

2.6.1 Définitions

L'annuaire des services *Universal Description Discovery and Integration* (UDDI) est un standard pour la publication et la découverte des informations sur les services Web. La spécification UDDI est une initiative lancée par ARIBA, Microsoft et IBM. Cette spécification n'est pas gérée par le W3C mais par le groupe OASIS. La spécification UDDI vise à créer une plateforme indépendante, un espace ou cadre de travail (framework) ouvert pour la description, la découverte et l'intégration des services des entreprises.

2.6.2 Consultation de l'annuaire

L'annuaire UDDI se concentre sur le processus de découverte de l'architecture orientée services (SOA) et utilise des technologies standard telles que XML, SOAP et WSDL. Ces technologies permettent de simplifier la collaboration entre partenaires dans le cadre des échanges dans la plus part commerciaux. L'accès au référentiel s'effectue de différentes manières. Voici quelques exemples :

— Les pages blanches : elles comprennent la liste des entreprises ainsi

que des informations associées à ces dernières (exemples : coordonnées, description de l'entreprise, identifiants, etc.).

— Les pages jaunes : elles recensent les services Web de chacune des entreprises sous le standard WSDL.

— Les pages vertes : elles fournissent des informations techniques précises sur les services fournis.

Le scénario général est le suivant. Les entreprises publient des descriptions de leurs services Web en utilisant un annuaire UDDI sous la forme de fichiers WSDL. Ainsi, les clients peuvent plus facilement rechercher les services Web dont ils ont besoin en interrogeant le registre UDDI. Lorsqu'un client trouve une description de service Web qui lui convient, il télécharge son fichier WSDL depuis le registre UDDI. Ensuite, à partir des informations inscrites dans le fichier WSDL, notamment la référence vers le service Web, le client peut invoquer le service Web et lui demande d'exécuter certaines de ses fonctionnalités ou opérations.

Le scénario classique d'utilisation de UDDI est illustré dans Figure 2.5. Dans cet exemple, l'entreprise B a publié le service Web S, et l'entreprise A est cliente de ce service. L'entreprise A peut donc trouver le service, grâce à l'annuaire, et invoquer les opérations requises en sollicitant l'entreprise B directement. Comme expliqué précédemment, les requêtes et les réponses obéissent à un protocole donné tels que http ou SOAP.

2.7 Les architectures orientée service (SOA)

Dans cette section, nous nous intéressons à l'organisation de l'écosystème des services sous forme architecturale et organisation des différentes composantes impliquées.

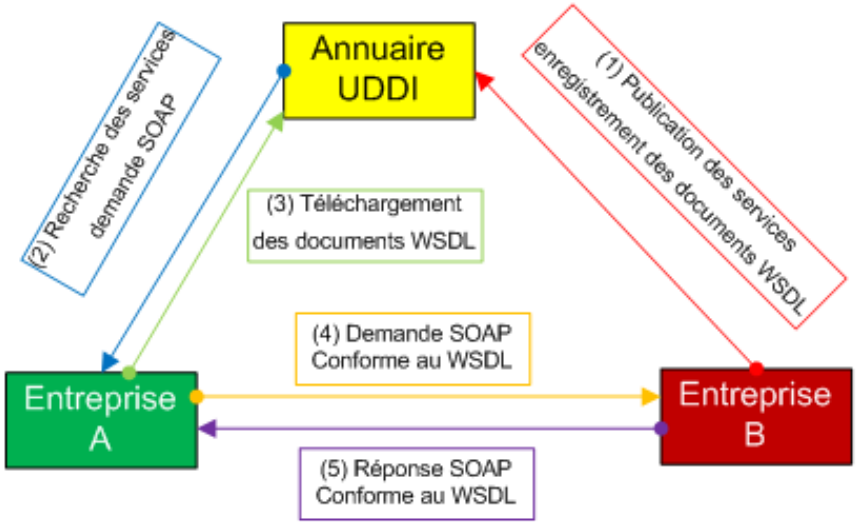

FIGURE 2.5 – Scénario classique d'utilisation d'un annuaire UDDI

2.7.1 Définition

Les architectures orientée service (SOA) représentent un nouveau style architectural qui facilite l'intégration de plusieurs entités logicielles en les organisant en un ensemble fonctionnel appelées services. L'objectif de cette intégration c'est de réaliser une fonctionnalité dont l'utilisateur à besoin. Cette architecture repose sur un concept basique généralisé qui est la notion de «service». Ici, un service peut être vu comme une application définie par un contrat et qui est accessible par une interface qui offre une ou plusieurs fonctionnalités. Les services permettent de communiquer via l'échange de messages afin de cacher l'hétérogénéité des environnements de développements des applications. L'utilisation de l'architecture orientée service a apportée plusieurs avantages tel que :

— Le couplage faible : les acteurs partagent entre eux seulement les descriptions de services afin de réduire la dépendance entre les applications.

— La substituabilité : en cas d'une mise à jour d'un tel service (exemple retrait d'un service), la SOA permet de le remplacer facilement.

— La facilité d'intégration des services : permet de masquer l'hétérogénéité

des systèmes et des plateformes de développement des services en les représentants comme des boites noires.

— La réutilisabilité : un service peut être combiné (on dit parfois «composé») avec d'autres services afin de réaliser une certaine fonctionnalité plus riche ou plus complexe.

— La facilité des communications externes : le client peut communiquer avec des outils applicatifs externes à l'entreprise.

2.7.2 Les acteurs SOA

Les organisations SOA s'articulent autours de trois principaux acteurs (fournisseurs, registres et clients) qui sont présentés dans cette section dans l'ordre chronologique de leurs interactions. La Figure 2.6 présente le modèle d'une architecture SOA.

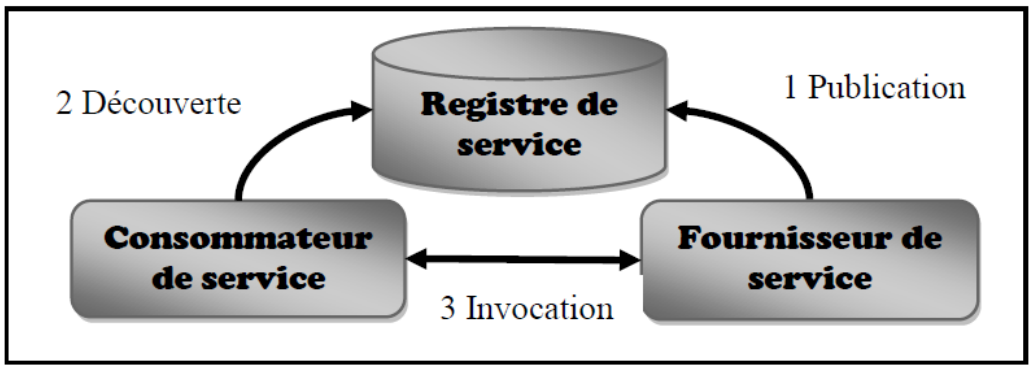

FIGURE 2.6 – Les architectures orientée service (SOA)

Les acteurs qui sont en interactions dans ce type d'architecture sont :

— Le fournisseur de service : il représente une personne ou une organisation qui permet d'exposer la description (effectuer la publication) de son service dans un registre de service afin de le rendre accessible aux utilisateurs potentiels.

— Le registre de service : c'est un répertoire de service (appelé aussi annuaire ou courtier de service). Il permet aux fournisseurs de stocker leurs descriptions et aux consommateurs de rechercher et de choisir le service correspondant à son besoin. Ce registre joue un rôle intermédiaire

entre le fournisseur et le consommateur de service.

— Le consommateur de service : c'est l'utilisateur de service qui peut être une personne, une application ou un serveur. Il lance une recherche (découverte) dans l'annuaire afin de voir s'il existe un service qui répond à son besoin. Il utilise la description de ce service pour établir une connexion avec son fournisseur afin de l'utiliser (invocation).

Les SOA suivent de près le modèle client/serveur avec en plus un répertoire de service. Pour récapituler :

1. Le fournisseur de services publie la description de son service.

2. Le client du service fait une recherche du service correspondant à ses besoins et critères.

3. Le client reçoit la description du service et utilise cette description afin d'invoquer le service. Il invoque le service, sous forme d'un message auprès du site du fournisseur.

2.7.3 SOA et la dynamicité des services

SOA propose des moyens pour interagir avec les environnements dynamiques afin de satisfaire les besoins des consommateurs. En effet, les services sont caractérisés par un aspect dynamique. Cette dynamicité peut représenter des changements qui touchent l'environnement d'exécution des services. Ces changements peuvent se traduire par des opérations d'ajout, de suppression ou de mise à jour d'un service.

2.8 Réseaux sociaux

Dans cette section, nous allons présenter le contexte des réseaux sociaux et sa principale caractérisation.

2.8.1 Définition

D'une manière générale, Il existe plusieurs types de réseaux en pratique. On peut définir un réseau comme "un ensemble d'entités avec un contenu (caratère) spécifique et des relations entre les entités". On peut définir le réseau social comme une communauté d'individus (ou d'entités au sens large) reliés entre eux, selon les cas, par des critères basés sur les origines, des centres d'intérêts, des besoins et des points de vue proches ou similaires. Cette communauté peut être représentée par un ensemble de nœuds (ou nodes) reliés entre eux par des liens (Figure 2.7).

Comme tout réseau, le réseau social est voué à se développer rapidement. Les réseaux sociaux peuvent être vus comme une sorte de réseau formé d'entités et de relations entre elles, où les entités peuvent interagir entre elles avec de multiples manières et pour différentes raison fondées sur des motivations sociales. Il existe de nombreux réseaux sociaux typiques, tels que Facebook, WeChat, Twitter, etc. Bien qu'un réseau social peut être vu, par les individus, comme un monde virtuel, ses caractéristiques, sa variation et sa structure communautaire reflètent le fonctionnement du monde réel. Comme nous allons voir dans la suite de ce mémoire, de plus en plus de recherche académique et industrielle se focalisent sur ce phénomène et sur la maîtrise des communautés et de leur évolution.

Le principe de communauté, principe fondateur des réseaux sociaux, peut être défini comme une structure reliant plusieurs entités (sommets) et formant ainsi une structure de graphe. On peut s'intéresser à l'aspect graphique et définir une communauté comme un sous ensemble dense où plusieurs connexions existent entre un sous ensemble de sommets. On peut aussi penser à que les sommets d'une même communauté ont, en quelques sortes, plusieurs similitudes. Il est clair, qu'à ce jour, aucune définition n'est universellement acceptée. En effet, la définition de réseaux sociaux ainsi que celle des communautés dépend profondément du système ou de l'application spécifique qu'un chercheur ou développeur aborde.

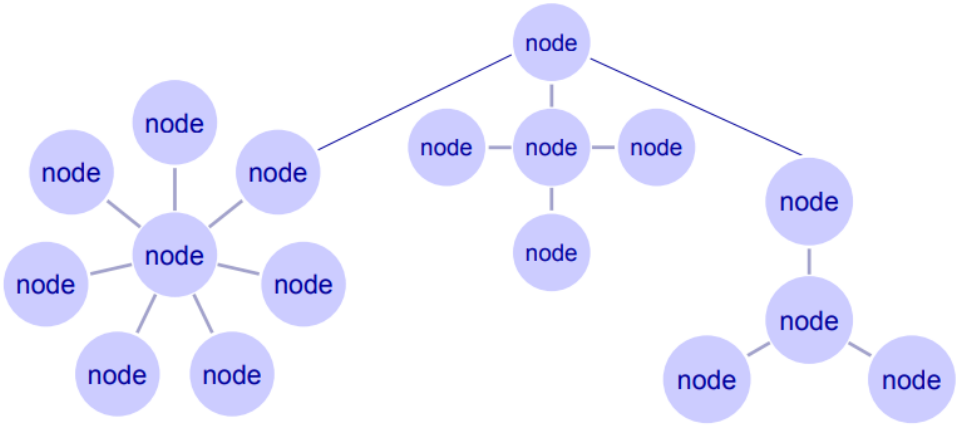

FIGURE 2.7 – Réseaux sociaux

2.8.2 Caractérisation d'un réseau social

Pour simplifier la présentation de ce concept complexe, nous pouvons caractériser un réseau social par deux principales propriétés :

— Un réseau social est un ensemble d'acteurs (individus, nœuds, groupes ou organisations) reliés par des interactions sociales.

— Ces interactions sociales peuvent être de différentes natures : familiales, sentimentales (liens forts) ou plus distantes : affinité, relation d'affaire, de travail (liens faibles), possibilités et fréquences d'interaction, compatibilités, etc.

2.9 Conclusion

Dans ce Chapitre, nous avons situé le contexte général de notre travail et présenter les principaux outils nécessaires à la compréhension de ce sujet. Comme nous l'avons montré, comprendre et pratiquer ce domaine nécessite la maîtrise de nombreuses technologies et normes impliquées telles que le XML, SOAP, WSDL, UDDI, SOA, sans oublier bien sûr les réseaux sociaux.

Chapitre 3

État de l'Art

Sommaire

3.1 Introduction

Dans ce Chapitre, nous allons discuter les principaux travaux existants par rapports aux services Web et notamment dans un contexte des réseaux sociaux qui représente un nouveau paradigme investigué par certaines études principalement de type recherche scientifique.

3.2 Les services Web et réseaux sociaux

Aperçu sur les réseaux sociaux

Les réseaux sociaux ont été utilisés dans des domaines différents comme les sciences sociaux et politiques, Intelligence Artificiel (IA), et e-business. Selon Ethier [10], *l'étude des réseaux sociaux est importante tant qu'elle nous aide à mieux comprendre comment et pourquoi nous interagissons, ainsi que la façon dont la technologie peut modifier cette interaction. Le domaine de la théorie des réseaux sociaux s'est considérablement développé au cours des dernières années, la technologie informatique avancée ayant ouvert la voie à de nouvelles recherches.* Généralement, un réseau se compose de nœuds et de bords. Les nœuds réfèrent à tout type d'objet (ou d'entité) comme les individus ou les organisations, et les bords réfèrent aux relations (ou associations) entre ces nœuds tels que le degré d'amitié entre deux personnes ou la distance entre deux villes. Les relations sont parfois directionnelles, bidirectionnelles, avec un poids, ou un mélange de tous ces éléments. La recherche dans des domaines académiques a révélé que les réseaux sociaux opèrent à plusieurs niveaux : des familles jusqu'au niveau des nations, et jouent un rôle essentiel dans la résolution des problèmes, la gestion des organisations et le succès des individus dans la réalisation de leurs objectifs. [16]

Les réseaux sociaux des Web services (WS)

Ce type de réseaux est différent des types classiques des réseaux sociaux de personnes (ex : LinkedIn, Plaxo, Facebook). Ces derniers sont basés sur la collaboration absolu et l'entraide entre leurs membres dans le sens où il n y a pas en général de compétition entre les membres. Par contre, dans les réseaux sociaux des Web services (WS), les membres sont principalement compétitifs car chaque WS souhaite faire partie des compositions, remplacer un WS défectueux, ou être nommé comme un WS supplémentaire. Exemple, un WS pour organiser la participation dans une conférence, et un WS supplémentaire pour organiser une excursion [16].

Dans les travaux de [16], les auteurs expliquent que construire un réseau

social revient à identifier les types de nœuds et de bords qui constituent ce réseau. Dans ce contexte, les services Web sont les seuls constituants et ils désignent des nœuds. En termes de bords, ils proposent trois types d'association pertinente entre les services Web, à savoir : Recommandation (R), Similarité (S) et Collaboration (C).

Dans l'article [15], les auteurs mettent en avant la valeur de l'ajout des réseaux sociaux aux services Web dans le sens où quand les entreprises découvrent et engagent les services Web pour des besoins de travail, ils sont inclus dans les compositions des service basés à la fois sur les fonctionnalités qu'ils offrent et de la qualité du service qu'ils peuvent garantir. Ce qui implique forcément le besoin de contrats. Cependant, lorsque les consommateurs s'engagent et composent des services, c'est beaucoup plus informel et dynamique, tout comme la façon dont les utilisateurs téléchargent des applications iPhone. Mais contrairement aux applications iPhone, qui sont monolithique et fonctionnent indépendamment les unes des autres, les services Web sont destinés à cohabiter et à être composés, et leurs fonctionnalités et QoS sont interdépendantes avec d'autres services. De plus, ils sont exécutés à distance et avec un certain degré d'autonomie. Leur découverte et leur engagement ultérieur deviennent ainsi des activités sociales, tout comme la collaboration et les interactions soutenues dans les réseaux sociaux.

3.3 Approches de découverte des services Web

Dans cette section nous nous intéressons aux approches existantes pour la découverte de services Web.

3.3.1 Approches lexicales

Le travail de [7] propose une nouvelle approche centrée sur les exigences pour modéliser les applications basées sur les services ou *Service-Based Applications - SBA*. Ces exigences sont listées en termes d'exigences fonctionnelles et non

fonctionnelles des utilisateurs, en termes de découverte des services potentiels selon les besoins prévus, et enfin en terme de sélection de services basée sur la qualité de service (QoS) qui doit être pertinente et élevée. Cette approche permet une modélisation intentionnelle des SBA en utilisant le formalisme basé sur un graphe appelé MAP [2]. Le formalisme MAP analyse les besoins des utilisateurs dans un ensemble de graphes composés d'intentions et de stratégies. Cette approche permet la découverte de services opérationnels en interrogeant le moteur de recherche WS (appelés *Service-Finder*) en utilisant des mots-clés extraits des modèles *Intentional Service Model* (ISM).

Dans le travail de [12], une nouvelle méthode a été proposée pour résoudre le problème de la découverte de services Web. Cette méthode utilise une architecture basée sur des registres des services Web appelés aussi communautés. Le registre des services Web représente un groupe de services Web ayant le même domaine d'intérêt et ils sont différenciés les uns des autres en fonction de leurs propriétés non fonctionnelles (telle que la sécurité, le temps de réponse, etc.). Le travail utilise l'algorithme *Bees Algorithm* comme un moyen de recherche intelligent. L'approche consiste en trois étapes principales :

1. La configuration de l'environnement distribué avec l'exploitation de l'architecture peer-to-peer (P2P). Chaque groupe de pairs représente un registre de services Web.
2. La découverte du registre approprié ayant le même domaine d'activité avec le service Web recherché. Dans cette étape, on utilise une méthode basée sur un nouvel algorithme d'optimisation «Bees» basé sur la population appelé.
3. La sélection des services Web optimaux présents dans le registre découvert. Le service Web sélectionné a le niveau de QoS le plus proche de la valeur demandée par le client.

Le travail présenté dans [21] propose un environnement de travail (framework) pour la recherche des services Web sémantiques en utilisant les techniques du traitement du langage naturel. L'approche proposée permet de rechercher, à partir d'un ensemble de services Web sémantique, une

correspondance avec une requête de l'utilisateur constitué de mots-clés. En précisant le but de la recherche avec des mots-clés, les utilisateurs finaux n'ont pas besoin d'avoir des connaissances sur les langages sémantiques.

3.3.2 Approches sémantiques

Une autre approche qui peut complémenter la recherche syntaxique est l'approche basée sur la sémantique. Dans [22], les auteurs décrivent les défis posés par les normes de service Web existantes pour découvrir et interagir automatiquement avec les services Web. Ensuite, ils discutent les avantages de la norme OWL-S par rapport aux normes existantes. Le travail s'est concentré sur la prise en charge de la découverte dans un IDE de OWL-S et les modifications apportées au registre UDDI existant. Les auteurs présentent l'architecture d'entremetteur OWL-S / UDDI et ses extensions pour effectuer une recherche de capacité. L'approche utilise un algorithme de correspondance (matching) dans le registre UDDI amélioré. L'algorithme définit un mécanisme de correspondance flexible basé sur le mécanisme de subsomption du langage OWL. Le travail a également mené quelques expériences préliminaires pour montrer l'évolutivité de l'implémentation proposée.

Le travail présenté dans [13] présente une approche OWLS-MX hybride de correspondance du OWL-S. Cette approche de correspondance de service Web sémantique hybride utilise à la fois le raisonnement logique et technique (IR) pour les services Web sémantiques dans OWL-S. Le travail implémente des variantes du matchmaker OWLS-MX version 1.1 en Java utilise la version bêta OWL-S API 1.1 avec l'outil de raisonnement OWL-DL appelé *Pellet*

La variante purement logique, appelée OWLS-M0, de OWLS-MX est similaire à celle de l'entremetteur OWLS-UDDI mais en diffère à plusieurs égards. Tout d'abord, ce dernier utilise une notion différente de correspondance de plug-in et n'effectue pas de correspondance supplémentaire subsumée. Deuxièmement, OWLSM0 classifie les concepts de requête arbitraires dans son ontologie en évolution dynamique avec un vocabulaire de base minimal

communément partagé par composants primitifs. L'évaluation expérimentale de ce travail montre que sous certaines contraintes, la méthode d'appariement proposée peut effectivement surpasser les approches basées uniquement sur la logique.

Dans le travail détaillé dans [23], une nouvelle approche pour la visualisation et la navigation des services Web a été proposée. Cette approche permet aux clients de comprendre et d'inspecter les services Web disponibles relatifs à des problèmes qui peuvent être résolus par abstraction faite des détails techniques qui sont au centre de la plupart des outils existants. L'approche utilise des objectifs comme élément constitutif et des graphiques appelées *Semantic Discovery Caching Technique* (SDC). SDC définissent les hiérarchies de subsomption des modèles d'objectifs comme structure d'indexation et capturent des connaissances sur l'adéquation des services Web disponibles aux résultats de la découverte.

Le graphe SDC est généré automatiquement par la mise en relation sémantique des descriptions formelles des buts et de services Web. Il fournit une structure d'index pour la recherche efficace de services Web appropriés et sert ainsi de structure de données pour la visualisation de l'espace de recherche. L'interface graphique de l'utilisateur est implémentée dans un outil appelé *Web Service Modeling Toolkit* (WSMT).

Le travail de [19] propose une nouvelle architecture de SOA qui permet la découverte sémantique de services Web avec une technique d'apprentissage adaptative appelée *apprentissage social*. Cette forme d'apprentissage produit des résultats de découverte plus pertinents au fil du temps. L'ontologie du fournisseur de services améliore son concept à travers les contributions sociales des consommateurs des services qui devienent ainsi plus facilement détectable au fil du temps.

Cette nouvelle architecture SOA est proposée avec le cahier des charges suivant :

— L'architecture devrait permettre la description sémantique des services dans la description concrète des services.

— Elle devrait permettre une recherche évolutive avec différents mécanismes de service à différents points d'extrémité. Ainsi différents algorithmes de recherche peuvent être utilisés à différents points d'extrémité.

— Elle devrait permettre au demandeur de service et au fournisseur de créer une ontologie de service de manière distribuée. Il est donc trivial que la chance de contact préalable et direct entre eux soit très faible. Donc, on favorise la création d'une description de service indépendante.

Le travail précédent propose un algorithme de mesure de similarité sémantique. L'algorithme proposé traite de l'ontologie pondérée. L'ontologie non pondérée peut être considérée comme une ontologie pondérée en indiquant un poids maximum également réparti sur tous les liens. Ce nouvel algorithme devrait également intégrer l'apprentissage social qui a été défini comme étant le processus par lequel, si un individu devient plus social, il peut se mélanger ou partager plus de choses avec les autres.

3.3.3 Approches basées sur le concept des réseaux sociaux

Si les deux classes précédentes de découvertes de services Web ont connu de nombreux travaux de recherche, d'expérimentation et de développement, les approches basées sur l'exploitation du paradigme des réseaux sociaux pour la découvertes sont relativement nouvelles. En effet, l'utilisation et la généralisation du concept des réseaux sociaux pour la modélisation, les interactions et la découverte des services Web remonte seulement à moins de sept ans. Notre recherche approfondie relative au sujet de ce mémoire révèle que les premiers travaux sont ceux de Zakaria Maamar de l'Université de Zayed (UAE) qui ont débuté en 2010 [17] et qui se s'ont précisés en 2011 avec le travail discuté dans [14].

Le travail présenté dans [17] propose une approche nommée *LinkedWS* qui définit un nouveau modèle de découverte de services Web basé sur la métaphore des réseaux sociaux au sens large du terme. Rappelons que, comme défini

dans notre précédent Chapitre, aucune définition universelle n'a été adoptée pour le concept des réseaux sociaux et que toutes les définitions dépendent profondément du contexte de leur application. Dans un premier temps, les auteurs de [17] définissent les réseaux sociaux dans le contexte particulier des services Web. Ensuite, ils montrent comment peut on exploiter de tels modèles pour la construction des services Web et assurer leur utilisation et leurs maintient.

Aperçu sur *LinkedWS*

Différents types de services Web peuvent exister et cohabiter sur Internet. Ce fait rend leur découverte en réponse aux demandes des utilisateurs finaux un sérieux défi qui peut se résumer avec les questions récurrentes suivantes : où peut on trouver des services Web ? Comment sélectionner les services Web pour un certain nombre de critères donnés ? Comment garantir la pertinence des services Web et comment suivre l'évolution de ces services ? La proposition de *LinkedWS* tente de répondre à certaines de ces questions. Cela nécessite, entre autres, de construire, de naviguer et d'assurer une maintenance des réseaux sociaux.

Adopter le paradigme des réseaux sociaux pour service Web signifie de pouvoir naviguer à travers les nœuds et les bords du modèle représentant ce réseau pour des fins de recherche d'information. En cas de réseau social positif, la navigation renvoie les pairs collaborateurs pour lesquels un service Web a fonctionné avec plus ou moins des compositions précédentes. L'établissement de nouvelles compositions ou connexions peut faire référence à ces pairs collaboratifs au lieu de filtrer les registres classiques (du style UDDI, voir Chapitre 2) pour chercher des pairs avec lesquels un service Web peut travailler. En cas de réseau social négatif, la navigation renvoie les homologues concurrents à un service Web donné. Par conséquent, ces homologues peuvent être sélectionnés pour ce service Web en cas de manque de compétitivité. Cette navigation renvoie également les homologues qui peuvent remplacer le service Web en cas d'échec ou de panne du service principal. Le remplacement d'un service Web défaillant dans un service en cours de composition peut donc se

référer à ces pairs au lieu de se limiter à de simples filtrages des annuaires pour rechercher des pairs similaires.

Comme un vrai réseau social conventionnel, un réseau social de service est voué à une évolution constante et à une dynamicité accrue. Maintenir le réseau social d'un service Web, est le résultat des participations supplémentaires du service Web dans de nouvelles compositions. Cela signifie créer de nouveaux nœuds et bords pour les pairs dans ces compositions. Ces nouveaux nœuds et bords sont connectés à ceux qui existent déjà dans le réseau social actuel de ce service Web. Une autre raison de maintenir un réseau social, c'est quand un service Web cesse de fonctionner, cela nécessite de le retirer du réseau et appliquer une stratégie de «best effort» qui vise à le remplacer.

Le modèle LinkedWS

Le modèle LinkedWS représente un réseau social (notons le SN pour *social network*) d'un service Web (noté WS) grâce au couple : $SN_{WS} = (N, E)$ où N et E sont l'ensemble des nœuds et des bords/liens respectivement. Chaque lien e est un tuple de la forme (WS_i, t, w, WS_j) où t est le nom du lien entre le service WS_i et WS_j. w est le poids du lien présenté par nombre positif compris entre 0 et 1. Un poids de lien devrait être utilisé par les algorithmes de recherche ou de classement qui naviguent à travers le réseau afin de trouver des services Web correspondants à une demande particulière comme la recherche de collaborateurs potentiels pour un service donné.

LinkedWS facilite la découverte de services Web supplémentaires basés sur ceux qui sont déjà inclus dans les compositions en cours c'est à dire les relations déjà existantes. La recommandation et les relations de collaboration dans les réseaux sociaux sont évaluées régulièrement pour refléter la valeur ajoutée de ces réseaux à la découverte de services Web. Chaque fois qu'un service Web est découvert (ou ajouté), le poids d'une relation est réévalué ou un nouveau nœud est établi et connecté au reste des nœuds. Ceci dynamicité peut exploiter les registres de la découverte traditionnelle (comme UDDI) mais avec les détails que les réseaux sociaux offrent. Tout service Web découvert en utilisant ces mécanismes est un point d'entrée à son propre réseau social et probablement

aux réseaux sociaux d'autres pairs s'ils sont traversés et s'ils accordent les droits au service Web principal (service Web du point d'entrée).

Le travail présenté dans [14], auquel nous nous appuyons fortement dans nos travaux de ce mémoire, est basé sur les fonctionnalités des services Web. Dans ce travail, les chercheurs proposent deux types de relations entre les réseaux sociaux des services Web : *similaire* et *différent* (Figure 3.1). Une *similarité* signifie qu'un groupe de services Web offre des fonctionnalités homogènes à la communauté externe indépendamment de la façon dont ces fonctionnalités sont satisfaites. Ils raffinent la notion de similarité dans un réseau social en deux sous-catégories de relations : *concurrence* et *substitution*.

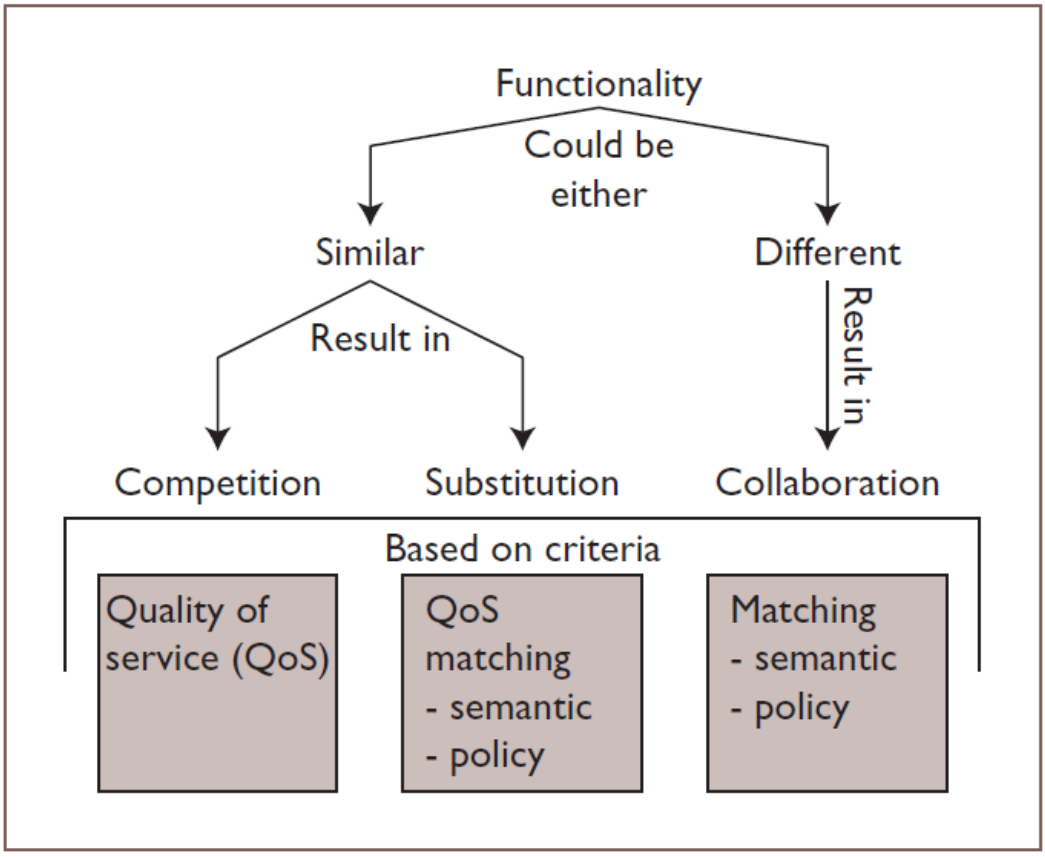

FIGURE 3.1 – Relations entre services Web avec le modèle des réseaux sociaux

Le modèle proposé défini six étapes pour le développement des services Web sociaux :

1. **Identification des composants des réseaux sociaux** : l'approche associe 3 valeurs à un service Web pour chaque type d'interaction

sociale. Ces interactions sont : les compétition, les collaborations ou les substitutions (en cas de défaillance).

2. **Analyse des correspondances entre services Web** : afin d'établir la nature des interactions sociales entre les services Web, on utilise leurs propriétés respectives fonctionnelles et non fonctionnelles en formant ainsi le profil d'un service Web.
3. **Gestion du réseau social** : l'achèvement de cette étape nécessite un type spécial de nœud appelé textitracine du service Web. Tout service Web qui va faire partie du futur réseau social est le racine. Ainsi, la sélection de racine peut être aléatoire car tous les autres services Web seront connectés à cette racine. Le choix de la racine peut par exemple se faire dans un contexte d'identification des nouvelles relations, poids et types d'interactions possibles pour un service donné (c-à-d la racine) avec le reste du réseau.
4. **Evaluation initiale du poids des liens** : le poids initial d'un lien (noté aussi WE) entre deux services S_i et S_j (où S_i est la racine du service Web) correspond au degré de similitude entre eux.
5. **La navigation dans les réseaux sociaux** : des moyens appropriés sont nécessaires pour aider les ingénieurs du service à naviguer à travers les réseaux sociaux de services Web. Ici, notons que les *utilisateurs finaux* (terme que nous avons utilisé au début de ce mémoire) correspond à des ingénieurs qui veulent trouver/composer des services pour développer des applications. Dans les réseaux sociaux, la notion d'*utilisateurs finaux* est large. Elle peut correspondre à un utilisateur lambda du réseau social ou, comme dans notre exemple, elle peut signifier un spécialiste de développement. Par ailleurs, elle peut aussi correspondre à une application distribuée visant à utiliser des services Web appartenant à un réseau social donné.
6. **Evaluation continue du poids des bords** : l'évaluation en cours reflète le rôle des réseaux sociaux dans la découverte de services Web. Ceci arrive quand on doit mettre à jour les poids de bord chaque fois qu'un remplaçant, collaboratif ou concurrent est découvert en utilisant ces réseaux.

Dans [9], un nouveau cadre pour la découverte et la publication de services Web est proposé. La méthode introduit l'idée de fusionner les utilisateurs et les services Web comme composants actifs du même réseau social mondial pour améliorer la publication et la découverte de services Web. L'incorporation des utilisateurs et des services Web des réseaux sociaux permet de résoudre de nombreux problèmes auxquels sont confrontés les services Web tels que la recherche du meilleur emplacement pour les annonces et le remplacement des services en cas d'échec.

L'approche utilise une méthode d'ingénierie en trois étapes pour aider à construire leur réseau social :

1. **Etape 1** : il s'agit de l'identification des composants du réseau social : les utilisateurs et les réseaux sociaux des services Web ont deux composants. Les nœuds représentant des utilisateurs et/ou des sites/service Web, et les liens représentent la relation entre ces nœuds.
2. **Etape 2** : établir des relations entre les utilisateurs et les services Web.
3. **Etape 3** : l'identification des utilisateurs et les caractéristiques des services Web.

Le travail de [4] construit des réseaux de service social basés sur des principes de données liées pour améliorer la sociabilité des services et pour fournir une meilleure qualité de service de découverte. L'approche propose des principes spécifiques liés au service social qui comprennent quatre éléments pour fournir une recette de base pour la publication et la connexion des services dans un réseau de service social. Voici les principaux éléments :

1. Les services sont publiés sur le Web ouvert (sans restrictions) en suivant les principes de données liées (*linked data*). Leur information sociale et connaissances sont décrites en termes de vocabulaires avec la norme W3C RDF(S). L'objectif étant que les applications puissent facilement découvrir et traiter leurs descriptions tout comme récupérer les données liées.
2. Les services sont construits sur le Web des données. En effet, ce dernier peut être utilisé comme une connaissance de base pour fournir des ontologies appropriées qui peut être utilisées, étendues et combinées pour

créer un domaine des ontologies pour les services d'annotation.

3. Les services doivent être liés à des services connexes ou liés de manière fonctionnelle à des services connexes, en utilisant un lien social afin qu'ils puissent être découverts et composés efficacement en suivant ces liens sociaux.
4. Les services, appelés dans ce travail *Interlink* doivent refléter la réalité sociale du service. Les interactions sociales passées du service et la popularité du service pour relier les services dans un modèle de réseau pour refléter réalité sociale du service.

Dans le travail étudié, une nouvelle plateforme a été proposée aussi pour la construction d'un réseau mondial de services sociaux pour connecter les «îles» du service isolé pour soutenir les activités du service social.

Une approche efficace de découverte de service appelée *Link-as-you-go* a été proposée pour permettre l'exploration de type service-à-service. Cette approche est basée sur le réseau de service social global et la navigation dans les propriétés d'un service puis la navigation au long des liens sociaux dans des services connexes à explorer. L'objectif est que les utilisateurs puissent explorer le réseau mondial de services sociaux plus profondément sur le Web ouvert. Link-as-you-go ouvre la boîte noire pour la découverte de service afin qu'un consommateur de services puisse découvrir des services en suivant les liens d'intérêt. C'est exactement le même principe que l'exploration des pages Web sur Internet ou la recherche de nouveaux amis grâce à des amis déjà connus sur Facebook.

Dans le travail de [8], les auteurs présentent un mécanisme de découverte de services Web collaboratifs fondé sur les relations sociales latentes derrière les utilisateurs et les services. En définissant quatre facteurs de lien, ils formalisent le lien social. Ainsi, les auteurs formalisent le lien social (SL) dans la découverte de services Web avec $SL =< RD, CL, PL, TL >$. Ces liens sont régis par les facteurs suivants :

1. **Le degré de réputation (RD)** : est la proportion de la quantité de concepts dans son modèle de personnalisation par rapport à la quantité de concepts entiers dans l'ontologie correspondante.

2. **Clustering Link (CL)** : contient deux sous-classes : Service Clustering Link (SCL) et utilisateurs Clustering Link (UCL). Le premier est l'ensemble des services et leurs similitudes.
3. **Lien de préférences (PL)** : pour un utilisateur, il est défini comme l'ensemble des services invoqués par lui/elle et l'expérience correspondante.
4. **Le Lien de confiance (TL)** : pour un utilisateur il est conçu pour formaliser la relation entre lui et son amis de confiance. c'est l'ensemble des utilisateurs qui ont recommandé le service à lui/elle et reflète le degré de confiance.

Dans le travail de [6], les auteurs exploitent d'autres relations présentes dans le réseau social pour identifier la similitude entre services. Ces relations sont également exploitées pour effectuer des regroupements pour augmenter la précision et pour les relations sociales facilitent la découverte de services.

Les services sont donc groupés par similarité pour déterminer leur points en commun (voir Figure 3.2).

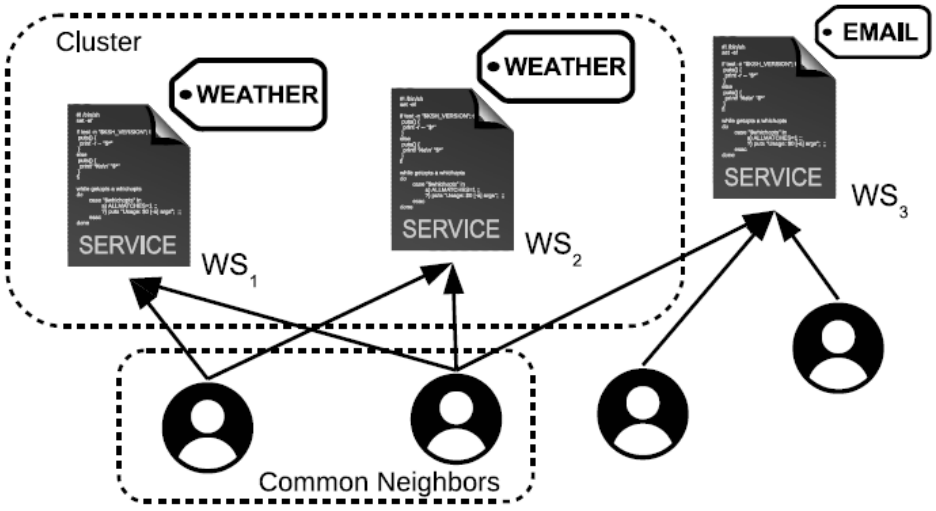

FIGURE 3.2 – Exemple de groupement des services basé sur les réseaux sociaux

La structure graphique d'un graph de réseau social peut elle-même prise comme une base pour mesurer la similarité entre deux nœuds de sorte qu'ils

sont réputés similaires si ils sont structurellement proches dans le graphique. A cette fin, les métriques disponibles sont des indices de similarité locale, qui considèrent le voisinage des deux nœuds d'intérêt, tandis que les indices mondiaux ou globaux impliquent le graphe entier. Dans cette approche on étudie les indices locaux car ils sont plus faciles à calculer et bien dimensionné pour les grands graphiques. Les auteurs de [6] présentent un clustering de services basé sur des exemples qui résument les indices de similarité locale utilisés dans leur travail. Dans ce type de regroupement, la similarité par paire de services est utilisée pour former des grappes de services. Un algorithme de clustering classique, appelé *K-Means*, a été utilisé. C'est un algorithme basé sur un prototype qui construit un nombre de clusters spécifié par l'utilisateur.

Dans [11], le travail de recherche est principalement lié à un service Web basé sur deux mécanismes complémentaires, à savoir, la détection de la confiance sociale (STD) et la recommandation de service (SR). Ces deux notions peuvent être résumés comme suit :

1. **La détection de la confiance sociale (STD)** : elle analyse le profil social d'un utilisateur cible pour extraire les intérêts et les interactions avec ses amis. Ces interactions sont représentées par un vecteur qui contient le type de chaque interaction (par exemple, message, commentaire ou publication), la date de l'interaction et les amis impliqués. Après cela, on calcule le niveau de confiance parmi les utilisateurs actifs et leurs amis. Les auteurs ont adopté l'analyse dite *égocentrique* du réseau social, et ils ont proposé le niveau de confiance informatique avec le filtrage dynamique des amis de confiance.

2. **La recommandation de service (SR)** : ici, on propose de considérer la recommandation de service social (RSS). Le mécanisme de recommandation de service (SR) fonctionne en six étapes : extraction de recommandeurs, niveau d'expertise, expérience antérieure d'extraction, filtrage de service Web, prédiction de notation, et classement du service Web.

Dans [29], les auteurs proposent une architecture décentralisée auto-organisée pour les futurs réseaux sociaux en ligne pour permettre une

découverte efficace des services dans des environnements sociaux dynamiques d'Internet des objets (Internet of Things ou IoT). Cette architecture décentralisée comble le fossé de la recherche entre les infrastructures de type P2P traditionnel et les architectures décentralisées De type *Online social networks*(OSN).

Le réseau social est modélisé comme un graphe non orienté $G = (V, E)$, où V représente les utilisateurs (nœuds) et E les relations d'amitié entre eux.

Dans ces graphes appelés *self-organized decentralized OSN*(ou SDOSN), les nœuds mettent à jour deux indices distincts : un indice de service local et un indice de connaissance. L'indice de service local est un indice inversé qui organise différents types de services locaux. L'indice des connaissances stocke les raccourcis de superposition des informations obtenues lors des interactions sociales. La structure de l'indice des connaissances est une table relationnelle. La découverte de service comprend deux étapes : la phase de publicité et la phase de découverte. Le but de la publicité est d'augmenter la probabilité que les nœuds trouvent un nœud homologue contenant le service demandé. Lors de l'étape de la découverte, les nœuds utilisent des messages de requête pour effectuer une recherche du réseau de manière décentralisée. On intègre également les intérêts de l'utilisateur à traiter avec des informations sociales personnalisées. Les intérêts sociaux sont les termes qui décrivent les préoccupations personnelles (préférences) ou l'implication d'un utilisateur dans un réseau social.

3.3.4 Comparaison et discussion

Dans cette section, nous résumons, dans le Tableau 3.1 quelques critères d'évaluation des approches de découvertes des services Web afin de les comparer et dresser un récapitulatif.

— La colonne «Technique du Matchmaking» indique quelles sont les principales techniques utilisées pendant le processus de découverte des services Web.

— La colonne «Outils» indique si l'approche en question est supportée par un outil implémenté ou pas.
— Le tableau présente aussi les avantages et les inconvénients des différentes catégories d'approches.

TABLE 3.1 – Les différentes approches relatives à la découverte des Services Web

	Approche	Technique de matchmaking	Outils	Avantages	Inconvénients
Lexicales	[7]	Techniques de classification	non	Approches simples et largement utilisées	Nécessitent l'interaction humaine
	[12]	Clustering + Algorithmes génétiques	non	Basée sur des standards comme WSDL et UDDI	Ne convient pas pour le traitement automatique
	[21]	Traitement du langage naturel	oui	Recherche par mots-clés est plus familière a l'utilisateur	
Sémantiques	[22]	Algorithmes de matching (mise en correspondance)	oui	Réduit l'effort manuel de découverte	Technique complexe et un tagging (annotation) sémantique des SW peut être nécessaire
	[13]	Syntactic IR based similarity computations	oui	Techniques efficaces et fiables	Mots-clés ne suffisent pas pour spécifier avec précision les besoins d'iformation de l'utilisateur
	[23]	Similarité sémantique	oui		
	[19]	Matching d'ontologies algorithmes de fusion	oui		
Réseaux sociaux	[17]	Découverte traditionnelle de registres (ex. UDDI) avec des propriété ajoutées par les réseaux sociaux	oui	Facilite la découverte de services Web supplémentaires	Exige une modélisation et une implémentation d'algorithmes parfois complexes Exige le suivi de l'évolution dynamique du réseau
	[14]	Clustering + degré de similarité	oui	Les utilisateurs peuvent trouver des SW, sans avoir à manipuler des registres UDDI	
	[9]	Ontological matching score	oui	Remplacement des services en cas de panne	
	[4]	Syntactic matching + Semantic matching	oui	Rendre la découverte plus facile et efficace.	
	[8]	Clustering Link and Preference Link	oui	Aide à rendre les services Web hautement disponibles	
	[6]	Clustering algorithms	oui		
	[29]	Semantic distance	oui		

3.4 Composition de service

Dans cette section, nous complétons l'aspect de la découvert de service avec le besoin de composition des services qui nous paraît important à présenter. Le plan de composition des services englobe les rôles et la fonctionnalité pour agréger plusieurs services en un seul service composite plus riche. Les services composites résultants peuvent être utilisés comme services de base dans d'autres compositions de services ou offert dans une application plus complète ou dans des solutions clés en main au service des clients. Les «agrégateurs» de services accomplissent cette tâche et deviennent ainsi des fournisseurs de services en publiant les descriptions du service composite qu'ils créent. Les agrégateurs peuvent également appliquer des règles sur les invocations agrégées du service [20].

La promesse visionnaire du paradigme *service-oriented computing* (ou SOC) est de rendre possible le but de vouloir assembler facilement des composants d'application dans un réseau faiblement couplé de services qui peuvent créer des processus dynamiques de business et des applications agiles des organisations et des plates-formes informatiques. Ces services iront bien au-delà du simple échange d'informations. C'est le mécanisme dominant pour l'intégration des applications aujourd'hui pour accéder, programmer et intégrer des services d'application encapsulés dans d'anciennes ou nouvelles applications.

La clé de la réalisation de cette vision est les architectures orientées services (SOA). Les SOA offrent un moyen logique de concevoir un système de logiciel pour fournir des services soit aux applications de l'utilisateur final ou pour d'autres services distribués dans un réseau, via des interfaces publiées et identifiables [20]

Dans le travail présenté dans [20], les auteurs identifient les défis suivants qui sont naturellement d'ordre de la recherche scientifique :

— Analyse de «composabilité» pour la «remplaçabilité», la compatibilité et la conformité des processus.

— Conception de processus dynamiques et adaptatifs.
— Compositions de services sensibles à la qualité de service (QoS-aware).
— Compositions automatisées axées sur les affaires (relations économiques ou commerciales).

Les chercheurs indiquent également que les développeurs utilisent largement les termes d'"orchestration" et de "chorégraphie" pour décrire les protocoles de l'interaction commerciale qui coordonnent et contrôlent les services collaboratifs. Ces termes peuvent être résumés comme suit :

— **L'orchestration** : Décrit comment les services interagissent au niveau des messages, y compris la logique du business et l'ordre d'exécution des interactions sous le contrôle d'un seul point final. Il s'agit d'un processus de business exécutable qui peut aboutir à un modèle de processus transactionnel à étapes multiples. Avec l'orchestration, l'une des parties du business, impliquées dans le processus, contrôle toujours les interactions processus-business. L'orchestration est réalisée via des modèles appelés BPEL4WS et d'autres basés sur le standard XML.
— **La chorégraphie** : est généralement associée aux échanges de messages du public (globalement visibles), aux règles des interactions, et aux accords qui se produisent entre plusieurs points finaux de processus-business plutôt qu'une entreprise spécifique. La chorégraphie de service est souvent réalisée via la norme «Web Services Description Language» qui est utilisé pour spécifier le comportement observable commun à tous les participants.

Le travaille de [18] présente une nouvelle approche pour la recommandation des services dans un environnement nommé *Mashup* basé sur une analyse d'un réseau social. La particularité de cette approche est la création d'un graphe social implicite basé sur les interactions entre utilisateurs et services. Mashup est certainement l'instanciation la plus intéressante de la composition des services des utilisateurs finaux. Concrètement, un Mashup est une application Web qui combine des services existants (API, sources de données, etc.) en un seul service intégré. Un exemple pourrait être l'utilisation des données cartographiques et des interfaces du service *Google Maps* pour ajouter des

informations de localisation sur les données immobilières. Le travail proposent un cadre général, appelé *Social Composer* (SoCo) pour la découverte et la composition de services. SoCo est basé sur la transformation de l'utilisateur, des interactions des services avec les utilisateurs, du réseau social des utilisateurs en plus de quels processus statistiques qui sont appliqués pour déterminer des recommandations. Ceci est fait pour aider un utilisateur donné dans la construction d'un Mashup c'est à dire une composition de services.

Dans le travail de [28], on propose de considérer un service de logistique comme V_i qui se réfère aux villes et S_i qui se réfère aux services de logistique. Chaque service détient sa propre qualité de service (QoS). Le modèle de composition du service logistique proposé est composé de deux parties : un modèle d'évaluation de la qualité de service du service logistique qui évalue la QoS du service logistique. L'autre est un service de logistique basé sur un réseau de coopération entre fournisseurs et qui permet la création d'un cercle de partenaires constitué avec un algorithme nommé *PartnerFirst Algorithme*. L'approche peut être résumée comme suit :

— **Le modèle d'évaluation de la qualité de service du service logistique** qui utilise un **Service logistique** représenté par un quintuple $si = (Id, Pro, Dep, Des, QoS)$ et qui fait référence à : Identity, Service Provider, Logistics Departure, Logistics Destination, QoS . Un **ensemble d'attributs QoS** est utilisé et qui est représenté par un quintuple Pr, Du, Co, Ac, Re, où Pr est le prix du service, Du est la durée du service, Co est l'exhaustivité des marchandises lorsque le service a été terminé, Ac est la précision de l'ordre de transformation une fois terminé, et Re se réfère à la réputation du fournisseur de services. Notons que Pr et Du sont des attributs de sommation et Co, Ac, et Re sont les multiplications d'attributs.

— **Réseau de coopération des fournisseurs de services logistiques** : ce réseau est créé en se basant sur l'historique des coopération entre fournisseurs de services. Un réseau social est composé d'acteurs sociaux et de leurs relations. Un acteur social peut être une personne, un groupe ou une organisation. Les fournisseurs du réseau de coopération de service peuvent être créés en fonction de l'historique de leur coopération

sachant que l'expérience de coopération entre les membres d'une équipe a un impact positif. Remarquons que cette relation sociale peut se décomposée dans le temps. Cela signifie que si le temps de la dernière coopération entre deux fournisseurs de services à partir de maintenant est trop long, il peut être considéré comme s'ils n'ont pas de relation sociale.

Comme mentionné précédemment, l'approche précédente propose d'utiliser l'algorithme **PartnerFirst** qui est basé sur le concept de cercle de partenaires. L'algorithme PartnerFirst atteint l'objectif de la planification du chemin à travers l'amélioration de l'algorithme. Cette amélioration peut réduire considérablement l'espace (la portée) de la recherche en cherchant préférentiellement les services dans le cercle des partenaires.

Dans [3], les auteurs utilisent *Klaim* : un langage formel spécifiquement conçu pour modéliser des systèmes distribués afin de spécifier formellement une nouvelle procédure pour la composition de services Web appartenant à des réseaux sociaux spécialisés. La nouveauté dans la façon dont les SWS sont composées repose sur la prise en compte de leur réputation. Les scores de réputation donnent une prédiction de la QoS dans les transactions futures et les rendre plus ou moins, digne de faire partie d'une composition nouvellement créée.

Soutenu par un ensemble d'outils associés à Klaim, le travail a ensuite analysé de manière stochastique les propriétés de composition des SW basées sur la réputation. Le travail démontre comment la spécification de Klaim présentée peut supporter l'analyse des compositions basées sur la réputation des SW. Les chercheurs ont présenté les résultats de la simulation et de l'analyse de la vérification du modèle, respectivement. L'approche s'est appuyée sur des outils formels qui ont enrichi la spécification de Klaim avec des aspects stochastiques (obtention d'une spécification StoKlaim). Ensuite, les propriétés d'intérêt (exprimées en MoSL) ont été vérifiées par rapport à la spécification StoKlaim au moyen d'un outil d'analyse appelé SAM.

Le travail a introduit les outils d'analyse stochastique de Klaim, qui a permis

d'effectuer une analyse quantitative des systèmes. En général, deux types principaux d'analyse peuvent être effectués sur des systèmes : quantitatifs ou qualitatifs. Dans l'analyse qualitative, il est vérifié si un certain événement peut se produire. Dans l'analyse quantitative, au contraire, il est vérifié quelle est la probabilité qu'un certain événement peut se produire. Nous récapitulons dans ce qui suit les outils (StoKlaim, MoSL et SAM) utilisés dans cette approche :

— **StoKlaim** : les actions de processus de Klaim sont enrichies avec un certain taux. Ce taux est le paramètre d'une variable aléatoire exponentiellement distribuée caractérisant la durée de l'exécution d'une action.

— **MoSL** : les propriétés souhaitées d'un système sous vérification sont formalisées à l'aide de la logique stochastique mobile nommée MoSL.

— **SAM** : la vérification des formules MoSL par rapport aux spécifications de StoKlaim est assistée par l'outil SAM. Ce dernier utilise un algorithme de contrôle de modèle statistique pour estimer les probabilités de satisfaction de la propriété en considérant un ensemble d'observations indépendantes.

3.5 Conclusion

Dans ce Chapitre, nous avons étudié de nombreux travaux existants liés de près ou de loin aux services Web et à l'exploitation du modèle des réseaux sociaux. Le passage en revue de ces travaux nous a permis d'acquérir pas mal de connaissances mais surtout de voir que l'exploitation du paradigme des réseaux sociaux appliqué aux services Web est loin d'être fini. Le potentiel de tels modèles reste à explorer pour proposer de nouvelles approches et résoudre les problématiques qui restent non résolues.

Chapitre 4

Contribution

Sommaire

4.1 Modélisation d'un réseau social de services Web

4.1.1 Introduction

Notre étude de la situation actuelle du Web, en général, et des services Web en particulier montre que le soutien des grosses compagnies aux standards liés aux annuaires de type UDDI s'est considérablement affaibli et ce depuis de nombreuses années (pratiquement depuis 2007). En effet, les annuaires UDDI publiques de IBM, Microsoft et SAP par exemple sont arrêtés et il reste donc seulement les utilisations privées dans certaines entreprises. De point de vue technique et recherche scientifique, nous pouvons récapituler les lacunes suivantes des approches basées sur registres UDDI par rapport à un apport du nouveau modèle des réseau sociaux. Les modèles UDDI sont donc :

1. Non adaptés à des formation de services dans une topologie de réseaux (et moins sous forme de réseaux sociaux),
2. Appliquent une recherche/découverte de type « annuaire » et par service isolé
3. Simplistes de point de vue critères de recherche/découverte
4. Moins soutenus par l'industrie depuis 2007 et rares sont les registres qui sont publiques et accessibles aujourd'hui,
5. Négligeants de l'historique des interactions interservices, des préférences et de l'évolution des services dans le temps.

4.1.2 Proposition d'une nouvelle modélisation

Notre objective, dans cette partie, est de proposer une modélisation des réseaux sociaux des services Web tout en considérant les possibles interactions

qui peuvent exister en services Web. Une bonne modélisation permet de refléter la nature d'un réseau social et de ainsi de l'expérimenter en évaluant les approches proposées avec la facilité de varier le réseau et de tester la sacalabilité (mise en échelle) des solutions proposées. De tels tests peuvent inclure des requêtes de découverte de services mais aussi d'inclure l'évolution du réseau, sa dynamicité, et tester les algorithmes d'exploration et de recherche affinée. Notre réflexion ainsi que notre étude approfondie des travaux existants, discutée principalement dans le chapitre précédent, nous mènent vers une modélisation basée sur les graphes orientés.

Nous proposons qu'un réseau social de services Web soit modélisé par un graphe orienté $G = (SW, E)$, où l'ensemble SW représente les services Web (les nœuds) et l'ensemble E représente les relations existantes entre ces services (les arcs). Chaque nœud du graphe, représentant un service Web, est **caractérisé** par un certain nombre de **propriétés** (exemple : les types des protocoles utilisés, le nombre de méthodes exposées, le nombre d'opérations élémentaires implémentés, sa qualité de service, etc.). Contrairement à certains travaux existants, nous ne négligeons pas l'orientation du graphe dans le sens où un arc émanant de SW_i vers SW_j signifie que c'est le service SW_i qui sollicite une méthode ou qui échange un message avec le service SW_j. Notre modélisation opte pour des graphes **multiples** où nous considérons le cas où il y a des arcs multiples, c'est-à-dire que plusieurs arcs différents peuvent relier la même paire de nœuds. En effet, un service Web donné peut exposer plusieurs méthodes et peut échanger plusieurs messages avec un autre service. Chaque possible interaction est donc différente et elle est représentée par un arc.

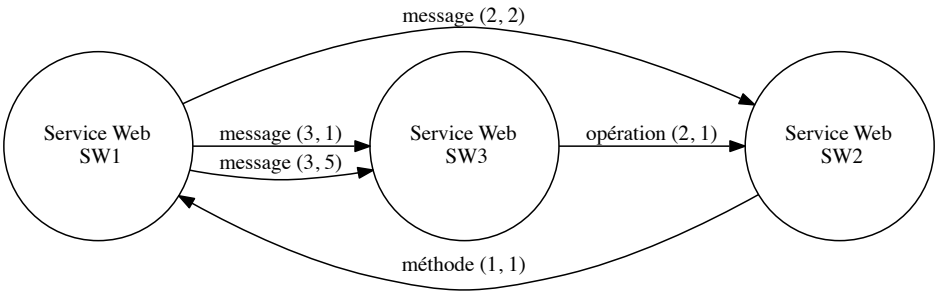

FIGURE 4.1 – Exemple d'*interactions* entre services Web dans un réseau social

L'exemple de la Figure 4.1 représente une instance de réseau social déjà formé et composé de 3 services SW_1 , SW_2 et SW_3. Les arcs présentent les possibles interactions entre services. Par exemple, SW_1 interagit avec SW_2 en utilisation le message numéro 2 exposé par SW_2. SW_2 utilise la méthode numéro 1 exposée par SW_1 . SW_1 a des relations multiples avec SW_3 et utilise les messages 1 et 5 de SW_3.

La Figure 4.2 représente le réseau social résultant de l'exemple de la Figure 4.1 en faisant abstraction du détail des interactions. Il est important de noter que, dans notre modélisation, de tels réseaux ne peuvent se former que si et seulement si les opérations et méthodes emploient des protocoles **compatibles** et supportés par la paire de nœuds concernés. A titre d'exemple, dans notre réseau social présenté par la Figure 4.1, le lien entre SW_3 et SW_2 ne peut se former qui si SW_2 expose l'opération 2 avec un protocole, exemple HTTP ou SOAP, supporté à la fois par SW_2 et SW_3. Une autre notion intuitive, prise en considération dans notre modélisation, est la **richesse** d'échange entre deux nœuds. Cette richesse, est extraite est évaluée en utilisant la notion de la théorie des graphe appelée *degré entrant* et *degré sortant* et qui dépend naturellement du nombre disponible (c-à-d exposé) de méthodes et messages compatibles pour un service donné. Le degré sortant nous renseigne sur la **forte activité** d'un service donné dans le réseau. Le degré entrant renseigne sur la **popularité** d'un service dans un réseau. En plus des valeurs des degrés, cette popularité, qu'on pourra aussi nommer **réputation**, dépendra dans

notre modélisation du nombre de protocoles supportés, nombre des messages définis, nombre d'opérations élémentaire et méthodes implémentées, et enfin, du nombre de point d'accès au sens WSDL (c-à-d URL) du service.

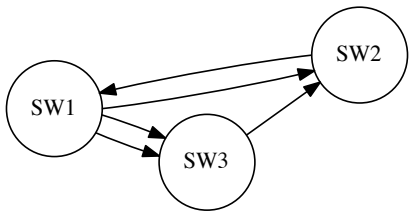

FIGURE 4.2 – Exemple d'abstraction de réseau social de services Web

Dans notre modélisation, les *arcs* représentent les interactions directes possibles entre services. Il est naturellement possible d'avoir des interactions **indirectes** entre services. En effet, à l'instar des réseaux sociaux conventionnels, une personne peut avoir besoin à communiquer indirectement avec une autre personne, par exemple à des fins de recommandation dans un réseau professionnel comme *LinkedIn*, en sous traitant une autre entité pour une tâche précise d'entreprise, ou en sollicitant par exemple une traduction linguistique d'une communication faite dans une autre langue. Dans notre modélisation, c'est la notion de *chemin* possible entre deux nœuds différents qui est utilisée. En effet, une communication indirecte entre services Web SW_i et SW_j dans une communauté sociale déjà formée, ne sera possible que s'il existe un chemin entre SW_i et SW_j. Cette communication indirecte peut être exploitée à des fins de découverte de possibilités de **composition de services** (voir Chapitre 2) même si les deux services en question ne sont pas directement compatibles. La Figure 4.3 montre une possible interaction indirecte entre les services SW_1 et SW_4 bien que les deux services n'emploient pas les mêmes protocoles de communication. Les services SW_2 et SW_3 peuvent donc jouer le rôle d'intermédiaires et appliquer les opérations nécessaires à cette composition comme les conversion de messages dans un format protocolaire donné.

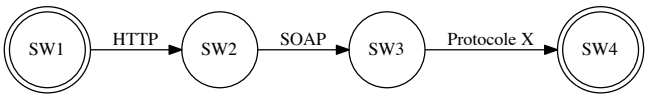

FIGURE 4.3 – Interaction indirecte entre services Web

Un réseau social est -par nature- changeant dans le temps. En effet, de nouveaux liens peuvent se créer ou disparaître, de même, pour les nœuds. Dans la Figure 4.4, nous avons –à l'instant t_1- un réseau social formé par trois services. Le réseau évolue –à l'instant t_2- avec l'apparition du nœud 4, une nouvelle interaction entre 3 et 4, et la disparition d'une interaction entre les nœuds 1 et 3. A l'instant t_3, le réseau évolue : le service 2 tombe en panne. Afin de refléter cette **dynamicité** et considérer l'évolution temporelle d'un réseau social de services Web, nous considérons les ensembles SW et E (présentés précédemment) comme des ensembles dynamiques dont le contenu varie dans le temps. Il est ainsi possible d'ajouter un service à la communauté en ajoutant un nœud à l'ensemble SW. De la même façon expliquée précédemment, les nouvelles interactions seront représentées par des arcs. Il est à noter, lorsqu'un SW_i disparaît (exemple pour des raisons de pannes ou de défaillances), le nœud correspondant sera retiré de l'ensemble SW ainsi que tous les arcs impliquant le nœud SW_i (voir Figure 4.4).

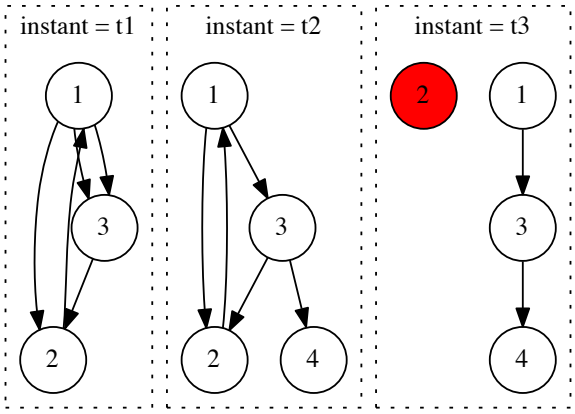

FIGURE 4.4 – Évolution d'un réseau social de WS dans le temps

Suivre l'évolution temporelle d'un réseau social de services Web est très important pour maîtriser l'**historique** et la réalité de ce réseau. Cette notion est très importante pour identifier la force des liens qui peut lier deux services. Cette force d'interaction basée sur l'historique favorise qu'un service passe par un service bien déterminé (avec lequel il a des liens forts) pour interagir avec le reste de la communauté du réseau social. Ceci est relativement similaire à la notion d'*amitié* dans les réseaux conventionnels qui définit les nœuds avec lesquels on a plusieurs interactions qui perdurent dans le temps. La notion d'historique est utilisée pour ne pas limiter notre modèle à l'état actuel d'un réseau social mais de tenir compte du passé pour les opérations ou les interactions à venir. Par exemple, pour une composition de service (ou une communication indirecte en général) et dans le cas où il existent plusieurs chemins possibles entre deux nœuds, il est évident que les chemins qui passent par des nœuds *amis* seront favorisés. Aussi, dans le cas de découverte de méthodes disponibles dans le réseau et s'il existent des choix multiples d'usage, l'usage des méthodes des nœuds amis devrait être favorisée dans toute stratégie de découverte.

Dans notre modélisation, nous considérons la notion de **qualité de services** (QoS) des SW en se basant sur les **caractéristiques** (ou propriétés) de

chaque nœud représentant un service Web. Comme mentionné précédemment, ces propriétés peuvent être diverses, elles participent à la découverte et l'identification (ou toute interaction possible) des services auxquels un service donné veut (souhait) et peut (compatibilité) interagir. Nous distinguons deux types de propriétés que nous associons aux services : les propriétés statiques et les propriétés dynamiques. Les propriété statiques sont celles qui caractérisent un service à un instant donné comme : le nombre de message disponibles, le nombre d'opérations, méthodes, points d'accès, etc. Ces propriétés peuvent êtres extraites des sources disponibles comme principalement les descriptions WSDL ou des annotations qui peuvent les enrichir. Ajouté à cela, certaines propriétés dynamiques (ou calculées) peuvent être utilisées, nous pouvons cité par exemple : la réputation (ou la popularité), l'historique, le temps de réponse perçu par un service SW_i des méthodes ou opérations exposées par un service SW_j, la taille des messages disponibles (et par conséquent le temps de transfert de ces messages), la réputation et le rangs (ranking) des points d'accès, par exemple, les statistiques publiques relatives aux noms de domaines qui sont la base des URL des points d'accès employés par un service donné, etc.

Les détails de modélisation, discutés jusqu'à maintenant, nous permettent aisément de traiter les différentes formes d'interactions qui peuvent exister dans les réseaux sociaux des services Web et qui ont été présentés et étudiés dans la littérature. Ainsi, il est facile de calquer les trois classes d'interactions déjà identifiées avec notre modélisation. Rappelons que ces trois classes sont les **collaborations**, les **substitutions** et les **compétitions** (voir Chapitre 3). Avec notre modélisation, il est possible de former un réseau social initial à l'instant t = t_0 en tenant compte des compatibilités entre services (en utilisant les propriété des nœuds) mais aussi en tenant compte des affinités existantes et qui sont antérieure à l'instant t_0 (des arcs existants pour diverses raisons, par exemple, des contrats de services, partenariats, etc.). A partir de l'instant t_0, notre modélisation permet de suivre l'évolution temporelle du réseau et la possibilité de former des nouveaux arcs ou nœuds selon la portée voulue par le développeur du réseau. Cette portée peut être axée sur un thème particulier (exemple un dataset/annuaire de services sur la santé) ou tout simplement en incluant un grand nombre de services qui s'inscrivent dans une communauté

ou un portail de services donné (exemple un site *facebook-like* pour les services ou tout simplement un annuaire d'une grande entreprise de service).

Pour un service donné SW_i, l'identification des interactions sociales de type **collaborations** revient, grâce à notre modèle, d'identifier en premier lieu les nœuds adjacents avec lesquels SW_i a au moins un arc (entrant ou sortant). De plus, pour ne pas ignorer les collaborations indirectes, il faut identifier l'ensemble des nœuds SW_j avec lesquels il existe au moins un chemin possible entre SW_i et SW_j. Ce problème est équivalent à chercher le graphe *connexe* qui inclut le nœud SW_i. En pratique, il est intéréssant à limiter la longueur du chemin séparant le nœud SW_i avec les autre nœuds en imposant un seuil ou une profondeur maximale qui représentera la distance maximale avec un service collaborateur.

L'identification des interactions de type **substitution** revient à comparer *les propriétés* des nœuds formant le réseau social afin d'identifier les nœuds fortement similaires aux nœud incriminé (c-à-d le nœud défaillant ou en panne qu'on voudrait remplacer). Vu que l'ensemble des propriétés est limité, il faut admettre que le degré de similarité ou de *matching* ne peut pas atteindre 100% dans tous les cas. Nous proposons donc d'ordonner les nœuds similaires à un nœud donné et recommander, pour substitution, les nœuds avec les scores les plus élevés. Il est intéressant de noter, qu'une perspective à cette approche, serait d'étendre la recherche à la similarité entre un nœud incriminé et une *composition de services* (chemin de nœuds) qui peut remplacer le nœud à remplacer. Cette perspective elle est à la fois intéressante mais complexe à implémenter et peut engendrer le besoin de beaucoup de ressources de calcul.

Enfin, grâce à notre modélisation, les interactions de type **compétitions** pour un nœud SW_i donné reviennent à combiner les approches précédentes utilisées pour les classes **collaborations** et **substitutions**. Ici il s'agit tout simplement de chercher les nœuds « très » similaires mais qui ne sont pas des collaborateurs du nœud SW_i.

4.2 Implémentation et Expérimentations

4.2.1 Analyse des descriptions des services Web

Afin d'implémenter la modélisation proposée dans la section précédente et par conséquent être capable de tester et d'expérimenter notre réseau social de services Web, nous nous sommes focalisés sur la norme *Web Service Description Language* (WSDL) discutée dans le Chapitre 2. En effet, c'est la brique de base pour la description des services Web ouverts et interopérables.

Nous avons donc implémenté une solution qui permet de lister les services existants ainsi que d'extraire les informations pertinentes pour caractériser ces services. Le jeu de données (dataset) utilisé a été cité dans de nombreux articles scientifiques. C'est le dataset du projet ASSAM [1] comportant 164 services Web complexes. Nous l'avons donc adopté comme une base de teste de services Web réels.

Il est très important de noter qu'à ce jour la dernière version du standard WSDL est la version 2.0 parue en juin 2007 [26]. Malheureusement, nous avons pu constater lors du travail de ce mémoire que beaucoup de sources bibliographiques et d'implémentations citent et emploient des services avec des versions différentes et souvent avec des versions anciennes qui ne tiennent pas compte des modifications majeures de la dernière version WSDL 2.0 ce qui rend certains outils notamment les outils d'analyse syntaxiques incompatibles ou non fonctionnels. L'exemple flagrant est l'élément pilier de la structure WSDL qui est l'élément *<PortType>* (voir sous-section 2.5.2). En effet, cet élément n'existe plus dans la version WSDL 2.0 et il n'est absolument pas cité dans cette dernière version officielle du W3C. C'est seulement en remontant le fil des changements du langage WSDL, en consultant la documentation officielle du W3C (en utilisant à chaque fois le lien *previous version*), qu'on s'aperçoit que la version «W3C Proposed Recommendation» (version en état de proposition, [25]) et non pas la version «W3C Recommendation» (c-à-d la version finale, [26]) cite les changements de la version WSDL 2.0 dans la section "F. Part 1 Change Log (Non-Normative) et WSDL 2.0 Specification Changes".

On apprend donc, à titre d'exemple, que l'élément *<PortType>* a été changé en l'élément *<interface>*. Pour être rigoureux, en analysant en détail l'historique de la norme WSDL (toujours avec les différentes spécifications du W3C) on découvre que ce changement spécifique de *<PortType>* a été annoncé pour la première fois dans la version brouillon (draft) du 11 juin 2003.

Par conséquent, il important de noter que le dataset du projet ASSAM [1] utilisé dans notre implémentation, bien que cité et utilisé par de sérieux travaux de recherche qui sont apparus après la sortie de la version 2.0 de WSDL comme dans [14], n'est pas compatible avec la version WSDL 2.0. A titre d'exemple, on y trouve le fameux élément *<PortType>*. Notre implémentation d'analyse des fichiers WSDL prend cela en considération. L'utilisation du dataset cité par de nombreux travaux ultérieure à la sortie de la version WSDL 2.0 est sans doute justifiée par le fait d'absence de datasets riches et publiques de services Web qui suivent la norme WSDL 2.0 et qui peuvent être expérimentés dans un travail de recherche. Un autre critique qu'on pourrait citer à l'égard du dataset ASSAM et qui ajoute une difficulté technique dans notre implémentation est que le codage WSDL des 164 services du dataset n'est pas homogène. En effet, certains éléments standard du WSDL sont parfois encodés différemment selon les services. Par exemple, dans le service nommé *84_Get_States.wsdl*, on trouve l'encodage suivant : *<wsdl :message>*, *<wsdl :portType>* et *<wsdlsoap :address..>* au lieu de *<message>*, *<portType>* et *<soap :address..>* respectivement. Dans notre implémentation, nous avons pris en considération les variations de cette syntaxe.

Puisque, la parties **«Message»** du langage WSDL représente la définition abstraite des données qui peuvent être échangées entre services ou entre un client et un service Web (voir sous-section 2.5.2), il nous paraît important de l'analyser et de l'extraire comme élément caractérisant la nature des services et les opérations qu'un service peut manipuler avec le reste de la communauté dans un réseau social donné.

Vu la multitude des protocoles d'échanges possibles entre services, nous nous

sommes focalisés sur les protocoles HTTP et SOAP qui offrent une meilleure interopérabilité entre plateformes, clients et applications hétérogènes et qui s'apprête bien au dataset utilisé. En effet, de nombreux messages du dataset sont prévus pour un échange avec le protocole HTTP et/ou SOAP avec des paramètres en entrée (arguments reçus dans les messages reçu par le service) et en sortie (argument ou données généré en output par le service sollicité par un utilisateur, un autre service ou par une application). Dans notre approche, nous prenons en considération le nombre de protocoles supportés qui peut être 1 (HTTP ou SOAP) ou bien 2 (HTTP et SOAP). L'idée est qu'un service qui supporte 2 protocoles a plus de chances d'interagir avec plus de monde (nœuds) dans le réseau social.

Grâce au codage des noms des messages employés par le dataset avec des noms de type *«*HttpPostOut»* ou *«*HttpGetIn»*, etc. (exemple *<message name="findZipCordinatesHttpPostOut">*), nous pouvons distinguer les messages, le sens du message (In/Out) et le protocole utilisé (exemple HTTP). Nous avons donc profité du codage simple utilisé par le dataset ASSAM. Cependant, pour être sûrs, plus général et ne pas limiter notre méthode d'analyse aux propriétés du dataset expérimenté, nous confirmons cette analyse avec ce qui a été prévu par le standard WSDL à savoir le listing des opérations possibles d'un services et la précision du sens des messages dans la balise <PortType> (ou <interface> dans la dernière version de WSDL 2.0). En effet, dans l'exemple suivant (Listing 4.1), nous montrons bien que le sens des messages est précisé par les éléments *<input message=...>* et *<output message=...>*. Partant de la constatation que plusieurs portType différents peuvent employer les mêmes messages, nous évitons de comptabiliser le nombre de messages en utilisant l'élément <portType> mais, à la place, nous utilisons la balise <message> afin d'identifier les messages différents et éviter de compter un même message plus d'une fois. L'élément <portType> sera donc notre base pour comptabiliser le nombre de méthodes différentes et disponibles en HTTP ou SOAP dans le contexte d'interactions avec d'autres nœuds du réseau social.

```
1 <portType name="zipCodeServiceHttpPost">
2     <operation name="findZipCodeDistance">
```

```
3               <input message="s0:findZipCodeDistanceHttpPostIn" />
4               <output message="s0:findZipCodeDistanceHttpPostOut" />
5       </operation>
6       <operation name="findZipDetails">...</operation>
7       <operation name="getCodeSet">...</operation>
8   </portType>
```

Listing 4.1 – Exemple de listing d'opérations dans l'élément <PortType> de WSDL

Dans notre implémentation d'analyse des services Web, nous avons également distingué le nombre de méthodes différentes qu'un service expose au reste du réseau social du nombre des opérations élémentaires qu'un service expose. Par construction du standard WSDL, et aussi comme le veut la logique de programmation, une méthode peut englober plusieurs opérations élémentaires qui peuvent être reprises, complètement ou partiellement, dans une autre méthode. Par exemple, une méthode X peut englober op_1, op_2 et op_3, alors qu'une autre méthode Y peut faire appel à op_2 avec d'autres méthodes. Il nous paraît donc intéressant de distinguer les services en termes de **méthodes exposées** (au sens de l'élément WSDL *<portType>* ou de *<interface>* de la version WSDL 2.0) et en termes d'**opérations élémentaires** différentes exposées (au sens de l'élément WSDL *<operation>*). Le nombre de méthodes nous paraît un indicateur intéressant qui montre les capacités pour un service d'interagir avec les autres services, tandis que le nombre d'opérations peut indiquer la richesse du service ce qui le favorise pour des opérations de types collaboration ou composition.

Dans notre analyse des services Web nous considérons également tous les **mots clés** forts du langage naturel qui peuvent exister dans une description de service Web. Ces mots clés peuvent être utilisés pour mieux décrire le fonctionnement du service au delà des noms d'opérations ou des méthodes qui peuvent être invoquées automatiquement par un processus d'exécution. Notre approche proposée peut fonctionner, de la même manière, sur la base d'autres sources de mots clés comme celles qui peuvent provenir d'une annotation manuelle de services et qui permettent d'avoir d'autres descriptions plus affinées sur les services Web. C'est l'approche adoptée dans certains travaux

comme dans [14] ; nous citons : "*After manually annotating the Web services in the data-set for similarity assessment, we identied two categories..*" (voir section *Experiments*, page 53 de [14]). Bien que, sans trop de modifications, notre implémentation peut utiliser d'autres sources de mots clés, dans notre travail nous n'avons pas effectué d'annotations manuelles, contrairement au travail de [14]. Notre objectif est de rendre le processus d'analyse des descriptions le plus automatique possible. Par conséquent, nous recommandons aux développeurs de service d'ajouter le maximum d'informations utiles pour la description de leurs services en utilisant, par exemple, l'élément WSDL *<documentation>*.

Enfin, nous considérons aussi le nombre de **point d'accès** qu'un service expose, ce sont les URL publiques avec lesquels les interactions peuvent se faire avec un service donné. Rappelons que les informations sur les *point d'accès* peuvent être extraites en utilisant l'élément WSDL nommé *port*, voir sous-section 2.5.2.

Le Tableau 4.1 donne un extrait du résultat riche de notre analyse des 164 services du dataset ASSAM avec les 14 paramètres suivants que nous avons adopté pour les interactions et manipulations des services dans un environnement de réseau social. Les paramètres sont : le nombre de messages possibles en entrée et en sortie (avec les protocoles HTTP, SOAP ou autres) ; le nombre de méthodes ainsi que les opérations élémentaires possibles (avec les protocoles HTTP, SOAP ou autres) ; le nombre de points d'accès pour un services (en HTTP ou en SOAP), et enfin, le nombre de mots clés forts qui décrivent le service. Pour des raisons de limitations de taille, nous nous limitons à l'extrait de 10 services Web Tableau 4.1. Cependant, afin d'avoir une vue globale des résultats de l'analyse, dans la Figure 4.5 nous présentons, sous forme de courbes, avec les 14 critères discutés précédemment. Dans Figure 4.6, nous présentons cette analyse en la focalisant sur seulement sur les protocoles HTTP et SOAP.

Dans la Figure 4.5, nous pouvons remarquer l'hétérogénéité des services considérés : utilisation de HTTP, SOAP pour les possibles messages qu'un

service peut échanger mais aussi d'autres protocoles pour les opérations élémentaires et les méthodes complexes exposées. Ceci reflète bien la réalité d'un réseau social où les protagonistes de tels réseaux (utilisateurs finaux) sont hétérogènes ce qui convient bien à notre environnement d'expérimentation des services Web dans les réseaux sociaux. Notons aussi qu'en termes d'expressivité et de différentiation, selon nos critères, certains services Web se font remarqués par une forte valeur par rapport à certains critères. Par exemple, le service #14 détient la valeur la plus grande par rapport au nombre de message non HTTP/SOAP avec 120 messages possibles, le service #145, quand à lui, détient la valeur la plus haute en terme de mots clés employés dans sa description (avec 89 mots clés).

La vue de notre analyse axée sur les support des protocoles HTTP et SOAP, présentée dans la Figure 4.6, nous montre qu'il y a de nombreux services qui supportent les deux protocoles cités, en termes de messages et d'opérations, ce qui augmente les chances de les faire connecter dans un contexte social ouvert. En terme de richesse, 38 services implémentent un minimum de 12 messages HTTP et le nombre de ce type de message peut augmenter jusqu'à 80 messages en HTTP ; 19 services assurent un minimum de 10 opérations possibles en HTTP (ce nombre peut augmenter jusqu'à 40 opérations). Le support de SOAP est présent mais moins flagrant que le HTTP : 23 services peuvent échanger au moins 10 messages possibles en SOAP et cela peut augmenter jusqu'à 40 messages ; 6 services définissent un minimum de 10 opérations possibles et ceci peut augmenter jusqu'à 20 opérations. En prenant compte de tous nos critères rassemblés (de point de vue HTTP et SOAP), le service #14 (un service de notification de mail) est le service le plus développé avec, par exemple, 80 messages HTTP, 40 messages SOAP, 40 opérations HTTP et 20 opérations SOAP.

TABLE 4.1 – Extrait des résultats de notre analyse de 164 services Web.

SW	**#mess. HTTP**		**#mess. SOAP**		**#autres**	**#ens. d'op.**	**#op. HTTP**	**#ens. d'op.**	**#op. SOAP**	**#autres**	**#autres**	**#pt. d'accès**	**#pt. d'accès**	**#mots**
	entrée	sortie	entrée	sortie	**mess.**	**HTTP**	**élémentaires**	**SOAP**	**élémentaires**	**ens. d'op.**	**op. élém.**	**HTTP**	**SOAP**	**clés**
100	2	2	1	1	6	1	2	1	1	3	3	1	1	2
101	4	4	2	2	12	1	4	1	2	3	6	1	1	3
102	0	0	0	0	14	0	0	0	0	1	7	0	1	23
103	6	6	3	3	21	1	6	1	3	3	9	1	1	7
104	2	2	1	1	6	1	2	1	1	3	3	1	1	2
105	2	2	1	1	6	1	2	1	1	3	3	1	1	2
106	4	4	2	2	12	1	4	1	2	3	6	1	1	3
107	12	12	6	6	36	1	12	1	6	3	18	1	1	7
108	4	4	2	2	12	1	4	1	2	3	6	1	1	3
109	14	14	7	7	42	1	14	1	7	3	21	1	1	8

Significaction des colonnes : ♦ **SW** : le numéro du service Web tel numéroté dans le dataset ASSAM [1]. Dans ce tableau, nous donnons ce numéro afin de faciliter l'examen du service en question à partir du dataset expérimenté. ♦**#mess. HTTP (SOAP)** : le nombre de messages possibles que le service peut échanger en utilisant le protocole HTTP (ou SOAP) en entrée ou en sortie. ♦**#autres mess.** : le nombre d'autres possibles messages (autres que HTTP et SOAP) que le service peut échanger. ♦**#ens. d'op. HTTP (SOAP)** : le nombre de groupement d'opérations (dans le sens *PortType* ou *Interface* du standard WSDL) que le service offre en HTTP ou en SOAP. ♦**#op. HTTP (SOAP) élémentaires** : le nombre d'opérations élémentaires différentes qu'un service offre en HTTP ou en SOAP. ♦**#autres ens. d'op.** : le nombre de groupement d'opérations (dans le sens *PortType* ou *Interface* du standard WSDL) que le service offre en utilisant un protocole autre que HTTP ou SOAP. ♦**#autres op. élém.** : le nombre d'opérations élémentaires différentes qu'un service offre en utilisant un protocole autre que HTTP ou SOAP. ♦**#pt. d'accès HTTP (SOAP)** : nombre de points d'accès que l'utilise définit en utilisant le protocole HTTP ou SOAP (au senes *port* du standard WSDL) ♦**#mots clés** : le nombre de mots clés que nous avons extrait en utilisant notre analyse syntaxique de la description du services Web.

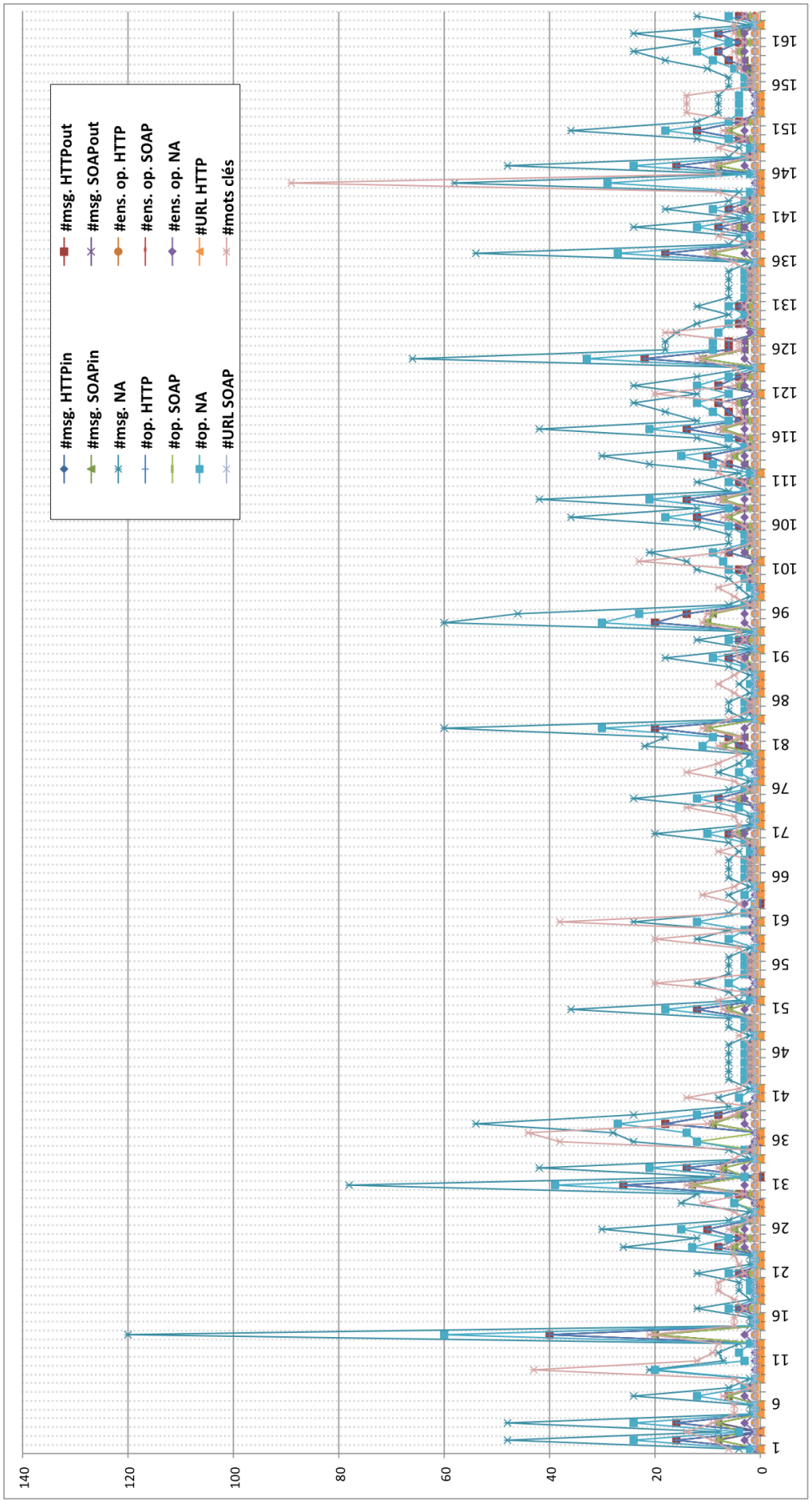

FIGURE 4.5 – Vue globale des résultats d'analyse de 164 services Web

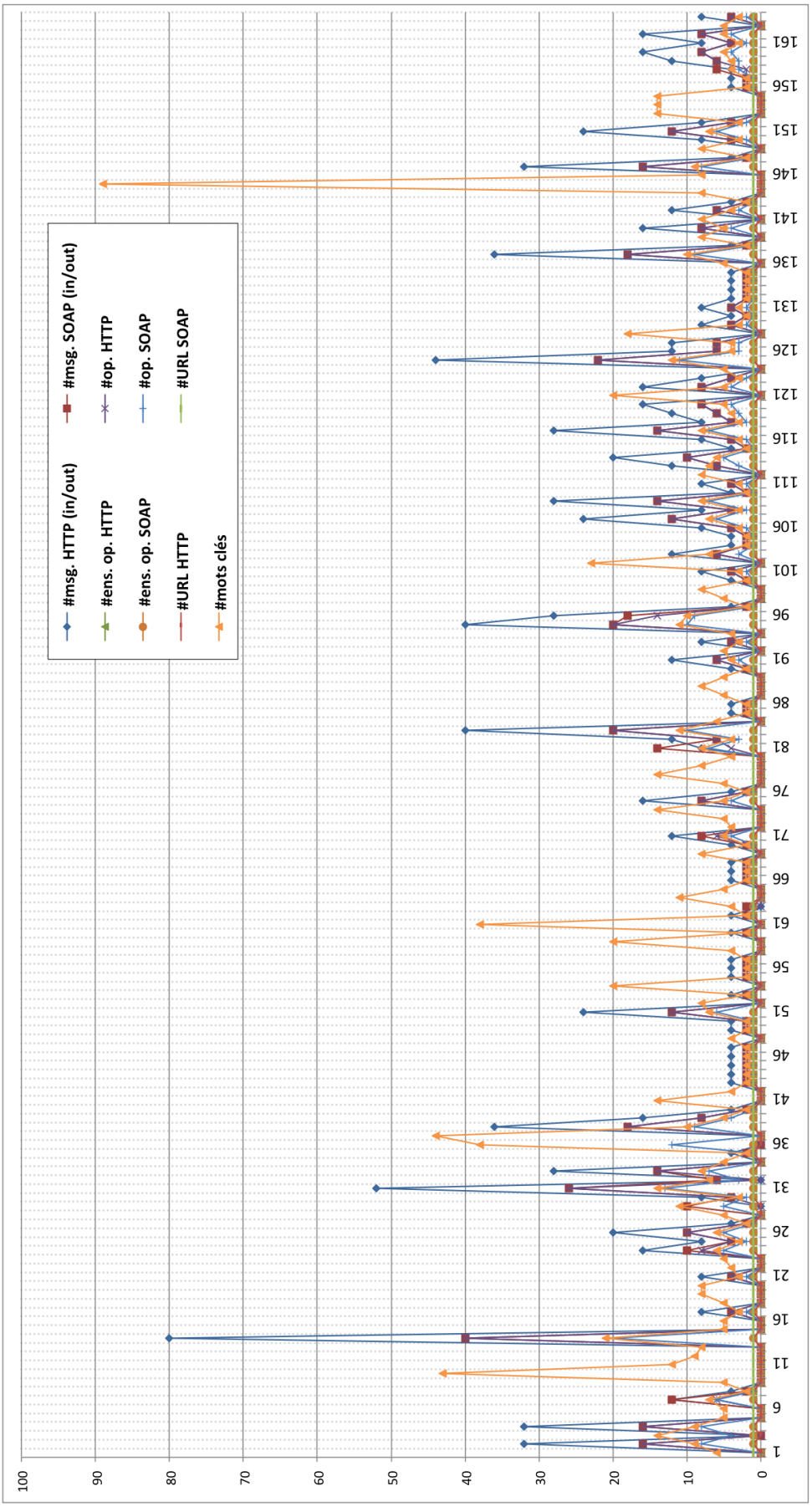

FIGURE 4.6 – Vue axée sur HTTP et SOAP des résultats d'analyse de 164 services Web

Le code présenté dans Listing 4.2 montre un extrait de notre implémentation du listing et de l'analyse des descriptions WSDL des services Web. C'est cette implémentation qui nous a permis d'obtenir, entre autres, les résultats présents dans Tableau 4.1, Figure 4.5, et Figure 4.6.

```
for i=1:size(WS_directories)
       numeroDuService = i; serviceName = '';
       cheminFichierWSDL = strcat ('chemin_absolu',WS_directories(i).name);
       docNode = xmlread(cheminFichierWSDL);
       rootNode = docNode.getDocumentElement;
       entries = rootNode.getChildNodes;
       node = entries.getFirstChild; % Obtenir le premier fils de l'arboresence
       compteur_MessageHTTP = 0; % nombre de <message> employant le protocole HTTP
       compteur_MessageSOAP = 0; % nombre de <message> employant le protocole SOAP
       compteur_MessageNA = 0; % nombre de <message> n'employant ni HTTP ni SOAP
       compteur_MessageHTTPin = 0; compteur_MessageSOAPin = 0; compteur_MessageHTTPout = 0;
       compteur_MessageSOAPout = 0; compteur_PortTypeHTTP = 0; compteur_OperationsHTTPWS = 0;
       compteur_PortTypeSOAP = 0; compteur_OperationsSOAPWS = 0; compteur_PortTypeNA = 0;
       compteur_OperationsNAWS = 0; operationsNAWS = {}; compteur_KeyWords = 0; compteur_URL_HTTP
       = 0; compteur_URL_SOAP = 0; syntacticContentOneService = {};
       operationsWS = {}; % les noms des operations pour ne garder que celles qui sont differents
       URL_HTTP_WS = {}; URL_SOAP_WS = {}; % les noms des points d'acces
       while ~isempty(node)
            %if strcmpi(node.getNodeName, 'message')
            if any([strfind(node.getNodeName, 'message')>0]) % nous employons strfind et non
            %pas strcmpi (une comparaison stricte) pour pallier aux differentes variantes
                nomMessage = char(node.getAttribute('name'));
                %Analyse des interaction HTTP en Input
                if any([strfind(nomMessage, 'HttpGetIn')>0, strfind(nomMessage, 'HttpPostIn')>0])
                  compteur_MessageHTTP = compteur_MessageHTTP +1;
                  compteur_MessageHTTPin = compteur_MessageHTTPin+1;
                  %On considere le nom des messages dans la recherche Syntaxique
                  %(QUE les keywords differents)
                  nomMessage = eliminateLightKeyWords(nomMessage); % nous avons defini cette
                  %fonction pour eliminer les mots non significatifs (ex. the, of, get,
                  %HttpPost), voir eliminateLightKeyWords.m
                  extractKeyWords(nomMessage);%etxrait et stoke les mots cles for dans
                  %"syntacticContentOneService", mets a jour aussi "compteur_KeyWords"
                end
            end ...
            node = node.getNextSibling; %passer au fils suivant
        end ...
       syntacticContentAllServices {numeroDuService} = syntacticContentOneService;
```

```
39        numeroFromDataset = strtok(WS_directories(i).name,'_');
40        ...
41  end %fin parcours des services WS
```

Listing 4.2 – Extrait simplifié de code d'analyse des descriptions WSDL des services Web

La fonction *extractKeyWords* (ligne 32, Listing 4.2) est la base de notre analyse syntaxique des descriptions WSDL. Dans Listing 4.3, nous donnons un extrait de notre implémentation de cette fonction. Comme, mentionné précédemment, nous appliquons notre fonction à tous les éléments XML (noms de messages, méthodes, opérations, noms de services, etc.) afin d'en extraire le maximum de mots clés significatifs. Nous profitons de l'encodage des noms XML utilisé dans l'encodage des fichiers WSDL tout en séparant les noms complets selon les positions des lettres majuscules. Par exemple, le nom *GetHTTPTemperatureOfTheCity* donnera les mots clés suivants $\{temperature, city\}$. Notons que les mots ou les acronymes non significatifs (exemple Get et HTTP) sont éliminés soit par la fonction *extractKeyWords* ou *eliminateLightKeyWords* (lignes 29 et 32, Listing 4.2).

Afin d'évaluer notre implémentation d'analyse des descriptions des services Web (ce qui inclus l'analyse des 14 paramètres détaillés dans Tableau 4.1 et l'analyse syntaxique), nous avons testé notre code sur plusieurs machines. Pour donner une idée sur la rapidité de notre analyse, la durée d'exécution de notre code appliqué sur le dataset ASSAM [1], comportant 164 services, d'une taille totale de 10,7 Mo, sur une machine de processeur 2,7 GHz et d'une mémoire vive de 8Go est de 8.85 secondes.

```
1  function extractKeyWords(str)
2      global syntacticContentOneService;
3      global compteur_KeyWords;
4      str = strrep(str,'URL','Url'); %remplacer certains mots acronyme pour ne pas etre splites
5      % en lettres isolees
6      str = ...
7      tableauMajuscules = isstrprop(str,'upper');%1 si majuscule, 0 sinon
8      indicesMajuscules = find(tableauMajuscules);%identifier les indices des lettres majuscules
9      if ~isempty(indicesMajuscules) % si le tableau d'indices n'est pas vide, extraire les mots
10         if (indicesMajuscules(1)~=1) %si la premiere lettre n'est pas majuscule
```

```
11            kw=lower(str(1:indicesMajuscules(1)-1));%considerer la premiere partie
12            %de noms qui debutent par un minuscule, ex. "pressionTemp"
13            if (ismember(kw,syntacticContentOneService)==0)
14                    compteur_KeyWords = compteur_KeyWords + 1;
15                    syntacticContentOneService{compteur_KeyWords} = kw;
16             end
17        end % si la premiere lettre est miniscule
18        for i = 1:length(indicesMajuscules)-1 %parcours des autres parties
19            kw=lower(str(indicesMajuscules(i):indicesMajuscules(i+1)-1));
20            if (ismember(kw,syntacticContentOneService)==0)
21                    compteur_KeyWords = compteur_KeyWords + 1;
22                    syntacticContentOneService{compteur_KeyWords} = kw;
23            end
24        end % fin parcours des autres parties du mot complet
25        kw=lower(str(indicesMajuscules(length(indicesMajuscules)):length(str)));
26        if (ismember(kw,syntacticContentOneService)==0)
27                compteur_KeyWords = compteur_KeyWords + 1;
28                syntacticContentOneService{compteur_KeyWords} = kw;
29        end % fin de traitement de la derniere partie du nom
30    else % cas ou il n y a pas de majuscules du tout dans le nom
31        kw=lower(str)
32        if (ismember(kw,syntacticContentOneService)==0)
33                compteur_KeyWords = compteur_KeyWords + 1;
34                syntacticContentOneService{compteur_KeyWords} = kw;
35        end
36    end
37 end
```

Listing 4.3 – Extrait de la fonction *extractKeyWords*

4.2.2 Formation d'un réseau social

En suivant notre modélisation, nous proposons un nouveau pseudo algorithme, détaillé dans algorithm 1, qui explique notre approche de formation d'un réseau social à l'état initial ($t = t_0$). La formation est faite délibérément d'une manière aléatoire que ce soit en ce qui concerne la taille du réseau ou les connexions possibles entre services Web. Cependant, dans cette formation aléatoire, la compatibilité et les préférences des services sont prises en considération. Si pour une raison donnée on souhaite forcer une taille fixe

du réseau social, à n par exemple, il suffit de changer la ligne 3, de notre algorithme, avec $tailleReseauSW \leftarrow n$.

Comme mentioné, notre expérimentation se base sur un dataset de services réels, nous considérons donc les compatibilités et les préférences des services avant de les connecter dans le réseau. Les préférences protocolaires et les préférences relatives aux mots clés (notées pp et pm) font partie de la notion de *propriétés* (notion introduite auparavant) de chaque nœud du réseau social (c-à-d chaque service). Elles indiquent : les protocoles qu'un service supporte pour communiquer avec les autres services du réseau social (ex. HTTP ou SOAP ou les deux), et les mots clés de recherche que le service souhaite trouver dans ce réseau respectivement. Il est donc nécessaire pour connecter un service Web SW_i à un autre service SW_j que le service SW_j expose des méthodes qui supportent les protocoles souhaités par SW_i, la même chose pour les mots clés, sans cela le test de matching entre SW_i et SW_j ne pourra être concluant (ligne 17). Rappelons que notre méthode d'analyse, présentée dans sous-section 4.2.1 nous permet d'avoir les propriétés des services qui contiennent les protocoles et les mots clés de chaque service (ligne 1).

Enfin, dans notre algorithme, le parcours des $tailleReseauSW^2$ nœuds est nécessaire (voir les deux boucles en lignes 14 et 15 de algorithm 1). En effet, nous souhaitons former un graphe orienté comme nous l'avons introduit dans notre modélisation. Dans le cas d'un graphe non orienté, le parcours de $\frac{tailleReseauSW^2}{2}$ serait suffisant car on se focalisera seulement sur le triangle supérieur (ou inférieur) de la matrice d'adjacence, l'autre triangle étant déduit par symétrie.

Algorithm 1: Former un réseau social initial

Data: dataset de services Web (DT), préférences protocoloaires (pp), préférences mots clés (pm)

Result: Matrice d'adjacence du réseau social (SM) , vecteur des identifiants des services ($idSW$)

1 $descriptions \leftarrow$ analyser (DT) ; ▷ l'analyse des SW, voir sous-section 4.2.1
2 $dts \leftarrow$ taille(DT) ; ▷ la taille du dataset
3 $tailleReseauSW \leftarrow$ randi([2 dts]) ; ▷ fixer une taille aléatoire du réseau
4 $idSW \leftarrow$ [] ; ▷ les identifiants des services formant le réseau
5 $i \leftarrow 0$;
6 **while** $i < tailleReseauSW$ **do**
7 $SW \leftarrow$ randi([1 dts]) ; ▷ sélectionner un service au hasard
8 **if** SW *not in* $idSW$ **then**
9 $i \leftarrow i+1$;
10 $idSW = [idSW, SW]$; ▷ l'ajouter à l'ensemble des SW sélectionnés
11 **end**
12 **end**
13 $SM \leftarrow$ zeros ($tailleReseauSW$, $tailleReseauSW$) ; ▷ matrice d'adjacence nulle
14 **for** $i = 1 : tailleReseauSW$ **do**
15 **for** $j = 1 : tailleReseauSW$ **do** ▷ établir les connexions orientées
16 **if** $j \neq i$ **then** ▷ ne pas comparer i avec lui même
17 $test \leftarrow$ match($pp(idSW(\text{i}))$, $pm(idSW(\text{i}))$, description($idSW(j)$)) ;
 ▷ $idSW$(j) matche $idSW$(i) ?
18 **if** $test$ **then** ▷ si ça matche on connecte aléatoirement
19 $test \leftarrow$ randi([0 1]) ; ▷ choisir 0 ou 1 aléatoirement
20 **if** $test$ =1 **then**
21 $SM(i,j) \leftarrow 1$; ▷ connecter i à j dans la matrice d'adjacence
22 **end**
23 **else**
24 $SM(i,j) \leftarrow 0$; ▷ ne pas connecter i à j car ça ne matche pas
 ▷ cette instruction est donnée pour simplifier le code, elle est inutile car SM est déjà nulle
25 **end**
26 **end**
27 **end**
28 **end**

La Figure 4.7 montre le résultat de notre algorithme pour une formation aléatoire d'un réseau social de services Web appartenant au dataset ASSAM [1]. Ici, les interactions entre services ne sont pas détaillées car, pour des raisons de simplification, nous considérons une vue abstraite du réseau social ou un arc représente une interaction possible entre une paire de services sans pour autant rentrer dans le détail des messages ou méthodes utilisés dans cette interaction. La matrice d'adjacence correspondante générée suivant notre modélisation est représentée dans Figure 4.8. Les identifiants sélectionnés sont donnés dans le vecteur $idSW$ présenté dans le tableau Tableau 4.2

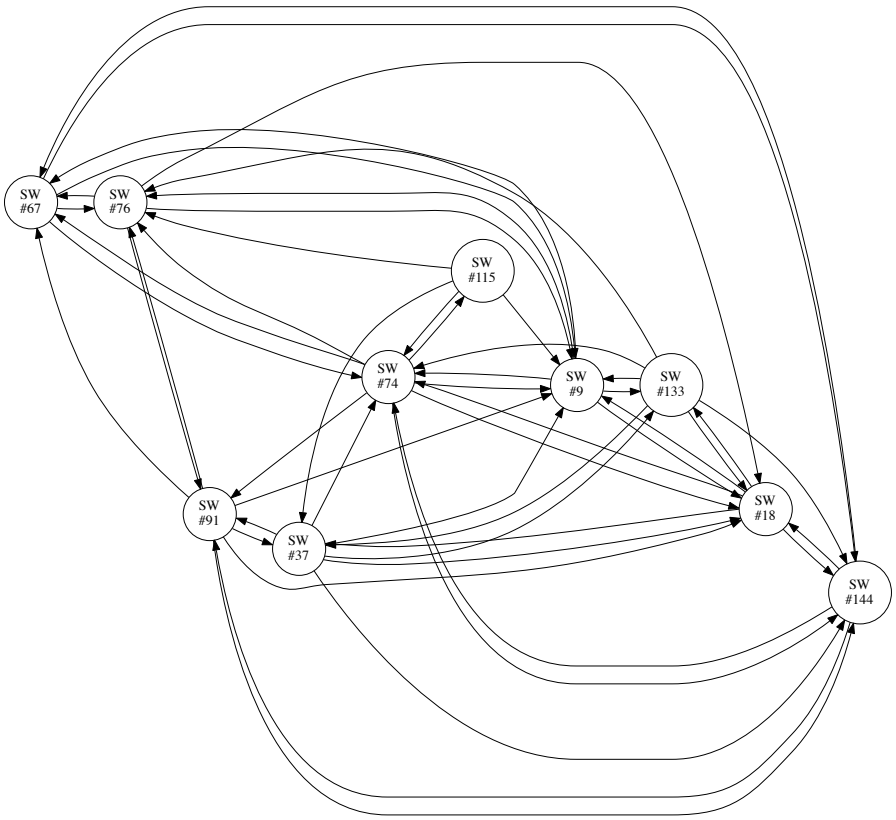

FIGURE 4.7 – Exemple de formation de réseau social de taille 10

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

FIGURE 4.8 – Matrice d'adjacence SM

$idSW$:	#67	#18	#76	#74	#91	#133	#115	#144	#9	#37

TABLE 4.2 – Identifiants des services Web aléatoirement choisis

4.2.3 Évolution temporelle d'un réseau social

Contrairement aux registres et annuaires classiques des services Web (ex. UDDI) qui se limitent à recenser les services et leurs descriptions, dans un modèle de réseau social il est important de suivre l'évolution d'un réseau social dans le temps et de pouvoir le modéliser par exemple pour identifier les liens forts entre services ou pour identifier les pannes des services pour pouvoir les remplacer, ou encore à des fins de composition de services. Nous proposons l'algorithme détaillé dans algorithm 2 et qui inclut cinq fonctions d'ajout ou de suppression d'interactions entre services, d'ajout ou de panne de service dans le réseau social, et d'ajout et initialisation d'interactions aléatoires (mais faisables) pour le service ajouté. Notons que la panne d'un service engendre, automatiquement, la suppression de toutes les interactions sociale possibles qui l'implique. Pour des raisons de simplification, la fonction de suppression d'interaction entre deux service (*SupprimerUneInteraction*) n'a pas été présentée car elle est très similaire à la fonction *AjouterUneInteraction*. En effet, la seule différence entre les deux fonctions est la mise à jour de la matrice d'adjacence avec la valeur nulle ($SM(idx_i, idx_j) \leftarrow 0$) au lieu de la valeur 1.

Algorithm 2: Faire évoluer un réseau social

Data: Matrice d'adjacence du réseau social (SM), vecteur des identifiants des services ($idSW$), préférences protocoloaires (pp), préférences mots clés (pm)
Result: SM, $idSW$

Function *AjouterUneInteraction* ($idSW_i$, $idSW_j$)
  ▷ trouver la position des services par leurs IDs :
  $idx_i \leftarrow \text{find}(idSW == idSW_i)$; $idx_j \leftarrow \text{find}(idSW == idSW_j)$;
  if *not isempty*(idx_i) *& not isempty*(idx_j) **then**
    $SM(idx_i, idx_j) \leftarrow 1$; ▷ mettre à jour la matrice d'adjacence
  end
end
Function *AjouterUnService* ($idSW_i$)
  $tailleReseauSW \leftarrow \text{taille}(SM)$;
  $SM(:, tailleReseauSW+1) \leftarrow \text{zeros}(tailleReseauSW, 1)$;
  ▷ ajouter une colonne vide dans la matrice d'adjacence
  $SM(tailleReseauSW+1, :) \leftarrow \text{zeros}(1, tailleReseauSW+1)$;
  ▷ ajouter une ligne vide dans la matrice d'adjacence
  $idSW \leftarrow [idSW, idSW_i]$; ▷ ajouter l'ID du service
end
Function *SupprimerUnService* ($idSW_i$)
  $idx \leftarrow \text{find}(idSW == idSW_i)$;; ▷ trouver l'indexe du service par son ID
  if *isempty(idx)* **then** ▷ si le service existe supprimer ID et noeud
    $idSW(idx) \leftarrow []$; $SM(:, idx) \leftarrow []$; $SM(idx, :) \leftarrow []$;
  end
end
Function *AjouterUnServiceEtInteractions* ($idSW_i$)
  AjouterUnService ($idSW_i$);
  for $k = 1 : tailleReseauSW$ **do**
    $test \leftarrow \text{match}(pp(idSW(k)), pm(idSW(k)), \text{description}(idSW_i))$;
    ▷ le service à ajouter est compatible avec ceux qui existent ?
    if *test* **then**
      $test \leftarrow \text{randi}([0\ 1])$; ▷ choix aléatoire
      if *test == 1* **then**
        $SM(k, tailleReseauSW + 1) \leftarrow 1$; ▷ connecter le réseau à $idSW_i$
      end
    end
    $test \leftarrow \text{match}(pp(idSW_i), pm(idSW_i), \text{description}(idSW(k)))$;
    ▷ les services existant sont compatibles avec $idSW_i$?
    if *test* **then**
      $test \leftarrow \text{randi}([0\ 1])$; ▷ choix aléatoire
      if *test == 1* **then**
        $SM(tailleReseauSW + 1, k) \leftarrow 1$; ▷ connecter $idSW_i$ aux réseau
      end
    end
  end
end

La Figure 4.9 montre l'ajout du service #164 au réseau social présenté déjà dans Figure 4.7 en employant la fonction $AjouterUnServiceEtInteractions(idSW_i)$ de algorithm 2. Ici, $idSW_i$ est égal à #164, le service #164 étant un service Web réel du dataset ASSAM. Notons que, grâce à la fonction *AjouterUnServiceEtInteractions*, le service #164 est ajouté et des interactions aléatoires mais compatibles (au sens des préférences protocolaires et des mots clés) sont crées.

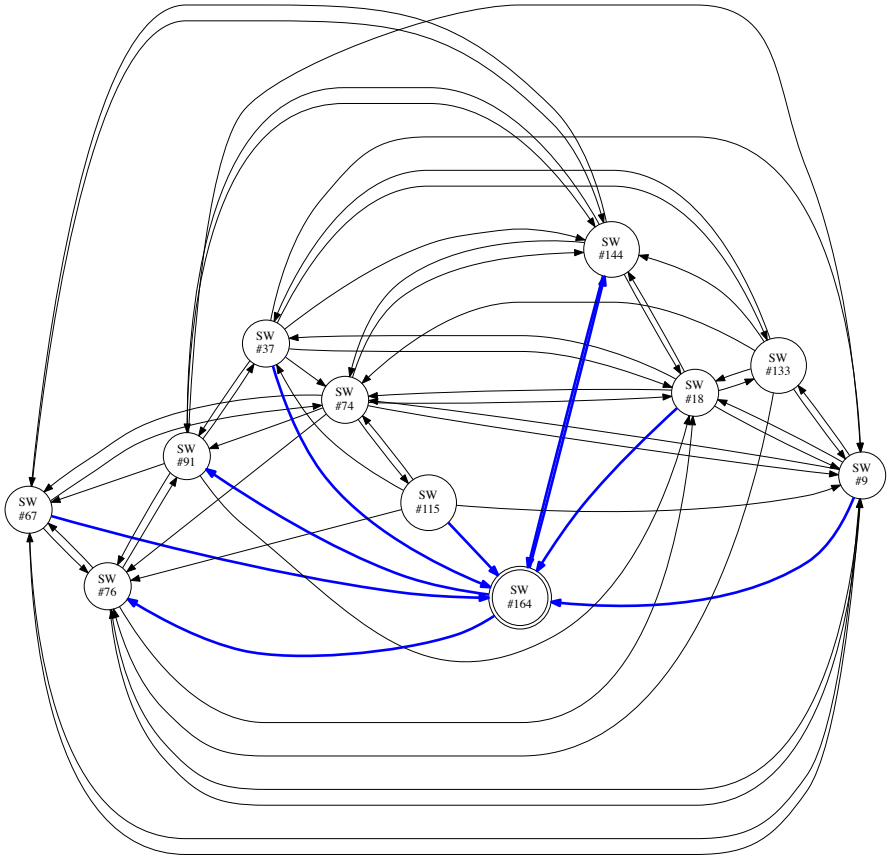

FIGURE 4.9 – Évolution du réseau social (de la Figure 4.7) avec l'ajout d'un service

4.2.4 Prise en compte de l'historique des interactions sociales

Les implémentations précédentes de formation d'un réseau social initial et la possibilité de le faire évoluer dans le temps (sous-section 4.2.2 et sous-section 4.2.3) sont très intéressantes pour représenter est expérimenter notre modélisation mais aussi pour exploiter avec aisance le paradigme social dans les interactions possibles entre services Web. Comme mentionné dans la sous-section 4.2.3, faire évoluer et suivre cette évolution du réseau peut servir à atteindre plusieurs objectifs notamment pour identifier les liens forts entre services. Cette identification peut servir à des fins de composition de services ou pour favoriser un service avec lequel on a des liens forts dans le cas où il existent de multiples choix pour une utilisation de méthodes ou d'opérations du réseau. On retrouve une notion similaire dans les réseaux sociaux conventionnels qui est la notion d'*amitié* entre individus ou de *partenariat* entre entreprises.

Notre modélisation nous permet d'identifier la force des liens sociaux dans le temps d'une manière très simple, peu gourmande en calcul mais efficace. En effet, pour résoudre ce problème, il suffit d'additionner les matrices d'adjacence SM_t (la matrice SM à l'instant t) obtenues au fil de l'évolution du réseau (c-à-d à chaque changement du réseau).

$$S_t = \sum_{i=1}^{t} SM_i \tag{4.1}$$

En effet, en faisant de la sorte, chaque lien qui persiste est comptabilisé dans une matrice somme S (voir équation 4.1). Par conséquent, les liens qui perdurent dans le temps seront représentés par une valeur importante dans S pour une paire de service donnée. Notons qu'à force que le réseau évolue, les valeurs de la matrice S augmentent mais pas de la même manière. Une valeur $S(i,j)$ de la matrice S n'a donc pas de sens que si elle est comparée avec les autre valeurs de S pour comparer la force des interactions entre services i et j et le reste du réseau. Notons que S est initialisée à SM, lors de la toute première formation du réseau initial (voir algorithm 1).

Le pseudo algorithme présenté dans algorithm 3 montre les modifications

d'implémentation que nous appliquons à l'algorithme d'évolution du réseau social présenté précédemment dans algorithm 2. Ces modifications sont appliquées à chaque fois où une évolution ou un changement du réseau social se produit. Concrètement, ceci concerne les fonctions : *AjouterUneInteraction*, *AjouterUnService*, *SupprimerUnService* ou *AjouterUnServiceEtInteractions* de algorithm 2. La fonction *SupprimerUneInteraction* n'a pas d'impact sur S car si un lien est supprimé il ne sera de toute façon pas comptabilisé dans les futures évolutions du réseau.

Algorithm 3: Prise en compte de l'historique d'évolution d'un réseau social

Data: SM, $idSW$, pp, pm, S
Result: SM, $idSW$, S

Function *AjouterUneInteraction ($idSW_i$, $idSW_j$)*
  ...
  if ... **then**
    $SM(idx_i, idx_j) \leftarrow 1$;
    $S(idx_i, idx_j) \leftarrow S(idx_i, idx_j) + 1$; ▷ mettre à jour la matrice somme
  end
end
Function *AjouterUnService ($idSW_i$)*
  ...
  $SM(:, tailleReseauSW+1) \leftarrow \text{zeros}(tailleReseauSW, 1)$;
  $S(:, tailleReseauSW+1) \leftarrow \text{zeros}(tailleReseauSW, 1)$;
  ▷ ajouter une colonne vide dans la matrice somme
  $SM(tailleReseauSW+1, :) \leftarrow \text{zeros}(1, tailleReseauSW+1)$;
  $S(tailleReseauSW+1, :) \leftarrow \text{zeros}(1, tailleReseauSW+1)$;
  ▷ ajouter une ligne vide dans la matrice somme
  ...
end
Function *SupprimerUnService ($idSW_i$)*
  ...
  if ... **then**
    $idSW(idx) \leftarrow []$; $SM(:, idx) \leftarrow []$; $SM(idx, :) \leftarrow []$;
    $S(:, idx) \leftarrow []$; $S(idx, :) \leftarrow []$ ▷ supprimer pour éviter le décallage des SW dans S
  end
end
Function *AjouterUnServiceEtInteractions ($idSW_i$)*
  ...
  for ... **do**
    ...
    if *test* **then**
      ... **if** *test == 1* **then**
        $SM(k, tailleReseauSW + 1) \leftarrow 1$;
        $S(k, tailleReseauSW + 1) \leftarrow S(k, tailleReseauSW + 1) + 1$;
        ▷ comptabiliser le lien
      end
    end
      ...
      $SM(tailleReseauSW + 1, k) \leftarrow 1$;
      $S(tailleReseauSW + 1, k) \leftarrow S(tailleReseauSW + 1, k) + 1$;
      ...
  end
end

L'avantage de notre approche d'addition de matrices est qu'elle est simple à implémenter et peu gourmande en ressource (simples additions). L'inconvénient est que les dimensions des deux matrices SM et S doivent être identiques ce qui engendre la perte d'historique pour les services qui quittent le réseau social (suppression de nœud dans SM et par conséquent dans S, voir ligne 20 dans algorithm 3).

4.2.5 Similarités entre services

En l'absence de données sémantiques dans les descriptions de services, nous proposons une approche de calcul de similarités entre services basée sur l'analyse syntaxique du langage naturel utilisé dans les descriptions WSDL des services. Nous nous basons donc sur l'ensemble des mots clés extraits par notre fonction *extractKeyWords* discutée précédemment (voir Listing 4.3). De ce fait, si un développeur fournit une description riche de son service, le calcul de similarité est affiné. Le choix de cette approche est délibéré. En effet, nous voulons que ce type de traitement soit automatique et non pas manuel. Comme mentionné auparavant dans ce mémoire, d'autre travaux (ex. [14]) ont appliqué une annotation manuelle des dataset de services puis ont proposé une similarité sémantique.

Après examen attentif des méthodes qui se présentent à nous comme l'utilisation du coefficient de *Dice*, coefficient de *Jaccard* ou le coefficient de recouvrement [5], nous avons opté pour un calcul basé sur la *Similarité cosinus* qui pénalise moins pour les ensembles de tailles très différentes. C'est en effet notre cas, notre méthode d'analyse présentée au début de ce chapitre génère des ensembles de mots clés de tailles différentes pour tous les services Web. Pour calculer la similarité entre deux ensembles, nous appliquons donc l'équation 4.2.

$$cosinus(SW_i, SW_j) = \frac{|keySW_i \cap keySW_j|}{\sqrt{|keySW_i|.|keySW_j|}} \tag{4.2}$$

Une des améliorations possibles à notre implémentation est l'emploi du facteur *term frequency-inverse document frequency* (nommé TF-IDF) qui donne la valeur d'importance d'un mot dans un ensemble (c-à-d dans un corpus). TF-IDF vise à donner un poids plus important aux termes les moins fréquents, considérés comme plus discriminants (c-à-d qui distingue mieux les services par leurs descriptions). Cette perspective éliminera donc les mots non signifiants dans les descriptions WSDL des services Web qui forment un réseau social à un instant donné. Car, en effet, à chaque évolution du réseau social, il faudrait recalculer les TF-IDF ce qui engendrera plus de calcul.

Le code présenté dans Listing 4.4 montre notre implémentation de la similarité cosinus entre deux services d'identifiants $idSW_i$ et $idSW_j$. L'évaluation de notre code pour le calcul de toutes les similarités pour toutes les paires de services du satset ASSAM [1], soit 13366 calcul de similarités (c-à-d $\frac{164^2-164}{2}$), sur une machine de processeur 2,7 GHz et d'une mémoire vive de 8Go donne un temps d'exécution de 5.315 secondes.

```
...
keySW_i = syntacticContentAllServices{idSW_i};
%"syntacticContentAllServices" est le tableau de toutes les listes "syntacticContentOneService"
% obtenus par notre analyse (voir section "Analyse des description des services Web")
keySW_j = syntacticContentAllServices{idSW_j};
INT = intersect(keySW_i,keySW_j);
card_INT = length(INT);
card_SW_i = length(keySW_i);
card_SW_j = length(keySW_j);
similariteI_J = card_INT / (sqrt(card_SW_i*card_SW_j));
...
```

Listing 4.4 – Extrait simplifié du code de calcul de la similarité

4.2.6 Découverte basée sur les préférences et compatibilités pour les interactions

Le problème des préférences et compatibilités peut se résumer comme suit : dans un contexte de réseau social de services Web, soit un service Web SW_iavec un ensemble de protocoles de communication qu'il préfère utiliser dans ses

interactions (notons les avec l'ensemble pp) et un ensemble de mots clés qu'il cherche (notons les avec l'ensemble pm), quels sont les services SW_j du réseau qui peuvent répondre, au mieux, à ces critères présentés par les ensembles pp et pm ? Nous allons donc chercher les services SW_j qui correspondent (ou qui matchent) le plus les attentes de SW_i. Afin d'atteindre cet objectif, nous allons nous baser sur notre modélisation et les différentes fonctions implémentées jusqu'à maintenant.

En fait, cette fonctionnalité de matching a été déjà présentée, mais sans discussion, dans les algorithmes et implémentations précédentes avec, par exemple, la ligne 17 dans algorithm 1 :

$$match(pp(idSW(i))\ ,\ pm(idSW(i)),\ description(idSW(j)))\ ;$$

Dans l'exemple de la ligne 17 de algorithm 1, nous testions si le service d'identifiant $idWS(j)$ correspondait bien aux attentes du service d'identifiant $idWS(j)$. Autrement dit, est ce que la description du service $idWS(j)$ (telle obtenue par notre méthode d'analyse des descriptions WSDL, voir sous-section 4.2.1) peut correspondre aux ensembles pp (préférences protocolaires) et pm (mots clés recherchés par le service $idWS(i)$). Notons qu'ici nous faisons bien la distinction entre la variable i (ou j) qui est l'indice du service dans la matrice d'adjacence (exemple, ligne et colonne 1) et $idWS(i)$ (ou $idWS(j)$) qui est l'identifiant du service dans le dataset (exemple, service #164).

Notre implémentation de la fonction *match* est représentée dans le Listing 4.5. Le cœur de notre approche s'appuie sur la similarité cosinus (présentée dans la sous-section 4.2.5) entre l'ensemble pm et la description du service $idSW(j)$. Cette description inclut la liste $syntacticContentAllServicesidSW(j)$ qui est la liste des mots clés du service $idSW(j)$. Si la similarité dépasse un certain seuil (fixé par le "chercheur" d'interactions dans le réseau social), le service est sélectionné. Reste donc l'ensemble pp à satisfaire et que nous considérons, dans notre implémentation, comme facteur discriminant qui s'ajoute à l'ensemble pm, c'est à dire :

l'ensemble *pp* pourra troncher en faveur (ou pas) d'un service trouvé. Pour ce faire, nous utilisons encore la description du service $idSW(j)$ qui inclut également les capacités du service en termes de support des protocoles, nombre de méthodes, nombre d'opérations élémentaires, etc. (voir Tableau 4.1). Il est évident que nous pouvons intervertir l'ordre de la discrimination, c'est à dire s'assurer d'abord d'une satisfaction protocolaire (*pp*) puis une satisfaction de mots clés (*pm*). Il est également possible d'affiner les résultats de la recherche en ajoutant un critère qui est les valeurs de la matrice somme (S) représentant l'historique des relations entre $idSW(i)$ et $idSW(j)$. Du coup, le nœud « lanceur » de recherche (ex. $idSW(i)$) pourrait favoriser les services $idSW(j)$ dont la valeur $S[idSW(i), idSW(j)]$ est forte (c-à-d, une fort lien qui perdure dans le temps).

```
function evaluerMatching(pmSW_i, ppSW_i, idSW_j)
    % INFO sur les variables :
    % -- syntacticContentAllServices{x}: contient la liste des mot cles du
    % service decrit par le fichier WSDL numero x.
    % Exemple : syntacticContentAllServices{x}(3) contient le 3eme mot cles du service numero x
    % -- descriptions{x,y} : contient le parametre numero y du service numero x
    % on a defini 16 parametres qui sont :
    % 1. nom du fichier WSDL du service
    % 2. # messages HTTP total
    % 3. # messages HTTP in
    % 4. # messages HTTP out
    % 5. # messages SOAP total
    % 6. # messages SOAP in
    % 7. # messages SOAP out
    % 8. # messages NA (ni HTTP ni SOAP)
    % 9. # methodes (PortType) HTTP
    % 10. # operations HTTP
    % 11. # methodes (PortType) SOAP
    % 12. # operations SOAP
    % 13. # methodes (PortType) NA (ni HTTP ni SOAP)
    % 14. # operations SOAP NA (ni HTTP ni SOAP)
    % 15. URL du point d'acces HTTP, sinon '' (vide)
    % 16. URL du point d'acces SOAP, sinon ''
   global syntacticContentAllServices;
   global descriptions;
   global similariteI_J_PM;
   global similariteI_J_PP;
   ppSW_j={};compteurPP_SW_j = 0; % representation des caracteristiques protocolaire de J
```

```
29      if descriptions{idSW_j, 9} > 0 %descriptions{idSW_j, 9} contient le nombre de methode HTTP
30          compteurPP_SW_j = compteurPP_SW_j +1;
31          ppSW_j{compteurPP_SW_j} = 'HTTP';
32      end
33      if descriptions{idSW_j, 11} > 0 %descriptions{idSW_j, 11} est le nombre de methodes SOAP
34          compteurPP_SW_j = compteurPP_SW_j +1;
35          ppSW_j{compteurPP_SW_j} = 'SOAP';
36      end
37      keySW_j = syntacticContentAllServices{idSW_j};
38      similariteI_J_PM = calculDeSimilariteGenerale(pmSW_i, keySW_j);
39      similariteI_J_PP = calculDeSimilariteGenerale(ppSW_i, ppSW_j);
40
41  end
```

Listing 4.5 – Extrait de la fonction de matching entre préférences (termes et protocoles) et description de service

Le tableau Tableau 4.3 montre le résultat de la recherche dans une portée de réseau social comportant 164 services Web avec les ensembles de termes de recherche *pm*={'mail', 'sms', 'phone'} et l'ensemble des protocoles préférés *pp*={'HTTP', 'SOAP'}. Le temps de réponse de notre implémentation est de **0.151 secondes** (sous la même configuration matérielle précédemment mentionnée). Le tableau montre un résultat de 7 services trouvés dont le plus pertinent détient un score de 0.333 pour les mots clés de recherche et 1 pour les protocoles recherchés. Ce service est le service d'identifiant #1 (de nom : *100_ GeoPhone*) du dataset experimenté. Les services #44 et #72 sont *3_ MailLocate* et *16_ SendanEmail*. Notez que ces derniers ne matchent pas notre recherche au sens protocolaire.

4.2.7 Découverte de « collaborations » entre services

Il est important de noter que c'est grâce à notre modélisation efficace (discutée auparavant) et l'implémentation des différentes briques de fonctionnalités détaillées dans les sections précédentes, que les aspects *découvertes* de tous les types des interactions sociales entre services Web (matching, collaboration, compétition, substitution et composition) deviennent

Résultat de découverte	Score de matching (termes)	Score de matching (protocoles)
service #1	0.333	1.000
service #44	0.204	0.000
service #72	0.204	0.000
service #86	0.149	0.707
service #55	0.140	1.000
service #96	0.132	1.000
service #3	0.118	0.000
Termes	*mail*, *sms*, *phone*	
Protocoles	HTTP, SOAP	

TABLE 4.3 – Résultat de recherche sur un réseau de 164 services Web

plus simples et plus efficace à effectuer. C'est l'ultime but recherché par notre contribution.

Par conséquent, la découverte de « collaborateurs » revient à trouver, dans un réseau social déjà formé, l'ensemble des services avec lesquels un service donné j a un lien d'interaction. Afin d'implémenter de telle identification, il revient à identifier l'ensemble des nœuds adjacents au nœud i dans la matrice d'adjacence SM (discutée auparavant). Partant du principe social *« les amis de mes amis sont mes amis »*, il est également intéressant de considérer d'autres nœuds en relation avec les nœuds adjacents du nœud i. En théorie de graphes, ce problème est équivalent à déterminer tous les nœuds appartenant à l'arbre dont la racine est i et avec une *profondeur* donnée. La pondeur est la distance maximale entre la racine et les nœuds lointains (appelés aussi « feuilles »). Nous implémentons donc une recherche généralisée de collaborateurs à une distance maximale d. Pour les expérimentation, nous optons pour la méthodologie suivante : d'abord nous formons un réseau social puis nous appliquons notre recherche de collaborateurs de pronfondeur deux (2) dans ce réseau en utilisant notre implémentation. C'est cette même méthodologie qu'on va suivre dans la découverte de type *substitution*, *compétition* et *composition*.

Il nous paraît important d'évaluer notre implémentation en terme de formation de réseau sociaux. La Figure 4.10 présente le temps de formation de

réseau en fonction de sa taille. Nous varions les taille de 10 à 160 services Web avec un pas de 10. Le temps de formation n'excède pas les 0.2 secondes dans le pire des cas (dans les mêmes conditions matérielles de test).

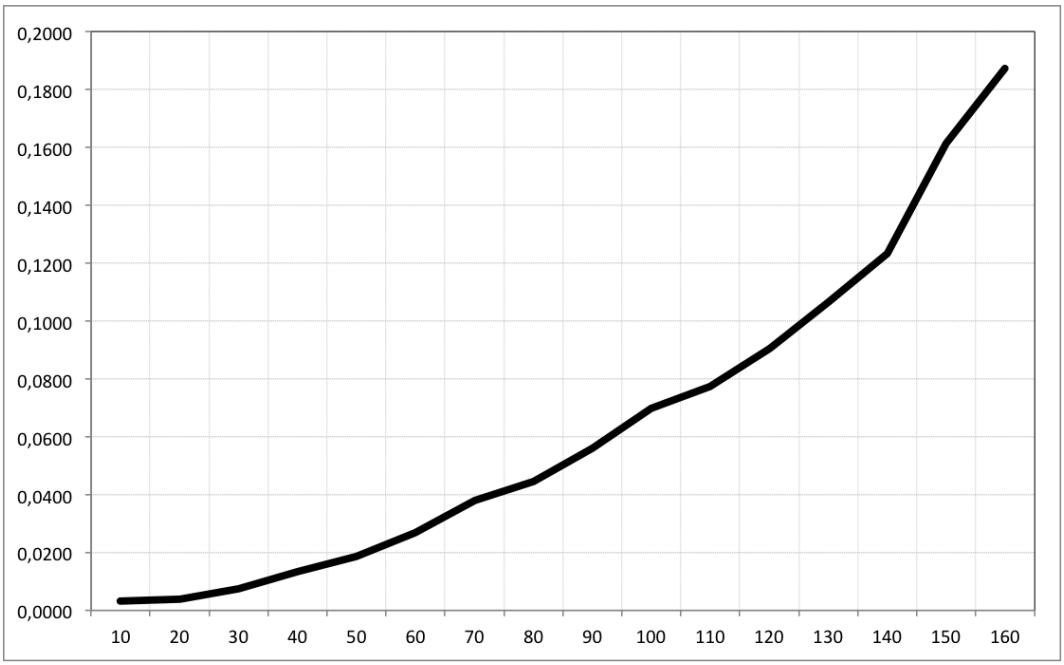

FIGURE 4.10 – Temps de formation des nœuds collaborateurs : nombre de collaborateurs (axe des X) en fonction du temps de formation du réseau social (en secondes)

Notre implémentation de la découverte de collaborateur est détaillée dans Listing 4.6.

```
function collab = chercherCollaborateurs(SM, root)
    %renvoi un tableau de collaborateur du noeud "root" (root etant l'identifiant du service)
    %
    global idSW; %vecteur des identifiants des services qui forment le reseau
    collab=[]; % variable de retour de cette fonction
    tailleSM = length(SM);
    indiceRoot = find(idSW == root); %extraire l'indice du service, car la fonction est appelee
                                     %par l'id du service
    voisins = chercherVoisins(SM, indiceRoot);
    collab = union(collab,voisins);%ajouter les voisins immediats aux collaborateurs

    %analyser les voisins sortants
    voisinsS = chercherVoisinsSortants(SM, indiceRoot);
    collab = union(collab,voisinsS);%ajouter les voisins sortants immediats aux collaborateurs
```

```
    tailleVoisins = length(voisinsS);
    for i=1:tailleVoisins
        x= chercherVoisinsSortants(SM, voisinsS(i));
        collab = union(collab,x);%ajouter les voisins sortants des voisins aux collaborateurs
    end
    %analyser les voisins entrants
    voisinsE = chercherVoisinsEntrants(SM, indiceRoot);
    collab = union(collab,voisinsE);%ajouter les voisins sortants immediats aux collaborateurs
    tailleVoisins = length(voisinsE);
    for i=1:tailleVoisins
        x= chercherVoisinsEntrants(SM, voisinsE(i));
        collab = union(collab,x);%ajouter les voisins sortants des voisins aux collaborateurs
    end
    %remplacer les indices par les identifiants des services Web
    tailleCollab = length(collab);
    for i=1:tailleCollab
        collab(i) = idSW(collab(i));
    end
    %exclure le noeud root lui meme des collaborateurs
    index= find(collab == root);
    if ~isempty(index)
        collab(index)=[];%supprimer le noeud lui meme de la liste de ses collaborateurs
    end
end
```

Listing 4.6 – Extrait de la fonction la recherche de collaborateurs

Notre méthode, de découverte de collaborateurs, appliquée pour le service Web #90 dans un réseau social formé de 15 services Web et avec une profondeur maximale de recherche de 2 donne le résultat présenté dans la Figure 4.11 en **0.008 secondes** sous les mêmes conditions matérielles précédentes. Notre implémentation de la découverte identifie les 13 services suivant : #2, #148, #1, #9, #13, #140, #149, #110, #51, #130, #107, #32 et #98. Notons que notre découverte prend en considération les deux sens d'interactions. En effet, un service « collaborateur » peut rendre un service mais aussi en demander un. Nous rappelons encore que ces services sont des services « réels » avec des descriptions WSDL complètes et qui proviennent du dataset ASSAM [1].

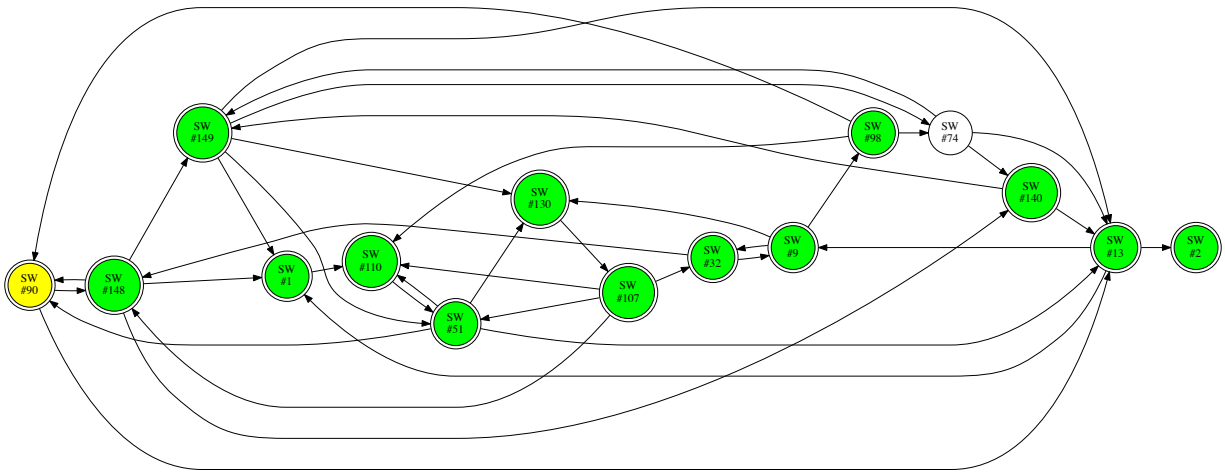

FIGURE 4.11 – Découverte de collaborateurs (pronfondeur 2) pour le service #90

4.2.8 Découverte des « substitutions » possibles

Comme nous l'avons établi dans la sous-section 4.1.2, la découverte de services Web pour les relations sociales de type «substitutions» d'un service SW_i revient à identifier les nœuds fortement similaires. Pour cela, nous utilisons la notion de similarité telle que nous l'avons défini et implémenté dans la sous-section 4.2.5. Comme nous pouvons le constater dans la Figure 4.12, en appliquant notre approche de découverte de substitution puis en sélectionnant seulement les 20 (parmi 164) premières fortes relations de substitution entre services Web, nous observons que plusieurs relations de substitution équivalentes existent. C'est le cas, par exemple pour les relations entre le service #66 et les services #120 et #74 (avec une métrique de 0.756) ou encore entre le service #59 et les services : #60, #61, #100 et #140 (avec une métrique parfaite de1). Comme nous l'avons mentionné précédemment, c'est dans ce genre de situations à choix multiples où l'historique (en utilisant la matrice somme S) peut servir d'outil discriminatoire dans le sens où on favorisera les paires avec des relations plus persistantes dans le temps. Enfin,

l'application de notre approche de similarité, sur le dataset ASSAM [1], nous a révélé qu'il existent 9 services qui n'ont pas du tout de possibles services Web de substitution (c-à-d la métrique calculée avec le reste du réseau est nulle). Ces services sont les services : #9, #13, #123, #132, #145, #148, #149, #154 et #163.

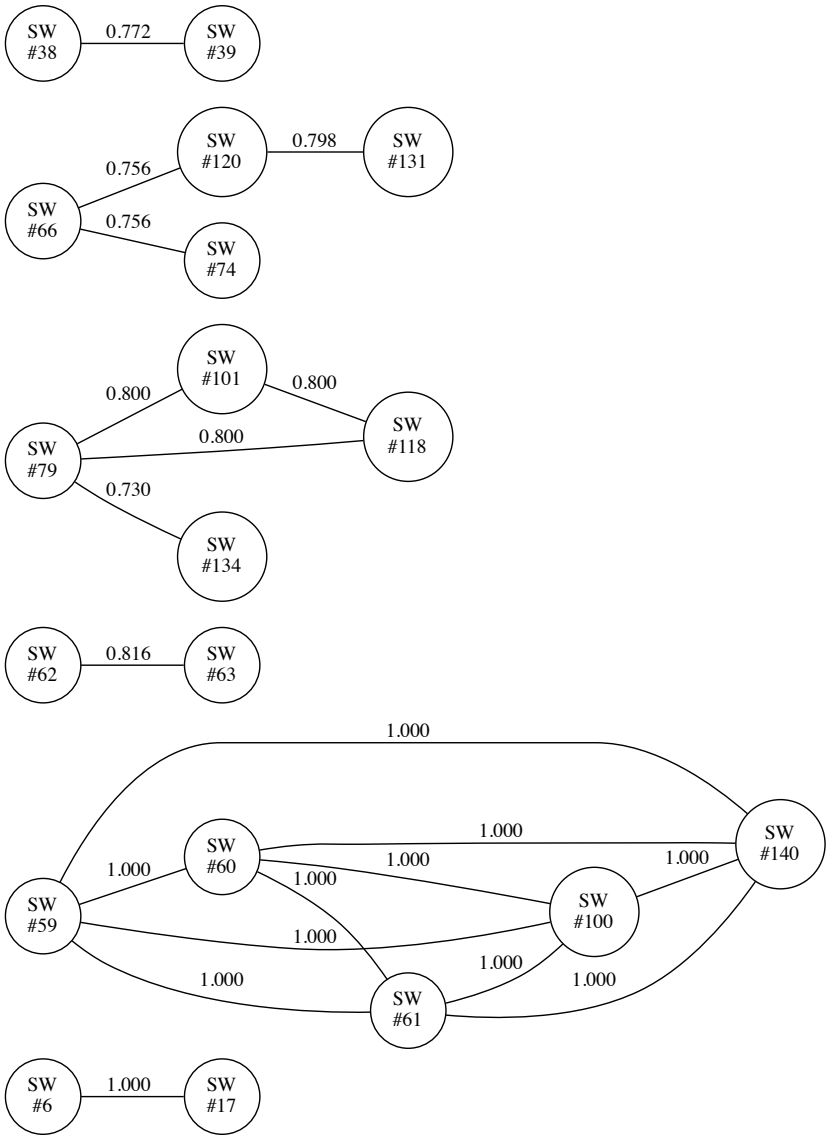

FIGURE 4.12 – Découverte des nœuds de substitution (les 20 premières plus fortes relations dans le dataset avec le coefficient *cosinus*)

4.2.9 Découverte de « compétitions » entre services

Comme expliqué à la fin de la sous-section 4.1.2, pour un nœud SW_i, identifier les services Web au sens relation sociale de type «compétition» revient tout simplement à identifier les nœuds « très » similaires mais qui ne sont pas des collaborateurs du nœud SW_i. Nous proposons donc de profiter des nos fonctions implémentées jusqu'à maintenant pour simplifier le développement de telle recherche. L'idée est d'identifier d'abord les nœuds «collaborateurs» puis les soustraire du réseau social, enfin de calculer le degré de similarité entre ces services et SW_i. Si ce degré dépasse un certain seuil, les services sont considérés comme «compétiteurs» de SW_i. L'implémentation revient donc à ce que nous présontons dans Listing 4.7. Exécuté sur le même exemple de la Figure 4.11 (c-à-d pour le service #90), le seul compétiteur identifié est le service #74 avec un degré de similarité de 0.2857. Ce qui veut dire, que si nous cherchions un compétiteur pour le service #90 avec un seuil supérieur strictement à 0.2857, la recherche se soldera par l'ensemble nul.

```
function compet = chercherCompetiteurs(SM, root, seuil)
    global idSW; %vecteur des identifiants des services qui forment le reseau
    compet = []; % le tableau des competiteurs
    x = setdiff(idSW, chercherCollaborateurs(SM, root)) % soustraire les noeuds collaborateurs
    %la fonction chercherCollaborateurs est detaillee plus haut
    x(find(x == root))=[];%rechercher la position de root et l'enlever pour l'exclure de la comparison
    for i=1:length(x)
        sim = calculDeSimilarite(root, x(i));%calcul de la similarite
        if (sim) > seuil % si le non collaborateur est tres similaire
            compet = [compet, x(i)];%ajouter le service aux competiteurs de root
        end
    end
end
```

Listing 4.7 – Fonction de découverte de compétiteurs

4.2.10 Découverte de possibilités de « compositions » de services

Pour implémenter la découverte des possibilités de composition des services nous réutilisons le résultat de la sous-section 4.2.7. En effet, comme mentionné dans sous-section 4.1.2, un service composite nécessite l'identification des chemins orientés possibles émanant du service SW_i vers le service SW_j avec une distance donné. Il revient donc à utiliser l'identification de collaborateurs (implémentée dans sous-section 4.2.7) mais seulement dans le sens sortant pour pouvoir composer un service pour SW_i. Par exemple, le chemin orienté SW_1-SW_2-SW_3 (s'il existe) permettrait de composer un nouveau service utilisable par SW_1 et qui implique les services SW_2 et SW_3.

Le code que nous avons développé est sous forme d'une fonction appelée *chercherCompositions* qui est exactement le meme code que la function *chercherCollaborateurs* (voir Listing 4.6), exempté des lignes 20 à 27 du Listing 4.6.

L'application de notre recherche de service Web candidats à la composition d'une distance maximale de 2 et appliqué sur le même exemple que la Figure 4.11 (c-à-d pour le service #90) donne le résultat présenté dans la Figure 4.13 (exécution en **0.005 secondes** sous les mêmes conditions matérielles précédentes). Notons dans l'exemple, qu'à la différence de la collaboration, on identifie seulement le sens sortant du service racine (c-à-d le #90) pour les chemins possibles de longueur maximale de 2.

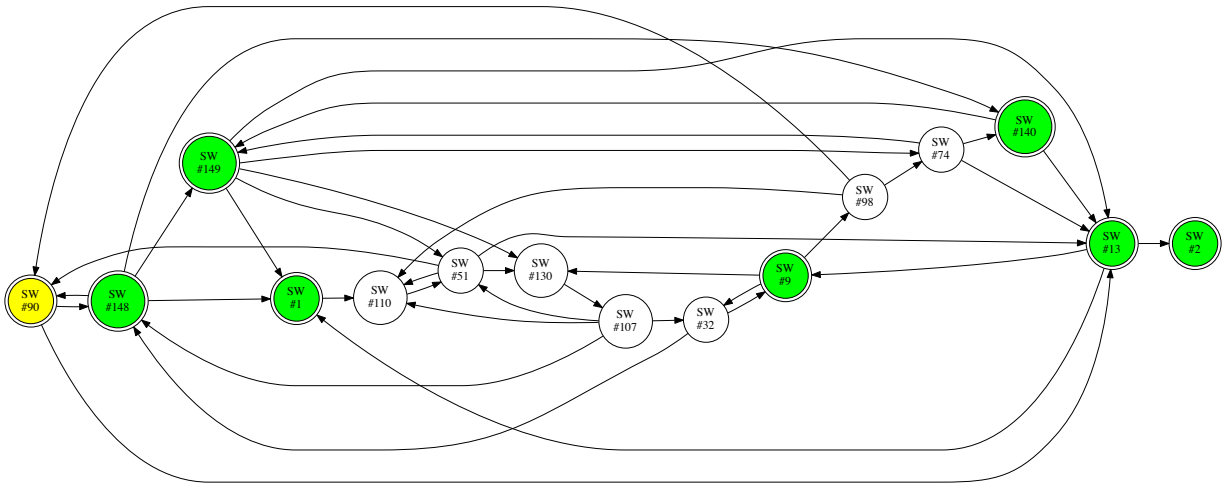

FIGURE 4.13 – Découverte de composition (distance maximale : 2) pour le service #90

4.2.11 Récapitulatif de notre implémentation

En plus de nos travaux de réflexion et de nos proposition originales d'ordre recherche scientifique, nous récapitulons dans les tableaux Tableau 4.4 et Tableau 4.5 les fichiers de code informatique que nous avons réalisé avec le langage Matlab en plus de la partie installation, configuration, publication de certains services Web et tests d'interrogation du framework jUDDI. Le nombre de lignes de code que nous avons réalisé sous Matlab s'élève à 979 lignes de code approximativement.

Fichier : 0. jUDDI	**Nombre de lignes de code** : N/A
Déscription : nous avons installé, configuré et expérimenté le framework jUDDI et lui ajouter certains services Web du dataset ASSAM, ensuite tester la recherche de service sous ce cadre de travail.	
Fichier : 1. WSDL_Parser.m	**Nombre de lignes de code** : 387
Déscription : parser un ensemble de fichiers WSDL et extraire un ensemble de paramètres caractérisant les services	
Fichier : 2. extractKeyWords.m	**Nombre de lignes de code** : 41
Déscription : extrait l'ensemble des mots du langage naturelle inclus même ceux dans les noms des messages et des méthodes des descriptions WSDL	
Fichier : 3. eliminateLightKeyWords.m	**Nombre de lignes de code** : 17
Déscription : élimine certains mots du langage qui ne sont pas significatifs (ex. certains déterminant)	
Fichier : 4. testWSDL_Parser.m	**Nombre de lignes de code** : 30
Déscription : tester le parsing des fichiers WSDL effectué par WSDL_Parser.m	
Fichier : 5. testExtractKeyWords.m	**Nombre de lignes de code** : 6
Déscription : tester le parsing de la fonction extractKeyWords.m avec des mots variées pour affiner son parsing	
Fichier : 6. formerUnReseauSocial.m	**Nombre de lignes de code** : 54
Déscription : former un réseau social avec un nombre aléatoire de nœuds et des interactions aléatoires compatibles	
Fichier : 7. AjouterUneInteraction.m	**Nombre de lignes de code** : 13
Déscription : ajouter une interaction entre deux services en utilisant les identifiants du dataset des deux services (et non pas leurs indices dans la matrice d'adjacence)	
Fichier : 8. SupprimerUneInteraction.m	**Nombre de lignes de code** : 12
Déscription : supprimer une interaction entre deux services en utilisant les identifiants du dataset des deux services (et non pas leurs indices dans la matrice d'adjacence)	
Fichier : 9. AjouterUnService.m	**Nombre de lignes de code** : 13
Déscription : ajouter un service en utilisant son identifiant du dataset (et non pas son indice dans la matrice d'adjacence)	
Fichier : 10. SupprimerUnService.m	**Nombre de lignes de code** : 16

TABLE 4.4 – Récapitulatif de notre implémentation (partie 1/2)

Fichier : 11. AjouterUnServiceEtInteractions.m	**Nombre de lignes de code** : 48
Déscription : ajouter un service et des interactions impliquant ce service en utilisant son identifiant du dataset (et non pas son indice dans la matrice d'adjacence)	
Fichier : 12. evaluerMatching.m	**Nombre de lignes de code** : 30
Déscription : évaluer la correspondance entre deux ensembles, l'ensemble des mots et l'ensemble des protocoles, et un service (donné par son identifiant dans le dataset)	
Fichier : 13. calculDeSimilarite.m	**Nombre de lignes de code** : 20
Déscription : calculer la similarité cosinus entre deux services donnés par leurs identifiants dans le dataset	
Fichier : 14. calculDeSimilariteGenerale.m	**Nombre de lignes de code** : 12
Déscription : calculer le degré se similarité cosinus entre deux ensemble quelconques	
Fichier : 15. chercherCollaborateurs.m	**Nombre de lignes de code** : 36
Déscription : chercher, dans un réseau social déjà formé (matrice d'adjacence prête), les tableau des nœuds collaborateurs (profondeur maximale 2) pour un nœud racine donné	
Fichier : 16. chercherCompetiteurs.m	**Nombre de lignes de code** : 15
Déscription : chercher, dans un réseau social déjà formé (matrice d'adjacence prête), les tableau des nœuds compétiteurs pour un nœud racine donné	
Fichier : 17. chercherCompositions.m	**Nombre de lignes de code** : 28
Déscription : chercher, dans un réseau social déjà formé (matrice d'adjacence prête), les tableau des nœuds candidats à la compoisiton (de distance maximale 2) pour un nœud racine donné	
Fichier : 18. chercherVoisinsEntrants.m	**Nombre de lignes de code** : 11
Déscription : chercher les identifiants des services adjacents, dans le sens entrée (arcs entrants), dans une matrice d'adjacence représentant le réseau social	
Fichier : 19. chercherVoisinsSortants.m	**Nombre de lignes de code** : 11
Déscription : chercher les identifiants des services adjacents, dans le sens de la sortie (arcs sortants), dans une matrice d'adjacence représentant le réseau social	
Fichier : 20. testScenariosReseauSocial.m	**Nombre de lignes de code** : 179
Déscription : effectuer un ensemble de tests sur toutes les fonctionnalités implémentées (parsing, formation, etc.). Ce code sert aussi à générer automatiquement les codes requis pour générer les tableaux et les graphiques (graphes et courbes) sous LaTeX pour le mémoire	

TABLE 4.5 – Récapitulatif de notre implémentation (partie 2/2)

Chapitre 5

Conclusion

Depuis leur apparition, l'utilisation des services Web sur Internet n'a pas céssé de connaître des évolutions et ces dernières n'ont pas concerné seulement les standards et les technologies qui gravitent autour cet écosystème mais aussi les outils. En effet, beaucoup d'outils ont également évolué et dans ce contexte se trouvent les manières dont ont doit gérer efficacement les services Web. A titre d'exemple, nous pouvons citer le sort proche de l'abandon des standards UDDI utilisés pour recenser les services Web et permettre aux utilisateurs, en particulier aux développeurs, de manipuler les services Web.

Depuis l'engouement continu des réseaux sociaux, les chercheurs ne cessent de s'intéresser à leurs propriétés et leurs évolution et pourquoi pas d'en profiter pour les appliquer dans d'autres domaines comme : l'application de ce paradigme dans une communauté de services Web. C'est le thème de notre travail auquel nous avons réussi à contribuer avec une nouvelle modélisation et de nombreuses briques d'implémentation qui permettent de valider les approches de découvertes et de manipulation des services Web.

Après avoir été confronté à l'abandon des UDDI, nous avons quand même voulu l'expérimenter (avec le framework jUDDI) pour se rendre compte par nous mêmes de leurs limitations en ce qui concerne la manipulation des services Web mais surtout des interactions qui peuvent y avoir entre services. Les quelques travaux originaux existant dans l'état de l'art, en ce qui concernent

les réseaux sociaux et les services Web, ont été examinés avec attention et nous avons pu identifier que l'union des principales interactions sociales entre services sont les *collaborations*, *compétitions*, *substitution*, et *composition*. Ces travaux existants ont beaucoup de valeurs mais détiennent aussi beaucoup de lacunes et resettent, de notre point de vue, incomplets car beaucoup de pistes de recherche scientifique et d'implémentation restaient à faire.

Nous avons donc voulu fournir un travail à la fois de recherche scientifique original mais aussi d'implémentation pour valider nos propositions. Ces dernières ont prouvé leurs excellentes performances et ont couvert toutes les interactions sociales possibles (citées précédemment). Elles fournissent les briques essentielles implémentées pour un vrai cadre de travail (framework) qui sert à valider et expérimenter nos approches mais aussi des éventuelles futures approches à proposer par la communauté des chercheurs dans le domaine des réseaux sociaux.

Les résultats obtenus dans ce travail nous encouragent à continuer à explorer de nombreuses pistes qui se sont ouverts à nous et qui nous permettrons de valoriser encore nos propositions. A très court terme, nous planifions d'organiser nos différentes briques logicielles de telle sorte à que nous puissions fournir à la communauté scientifique, avec la plateforme *GitHub* par exemple, un environnement de simulation complet pour manipuler d'une manière riche, flexible et *scalable* les réseaux sociaux des services Web.

Sur le fondement des problématiques scientifiques qui méritent encore une étude approfondie, nous comptons améliorer la prise en considération de l'**historique** d'un réseau social en cas de panne d'un service Web. Ceci revient à améliorer notre approche de sommation des matrices d'adjacence qui représentent le réseau social. Il serait également intéressant d'améliorer notre approche de calcul des **similarités** pour intégrer, à minima, la prise en compte de la fréquence inverse des termes avec le facteur TD-IDF. Une autre perspective et extension intéressante et facilement implémentable en se basant sur notre code réalisé, est de considérer plus de critères dans les requêtes de recherche en ce qui concerne le **matching entre services**. Par exemple, en

considérant l'historique (c-à-d la matrice somme S) et le nombre de messages ou d'opérations, etc. C'est d'autant plus facile que notre vecteur de description (issue du parsing des descriptions WSDL) extrait déjà 16 paramètres de description. Ces paramètres pourront servir à enrichir la syntaxe des requêtes de recherche dans le réseau social.

Il existe beaucoup d'autres perspectives d'**évaluation de notre code** de matching afin de fournir une évaluation encore plus rigoureuse de notre approche et de voir ses performances. Par exemple, en variant les requêtes de recherche (l'ensemble pm) en fonction du nombre de mot clés mais aussi en sélectionnant les mots clés en relation avec les termes utilisés dans la communauté des services (le corpus) par exemple en choisissant les mots les plus fréquents, les moins fréquents, etc. Dans ce contexte, une autre perspective à explorer et qui permettrait de gagner en scalabilité et en performance et l'utilisation d'une approche d'**indexation** des services dans le contexte d'un réseau social. Dans notre cas, l'indexation nous paraît une fonctionnalité à ajouter lors de chaque évolution du réseau social en particulier lorsqu'un service rejoint ou quitte le réseau.

Enfin, deux dernières perspectives nous paraissent intéressantes et qui sont liée à la recherche **sémantique** et la **qualité de service** (QoS). Pour la première perspective, nous envisageons l'exploration des ontologies tout en réduisant leur portée de la recherche, par exemple, en choisissant un thème comme la **santé numérique**. En ce qui concerne la QoS, le travail reviendrait à considérer certains paramètres de QoS pour distinguer les services dans les différentes interactions. De tels critères peuvent être implémentés en considérant le temps de réponse des points d'accès des services au sens WSDL (c-à-d les URL déjà extraits par notre code) et le ranking de leurs noms de domaine (informations qui peuvent se trouver sur le Web dans des sites spécialisés). Pour ce point particulier, nous avons déjà identifié la bibliothèque nommée *Beautifulsoup* qui facilite énormément le parsing des pages Web et qui permet, par conséquent, d'extraire les informations nécessaires de rankin en l'absence d'API standards fournis par les sites spécialisés.

Bibliographie

[1] A. Heß, E. Johnston, and N. Kushmerick. ASSAM : A Tool for Semi-Automatically Annotating Semantic Web Services. *Springer Verlag, Lecture Notes in Computer Science, ISWC 2004*, 2004.

[2] C. Rolland and N. Prakash. Bridging the gap between organizational needs and ERP functionality. *Requirements Engineering*, 5(3) :180–193, 2000.

[3] A. Celestini, G. Costantino, R. De Nicola, Z. Maamar, F. Martinelli, M. Petrocchi, and F. Tiezzi. Reputation-based composition of social web services. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 735–742. IEEE, 2014.

[4] W. Chen, I. Paik, and P. C. Hung. Constructing a global social service network for better quality of web service discovery. *IEEE transactions on services computing*, 8(2) :284–298, 2015.

[5] M. Constant. *Cours Master de Traitement Automatique des Langues, Université Paris-Est Marne-la-Vallée, accès mars 2018,*. http://igm.univ-mlv.fr/ens/Master/M2/2007-2008/TAL/cours/mstal-1-3-m2.pdf.

[6] A. Corbellini, D. Godoy, C. Mateos, A. Zunino, and I. Lizarralde. Mining social web service repositories for social relationships to aid service discovery. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 75–79. IEEE Press, 2017.

[7] M. Driss, N. Moha, Y. Jamoussi, J.-M. Jézéquel, and H. H. B. Ghézala. A requirement-centric approach to web service modeling, discovery, and selection. In *International Conference on Service-Oriented Computing*, pages 258–272. Springer, 2010.

[8] L. Duan and H. Tian. Collaborative web service discovery and recommendation based on social link. *Future Internet*, 9(4) :63, 2017.

[9] H. Fallatah, J. Bentahar, and E. K. Asl. Social network-based framework for web services discovery. In *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pages 159–166. IEEE, 2014.

[10] J. Ethier. Current research in social network theory. In *http ://www.scribd.com/doc/11171859/Current-Research-in-Social-Network-Theory, (accès en février 2018)*, 2006.

[11] A. Kalaï, C. A. Zayani, I. Amous, W. Abdelghani, and F. Sèdes. Social collaborative service recommendation approach based on user's trust and domain-specific expertise. *Future Generation Computer Systems*, 80 :355–367, 2018.

[12] A. Karray, R. Teyeb, and M. B. Jemaa. A heuristic approach for web-service discovery and selection. *arXiv preprint arXiv :1305.2684*, 2013.

[13] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with owls-mx. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922. ACM, 2006.

[14] Z. Maamar, N. Faci, L. Wives, Y. Badr, P. Santos, and J. P. M. de Oliveira. Using social networks for web services discovery. *IEEE internet computing*, 15(4) :48–54, 2011.

[15] Z. Maamar, H. Hacid, and M. N. Huhns. Why web services need social networks. *IEEE Internet Computing*, 15(2) :90–94, 2011.

[16] Z. Maamar, L. K. Wives, Y. Badr, and S. Elnaffar. Even web services can socialize : A new service-oriented social networking model. In *Intelligent Networking and Collaborative Systems, 2009. INCOS'09. International Conference on*, pages 24–30. IEEE, 2009.

[17] Z. Maamar, L. K. Wives, Y. Badr, S. Elnaffar, K. Boukadi, and N. Faci. Linkedws : A novel web services discovery model based on the metaphor of "social networks". *Simulation Modelling Practice and Theory*, 19(1) :121–132, 2011.

[18] A. Maaradji, H. Hacid, R. Skraba, A. Lateef, J. Daigremont, and N. Crespi. Social-based web services discovery and composition for step-by-step mashup completion. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 700–701. IEEE, 2011.

[19] S. K. Mistry, M. H. Kamal, and D. Mistry. Semantic discovery of web services through social learning. *Procedia Technology*, 3 :167–177, 2012.

[20] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing : State of the art and research challenges. *Computer*, 40(11), 2007.

[21] J. Sangers, F. Frasincar, F. Hogenboom, and V. Chepegin. Semantic web service discovery using natural language processing techniques. *Expert Systems with Applications*, 40(11) :4660–4671, 2013.

[22] N. Srinivasan, M. Paolucci, and K. Sycara. Semantic web service discovery in the owl-s ide. In *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 6, pages 109b–109b. IEEE, 2006.

[23] M. Stollberg and M. Kerrigan. Goal-based visualization and browsing for semantic web services. In *International Conference on Web Information Systems Engineering*, pages 236–247. Springer, 2007.

[24] W3C. *Web Services Activity*. https://www.w3.org/2002/ws/.

[25] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1 : Core Language, W3C Proposed Recommendation 23 May 2007.*

[26] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1 : Core Language, W3C Recommendation 26 June 2007.* https://www.w3.org/TR/wsdl20/.

[27] W3C. XML Protocol Working Group, *SOAP Version 1.2.* https://www.w3.org/TR/soap/.

[28] Y. Yu, J. Chen, S. Lin, and Y. Wang. A dynamic qos-aware logistics service composition algorithm based on social network. *IEEE transactions on emerging topics in computing*, 2(4) :399–410, 2014.

[29] B. Yuan, L. Liu, and N. Antonopoulos. Efficient service discovery in decentralized online social networks. *Future Generation Computer Systems*, 2017.