RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mira Abderrahmene de Béjaia

INFRASTRUCTURES POUR LA CRÉATION DE DÉPÔTS DE PROTOCOLES DE SERVICES WEB

PAR

Boubakir Mohamed

PRÉSENTÉ À LA FACULTÉ DES SCIENCES ET SCIENCES DE L'INGÉNIEUR
DÉPARTEMENT D'INFORMATIQUE, ÉCOLE DOCTORALE D'INFORMATIQUE
ReSyD
(Réseaux et Systèmes Distribués)
POUR l'OBTENTION DU GRADE DE MAGISTÈRE EN INFORMATIQUE
À
L'Université Mira Abderrahmene
Béjaia, 06000

Acceptée sur proposition du jury

Président :	M.Mendil Boubekeur, Maitre de conférence, université de Bejaia, Algérie
Rapporteur :	M.Mohand-Saïd Hacid, Professeur à l'université Claude Bernard, France
Examinateurs :	M.Melit Ali, Maitre de conférence, université de Jijel, Algérie
	Mme Tas Saadia, Maitre de conférence, université de Bejaia, Algérie

UNIVERSITÉ MIRA ABDERRAHMENE

École doctorale d'informatique, ReSyD.

Date: **avril 2007**

Nom :	**Boubakir Mohamed.**
Titre :	**Infrastructures pour la création de dépôts de protocoles de services web.**
Département :	**Informatique.**
Grade :	**Magistère en informatique.**

A ma mère

A mon père

A ma grand-mère

A toute ma famille

Table des Matières

Table des matières **iv**

Liste des Tableaux **vii**

Liste des Figures **viii**

Remerciements **x**

Résumé **xi**

Abstract **xii**

Introduction générale **1**

1 Les services web **4**
1.1 Introduction 4
1.2 Définition et objectifs des services web 5
1.3 Caractéristiques et avantages 6
1.4 Architecture des services web 7
1.4.1 L'architecture SOA 7
1.4.2 Modèle de fonctionnement 7
1.4.3 Pile protocolaire 8
1.5 Infrastructure basique des services web 9
1.5.1 WSDL 10
1.5.2 SOAP 13
1.5.3 UDDI 15
1.6 Conclusion 18

2 Protocoles de services web **19**
2.1 Introduction 19
2.2 Applications des protocoles de services web 20
2.3 Modèle de protocoles 21
2.3.1 Présentation 21
2.3.2 Définition Formelle 23
2.3.3 Représentation XML 23
2.4 Propriétés d'activation et contraintes temporelles 24
2.4.1 Propriétés d'activation 24

2.4.2 Contraintes temporelles . . . 24
2.5 principaux types d'analyse de protocoles . . . 25
2.5.1 Compatibilité (*compatibility*) . . . 25
2.5.1.1 Compatibilité totale (*Full compatibility*) . . . 26
2.5.1.2 Compatibilité partielle (*Partial compatibility*) . . . 26
2.5.2 Remplaçabilité (*Replaceability*) . . . 26
2.5.2.1 Equivalence . . . 27
2.5.2.2 Subsumption . . . 28
2.6 Présentation des dépôts de protocoles . . . 28
2.7 conclusion . . . 30

3 Publication et recherche des descriptions WSDL 31
3.1 Introduction . . . 31
3.2 Description WSDL . . . 31
3.2.1 Interface de service . . . 32
3.2.2 Implémentation de service . . . 32
3.3 Publication . . . 34
3.3.1 Publication des interfaces de service . . . 35
3.3.2 Publication des implémentations de service . . . 36
3.4 Recherche . . . 37
3.4.1 Recherche des descriptions d'interface de service WSDL . . . 37
3.4.2 Recherche des descriptions d'implémentation de service WSDL . . . 39
3.4.3 Conclusion . . . 41

4 Les bases de données XML 43
4.1 introduction . . . 43
4.2 XML . . . 44
4.3 Stockage de documents XML . . . 45
4.3.1 SGBD à support relationnel . . . 45
4.3.2 SGBD XML natif . . . 48
4.4 Le langage XQuey . . . 49
4.4.1 Présentation . . . 49
4.4.2 Expression XPath . . . 49
4.4.3 Les expressions FLWOR . . . 50
4.5 Conclusion . . . 51

5 Etude expérimentale et implémentation 52
5.1 Introduction . . . 52
5.2 Etude de persistance . . . 53
5.2.1 Critères de choix . . . 53
5.2.2 Bases de données testées . . . 53
5.2.2.1 MySQL . . . 53
5.2.2.2 Xindice . . . 54
5.2.2.3 Berkeley . . . 54
5.2.3 Presentation des tests . . . 56

5.2.3.1 Description des données . . . 56
5.2.3.2 Description des requêtes . . . 56
5.2.4 Résultats et Interprétation . . . 57
5.2.4.1 Test d'ajout . . . 57
5.2.4.2 Test de recherche . . . 58
5.2.5 Résumé des résultats de l'étude de persistance . . . 61
5.3 implémentation . . . 61
5.3.1 Stockage des documents XML . . . 62
5.3.2 Implémentation de la bibliothèque . . . 63
5.3.3 Exposition de dépôts . . . 66
5.3.3.1 Service web . . . 66
5.3.3.2 Application web . . . 66
5.4 Conclusion . . . 67

Conclusion générale et Perspectives **69**

A **71**
A.1 Exemples de DTD . . . 71
A.2 Exemple de protocoles de services . . . 71

Bibliographie **73**

Liste des tableaux

2.1 Exemple de contraintes temporelles 25

4.1 Exemple de produits XML natifs 50

5.1 Résultat du premier type de test de recherche 59
5.2 Résultat du deuxième type de test de recherche 60

Table des figures

1.1 Modèle de fonctionnement de l'architecture des services web 9
1.2 Pile protocolaire des service web 10
1.3 Schématique d'un échange SOAP 14
1.4 Format d'un message SOAP 15
1.5 Modèle structurel des données de l'annuaire UDDI 17

2.1 Exemple de modèle de protocole de service 22
2.2 Exemple de compatibilité totale 26
2.3 Exemple de compatibilité partielle 27
2.4 Exemple d'équivalence 27
2.5 Exemple de remplaçabilité de type *subsumption* 28
2.6 Exemple de remplaçabilité par rapport au client 29
2.7 Architecture d'un dépôt de protocole 29

3.1 Relation entre la partie interface et implémentation WSDL 32
3.2 Exemple de document interface de service WSDL 33
3.3 Exemple de document implémentation de service WSDL 34
3.4 Correspondances entre les éléments WSDL et les structures de données UDDI 35
3.5 Exemple de structure de *tModel* 36
3.6 Exemple de structure de *businessService* 39
3.7 Résumé du mapping WSDL/UDDI 41

4.1 Exemple d'un simple document XML 45
4.2 Exemple d'un arbre XML 46
4.3 Architecture d'un SGBD à support relationnel 46
4.4 Architecture d'un SGBD XML natif 49

5.1 Architecture de BDB XML 55
5.2 Arbre représentant la structure d'un protocol de service 57
5.3 Résultat du test d'ajout 58
5.4 Résultat du premier type de test de recherche 59
5.5 Résultat du deuxième type de test de recherche 60
5.6 Structure de stockage de Berkeley 64
5.7 Diagramme de classes de la bibliothèque de dépôt 65

5.8 Interactions entre le client et le dépôt 67

Remerciements

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

Je voudrais tout d'abord exprimer mes remerciements et ma gratitude au Professeur Mohand-Said Hacid pour m'avoir encadré durant ce projet, son aide et sa confiance m'ont grandement aidé à mener à bien mon travail.

Je remercie les membres de jury qui ont accepté de juger ce travail. J'adresse mes très sincères remerciements à Mr Melit Ali, Mr Mendil Boubekeur et à Mme Tas Saadia de me faire l'honneur de s'intéresser à ce travail et d'avoir accepté de faire partie de jury.

Je remercie également Mr TARI Kamel, le chef de département d'informatique de l'université de Bejaia pour son aide, ses conseils et sa disponibilité. Un merci bien distingué à mes enseignants de l'école doctorale ReSyD. J'aimerai remercier également tous les étudiants de l'école doctorale pour l'environnement de travail très agréable durant ces deux dernières années.

Enfin, et surtout, je remercie vivement ma famille qui m'a beaucoup soutenu et encouragé, en particulier mes parents, mes soeurs et mes frères qui ont été toujours derrière moi pour m'encourager et me soutenir. Merci encore.

Résumé

LES protocoles de services (*business protocols*) représentent une partie importante de la description des services web, ils servent à décrire le comportement externe de ces derniers. L'utilisation des protocoles de services permet de simplifier considérablement le cycle de vie entier des services web. Ce mémoire porte sur la création de dépôts permettant d'effectuer des études sur les protocoles de services web, en s'intéressant en particulier à la manière de faire le lien entre ces dépôts et les annuaires UDDI, ainsi qu'au mode de persistance. Ainsi, nous avons décrit en premier temps, un processus permettant de publier la description WSDL de services web dans un annuaire UDDI et de la retrouver au moment de la recherche. En second temps, nous nous sommes intéressé au problème de stockage de XML. A ce propos, les solutions existantes peuvent être classées en deux catégories : La première consiste à utiliser les bases de données relationnelles, tandis que la deuxième consiste à créer entièrement des bases de données XML natives. Pour justifier le choix du mode de persistance, nous avons procédé à une étude comparative d'un ensemble de base de données. Le travail réalisé a consisté aussi à l'implémentation d'un ensemble de classes Java constituant la bibliothèque de dépôt. Cette bibliothèque sert comme une interface entre la base de données et les utilisateurs de dépôt.

Mots clés : services web, architectures orientées services (SOA), protocoles de services web, XML, bases de données relationnelles, base de données XML natives.

Abstract

BUSINESS protocols represent an important part of web service descriptions ; they are used to describe external behavior of web services. The use of business protocols simplifies considerably the whole service life-cycle. This thesis deals with the creation of deposits making it possible to carry out studies on business protocols, while being interested in particular in the manner of establishing the link between these deposits and UDDI registries, and in the mode of persistence. Thus, we firstly described a process allowing to publish WSDL service description in a UDDI registry and to find it at the time of research. Secondly, we were interested in the XML storage problem. The existing solutions to this problem can be classified into two categories : The first consists in using relational databases, while the second consists in entirely developing native XML databases. In order to justify the choice of persistence mode, we made a comparative study of a set of database products. The work carried out also included an implementation of the set of Java classes which constitute the library of the deposit. This library is used as an interface between database and the users of the deposit.

Keywords : web services, service-oriented architecture (SOA), business protocol, XML, relational databases, native XML databases.

Introduction générale

ACTUELLEMENT, les entreprises se retrouvent avec des systèmes de plus en plus gros et complexes. Ces systèmes sont souvent incompatibles entre eux, ils ont été construits progressivement en s'appuyant sur des technologies hétérogènes, développées pour des besoins spécifiques par des équipes différentes. Lorsqu'il survient le besoin de faire interagir ces systèmes hétérogènes, l'une des solutions possibles consiste à modifier le code des applications, voir le réécrire complètement ce qui est très coûteux, très complexe et très lent. Il faut donc impérativement s'accommoder de cette hétérogénéité de façon à permettre l'interopérabilité des différentes applications sans toucher au coeur du dispositif métier de chacune d'entre elles.

Dans le but de permettre l'intégration des systèmes hétérogènes, plusieurs solutions ont été proposées (eg., CORBA, DCOM). Cependant, ces solutions sont souvent complexes, ou pas flexibles car elles sont fortement attachées à des plates-formes particulières. Les développeurs doivent créer des passerelles pour convertir les protocoles et les formats de données pour permettre aux Plates-formes d'interopérer [32].

Dans ce cadre, les services web, et plus en général les architectures orientées-services (SOA) sont entrain d'émerger comme technologies et architectures de choix pour implémenter les systèmes distribués et effectuer l'intégration des applications à l'intérieur et à travers les frontières des entreprises [5]. Le principe de base de l'architecture SOA consiste à exposer les fonctions d'un système comme des services décrits à l'aide de langages standards et communiquent à travers des protocoles standards. Les services web sont un ensemble de technologies standards permettant d'implémenter une architecture SOA. Les services web apportent par rapport aux solutions précédentes les avantages suivants :

- Ils sont faiblement couplés et offrent un bon support pour les interactions décentralisées.
- Ils sont basés sur des standards ouverts à différents niveaux, tels que le format des massages (XML), le langage de définition de l'interface (WSDL), et le mécanisme de transport (SOAP).

Les services web fournissent des abstractions pour simplifier l'intégration au niveau

des couches basses. Les trois protocoles : SOAP, WSDL et UDDI qui constituent l'infrastructure de base des services web permettent de simplifier la description, la découverte et la communication entre les services. Même si cela a résolu un certain nombre de problèmes, les services web n'ont toujours pas contribué à la simplification de l'intégration à des niveaux d'abstraction plus élevés (par exemple les types de données et de messages ou bien encore les protocoles métiers d'intégration). En effet, Les application réelles nécessitent d'invoquer un ensemble de services dans un ordre précis et selon une logique bien définie. Or SOAP, WSDL et UDDI ne s'intéressent pas à ce problème et se situent plutôt au niveau transport et données. A ce propos, plusieurs technologies ont été proposées et plusieurs travaux de recherche ont été publiés.

Notre travail fait partie d'un projet de recherche international nommé *ServiceMosaic*[1]. L'objectif de ce projet est de fournir un ensemble d'outils de haut niveau pour faciliter l'intégration des services web. Une part des travaux de recherche de ce projet consiste à s'intéresser au comportement externe des services. Pour cela, on a développé un modèle de protocole de services web. Les protocoles de service (*business protocols*)[2] sont des protocoles de communication de haut niveau permettant de spécifier les suites de messages qui peuvent être émis et reçus par un service. Les protocoles de service sont inclus dans la description des services web et servent principalement à aider les développeurs à écrire les clients qui peuvent interagir correctement (en terme d'échange de messages) avec un service particulier. Ils ont aussi d'autres utilisations permettant de faciliter le cycle de vie entier des services web. Cela nécessite la gestion et l'analyse de ces protocoles, on s'intéresse en particulier à la notion de compatibilité et de remplaçabilité. Ces notions correspondent à la capacité de deux services à interagir et à la capacité d'un service à remplacer un autre.

On souhaite fournir des dépôts permettant d'effectuer des études sur les protocoles de services. Dans notre travail nous nous intéressons à la création de ces dépôts. Les dépôts auront potentiellement un très grand nombre de protocoles et pourront être soumis à une forte charge en lecture et en écriture et doivent offrir d'excellents paramètres de performances. En premier temps, nous allons nous intéresser à la façon de faire le lien entre les annuaires de services UDDI et les dépôts de protocoles. La persistance des dépôts est d'une importance majeure et peut avoir un impact significatif sur les performances. Ainsi, nous allons en second temps déterminer le mode de persistance le plus approprié. Les protocoles de services sont décrits à l'aide des documents XML. Ainsi, nous allons effectuer une étude sur le stockage des documents XML afin de choisir une base de données pour assurer le stockage des protocoles au niveau des dépôts.

[1] http ://servicemosaic.isima.fr/

[2] Dans ce mémoire nous allons utiliser le terme protocole de service pour désigner un *business protocol*.

Structure du mémoire

Le présent mémoire est composé de cinq chapitres. Dans le premier chapitre, nous présentons les services web en mettant l'accent sur leurs caractéristiques qui ont permis à cette nouvelle technologie de devenir la technologie de choix pour le développement et l'intégration des applications. Nous présentons aussi les trois standards WSDL, SOAP et UDDI qui constituent l'infrastructure de base des services web. Dans le deuxième chapitre, nous introduisons les protocoles de services web en mettant l'accent sur leurs utilités et en présentant les principaux types d'analyse qu'on peut effectuer sur ces protocoles. Dans ce même chapitre, nous introduisons une présentation générale de nos dépôts. Cela permet de mieux comprendre leur fonctionnement. Dans le troisième chapitre, nous décrivons un processus permettant de publier la description WSDL d'un service web dans un annuaire UDDI et de la retrouver au moment de la recherche. Le quatrième chapitre est consacré au problème de stockage des documents XML. Cela présente un cadre théorique pour notre étude de persistance qui fait l'objet du dernier chapitre. Ce dernier chapitre comporte aussi une partie implémentation.

Chapitre 1

Les services web

1.1 Introduction

L'une des tendances historiques qui a conduit à l'apparition des services web est l'utilisation de l'architecture par composants comme approche d'intégration des applications [37]. Les composants sont des entités logicielles indépendantes fondées sur une interface et une sémantique bien définie. Ils interagissent via une infrastructure qui permet de gérer la communication entre des composants au sein d'un même système ou à travers un réseau via une décomposition de la logique applicative en composants distribués [13] [41]. L'architecture de composants distribués a engendré un développement rapide et évolutif d'applications distribuées et complexes. Parmi ces architectures, on trouve CORBA (Common Object Request Broker Architecture) de l'OMG et DCOM (Distributed Commun Object Model) de Microsoft.

Cependant, la mise en oeuvre de ces architectures soulève des difficultés, surtout dans le cadre d'une infrastructure ouverte telle que Internet. En effet, ces architectures, proposent chacune sa propre infrastructure. Ce qui impose un couplage fort entre les services offerts par les composants et leurs clients. Ainsi, on ne peut assembler que des objets CORBA (ou DCOM) entre eux. Le résultat est que les systèmes construits à base de ces architectures sont monolithiques. L'intégration de ces systèmes nécessite la conception des ponts logiciels ad-hoc coûteux et à fort couplage. L'entretient de telles solutions nécessite une forte cohésion entre les différents acteurs impliqués (concepteurs, intégrateurs, etc.).

En Parallèl, après l'avènement du B2C (*Business To Consumer*), où les entreprises mettent en ligne leurs services pour leurs consommateurs à travers des applications web, celles-ci souhaitent augmenter leur productivité à l'aide du paradigme B2B (*Business To Business*). Le B2B repose sur l'échange de produits, d'informations et de services entre

entreprises. Ceci implique l'utilisation de services et la collaboration avec des systèmes proposés par d'autres concepteurs et par conséquent une maîtrise de l'hétérogénéité.

L'interopérabilité est ainsi devenue une nécessité pour l'entreprise. Les services web ont ouvert la voie à l'intégration de systèmes d'information hétérogènes à faible couplage et à faible coût. La solution des services web repose sur des protocoles web existants et se base sur des standards ouverts (XML). Alors que les autres architectures reposent chacune sur sa propre infrastructure.

1.2 Définition et objectifs des services web

Les services web sont une technologie récente et par conséquent, il n'existe pas encore une définition précise de la notion de service web. Le groupe de W3C (*World Wide Web Consortium*) [1] qui travaille sur les services web a utilisé dans un document appelé *web Services Architecture* la définition de service web suivante :

Un service web est un système logiciel destiné à supporter l'interaction ordinateur-ordinateur sur le réseau. Il a une interface décrite en un format traitable par l'ordinateur (spécifiquement WSDL). Autres systèmes réagissent réciproquement avec le service web d'une façon prescrite par sa description en utilisant des messages SOAP, typiquement transmis sur le protocole HTTP et une sérialisation XML, en conjonction avec d'autres standards relatifs au Web [24].

Nous allons dans ce mémoire adopter la définition suivante : La notion de service web désigne essentiellement une application (un programme) mise à disposition sur Internet par un fournisseur de service, et accessible par les clients à travers des protocoles Internet standards [21] [14].

Cette nouvelle technologie, initiée par IBM et Microsoft, puis en partie normalisée sous l'égide du W3C, vise à permettre aux applications d'échanger facilement les données à travers des réseaux informatiques (généralement le Web), et ceci indépendamment des plates-formes et des langages sur lesquels elles reposent. Les modules applicatifs vont pouvoir invoquer d'autres modules en ignorant leurs structures internes, uniquement via leurs interfaces [22]. Ils s'inscrivent ainsi dans la continuité des autres initiatives telles que CORBA en apportant toutefois une réponse plus simple, en s'appuyant sur des technologies et standards reconnus et maintenant acceptés de tous.

[1] Consortium international dont le but est de promouvoir l'évolutivité du Web et de garantir son interopérabilité. Le site web du W3C est : http ://www.w3.org

1.3 Caractéristiques et avantages

Les services web constituent un cadre robuste pour assurer l'interopérabilité entre des applications hétérogènes, accessibles en ligne, car ils n'imposent pas de restriction sur les caractéristiques techniques de l'application qui implémente le service. Les services web ont des objectifs similaires aux autres solutions. Cependant, leurs mis en oeuvre est un peu différente et ils présentent des avantages plus intéressants.

Les services web sont basés sur des standards ouverts

Le fait que le Web est constitué de plates-formes totalement hétérogènes où les intérêts des différents acteurs du marché s'entremêlent ne l'a pas empêché de se développer et d'être universel, Ce succès est dû essentiellement à l'utilisation d'un ensemble de standards ouverts, dont le plus connu est le protocole HTTP. Les services web suivent la même approche que le Web en se basant sur des standards ouverts. Ceci permet de réduire le coût d'intégration des applications qui est essentiellement dû au fait que les modules interactifs ont des interfaces différentes, supportent différents protocoles de communication et différents formats de données et modèles d'interaction [5].

Les services web sont faiblement couplés

Récemment, la comparaison entre les approches d'intégrations à couplage faible et à couplage fort a attiré beaucoup d'attention. D'un point de vue technologique, ceci est principalement guidé par la capacité des services web à découvrir et invoquer dynamiquement d'autres services. D'un point de vue métier, ceci est guidé par le besoin croissant de l'entreprise à augmenter la flexibilité en fonction des changements de ses processus métiers et des manières avec lesquelles elle interagit avec ses partenaires [18].

Le couplage fort est une source de plusieurs problèmes de maintenance. Dans les environnements distribués traditionnels, les composants logiciels fonctionnent de manière fortement couplée. C'est-à-dire, chaque composant est connecté avec les autres à travers une combinaison d'interfaces et de protocoles propriétaires. Ce qui rend les modifications dans le système difficiles, et coûteuses.

Par contre, les services web sont faiblement couplés. Par couplage faible, on désigne minimiser le nombre de choses que le code de l'application cliente et le code de l'application fournisseur connaît l'un sur l'autre [27]. L'interface du service web fournit une couche d'abstraction entre le client et le serveur. Ainsi, un changement dans l'un ne force pas nécessairement un changement dans l'autre. Le couplage faible minimise l'impact des changements aux applications. Cette abstraction de l'interface rend aussi facile la réutilisation du service dans une autre application.

En conclusion, le couplage faible réduit le coût de maintenance, augmente la réutilisabilité et facilite le processus d'intégration des différents systèmes.

Les services web sont basés sur l'existant

Les services web ne sont pas un changement fondamental remplaçant les infrastructures existantes [34]. Au contraire, ils présentent une bonne manière pour permettre l'interopérabilité des systèmes existants et leur évolution à côté de nouvelles applications. Les entreprises deviennent plus flexibles face aux évolutions du marché et peuvent rapidement s'adapter aux nouveaux besoins tout en capitalisant ses systèmes existants. C'est pour cela que les architectures orientées services s'inscrivent dans un axe de valorisation de l'existant par l'augmentation de sa capacité de communication avec un environnement hétérogène.

1.4 Architecture des services web

1.4.1 L'architecture SOA

Les services web sont basés sur une architecture SOA (Service-Oriented Architecture) dans laquelle les systèmes logiciels sont distribués sous forme d'un ensemble de services faiblement couplés et dynamiquement accessibles. Un service est une fonction bien définie, autonome et indépendante du contexte ou de l'état d'un autre service [6].

L'un des aspect les plus importants de SOA est la séparation entre l'implémentation du service et son interface. En d'autres termes, elle sépare entre quoi et comment [32]. Un service expose sa fonctionnalité à travers une interface qui cache son fonctionnement interne. Une application cliente n'a pas besoin de comprendre la façon dont le service exécute son travail, elle a besoin uniquement de comprendre comment utiliser l'interface. La vraie puissance de SOA et des services web devient évidente quand plusieurs constituants sont ajoutés, enlevés, remplacés, ou améliorés sans affecter défavorablement le système entier. Ceci est difficilement possible lorsque chaque partie de l'architecture se fonde sur une connaissance approfondie du fonctionnement interne des autres parties.

En conclusion, l'architecture SOA sert de base pour des applications interopérables, flexibles et fortement réutilisables.

1.4.2 Modèle de fonctionnement

Une architecture orientée service fait intervenir trois acteurs [28] [31] :

1. **Fournisseur de service (*provider*) :**

   Le fournisseur correspond au propriétaire du service, sa tâche consiste à implémenter les fonctions du service web, décrire l'interface de ces fonctions en utilisant une manière standard puis publier l'interface dans un annuaire de service pour permettre aux consommateurs de découvrir les services. D'un point de vue technique, le fournisseur est constitué par la plate-forme d'accueil du service.

2. **Le client ou consommateur (*requester*) :**

   Le client correspond au demandeur du service. D'un point de vue technique, il est constitué par l'application qui va rechercher et invoquer le service. Le client peut être lui-même un service web.

3. **L'annuaire (*Service registry*) :**

   L'annuaire du service peut être considéré comme une bibliothèque de services web. Il permet de publier des nouveaux services et de découvrir des services existants. Ainsi, il permet au consommateur de trouver les informations qui lui permettent d'invoquer le service web.

La dynamique de l'architecture est en général la suivante :
- Le fournisseur publie le service dans un annuaire (*publish*).
- Le client contacte l'annuaire pour chercher les services dont il a besoin (*find*).
- L'annuaire fournit au client l'interface de ce service et la référence du fournisseur.
- Le client contacte le fournisseur puis invoque le service web (*bind* et *invoke*).
- Le service web effectue l'opération demandée puis renvoie le résultat au client.

Chaque service web a un consommateur et un fournisseur, la présence de l'annuaire n'est pas toujours indispensable pour que l'échange ait lieu [33]. La figure (Fig. 1.1) synthétise les aspects statiques et dynamiques du modèle des services web. La communication entre les trois acteurs se fait en général par échange de messages SOAP.

1.4.3 Pile protocolaire

La pile protocolaire des services web est constituée de plusieurs couches, chaque couche s'appuyant sur un standard particulier. Les services web représentent un domaine de recherche jeune et par conséquent, la pile protocolaire ne trouve pas encore de consensus quand à sa définition standardisée. La figure (Fig. 1.2) propose une vue globale simplifiée de la pile des services web.

Dans la couche la plus basse, on trouve les protocoles de transport qui permettent d'assurer l'échange de messages entre les services. On utilise en général HTTP mais on peut aussi utiliser d'autres protocoles tels que FTP, SMTP, etc.

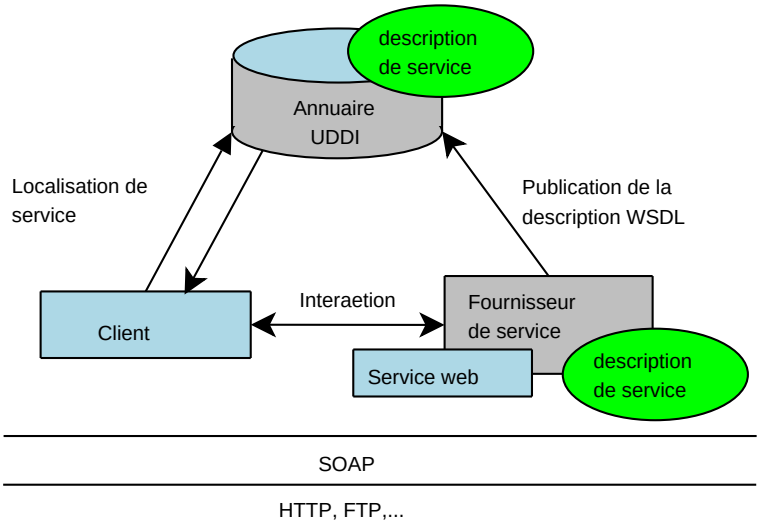

FIG. 1.1 – Modèle de fonctionnement de l'architecture des services web

Au-dessus de la couche de transport, on trouve les trois couches formant l'infrastructure de base des services web qui permet de répondre aux problèmes d'intégration technique des applications. Ces couches s'appuient sur les standards émergents SOAP, WSDL et UDDI qui font l'objet de la section suivante.

L'infrastructure de base des services web seule ne répond pas à tout les problèmes d'intégration. Elle est complétée par d'autres couches : la couche *Business processes* et les couches dites transversales. La couche *Business processes* permet d'utiliser les services web dans le domaine du e-business d'une manière effective. Elle représente un *business process* comme un ensemble de service web. les couches transversales contiennent un ensemble de standards qui rendent viable l'utilisation effective des services web dans le monde industriel (eg., sécurité, administration, transactions et qualité de services (QoS)) [31] [23].

1.5 Infrastructure basique des services web

Aujourd'hui, l'infrastructure de base des services web s'est concrétisée autour de trois spécifications considérées comme des standards du W3C et généralement acceptées par l'industrie informatique. Ces standards définissent la manière avec laquelle un service web peut être décrit et découvert et la manière avec laquelle il peut communiquer avec d'autres services [34] :

– **WSDL** (*Web Services Description Langage*) est responsable de la description des services.

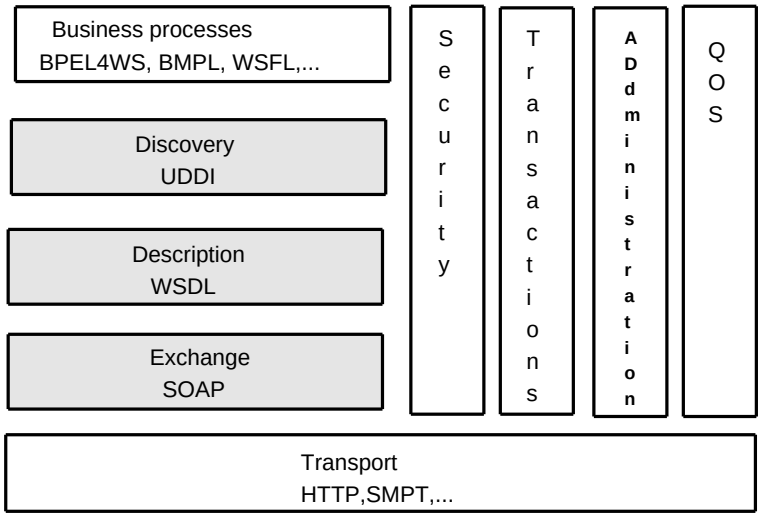

FIG. 1.2 – Pile protocolaire des service web

- **SOAP** (*Simple Object Access Protocol*) sert de moyen de communication entre le consommateur, le fournisseur et l'annuaire de service.
- **UDDI** (*Universal Description, Discovery and Integration*) fournit des mécanismes pour la publication et la découverte des services web.

1.5.1 WSDL

Le standard WSDL est un langage basé sur XML permettant de décrire les services web, spécifier leurs localisations et les opérations qu'ils exposent [22]. WSDL représente les services web sous forme de points de communications ou de ports échangeant des messages [43]. Ces ports sont mis bout à bout et reliés par un protocole de transport.

WSDL permet de séparer clairement entre la définition abstraite et les mécanismes de liaison (protocoles de transport et format de données). Cela permet de réutiliser les définitions abstraites : *messages* et *portTypes*. Les formats de données et les protocoles de transport concrets d'un *portType* constituent un *binding* réutilisable. SOAP est généralement utilisé comme protocole d'accès, mais on peut aussi utiliser HTTP, ou tout autre protocole. Un port est défini en associant une adresse réseau à un *binding*. Un ensemble de ports définit un service [22][43].

De façon plus précise, Un document WSDL utilise les éléments suivants pour décrire un service web [22] :

1. **Types :** les types de données que l'on peut employer dans les messages, en référençant en général les types standards des schémas XML.

2. **Message :** les messages sont des descriptions abstraites des données échangées.
3. **Operation :** les opérations sont des actions élémentaires que l'on peut effectuer auprès du service.
4. **PortType :** définit un ensemble d'opérations abstraites.
5. **Binding :** spécifie un protocole et un format de données concret pour un *portType*.
6. **Port :** les ports sont des points de communication constitués d'un *binding* et d'une adresse réseau.
7. **Service :** les services sont des collections de ports.

L'exemple suivant donne la définition WSDL d'un service web simple, ce service contient une seule opération nommée *GetLastTradePrice* déployée en utilisant SOAP 1.1 sur HTTP.

Le service sera défini comme suit :

```
<?xml version="1.0"?>
<definitions name="StockQuote"
  targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd1="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="GetLastTradePriceInput">
    <!--Déclaration des messages-->
  </message>

  <portType name="StockQuotePortType">
    <!--Déclaration des opérations-->
  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <!--Déclaration de la liaison-->
  </binding>

  <service name="StockQuoteService">
    <!--Déclarations des ports-->
  </service>

</definitions>
```

Chaque service est défini à l'aide d'un élément *service*. A l'intérieur de cet élément, on définit les différents ports par lesquels le service est accessible. Un port est caractérisé par un nom et une liaison (*binding*). Une adresse HTTP de service lui est associée :

```
<service name="StockQuoteService">
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
  </port>
</service>
```

Pour définir la structure d'un message, on utilise l'élément *message*. Un message contient un ensemble de parties, chacune étant définie par un élément *part* qui correspond à un paramètre d'appel ou de retour. Chaque paramètre doit avoir le même nom et le même type que le paramètre de la méthode correspondante :

```
<message name="GetLastTradePriceInput">
  <part name="body" type ="xsd1: string"/>
</message>
<message name="GetLastTradePriceOutput">
  <part name="body" type = "xsd1: float"/>
</message>
```

Les opérations sont définies à l'aide de l'élément *operation* en précisant les messages d'entrée et de sortie de chaque opération. L'élément *porType* permet de regrouper l'ensemble des opérations exposées par un service sur un port :

```
<portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
  </operation>
</portType>
```

Pour spécifier un protocole et un format de données concret, on utilise l'élément *binding* dont le nom doit être celui référencé par le port. L'élément *soap :binding* permet de préciser le style de la requête (RPC ou document) et le protocole de transport utilisé (HTTP, SMTP, etc.). Pour chaque opération du port, il faut un élément *operation* et il faut préciser l'action SOAP en utilisant l'attribut *soapAction* de l'élément *soap :operation*. De plus, il faut spécifier l'encodage des messages d'appel et de retour de l'opération en utilisant les éléments *input* et *output* :

```
<binding name="StockQuoteBinding" type="tns:StockQuotePortType">
  <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
```

```
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

Grâce à des capacités offertes par WSDL, les services web sont connus comme des éléments logiciels auto descriptifs [38]. C'est à dire capables de fournir des informations permettant de comprendre comment les manipuler. Ainsi, en se basant sur la description WSDL, les applications sont capables de générer des souches de logiciels réalisant les échanges entre les services web. Ces deux souches encapsulent véritablement le service en masquant la plupart des détails techniques de bas niveau [22].

La spécification WSDL joue un rôle important dans l'interopérabilité des services web en permettant d'automatiser davantage le processus d'invocation des services web. Cependant, WSDL est utilisé uniquement pour décrire la partie statique. Ainsi, WSDL est généralement employé en combinaison avec une autre forme de description qui permet de décrire le comportement dynamique d'un service web.

1.5.2 SOAP

La communication par message constitue un point crucial dans toute architecture SOA. Le standard actuel qui assure la messagerie est le protocole SOAP. SOAP est un protocole léger de transmission de messages et des appels de procédures à distance (RPCs) dans un environnement totalement distribué et hétérogène, dont la simplicité et l'extensibilité sont deux objectifs primordiaux de sa conception [17] [26].

Les autres technologies, telles que CORBA, RMI Java et DCOM offrent des fonctionnalités similaires à celles offertes par SOAP, en permettant aux objets distribués de solliciter et d'obtenir des services rendus par d'autres objets. Cependant, les messages SOAP sont écrits entièrement en XML. Ainsi, ils sont exceptionnellement indépendants des plates-formes et des langages de programmation [15]. Par exemple, un client écrit en Microsoft VisualBasic peut utiliser SOAP pour invoquer une méthode d'un objet CORBA sur une plate-forme Unix. De plus, SOAP ne définit pas un nouveau protocole de transport, mais il travaille sur des protocoles existants (HTTP en particulier).

SOAP représente une brique de base de l'architecture des services web, en permettant à des applications diverses d'échanger facilement des données et des services [15]. Couplé à WSDL, SOAP fournit à l'application cliente un mécanisme qui lui permet d'invoquer les services distants en lui donnant l'illusion qu'ils sont locaux. La compilation de la description WSDL permet de générer une souche de procuration (*proxy*) récupérant les appels au service de l'application et générant les messages SOAP [22].

La figure (Fig. 1.3) illustre avec plus de détails l'invocation d'un service web par SOAP. Les messages SAOP générés par la souche de procuration lors de l'appel sont transférés vers un port de connexion. En réponse, ils circulent depuis le service vers l'application cliente. Les messages peuvent être transportés par HTTP permettant la traversée des pares-feu (*firewalls*) et des contrôles d'accès.

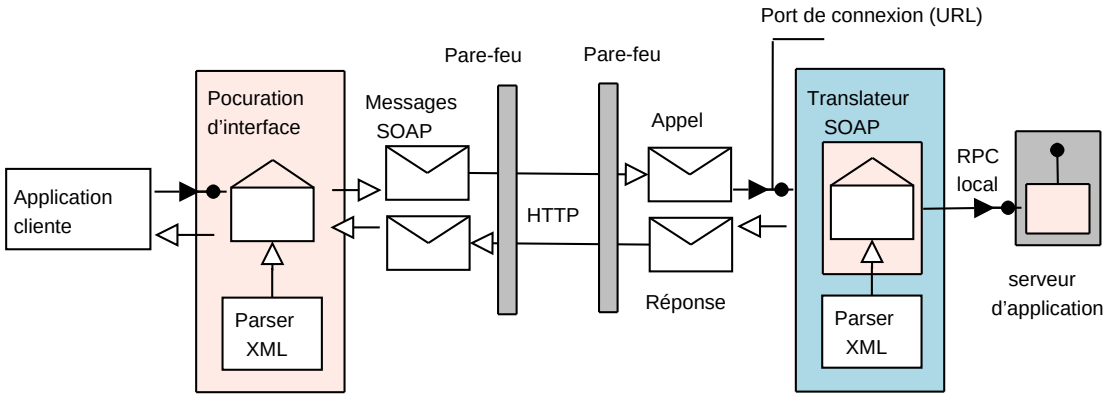

FIG. 1.3 – Schématique d'un échange SOAP

Structure d'un message SOAP [32] [42]

Un message SOAP (voir (Fig. 1.4)) est constitué d'une enveloppe indiquant que le document XML est un message SOAP et permettant au récepteur de déterminer le début et la fin de ce message. Il contient deux éléments fils : une entête optionnelle *header* et un corps obligatoire *body* appelé aussi charge utile.

L'élément *header* permet aux applications d'étendre les massages en ajoutant d'autres informations telles que l'authentification et la gestion des transactions. Il est encodé en tant que premier enfant de l'enveloppe SOAP. La charge utile est contenue dans un élément *body* et contient le massage destiné au récepteur final. L'élément *body* peut contenir n'importe quelle structuration XML à condition qu'elle obéisse à une grammaire dont l'espace de noms est référencé. Il peut en plus contenir un élément fils appelé SOAP *Fault*, qui sert à reporter les erreurs du message SOAP.

L'exemple suivant contient un message SOAP échangé sur HTTP et permet d'invoquer l'opération *GetLastTradePrice* du service web (*StockQuoteService*), conformément à la description WSDL vue ci-dessus.

```
POST /StockQuote HTTP/1.1 Host: http://example.com/stockquote
Content-Type: text/xml; charset="utf-8" Content-Length: nnnn
```

```
SOAPAction: http://example.com/GetLastTradePrice

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
       <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

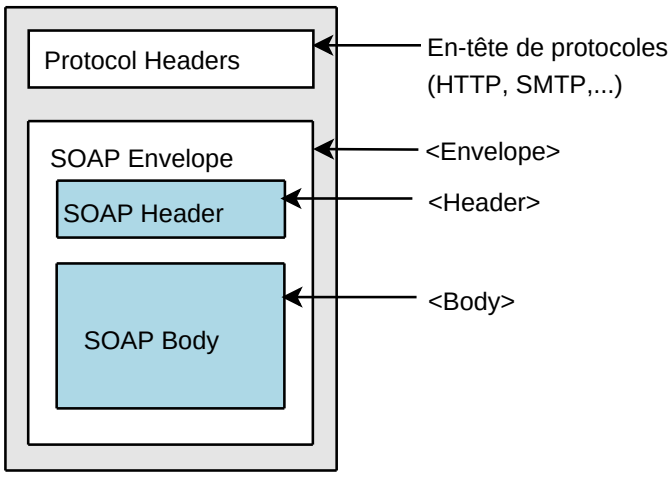

FIG. 1.4 – Format d'un message SOAP

1.5.3 UDDI

Jusqu'ici, nous avons présenté les deux standards qui définissent l'aspect le plus basique de l'infrastructure des services web. Le problème qui reste à résoudre, surtout dans un environnement ouvert comme Internet est comment permettre à un client de localiser aussi bien les services que leurs descriptions WSDL. Le standard UDDI a été conçu pour réduire cet écart entre les applications clientes et les services web.

UDDI est un standard basé sur XML qui fournit une méthode pour publier et découvrir des descriptions des services web [1]. Les annuaires UDDI offrent la possibilité de publier et de découvrir des informations sur une entreprise et ses services web [29]. Ces données peuvent être classifiées à l'aide de taxinomies standards. Cela permet d'effectuer une recherche par catégorie. Enfin et surtout, Les UDDI contiennent des informations sur les interfaces techniques des services d'une entreprise. Les annuaires

sont accessibles en SAOP comme des services web en permettant aux fournisseurs de s'enregistrer et de publier leurs services et aux clients de consulter et d'extraire les données concernant un service et/ou son fournisseur.

Les informations UDDI sont hébergées par des noeuds opérateurs organisés en réseau et répliquant les données entre eux. La publication d'un service chez un opérateur, donne lieu automatiquement à un processus de propagation des informations aux différents noeuds. L'accès à l'ensemble d'informations des annuaires peut se faire de n'importe quel opérateur UDDI [1].

Structure de l'annuaire UDDI

Le contenu d'un annuaire UDDI contient quatre type d'informations[2] : *businessEntity*, *businessService*, *bindingTemplate* et *tModel* . La figure (Fig. 1.5) illustre les relations entre ces structures.

- **businessEntity :** Correspond aux pages blanches et contient des informations concernant l'entreprise qui offre le service : nom de l'entreprise, sa description, l'adresse électronique des contacts qu'elle met à la disposition des clients et une ou plusieurs catégories auxquelles elle appartient (selon des taxonomies). Les éléments *categoryBag* et *identifierBag* permettent de distinguer les entreprises suivant certains critères et permettent de faciliter la recherche dans l'annuaire UDDI. Une entité *businessEntity* regroupe un ensemble d'éléments de service *businessService*.
- **businessService :** Correspond aux pages jaunes et contient des informations sur un service : nom du service, sa description, son code. Il fait référence à une catégorie de service pour permettre une recherche en fonction d'un type de service particulier. Un *businessService* contient une collection d'éléments *bindingTemplate*.
- **bindingTemplate :** Correspond aux pages vertes et contient des informations techniques sur un service web. Ces informations indiquent les références à un ou plusieurs *tModels*, ainsi que le point de terminaison (adresse Internet) du service.
- **tModel :** Un *tModel* permet de décrire les spécifications techniques du service web à enregistrer. Les *tModls* sont généralement des fichiers WSDL.

Grâce à un jeu d'API basé sur SOAP, il est possible d'interagir avec UDDI au moment de la conception et de l'exécution [29]. La spécification UDDI décrit deux APIs. Une pour la recherche et l'autre pour la publication. Chaque API définit un

[2]voir [3] pour plus de détails

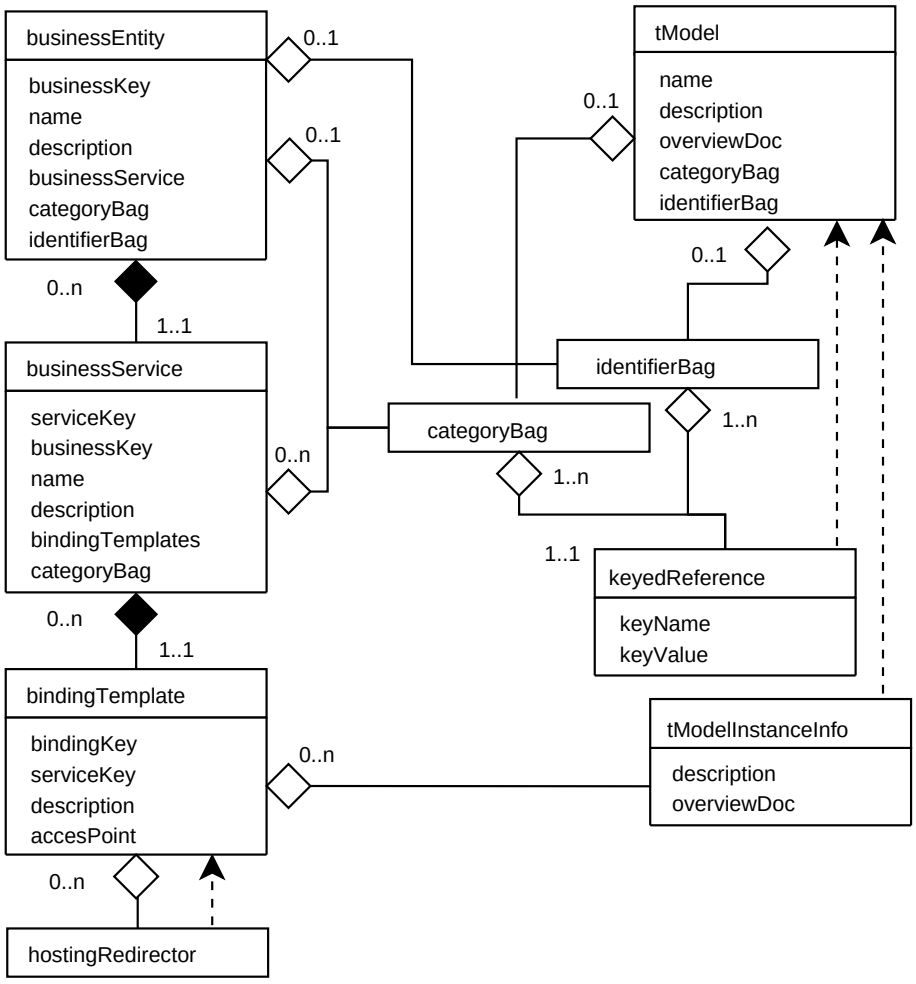

FIG. 1.5 – Modèle structurel des données de l'annuaire UDDI

ensemble de primitives dont on trouve la description détaillée dans [2]. Ces primitives sont utilisées dans des messages SOAP.

L'API de recherche ne nécessite pas un accès authentifié et définit des primitives de type *find_xxx* qui concernent les quatre structures de l'annuaire (*business*, *service*, *binding* et *tModel*). Par exemple *find_service* est utilisé pour chercher un service. Ces primitives sont complétées par des *get_xxxDetail* qui fournissent en réponse plus de détails sur les données.

Les primitives de l'API de publication requièrent des accès authentifiés et servent aux fournisseurs de services et aux entreprises pour mettre à jours les informations de l'annuaire. Les primitives de type *save_xxx* sont utilisées pour enregistrer une des quatre structures de l'annuaire et celles de type *delete_xxx* pour la supprimer.

1.6 Conclusion

Dans ce chapitre nous avons présenté les services web. Il s'agit d'une nouvelle technologie pour le développement et l'intégration d'applications. Les objectifs des services web ne sont pas nouveaux. Plusieurs solutions ont été proposées pour faciliter l'intégration d'applications hétérogènes. Cependant, aucune de ces solutions n'a réussi à fournir une intégration parfaite, surtout dans le cas d'un environnement ouvert comme Internet.

L'architecture orientée service et la standardisation sont les deux facteurs principaux qui ont permis l'expansion des services web comme une technologie de choix pour le développent et l'intégration des applications. En effet, L'architecture orientée service a rendu possible le développement d'applications faiblement couplées, la standardisation a permis de masquer l'hétérogénéité à large échelle. Ainsi, on peut facilement faire interagir des applications développées par des langages différents et tournant sur des plates-formes différentes.

Actuellement, l'infrastructure de base des services web est constituée de trois standards de W3C : SOAP, WSDL et UDDI. Ces standards facilitent la description, la découverte et la communication entre les services. Cependant, cela n'est pas suffisant pour répondre à tous les problèmes liés à l'intégration. En effet, cette infrastructure de base est principalement concernée par l'interopérabilité aux niveaux bas de la pile des services, la modélisation et l'analyse des abstractions dans les niveaux les plus élevés de la pile sont peu supportées, et en particulier la modélisation et la gestion des protocoles de services web qui font l'objet du prochain chapitre.

Chapitre 2

Protocoles de services web

2.1 Introduction

Comme nous l'avons vu précédemment, le couplage faible est l'une des caractéristiques essentielles des services web. Cela implique qu'un service n'est pas conçu pour interopérer avec des clients spécifiques. Les clients et les services peuvent être développés par des équipes différentes appartenant éventuellement à des entreprises différentes. La description du service est tout ce que le développeur a besoin pour comprendre le comportement du service et pour pouvoir développer les clients qui peuvent interagir correctement avec le service. Pour cette raison, la description de service inclus non seulement l'interface du service mais également le protocole de service métier (*business protocol* ou *conversation protocol*) supporté par le service [9] (contrairement aux *middleware* classiques comme CORBA). Ceci est important, car il se produit rarement que les opérations d'un service sont invoquées indépendamment les unes des autres. Les interactions entre les clients et les services sont souvent structurées en un ensemble d'invocations d'opérations, dont l'ordre doit satisfaire certaines contraintes pour que les clients puissent obtenir les services dont ils ont besoin. Un protocole de service décrit un ordre sur les messages échangés entre un client et un service lors de l'invocation d'un service web [30].

Une fois les protocoles sont décrits, le service peut savoir comment interagir avec d'autres services en terme d'échanges légaux de messages. Un service peut être simultanément impliqué dans plusieurs échanges de messages (conversations) avec différents clients, et doit donc gérer des instances multiples et concurrentes du protocole. En plus des *business protocols*, on peut associer à un service web d'autres protocoles, tels que la sécurité et les transactions qui doivent également être exposés en tant qu'éléments de la description de service de sorte que les clients sachent comment interagir avec le service [7].

Dans ce chapitre, nous nous intéressons aux protocoles de services web. Nous allons en premier temps présenter quelques domaines d'applications des protocoles de services, ainsi qu'un modèle de protocole basé sur les automates à états finis. Puis nous allons présenter les deux principaux types d'analyses qu'on peut effectuer sur les protocoles de service (analyse de compatibilité et analyse de remplaçabilité). Enfin, nous allons présenter les dépôts de protocoles qui permettent d'effectuer les différents types d'analyse sur les protocoles de services web.

2.2 Applications des protocoles de services web

L'analyse et la gestion des protocoles de services visent à faciliter le développement, le déploiement, l'exécution et la gestion des services web. En plus de permettre le développement des clients qui peuvent interagir correctement avec des services spécifiques, les protocoles de services ont aussi d'autres applications importantes.

Génération automatique du code

Les protocoles peuvent simplifier l'implémentation des services en permettant une génération automatique du squelette du code (*code skeleton*). Ce dernier n'inclut aucune logique métier spécifique au service, c'est-à-dire n'inclut aucune information concernant l'implémentation des opérations. Les invocations des opérations apparaissent comme des boites noire dans le squelette. L'extension du squelette est laissée au développeur qui va raffiner l'implémentation de chaque opération. Cela a l'avantage de simplifier considérablement le processus de développement, le développeur se concentre uniquement sur l'implémentation de la logique interne du service. La partie du code qui gère la conversation et vérifie la conformité des messages est automatiquement générée [30] [7].

Analyse de compatibilité

Le protocole du client et celui du service peuvent être analysés pendant le temps de développement pour déterminer les parties des protocoles qui sont compatibles et celles qui ne le sont pas. C'est-à-dire, spécifier les conversations possibles entre les deux services. De plus, il est possible de suggérer des modifications qu'on doit apporter si on veut augmenter le niveau de compatibilité. Il est aussi possible d'analyser les protocoles au moment de l'exécution. Par exemple, lorsqu'un service veut utiliser d'autres services, l'analyse des protocoles permet au moteur de recherche de limiter l'ensemble des services rendu en cherchant uniquement les services avec lesquels la communication est possible [7].

Simplification de l'évolution

Les services web fonctionnent de façon autonome dans des environnements dynamiques, leurs spécifications peuvent changer, par exemple à cause des changements des lois, des règlements ou des changements dans les stratégies des entreprises, etc. En particulier, certains services peuvent changer leurs protocoles, d'autres peuvent disparaître et d'autres peuvent émerger. Par conséquence, les services peuvent ne pas pouvoir invoquer des opérations demandées.

L'analyse des protocoles permet de [7] :
- Identifier statiquement ou dynamiquement les services alternatifs qui offrent des protocoles équivalents.
- Identifier les problèmes de compatibilité entre les protocoles et si possible créer des adaptateurs pour permettre des interactions entre les services incompatibles [8].
- Tester la compatibilité d'une nouvelle version d'un protocole de service avec un client spécifique.

Vérification de la conformité

Les protocoles permettent aussi de vérifier la conformité des services avec des standards ou des consortiums industriels. Cela est important, car la standardisation est l'un des points essentiels assurant l'interopérabilité des interactions entre les services web [7].

2.3 Modèle de protocoles

2.3.1 Présentation

Les automates à états finis sont un outil commode pour modéliser les protocoles de service, dont les états représentent les différentes étapes dans lesquelles un service pourrait être pendant son interaction avec un client. Les transitions sont déclenchées par l'envoi des messages par le client au service ou vice versa. Un message correspond à l'invocation d'une opération de service ou à sa réponse. Chaque transition est étiquetée par le nom et la polarité du message (la polarité plus (+) pour les messages entrants et la polarité moins (-) pour les messages sortants). L'automate contient un seul état initial, mais peut contenir plusieurs états finaux qui correspondent à des utilisations finalisées du service [7] [9].

Nous allons prendre comme exemple, le protocole représenté par la figure (Fig. 2.1)

qui décrit le comportement externe d'un service permettant de vendre des produits en ligne. La figure présente deux protocoles de conversation, le protocole du service et celui du client. Le service se trouve initialement dans l'état *start*, et lorsqu'un client commence la conversation en envoyant le message *login*, le service se déplace vers l'état *Logged*, puis vers l'état *Searching* lorsqu'il reçoit le message *searchGood*, et ainsi de suite jusqu'à arriver à un état final. On peut bien remarquer que le client doit respecter certaines contraintes pour pouvoir interagir correctement avec le service. Par exemple, il ne peut pas effectuer l'opération de confirmation de la commande s'il n'a pas effectué l'opération de sélection du produit. On peut aussi remarquer que le service peut être impliqué dans des conversations non supportées par le client (eg., l'opération *cancel*), le client peut à son tour être impliqué dans des conversations non supportées par le service (eg., l'opération *comparePrice*). Cela pose un problème de compatibilité qu'on doit prendre en considération lorsqu'on veut faire communiquer le client avec le service. D'une manière générale, pour que deux ou plusieurs services puissent interopérer, leurs protocoles doivent être compatibles.

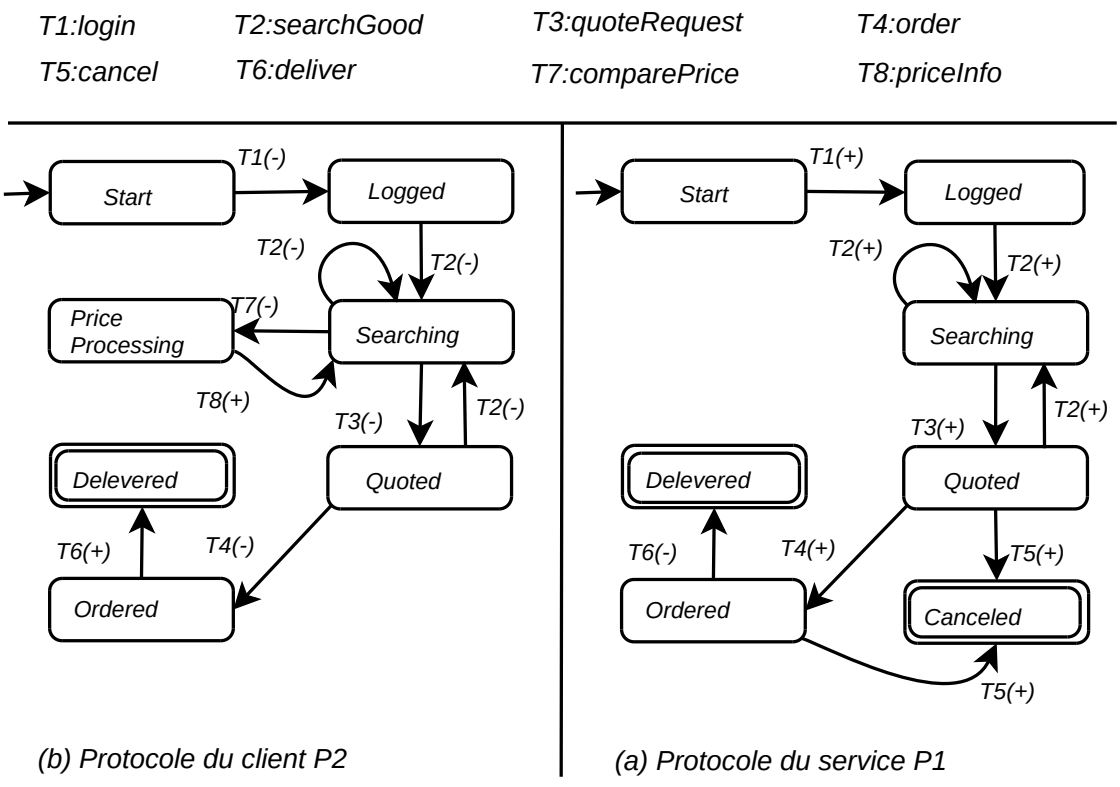

FIG. 2.1 – Exemple de modèle de protocole de service

2.3.2 Définition Formelle

Un protocole est défini par le tuple (S, s_0, F, M, R) [7] :

- S est un ensemble fini d'états.
- $s_0 \in S$ représente l'état initial.
- $F \subseteq S$ représente l'ensemble des états finaux.
- M est un ensemble fini de messages. Pour chaque message $m \in M$ on définit une fonction de polarité $Polarity(P, m)$ qui prend la valeur $(+)$ pour les message reçus et la valeur $(-)$ pour les massages envoyés. On utilise la notation $m(+)$ (respectivement $m(-)$) pour représenter la polarité du message m.
- $R \subseteq S^2 \times M$ est un ensemble de transitions. Chaque transition $(s, t, m) \in R$ est identifiée par l'état initial s, l'état final t et un message (entrant ou sortant) m.

2.3.3 Représentation XML

Les protocoles de services sont décrits à l'aide de documents XML, voici un morceau de code XML représentant le protocole $P1$ de la figure (Fig. 2.1). Cet exemple tient en compte les propriétés d'activation et les contraintes temporelles qui font l'objet de la section suivante.

```
<protocol>
  ...
  <transition name="T3" source="Searching" target="Quoted">
    <activation mode="user" event="quoteRequest ">
      <pre-conditions O-condition="True"
                      U-condition ="True"
                      T-condition="True"/>
    </activation>
  </transition>
  ...
  <transition name="T5" source="Ordered" target="Canceled">
    <activation mode="user" event="cancel">
      <pre-conditions O-condition="True"
                      U-condition ="True"
                      T-condition="C-Invoke(<,end(T4) + 7)"/>
    </activation>
  </transition>
  ...
</protocol>
```

2.4 Propriétés d'activation et contraintes temporelles

2.4.1 Propriétés d'activation

Les propriétés d'activation spécifient l'événement et le mode d'activation et des pre-conditions [9]. Une transition est activée par un événement, le mode d'activation indique si le déclenchement de la transition est explicite (mode utilisateur) ou implicite (mode fournisseur). Dans le premier cas, la transition est activée par l'invocation d'une opération du service par le client. Dans le deuxième cas, la transition est déclenchée automatiquement après un événement temporel. Une pre-condition est le triplet (O-condition, U-condition et T-condition) :

- O-condition : condition sur les objets du service.
- U-condition : condition sur le profil de l'utilisateur, elle est utilisée pour limiter les utilisateurs autorisés à invoquer l'opération.
- T-condition : spécifie des contraintes temporelles.

2.4.2 Contraintes temporelles

Une partie importante des conversations réelles sont soumises à des contraintes temporelles. Ces contraintes indiquent quand une opération doit ou peut être invoquée. Par exemple, un service offre au client la possibilité d'annuler sa demande dans un délai de 24 heures après l'envoi de cette demande. Pour modéliser ce type de conversation, deux types d'expressions temporelles sont possibles :

- Les intervalles de validité temporelle (C-Invoke [9]) : L'opération ne peut être effectuée que dans un laps de temps, elle devient invalide en dehors.
- Les délais d'expiration (M-Invoke [9]) : L'opération est effectuée automatiquement une fois le délai expiré.

Les deux contraintes temporelles peuvent être associées au mode d'activation de type fournisseur. En revanche, seules les contraintes de type intervalle de validité peuvent être associées au mode utilisateur. En effet, on ne peut attendre de l'utilisateur d'effectuer une opération à une date précise.

D'une manière formelle, une contrainte temporelle est représentée comme suit : $Preq(boolop, d)$ où :

1. $Pred$ peut être $C - Invoke$ ou $M - Invoke$.
2. *boolop* est un opérateur de comparaison (e.g., =,<,>).
3. d est une date (eg., $begin(t)$ ou $end(t)$ qui désignent respectivement le début et la fin de la dernière invocation de la transition T dans la même instance de conversation).

Transition	**T-condition**
$T5$	$C - invoke(<,\text{end}(T4)+7)$
$T6$	$M - inoke(>=,\text{end}(T4)+7)$

TAB. 2.1 – Exemple de contraintes temporelles

La table (Tab. 2.1) illustre un exemple de deux conditions temporelles. La première condition signifie que l'opération associée à la transition $T5$ peut être uniquement invoquée dans un intervalle de 7 jours après l'accomplissement de $T4$. La deuxième signifie que la transition $T6$ sera automatiquement déclenchée 7 jours après l'accomplissement de la transition $T4$.

2.5 principaux types d'analyse de protocoles

Dans cette section, nous allons présenter les principaux types d'analyses des protocoles des service web. En particulier, nous allons définir différents types de compatibilités qui correspondent à différentes capacités des services à interopérer, puis nous allons discuter les similitudes et les différences entre les protocoles, afin de comprendre si les services ont le même comportement ou si un service peut remplacer un autre.

2.5.1 Compatibilité (*compatibility*)

L'analyse de compatibilité consiste à vérifier si l'interopérabilité entre deux services est possible ou non, et déterminer celles des conversations qui sont possibles et celles qui ne le sont pas.

La notion de compatibilité est très utile dans le contexte des services web. Par exemple, les interactions entre les services qui ne sont pas compatibles (compatibilité partielle ou totale) n'ont aucun sens, car aucune conversation significative ne peut avoir lieu. En outre, s'il y a seulement une compatibilité partielle, le développeur et le *middleware* du service doivent se rendre compte de ceci, car dans ce cas on ne peut pas exploiter toutes les capacités du service, d'autres contraintes sont imposées et en conséquence certaines conversations ne sont pas permises. L'analyse de compatibilité facilite aussi l'évolution en permettant de vérifier si un client modifié peut encore interagir avec certains services.

On peut identifier deux classes de compatibilité [7] [10] :

2.5.1.1 Compatibilité totale (*Full compatibility*)

Un protocole $P1$ est totalement compatible avec un autre protocole $P2$ si toutes les exécutions de $P1$ peuvent interopérer avec $P2$, c'est-à-dire, toute conversation générée par $P1$ est comprise par $P2$.

Par exemple, toutes les conversations générées par le protocole $P1$ de la figure (Fig. 2.2) sont comprises par le protocole $P2$, ainsi $P1$ est totalement compatible avec $P2$, mais l'inverse n'est pas vrai, car $P2$ pourrait être impliqué dans des conversations non supportées par $P1$ (l'opération *cancelOrder*). Il est évident que la relation de compatibilité totale n'est pas symétrique.

2.5.1.2 Compatibilité partielle (*Partial compatibility*)

Un protocole $P1$ est partiellement compatible avec un autre protocole $P2$ si certaines exécutions de $P1$ peuvent interopérer avec $P2$, c'est-à-dire, s'il y a au moins une conversation possible entre le service qui supporte $P1$ et celui qui supporte $P2$.

Par exemple, le protocole $P1$ de la figure (Fig. 2.3) n'est pas totalement compatible avec $P2$, car $P1$ a la capacité d'envoyer le message *cancelOrder* qui n'est pas supporté par $P2$. Cependant, il y a des conversations possibles entre ces deux protocoles à condition que le client ($P1$) ne demande pas l'annulation de la commande (l'opération *cancelOrder*). Par conséquent, $P1$ est partiellement compatible avec $P2$.

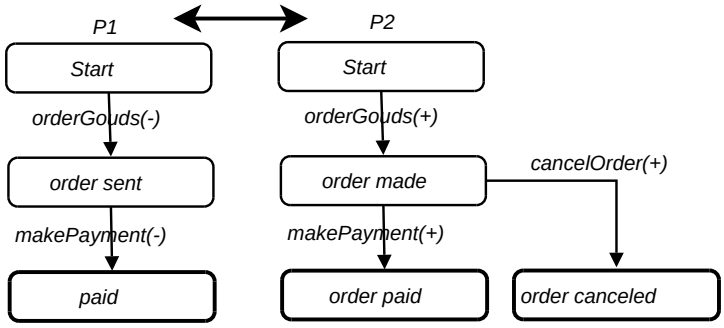

FIG. 2.2 – Exemple de compatibilité totale

2.5.2 Remplaçabilité (*Replaceability*)

L'analyse de remplaçabilité consiste à vérifier l'équivalence entre deux protocoles en terme de conversations dont ils supportent. Ce type d'analyse est très important et on peut le faire dans le cas général ou bien au moment de l'interaction avec certains clients.

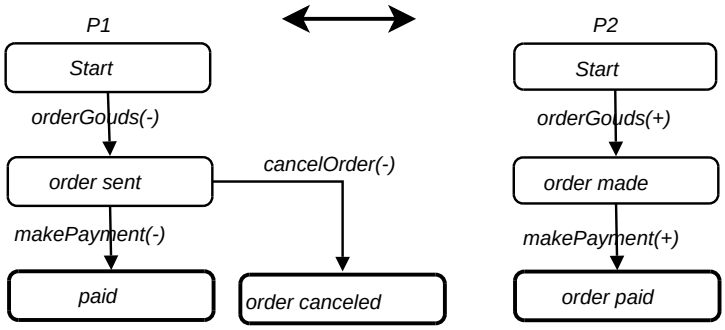

FIG. 2.3 – Exemple de compatibilité partielle

Par exemple, on peut vérifier si une nouvelle version d'un service peut supporter les mêmes conversations que la précédente, ou bien si un service nouvellement défini peut supporter les conversations exigées par la spécification d'un standard donné.

Selon la capacité d'un service à remplacer un autre, on peut classifier l'analyse de remplaçabilité en plusieurs classes [7] [10] :

2.5.2.1 Equivalence

Deux protocoles sont équivalents s'ils supportent le même ensemble de conversations. Chaque conversation légale selon le premier sera également légale selon le deuxième et vice-versa. Ceci signifie que chacun de ces deux protocoles peut être remplacé par l'autre dans n'importe quel contexte et d'une manière transparente pour les clients.

Par exemple, le protocole $P1$ et le protocole $P2$ de la figure (Fig. 2.4) sont équivalents.

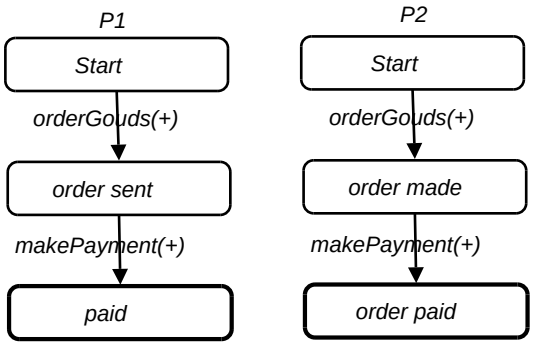

FIG. 2.4 – Exemple d'équivalence

2.5.2.2 Subsumption

On parle de ce type de remplaçabilité lorsqu'un protocole peut supporter au moins toutes les conversations supportées par un autre protocole. Dans ce cas, le premier protocole peut toujours remplacer le deuxième d'une manière transparente, mais l'inverse n'est pas nécessairement vrai.

Par exemple, il existe une remplaçabilité de ce type entre le protocole $P2$ et le protocole $P1$ de la figure (Fig. 2.5). On utilise dans ce cas la notation $Subs(P1,P2)$.

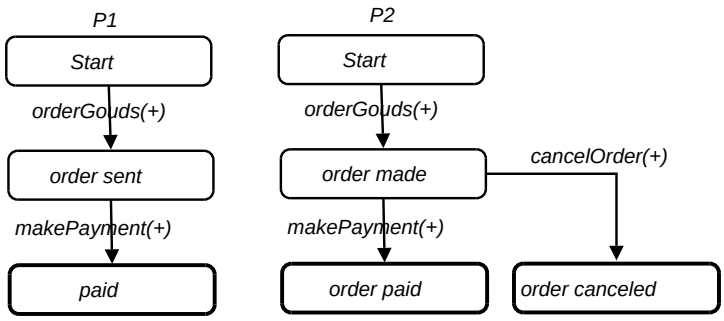

FIG. 2.5 – Exemple de remplaçabilité de type *subsumption*

Les deux définitions précédentes correspondent à la remplaçabilité dans le cas général. Cependant, on veut parfois savoir si un service peut remplacer un autre lors de l'interaction avec un client particulier. C'est-à-dire, on veut savoir si toutes les conversations légales entre un client et un service (service à remplacer) sont aussi des conversations légales entre le client et un autre service (service remplaçant). Dans ce cas, on s'intéresse à une remplaçabilié faible.

Par exemple, le protocole $P1$ de la figure (Fig. 2.6) peut remplacer $P2$ lors de l'interaction avec le client C. Mais, $P1$ et $P2$ ne sont pas équivalents et $Subs(P2,P1)$ n'est pas vrai. Ainsi, $P1$ ne peut pas remplacer $P2$ avec n'import quel client.

2.6 Présentation des dépôts de protocoles

Pour concrétiser les avantages des protocoles de services, nous avons besoins dans la pratique d'un ensemble d'outils. Dans notre travail, nous nous intéressons à la création de dépôts permettant aux utilisateurs d'effectuer des études sur les protocoles de services.

La figure (Fig. 2.7) présente l'architecture générale d'un dépôt. La base de données

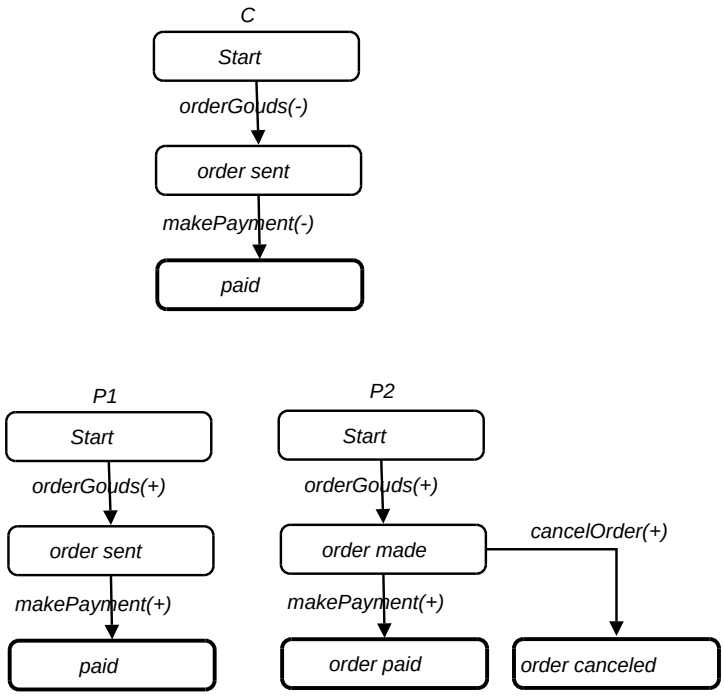

FIG. 2.6 – Exemple de remplaçabilité par rapport au client

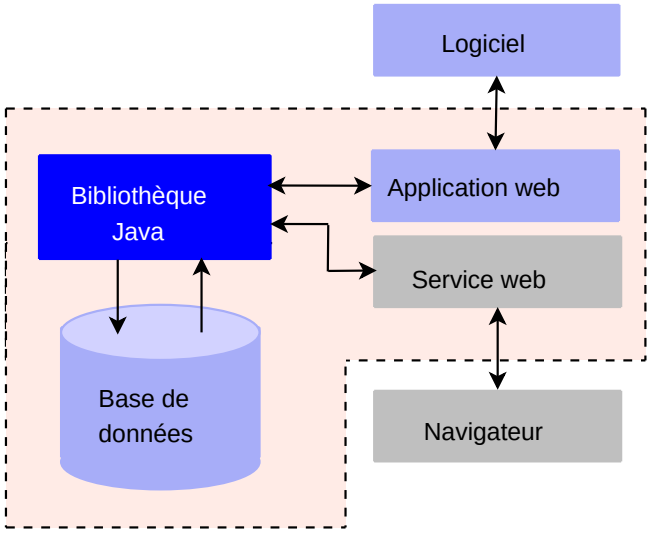

FIG. 2.7 – Architecture d'un dépôt de protocole

est utilisée pour stocker les protocoles de services sous forme de documents XML. La bibliothèque sert comme une interface entre les utilisateurs de dépôt et la base de données. Cette bibliothèque est implémentée en Java et offre un ensemble de fonctions telles que l'ajout, la suppression, la modification et la recherche d'un protocole. De

plus, elle permet de gérer des index sur les protocoles pour améliorer les performances du dépôt. Pour permettre aux utilisateurs d'interroger les protocoles de services, la bibliothèque sera exposée en tant que :

– Service web pour la rendre accessible par des logiciels ou des plug-in Eclipse.
– Application web pour la rendre accessible aux utilisateurs humains à travers des navigateurs.

2.7 conclusion

Dans ce chapitre, nous avons présenté un modèle de protocoles de services web basé sur les automates à états finis. Les protocoles de services représentent une partie importante de la description des services. Couplés à l'interface de services, ils permettent de décrire le comportement externe des services web. Les protocoles de services servent principalement à aider les développeurs à écrire des clients pouvant interagir correctement (en terme d'échange de messages) avec un service particulier, ils ont aussi d'autres utilisations permettant de faciliter le cycle de vie entier des services web. Nous avons aussi présenté les deux principaux types d'analyse qu'on peut effectuer sur les protocoles de services : l'analyse de compatibilité qui correspond à la capacité de deux services à interopérer et l'analyse de remplaçabailité qui correspond à la capacité d'un service à remplacer un autre que ce soit dans le cas général ou bien uniquement lors de l'interaction avec un client particulier. En pratique, pour réaliser tous ça, nous avons besoin d'un ensemble d'outils. Dans notre projet, nous nous intéressons à la création de dépôts permettant d'effectuer des études de nature variée sur les protocoles de services web.

Chapitre 3

Publication et recherche des descriptions WSDL

3.1 Introduction

UDDI fournit le support à plusieurs types de descriptions de services. De ce fait, UDDI n'offre pas un *mapping* direct pour WSDL ou tout autre mécanisme de description [36]. L'organisation UDDI a publié un document décrivant les pratiques qu'on doit suivre lors de la publication de la description d'un service web dans un annuaire UDDI [20].

Dans ce chapitre, nous allons décrire un processus permettant la publication et la recherche des descriptions de services WSDL en se basant sur le document précédent et en utilisant les deux interfaces de programmations (API) décrites par la spécification UDDI. Cela va nous aider à comprendre le fonctionnement des annuaires UDDI et ainsi choisir la façon de faire le lien entre les annuaires de services UDDI et nos dépôts de protocoles. Pour cela, nous allons d'abord présenter une manière de structurer les descriptions WSDL facilitant l'utilisation de WSDL avec UDDI.

3.2 Description WSDL

Les descriptions de services WSDL peuvent être structurées de différentes manières. Cependant, si les informations réutilisables sont séparées des informations spécifiques à une instance donnée d'un service, la publication et la recherche des descriptions de service WSDL dans les annuaires UDDI deviennent particulièrement simples. Pour cela, chaque document WSDL est divisé en deux parties : l'interface de service et l'implémentation de service (voir Fig. 3.1) [20].

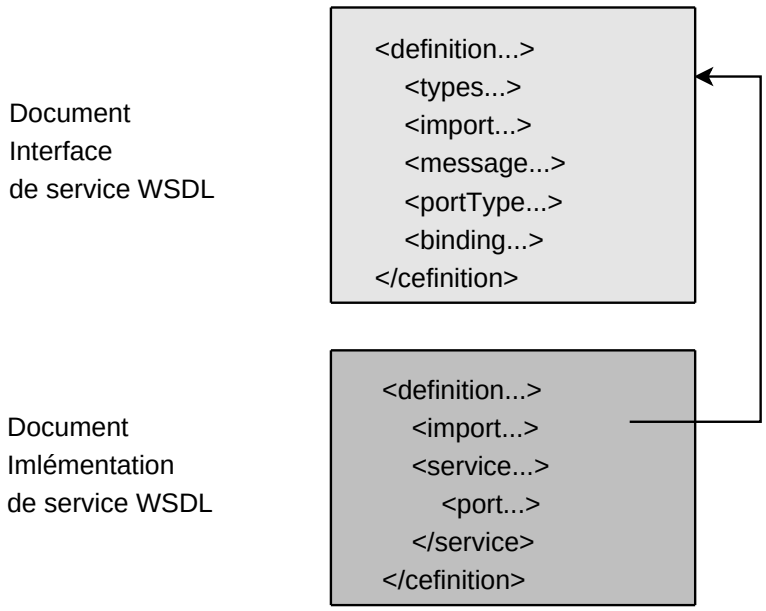

FIG. 3.1 – Relation entre la partie interface et implémentation WSDL

3.2.1 Interface de service

L'interface de service est décrite par un document WSDL contenant les éléments *types*, *import*, *message*, *portType* et *binding*. Cette interface contient la définition abstraite décrivant un type spécifique de service. Elle sera utilisée pour implémenter un ou plusieurs services. Un document interface de service peut référencer un autre document interface de service en utilisant l'élément *import*. Par exemple, une interface de service qui contient seulement les éléments *message* et *porType* peut être référencée par une autre interface qui contient seulement les liaisons pour le *potType* [36] [43].

3.2.2 Implémentation de service

Le document implémentation de service WSDL contient les éléments *import* et *service* et permet de décrire le service qui implémente une interface de service spécifique. Au moins un des éléments *import* contient une référence au document interface de service WSDL. Un document implémentation de service peut contenir des références à plusieurs documents interface de service. L'élément *import* dans le document implémentation de service contient deux attributs : l'attribut *namespace* qui correspond au *targetNamespace* dans le document interface de service et l'attribut *location* qui est une URL utilisée pour référencer le document interface de service. L'attribut *binding* de l'élément *port* contient une référence à une liaison spécifique dans le document interface de service [36] [43].

La figure (Fig. 3.2) contient un exemple de document interface de service.

```
<?xml version="1.0"?>
<definitions name="StockQuoteService-interface"
  targetNamespace="http://www.getquote.com/StockQuoteService-interface"
  xmlns:tns="http://www.getquote.com/StockQuoteService-interface"
  xmlns:xsd=" http://www.w3.org/2001/XMLSchema "
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <documentation>
    Standard WSDL service interface definition for a stock quote service
  </documentation>
  <message name="SingleSymbolRequest">
    <part name="symbol" type="xsd:string"/>
  </message>
  <message name="SingleSymbolQuoteResponse">
    <part name="quote" type="xsd:string"/>
  </message>
  <portType name="SingleSymbolStockQuoteService">
    <operation name="getQuote">
      <input message="tns:SingleSymbolRequest"/>
      <output message="tns:SingleSymbolQuoteResponse"/>
    </operation>
  </portType>
  <binding name="SingleSymbolBinding"
           type="tns:SingleSymbolStockQuoteService">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getQuote">
      <soap:operation soapAction="http://www.getquote.com/GetQuote"/>
      <input>
        <soap:body use="encoded"
          namespace="urn:single-symbol-stock-quotes"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
          namespace="urn:single-symbol-stock-quotes"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
</definitions>
```

FIG. 3.2 – Exemple de document interface de service WSDL

Un document implémentation de service correspondant à l'interface de service vue ci-dessus est présenté dans la figure (Fig. 3.3).

Les éléments encadrés dans les deux figures (Fig. 3.2) et (Fig. 3.3) seront utilisés lors de la publication pour construire certains éléments de *tModel* et de *businessService*.

```
<?xml version="1.0"?>
<definitions name="StockQuoteService"
  targetNamespace="http://www.getquote.com/StockQuoteService"
  xmlns:interface="http://www.getquote.com/StockQuoteService-interface"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <documentation>
    This service provides an implementation of a
    standard stock quote service
  </documentation>
  <import namespace= "http://www.getquote.com/StockQuoteService-interface"
          location= "http://www.getquote. com     /SQS-interface.wsdl"/>
  <service name="StockQuoteService">
    <documentation>Stock Quote Service</documentation>
    <port name="SingleSymbolServicePort"
          binding="interface:SingleSymbolBinding">
      <documentation>Single Symbol Stock Quote Service</documentation>
      <soap:address location= "http://www.getquote.com/stockquoteservice"/>
    </port>
  </service>
</definitions>
```

FIG. 3.3 – Exemple de document implémentation de service WSDL

3.3 Publication

Lorsque la description de service est achevée, on peut la publier grâce à UDDI afin de permettre aux partenaires futurs de découvrir les services proposés. La figure (Fig. 3.4) décrit les correspondances entre les éléments WSDL et les structures de données UDDI. Les interfaces de services doivent être publiées en tant que *tModels* avant que les implémentations ne soient publiées en tant que *businessServices* [36].

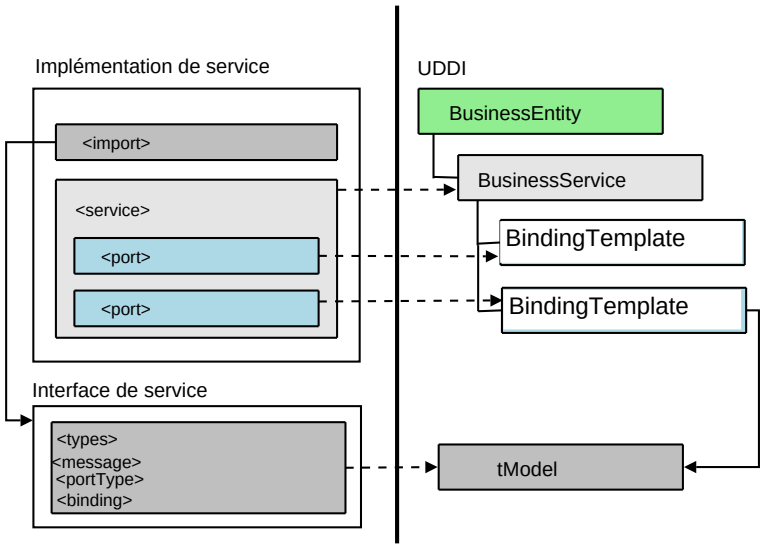

FIG. 3.4 – Correspondances entre les éléments WSDL et les structures de données UDDI

3.3.1 Publication des interfaces de service

L'interface de service représente une définition réutilisable de service, par conséquent elle est publiée dans un annuaire UDDI en tant que *tModel* [20]. Certains éléments du *tModel* sont construits en utilisant les informations de la description de l'interface. Les étapes de la création d'un *tModel* sont [20] [36] :

- Le nom du *tModel* est créé à partir de *targetNamespace* de l'interface de service.
- L'élément *description* est créé à partir de l'élément *documentation* qui est associé à l'élément definitions dans le document de l'interface de service.
- L'élément *overviewURL* est créé à partir de l'adresse réseau accessible pour le document interface de service WSDL. S'il y a plus d'un *binding* dans le document interface de service, alors le nom de ce *biniding* est concaténé avec le *overviewURL* en utilisant le séparateur "#".
- L'élément *categoryBag* doit contenir au moins un élément *keyedReference* contenant une référence au taxonomie *uddi-org :types* permettant de classifier le *tModel*. La valeur de l'attribut *keyValue* doit être *wsdlSpec* pour identifier le *tModel* comme une définition d'interface de service WSDL.

L'exemple de la figure (Fig. 3.5) contient le *tModel* créé lors de la publication de l'interface de service de la figure (Fig. 3.2).

```
<?xml version="1.0"?>
<tModel tModelKey="">
  <name>http://www.getquote.com/StockQuoteService-interface</name>
  <description xml:lang="en">
    Standard service interface definition for a stock quote service
  </description>
  <overviewDoc>
    <description xml:lang="en">
      WSDL Service Interface Document
    </description>
    <overviewURL>
      http://www.getquote.com        /SQS-interface.wsdl\
      #SingleSymbolBinding
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference
      tModelKey= "UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
      keyName= "uddi-org:types" keyValue="wsdlSpec"/>
    <keyedReference
      tModelKey= "UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
      keyName="Stock market trading services" keyValue="84121801"/>
  </categoryBag>
</tModel>
```

FIG. 3.5 – Exemple de structure de *tModel*

3.3.2 Publication des implémentations de service

L'implémentation de service est publiée en tant que *businessService* avec un ou plusieurs *bindingTemplates*.

businessService

Un nouveau *businessService* est créé pour chaque élément *service* défini dans le document implémentation de service. Les éléments du *businessService* sont construits en se basant sur le contenu du document implémentation de service de la manière suivante [20] [36] :

- L'élémnet *name* du *businessService* est créé à partir de l'attribut *name* tiré de l'élément *service* du document implémentation de service.
- L'élément *description* est créé à partir du contenu de l'élément *documentation* contenu dans l'élément *service*.

bindingTemplate

Un élément *bindingTemplate* est créé dans un *businessService* pour chaque élément *port* défini dans l'élément *service* [20] [36] :

- L'élément *description* est créé à partir de l'élément *documentation* de l'élément *port*.
- L'élément *accessPoint* est créé à partir de l'adresse du service associée à l'élément *port*. Il s'agit en général d'une URL permettant de localiser le service web en utilisant un protocole (HTTP par exemple). Le type de ce protocole est indiqué par l'attribut *URLType*.
- L'élément *tModelKey* continent une référence au *tModel* associé au document interface de service. Ce *tModel* peut être localisé en utilisant l'attribut *namespace* de l'élément *import*.
- l'élément *overviewURL* contient une référence directe au document implémentation de service. Si le document contient plus d'un seul port, alors le nom du port est concaténé avec le *overviewURL* en utilisant le séparateur "#".

L'exemple de la figure (Fig. 3.6) contient le *businessService* créé à partir du document implémentation de service de la figure (Fig. 3.3).

L'élément *categoryBag* contient une liste de couples non-valeur utilisés pour associer aux *businessEnties*, *businessServices* et aux *tModels* des informations de taxonomie précise. Ces informations peuvent être des codes produits, des codes services ou des codes géographiques. Ces informations sont optionnelles, mais leurs utilisations améliorent considérablement la recherche en utilisant les messages *find_ xxx* définis dans la spécification de l'API UDDI [2].

3.4 Recherche

La description de service est maintenant publiée dans un annuaire UDDI. Cette section décrit le processus choisi pour chercher et obtenir la description complète d'un service.

3.4.1 Recherche des descriptions d'interface de service WSDL

Les *tModels* sont catégorisés pour les identifier comme des descriptions de service basées sur WSDL [20]. Le message UDDI *find_ tModel* peut être utilisé pour trouver les *tModels* qui ont été catégorisés.

Le message *find_ tModel* présenté par le fragment du code XML ci-dessous peut être utilisé pour localiser toutes les descriptions d'interfaces de service WSDL :

```
<?xml version="1.0"?>
<find_tModel generic="1.0" xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference
      tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
      keyName="uddi-org:types"
      keyValue="wsdlSpec"/>
  </categoryBag>
</find_tModel>
```

Le message *find_ tModel* va envoyer une liste de clés *tModel keys*. Une description d'interface spécifique est obtenue en utilisant le message *get_ tModelDetail*. Ce dernier va retourner un *tModel* tel que celui présenté dans la figure (Fig. 3.5). Une fois qu'un *tModel* a été trouvé, l'élément *overviewURL* peut être utilisé pour obtenir le contenu du document interface de service WSDL.

Des *keyedReferences* complémentaires peuvent être ajoutés au *categoryBag* pour limiter l'ensemble de *tModels* retournés en réponse à cette requête. L'exemple suivant permet de localiser tous les services de catégorie *Stock market trading services* qui ont été définis en utilisant WSDL :

```
<?xml version="1.0"?>
<find_tModel generic="1.0" xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference
        tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
        keyName="uddi-org:types" keyValue="wsdlSpec"/>
    <keyedReference
        tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
        keyName="Stock market trading services"
        keyValue="84121801"/>
  </categoryBag>
</find_tModel>
```

```
<businessService businessKey="..." serviceKey="...">
  <name>StockQuoteService</name>
  <description xml:lang="en">
    Stock Quote Service
  </description>
  <bindingTemplates>
    <bindingTemplate bindingKey="..." serviceKey="...">
      <description>
        Single Symbol Stock Quote Service
      </description>
      <accesssPoint URLType="http">
        http://www.getquote.com/StockQuoteService
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="[tModel Key for Service Interface]">
          <instanceDetails>
            <overviewURL>
              http://www.getquote.com/services/SQS.wsdl
            </overviewURL>
          </instanceDetails>
        </tModelInstanceInfo>
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
  <categoryBag>
    <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-4411D14E384"
                    keyName="Stock market trading services"
                    keyValue="84121801"/>
  </categoryBag>
</businessService>
```

FIG. 3.6 – Exemple de structure de *businessService*

3.4.2 Recherche des descriptions d'implémentation de service WSDL

Il existe deux méthodes de base qui peuvent être utilisées pour trouver une description de service. Le message UDDI *find_service* peut être utilisé pour trouver les descriptions de service avec une classification spécifique, ou il peut être utilisé pour trouver les descriptions de service qui implémentent une interface de service spécifique [36].

Le message dans le fragment du code suivant peut être utilisé pour localiser les

businessServices qui sont des implémentations des services de catégorie *Stock market trading services*. Des *keyedReferences* complémentaires peuvent être ajoutés au *categoryBag* pour réduire la portée des descriptions de service qui sont retournées en réponse à ce message.

```
<?xml version="1.0"?>
<find_service businessKey="..." generic="1.0" xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference
       tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
       keyName="Stock market trading services"
       keyValue="84121801"/>
  </categoryBag>
</find_service>
```

Le message *find_service* peut être aussi utilisé pour localiser les *businessServices* qui sont des implémentations d'une interface de service spécifique [36]. Le fragment du code suivant contient un exemple d'un message *find_service* permettant de trouver tous les *businessServices* qui implémentent l'interface du service *stock quote*.

```
<?xml version="1.0"?>
<find_service businessKey="..." generic="1.0"  xmlns="urn:uddi-org:api">
  <tModelBag>
    <tModelKey>[tModelKey de l'interface de service WSDL]</tModelKey>
  </tModelBag>
</find_service>
```

Puisque l'interface de service est représentée par un *tModel*, un *tModelBag* est utilisé pour spécifier la clé du *tModel* de l'interface de service WSDL associée à ce service.

Le message *find_service* va retourner une liste de clés de services. La description du *businessService* est obtenue en utilisant le message *get_serviceDetail*. Ce dernier va retourner un *businessService* tel que celui présenté dans la figure (Fig. 3.6).

Une fois que le *businessService* a été obtenu, une *bindingTemplate* spécifique peut être utilisée pour invoquer le service web. Le *accessPoint* dans le *bindingTemplate* représente le point d'entrée du service. Le *overviewURL* contient une référence au document implémentation de service. Ce document peut être accédé pour obtenir plus de détails sur le service web implémenté. Le message *find_binding* permet de localiser un *bindingTemplate* spécifique.

Le fragment du code suivant contient un message *find_binding* qui peut être utilisé pour retrouver les *bindingTemplates* qui référencent un *tModel* spécifique. Ce *tModel* peut être associé à un *binding* spécifique dans une description d'interface de service WSDL.

```
<?xml version="1.0"?>
<find_binding serviceKey="..." generic="1.0" xmlns="urn:uddi-org:api">
  <tModelBag>
    <tModelKey>[tModelKey de l'interface de service WSDL]<tModelKey/>
  </tModelBag>
</find_service>
```

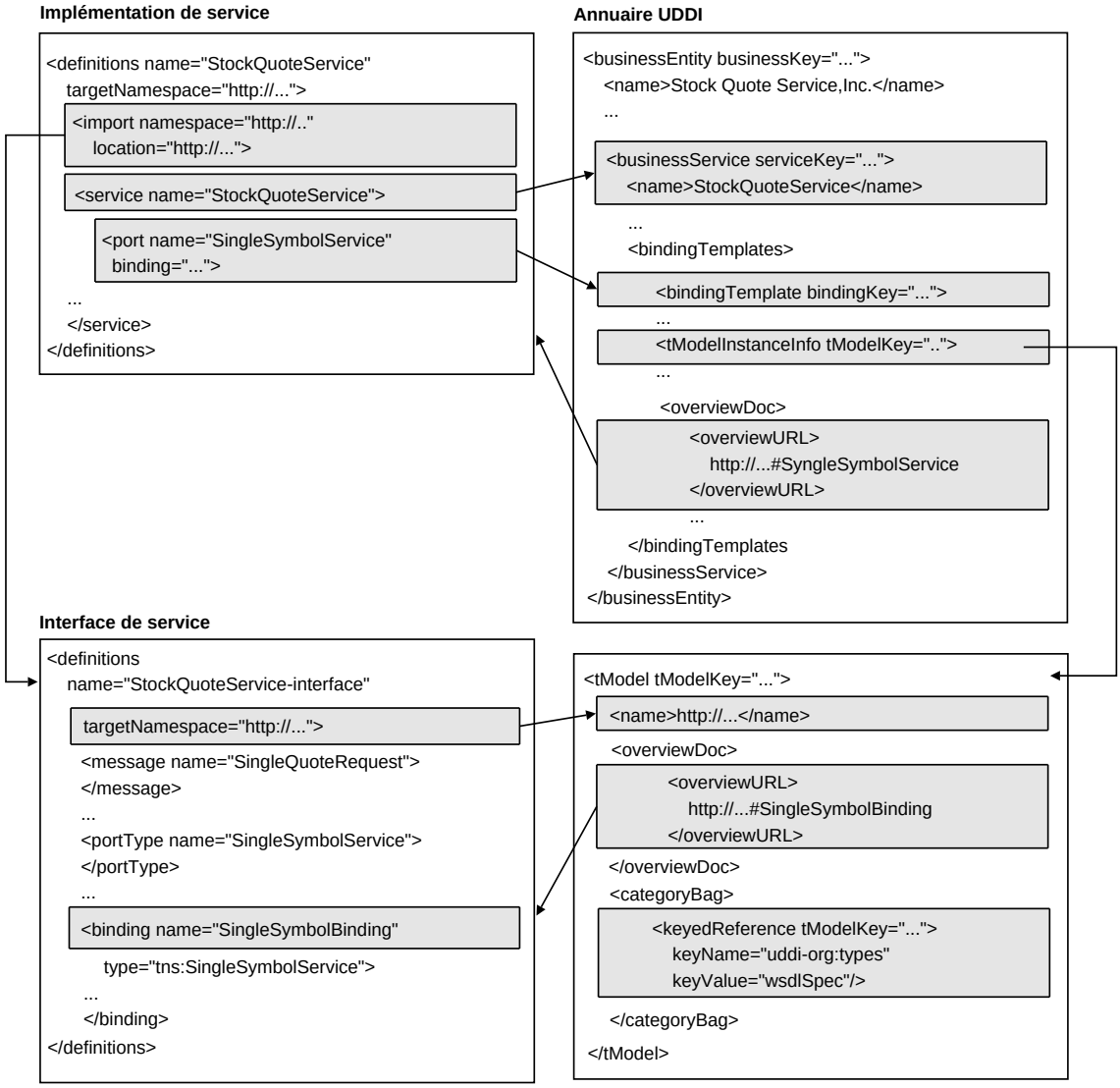

FIG. 3.7 – Résumé du mapping WSDL/UDDI

3.4.3 Conclusion

Dans ce chapitre, nous avons présenté un processus permettant la publication et la recherche des descriptions de services WSDL dans les annuaires UDDI. Pour simplifier

ce processus, la description de service WSDL est divisée en deux parties : l'interface de service et l'implémentation de service. L'interface de service contient la définition abstraite décrivant un type spécifique de service, elle sera utilisée pour implémenter un ou plusieurs services. L'interface de service est publiée dans l'annuaire UDDI en tant que *tModel*, tandis que l'implémentation de service est publiée en tant que*businessService* avec un ou plusieurs *bindingTemplates*.

La recherche se fait en se basant sur l'API UDDI qui définit un ensemble de primitives telles que *find_ tModel* et *find_ service*. Ces primitives sont complétées par d'autres primitives telles que *get_ tModelDetail* et *get_ serviceDetail* qui permettent de donner plus de détails sur les données.

Chapitre 4

Les bases de données XML

4.1 introduction

Vu la popularité de XML, les quantités d'informations stockées et échangées en utilisant XML ne cessent d'augmenter. Le stockage des documents XML sous leurs formes textuelles ralentit beaucoup la recherche et l'extraction de données. Alors, le recours à d'autres techniques plus performantes est inévitable. Pour ce problème de stockage, deux approches sont possibles : la première consiste à étendre un SGBD relationnel avec une couche XML, la deuxième consiste à créer entièrement un SGBD XML natif. Pour les deux approches, il faut choisir un langage de description et un langage de manipulation de données XML :

- Pour la description, on s'appuie en général sur les schémas XML [19] (*XML Schema*).
- Pour la manipulation, plusieurs langages ont été proposés (exemple : XQuery).

Les protocoles de services sont décrits à l'aide de documents XML. Pour cela, nous nous intéressons dans ce chapitre aux mécanismes de stockage des documents XML. Cela va nous aider à choisir un mode de persistance pour nos dépôts de protocoles. Dans un premier temps, nous allons présenter le langage XML, puis les deux approches de stockage de documents XML. Dans un second temps, nous allons présenter le langage de manipulation des documents XML proposé par le W3C (XQuery).

4.2 XML

XML [4] (*Extensible Markup Language*) a connu un succès fulgurant. La simplicité, la souplesse et l'universalité ont largement contribué à promouvoir cette nouvelle technologie issue du vieux standard industriel SGML. Le XML est le standard d'échange de données proposé par le W3C. Il permet d'échanger des informations marquées par des balises décrivant la structure et la sémantique des contenus [22].

XML est un langage de balises. Mais contrairement à HTML, qui est considéré comme un langage défini et figé (avec un nombre de balises limité). Les balises XML sont extensibles, c'est-à-dire que l'on peut toujours ajouter des balises XML. Ainsi, XML peut être considéré comme un métalangage permettant de définir d'autres langages. Grâce à son extensibilité, XML peut décrire n'importe quel domaine de données.

XML est aussi orienté contenu (contrairement à HTML). Il permet de séparer le contenu de la présentation. Ce qui permet par exemple d'afficher un même document sur des applications ou des périphériques différents sans pour autant nécessiter de créer autant de versions du document que l'on nécessite de représentations.

Un document XML est une suite d'éléments correctement imbriqués précédée d'un en-tête. Un élément est une unité d'information encadrée par deux balises (une balise d'ouverture et une balise de fermeture) qui peuvent contenir d'autres balises, du texte ou un ensemble des deux. Chaque élément peut âtre qualifié par un ou plusieurs attributs [22]. Un document XML peut être représenté sous une forme arborescente.

La figure (Fig. 4.1) contient un exemple simple de document XML. Ce document est représenté dans la figure (Fig. 4.2) sous forme d'un arbre.

De plus un document XML peut être associé à une description de sa structure sous la forme d'une DTD (*Document Type Definition*) ou d'un XML *Schema*. Cela permet de créer des familles de documents partageant une structure commune. Pour chaque élément, on définit les éléments qui peuvent être fils de cet élément et les attributs qui peuvent être attachés à cet élément.

La DTD associée au document XML de la figure (Fig. 4.1) est la suivante :

```
!ELEMENT title (#PCDATA)
!ELEMENT para (#PCDATA)
!ELEMENT section(title,(para)+)
!ELEMENT chapter (title,(section)+)
!ELEMENT book(title,(chapter)+)
```

Le W3C définit deux niveaux de contraintes sur un document XML [4] :

```
<book>
    <title>
        Un livre
    </title>
    <chapter>
        <title>
            Un chapitre
        </title>
        <section>
            <title>
                Une section
            </title>
            <para>
                Un paragraphe...
            </para>
        </section>
    </chapter>
</book>
```

FIG. 4.1 – Exemple d'un simple document XML

– Un document est bien formé s'il répond à certaines recommandations sur sa structure. Un document bien formé est constitué d'un élément racine unique, éventuellement précédé d'un en-tête, et incluant d'éventuels éléments imbriqués, chaque élément étant correctement ouvert et fermé.
– Un document est valide si une déclaration de type de document y est associée et si le document est conforme aux contraintes qu'elle exprime.

4.3 Stockage de documents XML

4.3.1 SGBD à support relationnel

Cette approche consiste à stocker les documents XML dans des tables relationnelles et traduire les requêtes XML en des requêtes SQL sur ces tables. L'exécution d'une requête XML (une requête XQuery en général) dans ce type de système se fait de la manière suivante :

1. Transformer la requête XML en une requête SQL.
2. Exécuter la requête SQL générée sur la base relationnelle.
3. Transformer le résultat retourné en un document XML.

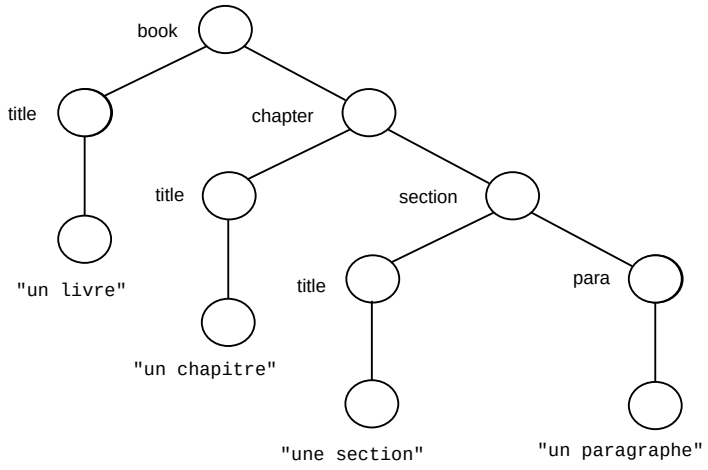

FIG. 4.2 – Exemple d'un arbre XML

La figure (Fig. 4.3) illustre l'architecture d'un SGBD à support relationnel.

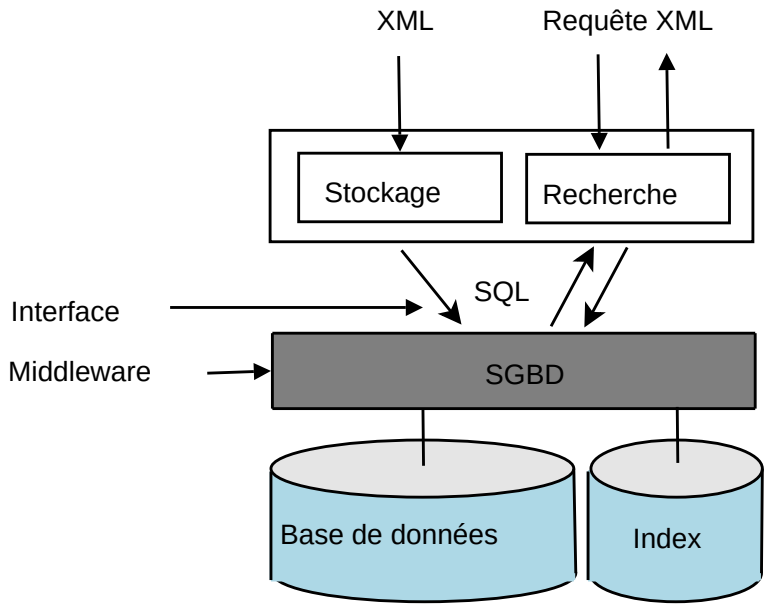

FIG. 4.3 – Architecture d'un SGBD à support relationnel

Cette approche présente les avantages suivants [44] :

– Elle s'appuie sur l'existant et permet aux utilisateurs d'accéder à des données relationnelles et XML dans un environnement intégré. Il n'est pas donc nécessaire d'acquérir et de mettre en oeuvre un nouveau SGBD.
– Elle permet de profiter des avantages et de la puissance de manipulation des données du modèle relationnel.

Globalement, elle propose deux solutions pour stocker les documents XML :

Première solution

Cette méthode consiste à stocker l'intégralité du document dans un attribut de type LOB(Large Object)[1]. Elle permet l'insertion et la récupération rapide d'un document complet (les opérations de jointure ne sont pas nécessaires). L'inconvénient de cette solution, c'est qu'elle ne tire pas profit des informations structurelles fournies par le balisage XML [39]. Le *parsing*[2] du document lors de l'exécution des requêtes influence considérablement les performances. Les opérations d'extraction des fragments des documents et des mises è jours au niveau des sous documents sont coûteuses en temps d'exécution [35].

Deuxième solution

Cette méthode consiste à distribuer les informations contenues dans le document XML sur les colonnes d'une ou de plusieurs tables en préservant les valeurs des données et les relations structurelles. Un *mapping* du Schéma XML vers le relationnel doit être défini. L'efficacité de ce *mapping* dépend en général de la structure des documents, plus la structure est moins régulière et prévisible, plus le *mapping* de ces documents en tables relationnelles devient moins efficace [39]. Plusieurs techniques de *mapping* XML/ relationnel ont été proposées et chaque technique a ses propres choix de conception.

Une fois le document éclaté dans différentes tables, l'accès aux éléments de données stockés comme des valeurs d'attributs relationnels souvent indexés est rapide. Cependant, la méthode présente quelques inconvénients [35] :

- Le *mapping* XML/relationnel est une tâche très compliquée. Le schéma XML peut être constitué de plusieurs éléments imbriqués. Ainsi, le schéma relationnel correspondant peut contenir des dizaines, voire des centaines de tables.
- L'insertion et la reconstruction des documents XML sont coûteuses en temps, surtout lorsque le volume de données à manipuler est important, et dans certains cas des requêtes complexes ne peuvent pas être traduites en requêtes SQL.
- Les changements des schémas relationnels en fonction des changements des schémas XML ne sont pas toujours faciles. Ceci peut restreindre la flexibilité qui représente l'un des principaux points forts de XML.

Parmi les bases de données relationnelles qui supportent XML, on peut citer principalement : Sybase ASE 12.5, Oracle 9i XDB, Microsoft SQL Server 2000, Informix et

[1] Les gros objets (LOB) sont utilisés pour gérer des données de gros volume non structurées(l'image, le sons, la vidéo et le texte.

[2] il s'agit d'une analyse syntaxique destinée à récupérer les informations contenues dans les balises d'un document XML.

DB2 d' IBM.

En conclusion, les bases de données relationnelles offrent un support pour le stockage, la recherche et la manipulation des données XML. Cependant, ces bases qui ne sont pas conçues pour XML présentent certains inconvénients. Ainsi, une technique plus générale de stockage est nécessaire pour tirer profit de l'information structurale offerte par XML.

4.3.2 SGBD XML natif

Les bases de données XML natives sont des bases conçues spécialement pour stocker et interroger des documents XML sans faire subir à ces données des transformations de formats (*mapping*) [39]. Eviter ces transformations peut être bénéfique dans la mesure où lors de l'exécution d'une requête, nous économisons le temps du changement du format de données.

L'idée de base de cette approche consiste donc à stocker les documents XML en conservant leurs structures logiques arborescentes et à leurs adjoindre des accélérateurs d'accès (index) pour améliorer les performances des requêtes [22]. L'évaluation d'une requête se fait directement sur les documents XML de la base de données. Le modèle logique ne spécifie aucune organisation particulière concernant le stockage physique. Différentes techniques de stockage physique peuvent être utilisées. L'efficacité des requêtes et des mises à jour en terme de temps et d'espace dépend de la technique utilisée [39].

La figure (Fig. 4.4) illustre l'architecture d'un SGBD XML natif.

Le fait que le document garde son intégrité facilite sa restitution. En général, les performances en extraction sont bonnes. Les systèmes natifs sont rapides en cherchant des documents entiers mais souvent lents en cherchant des éléments de taille réduite [22]. De plus, la création et la maintenance des index avec une base XML native sont plus complexes qu'avec un système purement relationnel. Ces index doivent pouvoir supporter l'accès aux données quelque soit le niveaux de l'hiérarchie des éléments, et doivent aussi supporter le fait que les objets indexés ont des cardinalités et des types différents. En général, les bases de données XML natives ont créé des nouveaux défis pour tous les aspects d'optimisation des requêtes [39].

La table (Tab. 4.1) contient quelques exemples de base de données XML natives[3] [11]. Dans notre travail, nous nous intéressons uniquement aux produits *open source*.

[3]La base Tamino supporte un lange de requête appelé X-Query, il s'agit d'une extension de XPath, il ne faut pas confondre avec le standard XQuery de W3C.

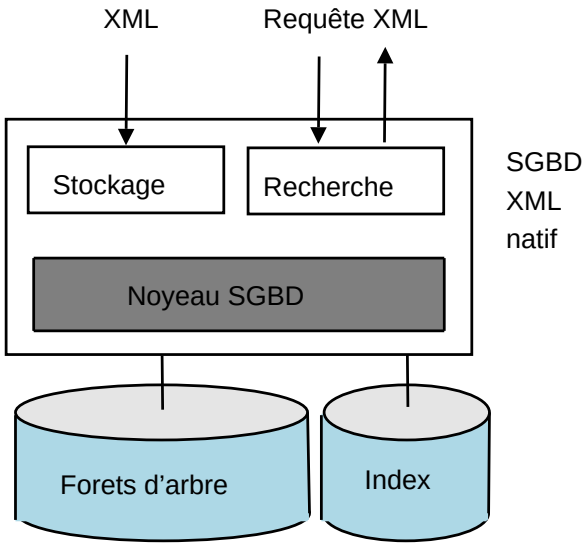

FIG. 4.4 – Architecture d'un SGBD XML natif

4.4 Le langage XQuey

4.4.1 Présentation

L'un des points forts de XML est sa flexibilité en représentant plusieurs types d'informations issues de diverses sources. Pour exploiter cette flexibilité, un langage de requêtes XML doit être capable de fournir des dispositifs pour chercher et interpréter efficacement ces informations. Plusieurs langages ont été développés comme Lorel, Quilt, XPath [16] et XQuery [25]. XQuery est un standard issu du langage Quilt et développé par le W3C dans la perspective de fournir aux utilisateurs une manière efficace d'interrogation et de transformation de données XML.

XQuery est conçu sous forme d'un langage dont les requêtes sont concises et faciles à comprendre. Il est également suffisamment souple pour interroger un large éventail de sources d'informations XML, en particulier les bases de données et les documents. XQuery est un langage fonctionnel, fortement typé et contenant un sous-ensemble considérable de XPath [25].

4.4.2 Expression XPath

Pour adresser les éléments imbriqués dans les arbres XML, XQuery utilise des expressions XPath. Ces expressions sont interprétées comme des requêtes appliquées sur un document ou un ensemble de documents dont le résultat est l'ensemble de sous arbres sélectionnés dans chaque document [22].

Produit	Développeur	Langage de requête	Licence
dbXML	dbXML Group	XPath, XUpdate	Open Source
eXtc	M/Gateway Developments Ltd	XPath	Commerciale
Berkeley DB XML	Sleepycat Software	XPath, XQuery	Open source
eXist	Wolfgang Meier	XPath, XQuery, XUpdate	Open source
DOMSafeXML	Ellipsis	XPath	Commerciale
Tamino	Software AG	X-Query	Commerciale
Xindice	Apache Software Foundation	XPath, XUpdate	Open Source

TAB. 4.1 – Exemple de produits XML natifs

Par exemple, la requête suivante qui porte sur le document *emps.xml* [4] permet de lister les noms de tous les employés :

```
document ("emps.xml")/emps/emp/nom
```

4.4.3 Les expressions FLWOR

Les expressions FLWOR permettent d'effectuer les itérations et la liaison des variables aux résultats intermédiaires. Ces expressions sont souvent utilisées pour réaliser les jointures entre les documents et pour restructurer les données [25]. Le nom FLWOR provient de : ***for***, ***let***, ***where***, ***order by*** et ***return*** (rappelle l'idée du ***select***, ***from***, ***where***, ***group by*** de SQL) :

- **for :** Fournit un mécanisme d'itération.
- **let :** associe le résultat d'une expression à une variable.
- **where :** permet de filtrer les tuples de varriables générés par les clauses *for* et *let* en utilisant une expression logique de prédicats.
- **order :** Permet de trier les résultats.
- **return :** Génère le résultat de l'expression.

[4] *emps.xml* est un document xml contenant des informations sur les employés d'une entreprise, le DTD associé à *emps.xml* est donné dans l'annexe.

Par exemple la requête qui va suivre et qui porte sur les deux documents : *emps.xml* et *depts.xml* [5] permet de donner les numéros des départements dont le nombre d'employés est supérieur à 15 avec le nombre des employés et la moyenne des salaires des employés de chaque département. Les résultats sont ordonnés par ordre décroissant des moyens des salaires.

```
for $d in doc("depts.xml")/depts/deptno
let $e := doc("emps.xml")/emps/emp[deptno = $d]
where count($e) >15
order by avg($e/salary) descending
return
  <big-dept>
  {
    $d,
    <headcount>{count($e)}</headcount>,
    <avgsal>{avg($e/salary)}</avgsal>
  }
  </big-dept>
```

4.5 Conclusion

Dans ce chapitre, nous avons présenté brièvement le métalangage XML et nous avons abordé le problème de stockage des documents XML, nous avons vu que les solutions existantes peuvent être classées en deux catégories : La première consiste à utiliser les bases de données relationnelles, cette solution est motivée par la réutilisation des techniques de stockage et d'interrogation efficaces développées pour le modèle relationnel, ainsi que l'enjeu économique en raison de la nature relationnelle de la majorité des bases de données existantes. Cependant, le fait que ces bases ne sont pas conçues pour XML pose certains problèmes et par conséquent une deuxième solution est proposée. Cette dernière consiste à créer entièrement des bases de données XML natives. Chacune de ces deux solutions a ses avantages et ses inconvénients et par consequent le choix d'une solution n'est pas toujours facile et dépend de l'application. La manipulation de données XML se fait en utilisant un langage. Ainsi, nous avons présenté le XQuery qui est le standard de manipulation de données XML proposé par le W3C. XQuery utilise les expressions XPath pour adresser des éléments imbriqués dans la structure d'un document XML.

[5] *depts.xml* est un document xml contenant des informations sur les département d'une entreprise, le DTD associé à *depts.xml* est donné dans l'annexe.

Chapitre 5

Etude expérimentale et implémentation

5.1 Introduction

Dans le chapitre précédent, nous avons présenté les deux approches de stockage des documents XML. Notre objectif est de choisir une base de données permettant l'interrogation des protocoles. Pour cela, nous allons procéder à une étude comparative d'un ensemble de base de données pour justifier le choix du mode de persistance. Ce choix se fait sur la base de critères tels que la facilité de maintenance et le passage à l'échelle, etc.

Ce chapitre comporte deux sections principales : La première section est consacrée à l'étude de persistance, nous allons en premier temps présenter les trois bases de données étudiées, ainsi que les critères de choix. Puis, nous allons présenter les tests que nous avons développés. Ensuite, nous exhibons les différents résultats obtenus avec les interprétations élaborées. La deuxième section est consacrée à l'implémentation, nous décrivons les différentes classes constituant la bibliothèque de dépôts, ainsi que les deux interfaces de dépôt (service et application web).

5.2 Etude de persistance

5.2.1 Critères de choix

Le choix doit se faire sur la base de critères tels que :

- **facilité de maintenance :** les opérations de maintenance, telle que le passage à une version plus évoluée, doivent être transparentes pour les utilisateurs.
- **possibilité d'évolution :** l'évolution de la structure interne de la base de données utilisée ne doit pas être limitée. Par exemple dans le cas d'une base de données relationnelle, on doit pouvoir ajouter de nouveaux attributs sans causer de conflits. Dans une base XML, l'ajout de nouveaux noeuds dans la représentation arborescente du fichier doit être possible afin de prendre en compte tous les modèles possibles et envisageables pour un protocole de service.
- **passage à l'échelle :** le dépôt doit offrir d'excellents paramètres de performance, même lorsqu'il est soumis à un très grand nombre de requêtes. Le passage d'une période de faible intensité de requêtes à celle d'une grande intensité ne doit pas influencer considérablement les performances de la base de données.
- **espace disque occupé :** la base de données servant de support physique pour le dépôt doit faire preuve d'économie de l'espace disque, cela est nécessaire pour des raisons de performances. On essaie de minimiser au maximum le nombre d'accès au disque.

5.2.2 Bases de données testées

Dans le chapitre précédent, nous avons cité un ensemble de base de données. Dans notre travail, nous allons uniquement étudier trois bases. Comme représentant des bases de données relationnelles, nous avons opté pour MySQL qui est très performante et très connue, et en plus, elle est distribuée gratuitement sous forme d'un produit *open source*. Pour le stockage natif, nous avons opté pour Xindice et Berkeley, ces deux bases sont aussi des produits *open source* et distribués gratuitement.

5.2.2.1 MySQL

MySQL [1] est une base de données relationnelle très populaire et très utilisée dans le Web. MySQL est un produit *open source* développé, distribué et supporté par *MySQL AB*[2]. Le serveur de bases de données MySQL est très rapide, fiable et facile à utiliser. La

[1] http ://www.mysql.com/

[2] MySQL AB est une société commerciale, fondée par les développeurs de MySQL, qui développent leur activité en fournissant des services autour de MySQL.

base de données MySQL est accessible en utilisant plusieurs langages de programmation tels que Java, C, C++, Delphi, Perl, PHP, etc. Des API spécifiques sont disponibles pour ces langages.

MySQL n'offre pas un support direct pour le stockage et la manipulation des données XML. La solution que nous avons utilisée dans notre travail consiste à stocker l'intégralité du document XML dans un seul attribut de type LONG TEXT.

5.2.2.2 Xindice

Xindice [3] est une base de données XML native *open source* qui fait partie du projet Apache XML. Xindice utilise le langage XPath et XUpdate pour effectuer les requêtes et les mises à jour. Xindice permet l'indexation des éléments et des attributs d'un document XML.

dans une base de données Xindice, les données sont stockées sous une forme arborescente. Cette arborescence est constituée d'un ensemble de collections équivalentes à des répertoires où l'on peut placer des documents XML, une collection peut contenir plusieurs documents. L'exécution des requêtes XPath se fait sur les collections contenant les documents, que l'on veut utiliser. L'administration de la base se fait en ligne de commande et permet de visualiser et de changer la configuration de la base. Pour le développement, on utilise l'API Java fournie par les développeurs de la base. Dans ce projet, nous allons travailler avec la version 1.0 de Xindice qui est écrite entièrement en Java.

5.2.2.3 Berkeley

Berkeley DB XML [4] (BDB XML) est une base de données XML embarquée développée par *Sleepycat Software*. Elle est conçue pour stocker les documents XML sous leurs formats natifs. Berkeley DB XML est implémentée comme une bibliothèque C++ au dessus de la base Berkeley classique (voir FIG. 5.1). La bibliothèque BDB XML est constituée de trois composantes principales : Le gestionnaire de stockage de documents, le gestionnaire des indexes et un moteur de traitement de requêtes. BDB XML expose ses fonctionnalités via des interfaces de programmation (API) permettant de gérer, d'interroger et de modifier les documents XML. Ils existent des APIs pour des langages de programmation tels que Java, C/C++, Perl, Python, etc. DB XML supporte le langage XQuery et XPath, et les schémas XML et permet le *parssing* et l'indexation des documents XML.

[3] http ://xml.apache.org/xindice/
[4] http ://sleepycat2.inetu.net/products/bdbxml.html

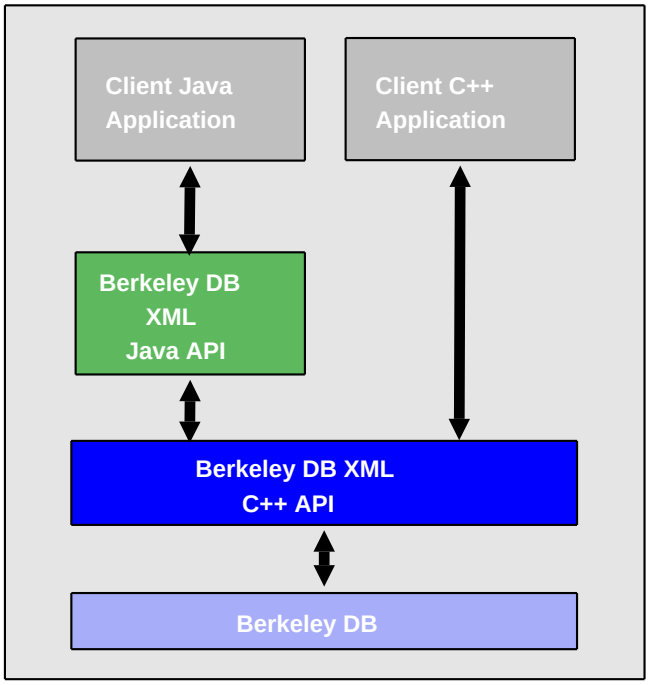

FIG. 5.1 – Architecture de BDB XML

BDB XML n'est pas un système de gestion de base de données client/serveur, mais une bibliothèque liée à l'application qui l'utilise et s'exécute dans son espace d'adresse. Cela permet d'éviter la surcharge de la base impliquée par les implémentations client/-serveur, et de faciliter le déploiement et l'administration de la base, l'application peut créer et utiliser directement la base de donnée Berkeley, sans nécessiter l'utilisation ou l'administration d'aucun serveur. Toutes les tâches administratives sont assurées par le développeur via des APIs. L'utilisation de Berkeley est transparente pour les utilisateurs.

A l'intérieur de BDB XML, un document XML est stocké dans un fichier appelé conteneur (*container*) selon deux manières :

- L'intégralité du document est stockée dans un seul noeud du conteneur. Cette méthode est généralement utilisée avec les documents de petite taille (inférieure à 1 Mo).
- Les documents sont décomposés automatiquement en un ensemble de noeuds dont chacun de ces noeuds sera stocké dans un enregistrement individuel dans le conteneur. Lors d'une requête, les noeuds qui constituent le même document sont rassemblés pour donner le même résultat que si le document a été stocké dans son intégralité. Tout ce processus se fait d'une manière transparente pour le programmeur. Cette méthode est utilisée avec les documents de grande taille. Elle a l'avantage d'éviter les limites sur la taille des documents pouvant être stockés

dans la base.

De plus, BDB XML peut traiter jusqu'à 250 To de données, ce qui est largement suffisant pour satisfaire les besoins de notre projet.

5.2.3 Presentation des tests

Afin de mesurer et comparer les performances des SGBD XML, plusieurs benchmarks ont été proposés (XMark [40] , X007 [12], XBench [45]. Chacun de ces benchmarks propose un ensemble de données et un ensemble de requêtes en essayant d'être aussi général et complet que possible.

Le choix de l'ensemble de données et des requêtes est crucial et peut avoir un impact significatif sur la qualité des tests. Pour cette raison et pour des raisons de simplicité, nous avons développé des tests spécifiques à notre application au lieu d'utiliser des solutions générales et complexes proposées par les benchmarks existants.

Les tests sont effectués en utilisant un programme Java qui se connecte aux bases de données testées via leurs APIs, exécute des requêtes de tests et calcule le temps d'exécution de chaque requête. Pour éviter tout influence sur le temps d'exécution, nous avons pris le soin de laisser uniquement ce programme en exécution sur la machine. De plus, on exécute une requête plusieurs fois, puis on prend la moyenne. Les tests ont été effectués sur une machine PC avec un système Windows XP et un processeur Pentium 4 de vitesse d'horloge de 1.7 GHz et de mémoire RAM de 256 Mo.

5.2.3.1 Description des données

Pour que le test à effectuer soit de bonne qualité, il faut tester la base avec des données de tailles considérables. Pour cela, un générateur de document XML est utilisé. Ce dernier permet de générer selon un schéma des documents XML de tailles variables. Le schéma utilisé dans ce travail correspond à une représentation possible des protocoles de services. La représentation graphique de ce schéma est donnée par la figure (Fig. 5.2).

5.2.3.2 Description des requêtes

Notre objectif derrière ces tests, n'est pas de choisir la meilleure base de données, mais c'est de choisir une base qui fonctionne mieux avec nos dépôts. Ainsi, le choix des requêtes de test est fait en tenant compte des fonctionnalités offertes par le dépôt. Deux types de requêtes sont choisis : les requêtes d'ajouts des documents et les requêtes de recherches.

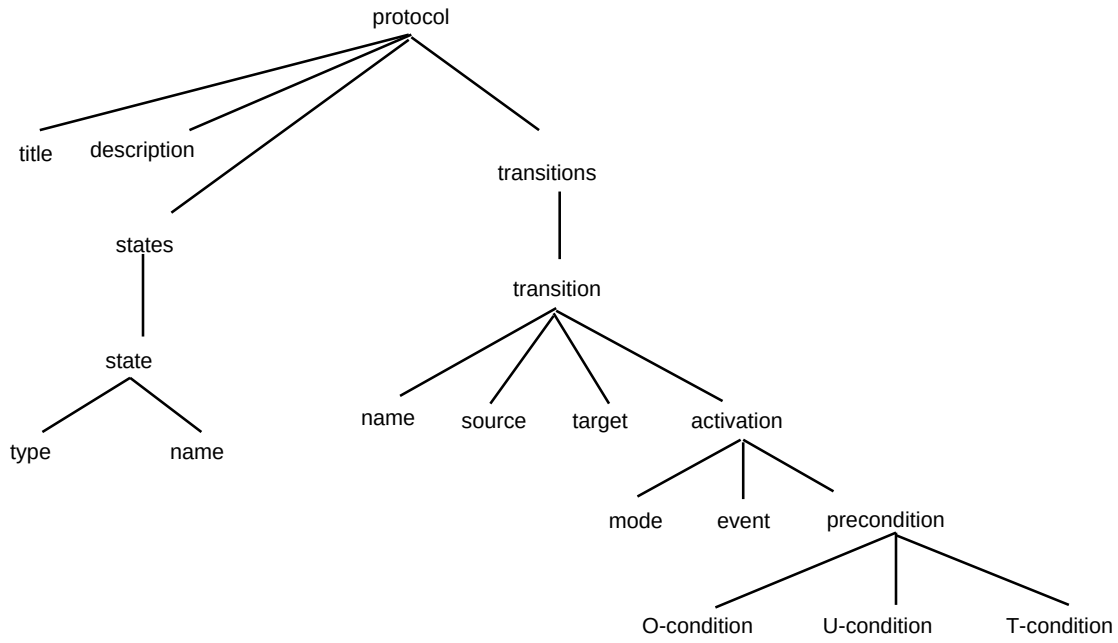

FIG. 5.2 – Arbre représentant la structure d'un protocol de service

5.2.4 Résultats et Interprétation

5.2.4.1 Test d'ajout

Ce test consiste à calculer le temps moyen nécessaire pour insérer un document dans chacune des trois bases. La courbe de la figure (Fig. 5.3) représente l'évolution du temps de réponse moyen(on fait l'exécution de chaque requête 10 fois puis on prend la moyenne) en fonction de la taille du document. L'axe horizontal représente la taille du document en Ko, l'axe vertical représente le temps d'exécution en milliseconds.

A partir du graphe, nous remarquons que les performances d'ajout de la base MySQL sont les meilleures avec les documents dont la taille ne dépassant pas 1024 Ko. Les limitations de MySQL portent sur la taille maximale du document pouvant être inséré.

Pour les deux autres bases, nous pouvons remarquer que les performances d'insertion de Berkeley sont meilleures que celles de Xindice. De plus, la taille des documents pouvant être insérés dans la base Xindice est limitée à 5 Mo (approximativement). Avec Berkeley, le problème de la taille des documents ne se pose pas, les documents avec les tailles supérieures à 1 Mo sont automatiquement décomposés avant d'être ajoutés à la base.

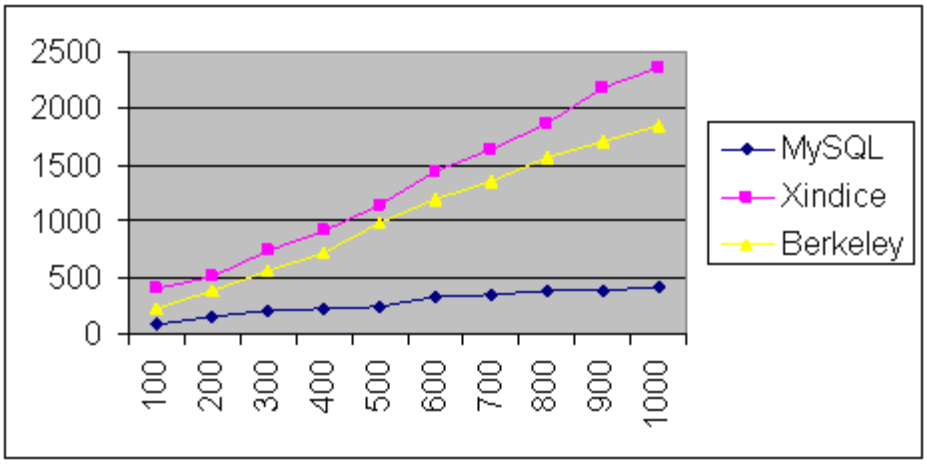

FIG. 5.3 – Résultat du test d'ajout

5.2.4.2 Test de recherche

Ce test consiste à calculer le temps moyen nécessaire pour trouver un document. Nous avons effectué deux types de recherche. Le premier consiste à chercher le document en utilisant son identificateur. Pour la base MySQL, cela se fait en ajoutant un nouveau champ indexé (*identificateur*) dans la table. Pour la base Xindice, l'API offre la possibilité de donner un identificateur à un document au moment de l'insertion et d'effectuer la recherche en utilisant cet identificateur. Pour la base Berkeley, on fait la recherche en utilisant le nom du fichier qui est stocké au moment de l'insertion du document comme une méta-donnée, la recherche se fait dans ce cas en utilisant une requête XQuery.

Le deuxième type de recherche consiste à effectuer la sélection d'un document en utilisant une expression XPath permettant de parcourir la structure du document en cherchant une information particulière. Voici l'expression XPath que nous avons utilisée. Elle permet de trouver le document dont l'attribut *title* de l'élément *protocol* est égal à une valeur particulière.

```
/protocol[@title='titre d'un protocole particulier']
```

Ce deuxième type de recherche est effectué uniquement pour les deux bases Xindice et Berkeley. La base MySQL ne supporte pas les expressions XPath et le langage XQuery.

Pour étudier l'impact de la taille de l'ensemble de données sur les performances de chaque base, nous avons effectué les tests sur quatre ensembles différents de données (Set1, Set2, Set3 et Set4). Pour la taille du document, nous avons choisi une taille moyenne de 64 Ko.

– Set 1 : la taille de la base est de 10 Mo.
– Set 2 : la taille de la base est de 100 Mo.
– Set 3 : la taille de la base est de 500 Mo.
– Set 4 : la taille de la base est de 1000 Mo.

Les résultats obtenus sont résumés dans les tables (Tab. 5.1) et (Tab. 5.2) et représentés graphiquement dans les figures (Fig. 5.4) et (Fig. 5.5). Les temps de réponse sont mesurés en millisecondes. De plus, on fait l'exécution de chaque requête 100 fois puis on prend la moyenne.

Premier type de recherche

Ce test montre que l'exécution de la requête de recherche avec Berkeley donne le meilleur temps de réponse lorsque la base de données contient une petite quantité de données. Cependant, lorsque la taille de l'ensemble de données augmente, le temps de réponse augmente proportionnellement avec la taille de ces données. Ainsi, les performances de MySQL deviennent meilleures. Mais les performances de Berkeley restent en général bonnes. Pour Xindice, nous pouvons bien remarquer qu'elle est la moins rapide, elle est environs 10 fois mois rapide que MySQL.

	MySQL	**Xindice**	**Berkeley**
Set1	27,162	295,95	22,031
Set2	27,455	315,99	34,541
Set3	31,131	321,62	75,959
Set3	32,566	323,67	85,561

TAB. 5.1 – Résultat du premier type de test de recherche

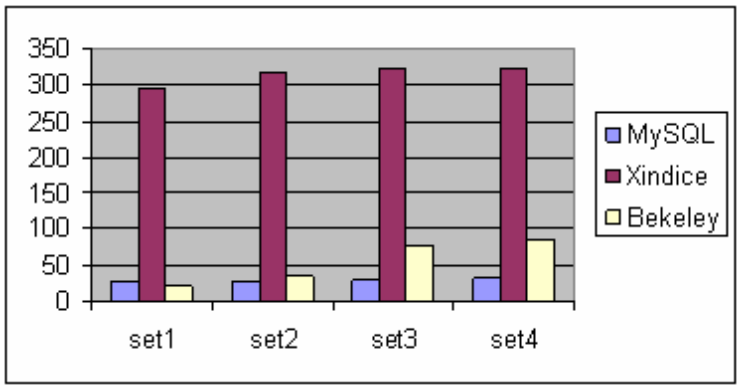

FIG. 5.4 – Résultat du premier type de test de recherche

Deuxième type de recherche

Ce test montre que le temps de réponse avec Berkeley est bien meilleur que celui de Xindice surtout lorsque la taille de l'ensemble de données n'est pas très grande. L'avantage de Berkeley sur Xindice en exécutant des requêtes XQuery est bien clair. Cependant, lorsque la taille de données augmente, les deux bases présentent une dégradation considérable de performances. Nous remarquons aussi que ce type de recherche est moins rapide que le précédent. Cela est évident, car l'exécution des requêtes XQuery doit prendre en considération la structure du document en cherchant un noeud particulier, ce qui demande plus d'exécution. L'utilisation des index permet d'améliorer considérablement le temps de réponse.

	Xindice	**Berkeley**
Set1	786,71	71,4
Set2	4192,8	696,34
Set3	29827	25530
Set3	62482	54325

TAB. 5.2 – Résultat du deuxième type de test de recherche

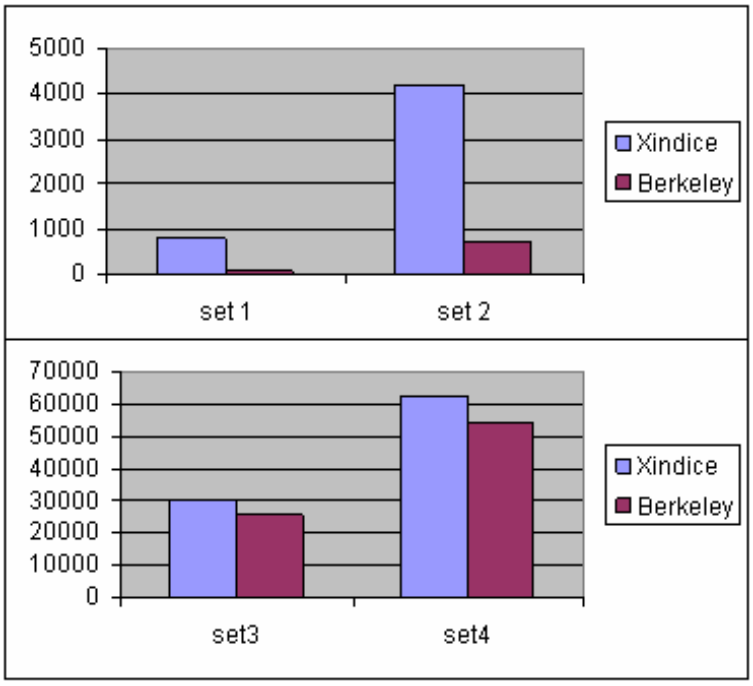

FIG. 5.5 – Résultat du deuxième type de test de recherche

5.2.5 Résumé des résultats de l'étude de persistance

Nous allons par la suite résumer les principales conclusions de l'étude de persistance.

MySQL offre de très bonnes performances lors de traitement d'un document entier (requêtes d'ajout, de suppression ou de recherche). Cependant, MySQL ne peut pas tirer profits de l'information structurelle de XML, un document XML est considéré comme un texte simple. Cela pose certains problèmes lors de traitement des requêtes qui portent sur les fragments des documents. Par exemple, une requête sur un seul noeud (recherche ou mise à jour) nécessite le traitement du document entier, ce qui peut entraîner des dégradations considérables de performances. Cela ne simplifie pas aussi l'évolution de la structure des documents (la modification de la structure du document nécessite le remplacement du document entier). De plus, la taille du document pouvant être stocké dans un seul attribut est limitée à 1024 ko. Ainsi, la base MySQL ne satisfait pas tous les critères de choix cités précédemment et par conséquent elle ne sera pas choisie pour assurer la persistance dans les dépôts.

Les tests montrent que les performances de Xindice ne sont pas intéressantes dans le cas de notre travail, si on les compare avec celles de BDB XML. Cette dernière présente des nombreuses fonctionnalités très intéressantes. Elle ne présente pas de limites sur la taille des documents XML et peut traiter efficacement des quantités de données considérables. Ainsi, les dépôts de protocoles peuvent fonctionner à grande échelle sans causer une dégradation considérable des performances.

Le fait que BDB XML stocke les documents sous leur format natif permet de profiter de la flexibilité de XML. Avec BDB XML, il est possible d'exécuter directement des requêtes sur des noeuds quelque soit leurs emplacements dans la structure du document XML. On peut aussi facilement changer la structure d'un document en ajoutant des nouveaux noeuds ou bien en supprimant/modifiant des noeuds existants. Cela simplifie grandement l'évolution des dépôts. BDB XML nous permet aussi de créer et de gérer facilement des index sur des noeuds des documents, ce qui permet d'améliorer considérablement le temps d'exécution des requêtes.

Enfin, les fonctionnalités offertes par la base de données BDB XML sont intéressantes pour notre travail et satisfont les critères de choix du mode de persistance. Ainsi, c'est DBD XML que nous avons choisie pour assurer le stockage des données au niveau de nos dépôts de protocoles.

5.3 implémentation

Le travail d'implémentation consiste à :

– Implémenter les différentes classes qui constituent la bibliothèque Java.

– Exposer la bibliothèque en tant que service web puis application web.

L'exposition de la bibliothèque en tant que service et application web ne fait pas l'objet de ce présent travail. De plus, il faut ajouter un autre service supplémentaire permettant d'effectuer des études sur les protocoles de services (analyse de compatibilité et de replaçabilité).

Avant de passer à l'implémentation de la bibliothèque Java, il est nécessaire de présenter quelques détails sur la manière de stockage des documents XML dans une base Berkeley XML. Cela permet de comprendre les choix que nous allons faire pour ce qui concerne les différentes classes qui constituent la bibliothèque Java.

5.3.1 Stockage des documents XML

Pour chaque document placé dans un conteneur, le conteneur contient toutes les données du document et tous ses index et ses méta-données. L'application cliente peut effectuer des opérations sur plusieurs conteneurs simultanément. Elle peut aussi stocker les données dans la base Berkeley classique.

De façon plus précise, une application cliente peut effectuer les opérations suivantes sur un conteneur :

– Ouvrir un conteneur pour l'utiliser à l'intérieur de l'application.
– Renommer un conteneur.
– Fermer un conteneur.
– Supprimer un conteneur.
– Exporter un conteneur sous forme de fichier texte.
– Charger un conteneur à partir d'un fichier texte.
– Vérifier que le conteneur possède une structure consistante.
– Déclarer un index sur un conteneur.
– Insérer un document dans un conteneur.
– Obtenir un document à partir d'un conteneur.
– Mettre à jour et modifier un document dans le conteneur.
– Interroger un conteneur en utilisant XQuery ou XPath.
– Supprimer un document du conteneur.
– Renommer un document du conteneur.

Le conteneur est géré par l'objet *XmlManager*. Ce dernier est un élément essentiel pour BDB XML, il permet de gérer les objets les plus importants dans les applications BDB XML. De façon plus précise, il permet de :

– Gérer les conteneurs (créer, ouvrir, supprimer et renommer).
– Créer les objets *InputStreams* qui sont utilisés pour charger les documents XML dans les conteneurs.

- Créer les objets : *XmlDocument*, *XmlQueryContext* et *XmlUpdateContext* qui représentent respectivement un document XML, un contexte de requête et un contexte de mise à jour.
- Préparer et exécuter les requêtes XQuery.
- Créer les objets *transactions* pour gérer les transactions.

De plus, BDB XML nécessite l'utilisation d'un environnement pour la base de données. Avant d'instancier un objet *XmlManager*, il faut prendre quelques décisions concernant cet environnement : L'environnement peut être géré d'une manière implicite par le constructeur de l'objet *XmlManager* ou bien d'une manière explicite. Dans le premier cas, l'environnement est configuré d'une manière automatique. Dans le deuxième cas, on peut spécifier une configuration spéciale pour l'environnement. Par exemple, on peut configurer l'environnement à supporter la gestion des transactions, d'authentification, ou bien les applications multi processus ou multi-thread. De plus, on peut choisir les emplacements où on peut stocker les conteneurs de l'application.

Avant d'utiliser un environnement, il faut d'abord l'ouvrir et identifier le répertoire dans lequel il réside. Un environnement peut contenir plusieurs conteneurs et possède une configuration spécifique.

La figure (Fig. 5.6) résume la structure de stockage de BDB XML. La base de données peut contenir un ou plusieurs environnements, chaque environnement est caractérisé par une configuration particulière et un chemin d'accès (emplacement de stockage de données) particulier et peut contenir un ou plusieurs conteneurs. Un conteneur sert à stocker les documents XML, ainsi que les index et toutes les méta-données de ces documents. Un document est stocké dans un conteneur selon deux manières : document composé ou bien document intégral.

5.3.2 Implémentation de la bibliothèque

Dans cette section, nous présentons les différentes classes Java qui constituent la bibliothèque de dépôt. La bibliothèque est conçue pour assurer un ensemble d'opérations de base d'entrée/sortie, telles que :

- L'ajout d'un nouveau protocole.
- La suppression d'un protocole.
- La modification d'un protocole.
- La recherche d'un protocole particulier.

La gestion des index permet une amélioration considérable des performances de stockage. Ainsi, la bibliothèque doit offrir un ensemble d'opérations de gestion des index (eg,. création et suppression des index d'un noeud particulier). Il faut en plus permettre l'exploitation des informations stockées comme des méta-données en permettant des

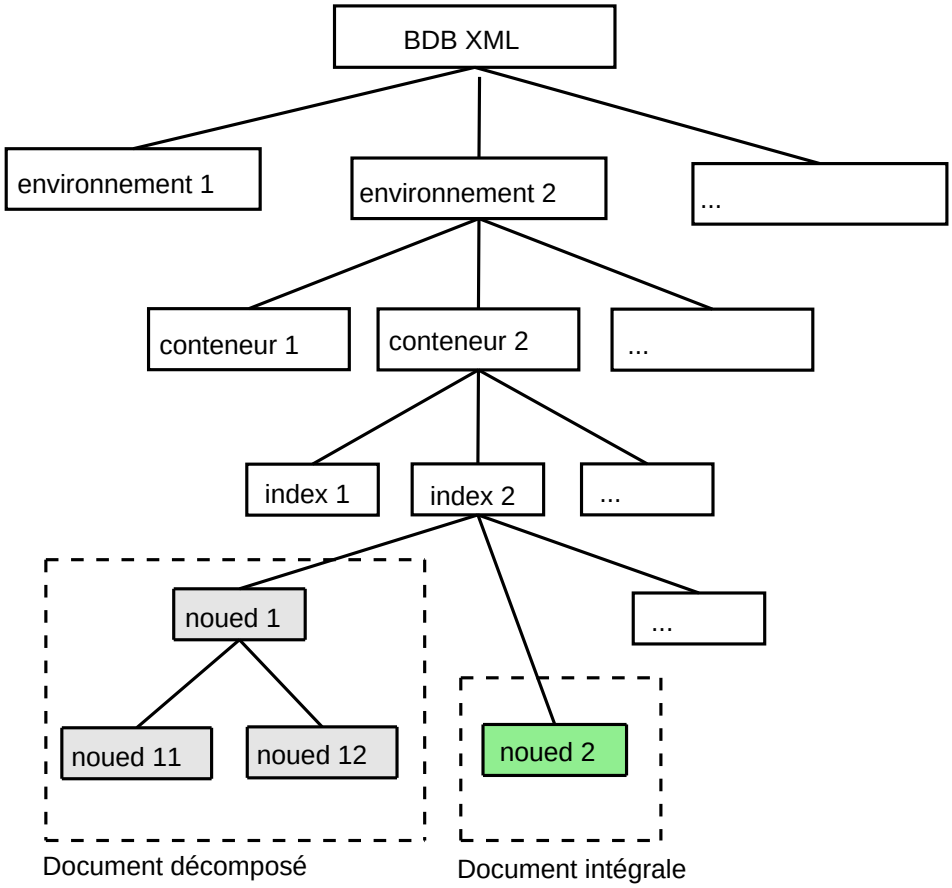

FIG. 5.6 – Structure de stockage de Berkeley

requêtes qui portent sur ces méta-données (eg,. rechercher la date et l'heure de l'insertion d'un protocole).

Pour assurer ces fonctionnalités, et après avoir présenté les fonctionnalités de stockage offertes par la base de données BDB XML, nous avons proposé un ensemble de classes dont la représentation UML est donnée par le diagramme de la figure (Fig. 5.7)

- **MyDbEnv**
  Cette classe est indispensable pour le fonctionnement des autres classes. Elle permet d'ouvrir un environnement existant et de créer un nouvel environnement en spécifiant une configuration spéciale pour cet environnement. Avant d'utiliser cette classe, il faut spécifier l'emplacement de l'environnement. La classe *MyDbEnv* permet en plus d'instancier l'objet *XmlManager* et de créer des nouveaux conteneurs.
- **QueryContext** Cette classe permet de créer un contexte de requête pour un conteneur. Cela permet d'exécuter des requêtes XQuery sur ce conteneur et d'afficher les résultats de ces requêtes. Cette classe utilise la classe *MyDbEnv* pour

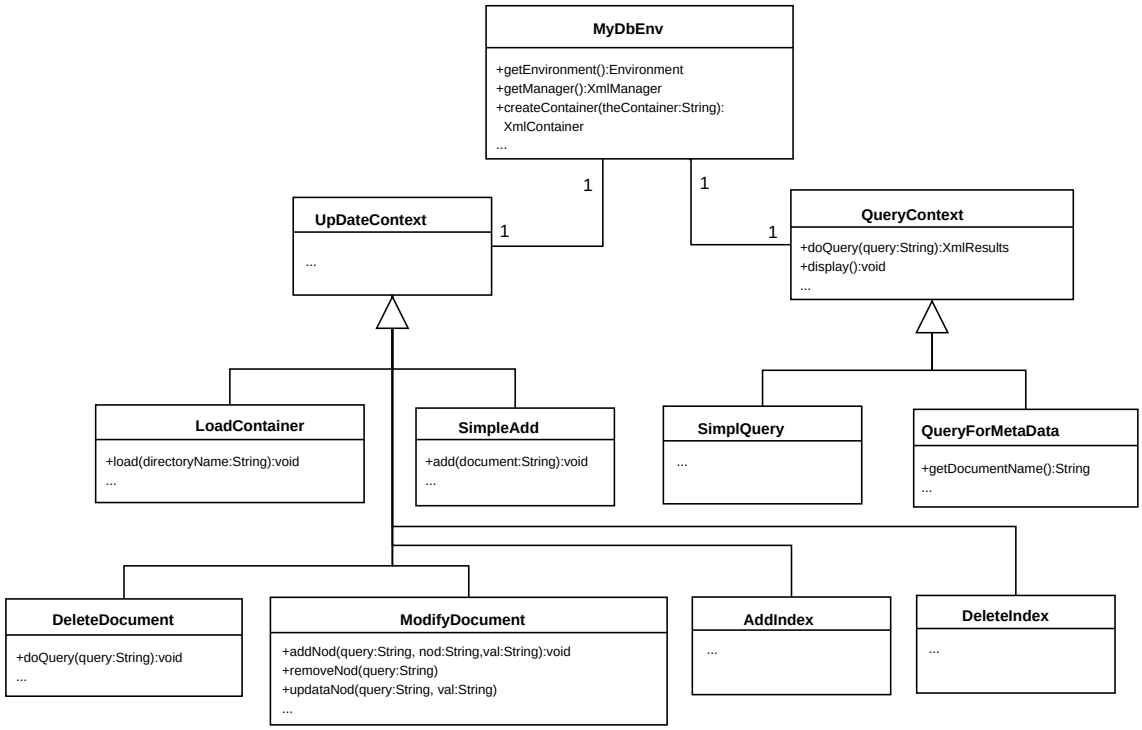

FIG. 5.7 – Diagramme de classes de la bibliothèque de dépôt

créer/ouvrir l'environnement de la base et pour instancier l'objet *XmlManager*.

- **SimpleQuery** Cette classe hérite de la classe *QueryContext* et permet d'exécuter une requête XQuery.
- **QueryForMetaData** Cette classe hérite de la classe *QueryContext* et permet de chercher les méta-données d'un document XML (nom du document, la date et l'heure d'insertion de ce document).
- **UpdateContext** Cette classe permet de créer un contexte de mise à jour pour un conteneur. Cela permet d'exécuter des requêtes de mises à jour sur ce conteneur. Cette classe utilise la classe *MyDbEnv* pour créer/ouvrir l'environnement de la base et pour instancier l'objet *XmlManager*.
- **LoadContainer** Cette classe hérite de la classe *UpdateContext* et permet de charger dans un conteneur tous les documents XML se trouvant dans un chemin d'accès spécifié. Pendant le chargement de chaque document, on lui attribue un nom qui sera stocké avec la date et l'heure d'insertion comme des méta-données.
- **SimpleAdd** Cette classe hérite de la classe *UpdateContext* et permet d'insérer un document XML donné sous forme d'une chaîne de caractère dans un conteneur.

  Les deux classes *SimpleAdd* et *LoadContainer* vérifient automatiquement si les

documents à insérer sont bien formés ou non et génèrent des messages d'erreurs lorsqu'on veut insérer un document malformé.

- **DeleteDocument** Cette classe hérite de la classe *UpdateContext* et permet de supprimer des documents XML à partir d'un conteneur. Pour choisir les documents à supprimer, on utilise des requêtes XQuery.
- **ModifyDocument** Cette classe hérite de la classe *UpdateContext* et permet de modifier un document retourné par une requête XQuery. Cette classe possède trois méthodes :
  - *addNod* : permet d'ajouter un nouveau noeud au document XML.
  - *removeNod* : permet de supprimer un noeud du document.
  - *updateNode* : permet de modifier la valeur d'un noeud.
- **AddIndex** Cette classe hérite de la classe *UpdateContext* et permet de créer un index sur un noeud particulier.
- **DeleteIndex** Cette classe hérite de la classe *UpdateContext* et permet de supprimer un index d'un noeud particulier.

5.3.3 Exposition de dépôts

5.3.3.1 Service web

Le service web offre l'ensemble de fonctionnalités suivantes :
- Recherche des protocoles.
- Ajout des protocoles.
- Suppression des protocoles.
- Modification des protocoles.

La figure (Fig. 5.8) montre les interactions entre le client et le dépôt à travers l'interface de service web.

Lorsque le client veut effectuer une opération sur des protocoles, il envoie au service web une requête encodée en SOAP. Le service web décode la requête puis invoque la méthode correspondante de la bibliothèque de dépôt. A son tour, la bibliothèque interroge la base de données BDB XML pour trouver la donnée demandée. Enfin, le service renvoie la réponse donnée par la bibliothèque au client sous forme d'un message SOAP.

5.3.3.2 Application web

L'application web offre les mêmes fonctionnalités que celles offertes par le service web (ajout, recherche et suppression/modification de protocoles). Cependant, il faut

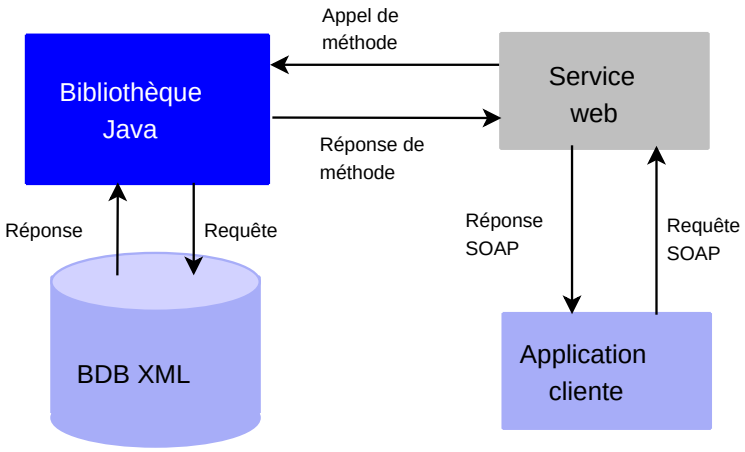

FIG. 5.8 – Interactions entre le client et le dépôt

adapter l'application aux utilisateurs humains. Par exemple, les documents XML sont difficilement lisibles pour un humain. Ainsi, le développement d'une interface graphique serait très utile pour simplifier l'utilisation de l'application.

5.4 Conclusion

Nous avons dans ce chapitre procédé à une étude comparative de trois bases de données afin de justifier le choix du mode de persistance de nos dépôts de protocoles. Nous avons étudié une base de données relationnelle (MySQL) et deux bases de données XML natives (Berkeley et Xindice). Cette étude a impliqué le développement d'un ensemble de tests. Ces tests proposent un ensemble de données et un ensemble de requêtes : Les données de tests correspondent à une représentation XML possible des protocoles de services. Deux types de requêtes sont utilisés : les requêtes d'ajout et les requêtes de recherche. L'étude a montré que les fonctionnalités offertes par la base Berkeley XML sont les plus intéressantes dans le cas de notre travail. Ainsi, nous avons opté pour la base Berkeley et pour le stockage natif.

Nous avons aussi implémenté un ensemble de classes Java constituant la bibliothèque de dépôt. Cette dernière offre un ensemble de fonctions telles que l'ajout/la suppression, la recherche et la modification des protocoles, ainsi que la gestion (création/suppression) des index sur des noeuds particuliers. Cette bibliothèque sera exposée en tant que service web puis application web pour permettre aux utilisateurs d'interroger le dépôt. Il est à noter que l'implémentation ne représente pas une version finale. En effet, la bibliothèque peut être enrichie pour prendre en considération d'autres fonctionnalités.

Conclusion générale et Perspectives

LES services web représentent actuellement la technologie de choix pour le développement et l'intégration des applications, surtout dans le cadre d'un environnement ouvert tel que Internet. Cependant, cette technologie relativement récente n'a pas encore tenu toutes ses promesses d'une intégration simple et à faible coût. La simplification de l'intégration des services web fait actuellement l'objet de nombreux travaux de recherche. Plus récemment, des propositions ont été faites pour ajouter plus d'informations dans la description des services, en particulier, les protocoles de services web (*business protocols*). Les protocoles de services web sont des protocoles de communication de haut niveau qui permettent de décrire un ordre sur les messages échangés entre les services web. Une fois utilisés, ils permettent de simplifier considérablement le cycle de vie entier des services web.

La principale motivation qui a régi ce travail est la création de dépôts de protocoles. L'intérêt de ces dépôts est de pouvoir effectuer des études de nature variée sur les protocoles de services web. En premier temps, nous avons décrit un processus permettant de publier la description WSDL de services web dans un annuaire UDDI et de la retrouver au moment de la recherche. En second temps, nous nous sommes intéressé à la persistance dans les dépôts. Les protocoles de service sont décrits à l'aide des documents XML. Ainsi, nous avons effectué une étude sur le stockage de XML, nous avons vu qu'ils existent deux approches : La première consiste à utiliser les bases de données relationnelles, tandis que la deuxième consiste à créer entièrement des bases de données XML natives. Chacune de ces deux approches a ses avantages et ses inconvénients. Le choix de l'une de ces approches n'est pas facile et dépend de l'application. Pour justifier le choix du mode de persistance, nous avons procédé à une étude comparative d'un ensemble de base de données.

Après avoir effectué notre étude, nous avons opté pour le stockage natif et la base de donnés Berkeley. Le résultat final est que le dépôt est capable de répondre aux exigences des utilisateurs en terme de performances et peut facilement évoluer vers une autre version possédant plus de fonctionnalités et utilisant des technologies nouvelles. Notre contribution a consister aussi à fournir une implémentation d'un ensemble de

classes Java permettant d'effectuer des opérations de base d'entrée/sortie (eg., ajout, suppression et recherche de protocoles, etc.).

Le travail de création de dépôts n'est pas encore achevé, il faut intégré d'autres modules qui ne font pas partie de ce présent travail. Il faut en particulier exposer les dépôts en tant que service web et application web pour permettre aux utilisateur d'interroger les protocoles de services. De plus, il faut ajouter un service web supplémentaire permettant l'analyse des protocoles. Le dépôt doit aussi assurer les fonctionnalités de sécurité afin de protéger les données.

Annexe A

A.1 Exemples de DTD

DTD du document *emps.xml*

```
< !--DTD du document emps.xml-->
<!ELEMENT emps (emp)+>
<!ELEMENT emp (emplNo,nom,deptno,salary)>
<!ELEMENT emplNo (\#PCDATA)>
<!ELEMENT nom (\#PCDATA)>
<!ELEMENT deptno (\#PCDATA)>
<!ELEMENT salary (\#PCDATA)>
```

DTD du document *depts.xml*

```
< !--DTD du document depts.xml-->
<!ELEMENT depts (dept)+>
<!ELEMENT dept (num,nom,ville)>
<!ELEMENT num (\#PCDATA)>
<!ELEMENT nom (\#PCDATA)>
<!ELEMENT ville(\#PCDATA)>
```

A.2 Exemple de protocoles de services

Voici un exemple d'un protocole généré aléatoirement, l'automate contient trois états et 3 transitions :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<protocol title="protocol7">

  <description>un texte quelconque</description>
```

```
  <states>

    <state type="initial">state_0</state>
    <state type="final">state_1</state>
    <state type="transactional">state_3</state>

  </states>

  <transitions>

    <transition name="transition_1" source="state_4" target="state_7">
      <activation mode="provider" event="methode_70">
        <precondition
          O-condition="true"
          U-condition="true"
          T-condition="true" />
      </activation>
    </transition>

    <transition name="transition_2" source="state_9" target="state_6">
      <activation mode="provider" event="methode_51">
        <precondition
          O-condition="true"
          U-condition="true"
          T-condition="C-Invoke(inf,end(transition7) + 88 days)" />
      </activation>
    </transition>

    <transition name="transition_3" source="state_7" target="state_8">
      <activation mode="provider" event="methode_29">
        <precondition
          O-condition="true"
          U-condition="true"
          T-condition="C-Invoke(inf,end(transition4) + 25 days)" />
      </activation>
    </transition>

  </transitions>

</protocol>
```

Bibliographie

[1] *Uddi technical white paper*, www.uddi.org, September 2000.

[2] *Uddi programmer's api 1.0 uddi published specification*, http ://www.uddi.org/pubs/ProgrammersAPI-V1.00-Published-20020628.pdf, June 2002.

[3] *Uddi version 2.03 data structure reference uddi committee specification*, http ://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm, July 2002.

[4] *Extensible markup language (xml)*, 2006 http ://www.w3.org/TR/2006/REC-xml-20060816, September 2006.

[5] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web services : concepts, architectures, and applications*, Springer Verlag, 2004.

[6] Douglas K. Barry, *Web services and service-oriented architecture : The savvy manager's guide*, Morgan Kaufmann, 2003.

[7] B.Benatallah, F.Casati, and F.Toumani, *Representing,analysing and managing web service protocols*, Data and Knowledge Engineering Journal (2005).

[8] B. Benatallah, F. Casati, D. Grigori, H. R. Motahari-Nezhad, and F. Toumani, *Developing adapters for web services integration*, Proc. of CAiSE'05, Springer. (2005).

[9] B. Benatallah, F. Casati, and F. Toumani, *Web service conversation modelling : A cornerstone for e-business automation*, IEEE Internet Computing (2004).

[10] B. Benatallah and H. R. Motahari-Nezhad, *Servicemosaic project : Modeling, analysis and management of web services interactions*, In Proc. Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Australia (2006).

[11] Ronald Bourret, *Xml database products*, Disponible sur http ://www.rpbourret.com/xml/XMLDatabaseProds.htm, dernière mise à jour 25 janvier 2007.

[12] S. Bressan, M. L. Lee, Y. G. Li, Z. Lacroix, and U. Nambiar, *The xoo7 benchmark*, in the Lecture Notes in Computer Science series (ISBN 3-540-00736-9) (2002).

[13] Alan W. Brown (ed.), *Component-based software engineering : Selected papers from the software engineering institute, 1st edition*, IEEE Computer Society Press Los Alamitos, CA, USA, 1996.

[14] F. Casati and M. C. Shan, *Models and languages for describing and discovering services*, In SIGMOD '01 : Proceedings of the 2001 ACM SIGMOD international conference on Management of data, ACM Press, New York, NY, USA (2001).

[15] E. Cerami, *Web services essentials, distributed applications with xml-rpc, soap, uddi & wsdl*, O'Reilly, February 2002.

[16] J. Clark and S. DeRose, *Xml path language (xpath) version 1.0. w3c recommendation*, http ://www.w3.org/TR/1999/REC-xpath-19991116, November 1999.

[17] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, *Unraveling the web services web : An introduction to soap, wsdl, and uddi*, IEEE Internet Computing (2002).

[18] D. Slama D. Krafzig, K. Banke, *Enterprise soa : Service-oriented architecture best practices*, Prentice Hall PTR, 2004.

[19] D. Fallside and P.Walmsley, *Xml schema part 0 : Primer second edition. w3c recommendation*, http ://www.w3.org/TR/2004/REC-xmlschema-0-20041028/, October 2004.

[20] F.Curbera, D.Ehnebuske, and D.Rogers, *Using wsdl in a uddi registry, version 1.07 uddi best practice*, http ://www.uddi.org/bestpractices.html, May 2002.

[21] D. Fensel, C. Bussler, and A. Maedche, *Semantic web enabled web services*, Invited paper (2002).

[22] G. Gardarin, *Xml des bases de données aux services web*, Dunod, 2002.

[23] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, *Introduction to web services architecture*, IBM Systems Journal (2002).

[24] W3C Working Group, *Web services architecture*, http ://www.w3.org/TR/2004/NOTE-ws-arch-20040211, February 2004.

[25] XML Query Working Group, *Xquery 1.0 : An xml query language w3c. candidate recommendation*, http ://www.w3.org/TR/2006/CR-xquery-20060608/, June 2006.

[26] M. Gudgin, M. Hadley, N. Mendelsohn, J.J Moreau, and H.F Nielsen, *Soap version 1.2 part 1 : Messaging framework. w3c recommendation*, http ://www.w3.org/TR/2003/REC-soap12-part1-20030624/, June 2003.

[27] IBM, *Patterns : Implementing an soa using an enterprise service bus*, International Technical Support Organization, 2004.

[28] R. Irani and S. Jeelani Basha, *Axis : The next generation of java soap*, Wrox Press Ltd, 2002.

[29] H. Kadima and V. Monfort, *Les web services*, Dunod, 2003.

[30] K.Baina, B.Benatallah, F.Casati, and F.Toumani, *Model-driven web service development*, Proc. of CAiSE'04 Conference, Springer (2004).

[31] P. Kellert and F. Toumani, *Les services web sémantiques. rapport de recherche, limos/rr 03-15*, (2003).

[32] J. McGovern, S. Tyagi, M. Stevens, and S. Matthew, *Java web services architecture*, Morgan Kaufmann, 2003.

[33] V. Monfort and S. Goudeau, *Web services et interopérabilité des si*, Dunod, 2004.

[34] E. Newcomer, *Understanding web services : Xml, wsdl, soap, and uddi.*

[35] M. Nicola and B. van der Linden, *Native xml support in db2 universal database*, ACM (2005).

[36] P.Brittenham, F.Cubera, D.Ehnebuske, and S.Graham, *Understanding wsdl in a uddi registry, part 1*, http ://www.ibm.com/developerworks/webservices/library/ws-wsdl/, Sep 2001.

[37] C. Pfister and C. Szyperski, *Why objects are not enough*, Proceedings International Component Users Conference (1996).

[38] E. Pulier and H. Taylor, *Understanding enterprise soa*, Manning Publications, 2006.

[39] M. Rys, D. Chamberlin, and D. Florescu, *Xml and relational database management systems : the inside story*, ACM Press (2005).

[40] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse, *The xml benchmark project. technical report ins-r0103*, (2001).

[41] C. Szyperski, *Component software : beyond object-oriented programming*, New York : ACM Press Harlow, England Reading, Mass : Addison- Wesley, 1997.

[42] D. Tidwell, J. Snell, and P. Kulchenko, *Programming web services with soap*, O'Reilly, December 2001.

[43] Web Services Description Working Group W3C, *Web services description language (wsdl)*, http ://www.w3.org/TR/2001/NOTE-wsdl-20010315, March 2001.

[44] J. Wang, K. Zhou, K. Karun, and M. Scardina, *Extending xml database to support open xml*, IEEE (2004).

[45] B. B. Yao, M. T. Özsu, and N. Khandelwal, *Xbench benchmark and performance testing of xml dbmss*, In Proceedings of 20th International Conference on Data Engineering, Boston, MA (2004).

Résumé

LES protocoles de services (*business protocols*) représentent une partie importante de la description des services web, ils servent à décrire le comportement externe de ces derniers. L'utilisation des protocoles de services permet de simplifier considérablement le cycle de vie entier des services web. Ce mémoire porte sur la création de dépôts permettant d'effectuer des études sur les protocoles de services web, en s'intéressant en particulier à la manière de faire le lien entre ces dépôts et les annuaires UDDI, ainsi qu'au mode de persistance. Ainsi, nous avons décrit en premier temps, un processus permettant de publier la description WSDL de services web dans un annuaire UDDI et de la retrouver au moment de la recherche. En second temps, nous nous sommes intéressé au problème de stockage de XML. A ce propos, les solutions existantes peuvent être classées en deux catégories : La première consiste à utiliser les bases de données relationnelles, tandis que la deuxième consiste à créer entièrement des bases de données XML natives. Pour justifier le choix du mode de persistance, nous avons procédé à une étude comparative d'un ensemble de base de données. Le travail réalisé a consisté aussi à l'implémentation d'un ensemble de classes Java constituant la bibliothèque de dépôt. Cette bibliothèque sert comme une interface entre la base de données et les utilisateurs de dépôt.

Mots clés : services web, architectures orientées services (SOA), protocoles de services web, XML, bases de données relationnelles, base de données XML natives.

Abstract

BUSINESS protocols represent an important part of web service descriptions ; they are used to describe external behavior of web services. The use of business protocols simplifies considerably the whole service life-cycle. This thesis deals with the creation of deposits making it possible to carry out studies on business protocols, while being interested in particular in the manner of establishing the link between these deposits and UDDI registries, and in the mode of persistence. Thus, we firstly described a process allowing to publish WSDL service description in a UDDI registry and to find it at the time of research. Secondly, we were interested in the XML storage problem. The existing solutions to this problem can be classified into two categories : The first consists in using relational databases, while the second consists in entirely developing native XML databases. In order to justify the choice of persistence mode, we made a comparative study of a set of database products. The work carried out also included an implementation of the set of Java classes which constitute the library of the deposit. This library is used as an interface between database and the users of the deposit.

Keywords : web services, service-oriented architecture (SOA), business protocol, XML, relational databases, native XML databases.