République **A**lgérienne **D**émocratique et **P**opulaire
Ministère de l'**E**nseignement **S**upérieur et de la **R**echerche **S**cientifique
Université **M'**hamed **B**ougara - **B**oumerdès

Faculté des Sciences
Département d'Informatique

MEMOIRE

Pour l'obtention du diplôme de MAGISTER

Spécialité : Systèmes Informatiques et Génie des Logiciels

Option : Spécification des Logiciels et Traitement de l'Information

Thème:

Interopérabilité sémantique des données échangées entre les services Web, engagés dans une composition
(Vers une approche de médiation de données basée sur le contexte)

Présenté et soutenu par

MAHFOUD Sami

Soutenu le : 10/10/2011.

Devant le jury composé de:

Mr BOULIF Menouar	**MCA**	**UMBB**	**Président**
Mr LOUDINI Malik	**MCA**	**ESI**	**Examinateur**
Mr GHOMARI Abdessamad-Réda	**MCA**	**ESI**	**Examinateur**
Mr HIDOUCI Khaled Walid	**MCA**	**ESI**	**Promoteur**
Mr ABBASSENE Ali	**AR**	**CDTA**	**Co-promoteur**

Année universitaire: 2010/2011

بسم الله الرَّحْمَنِ الرَّحِيمِ

Remerciements

Je remercie Allah le Tout Puissant de m'avoir permis de mener ce travail à son terme.

Je souhaite adresser mes sincères remerciements:

- *À mes promoteurs :*

  M. HIDOUCI Khaled Walid et M. ABBASSENE Ali, pour leur encadrement et toute l'aide qu'ils m'ont apportée, afin de rendre meilleur notre travail.

- *Aux enseignants du département d'informatique de l'INIM de l'université d'UMBB et aux chercheurs de la division ASM du CDTA.*

  Je tiens à les remercier pour l'effort qu'ils ont fourni afin d'assurer notre formation, pour leurs compétences, et surtout pour leur abnégation.

- *Je remercie également :*

  Le président du jury M.BOULIF Menouar et les membres du jury : M.LOUDINI Malik et M.GHOMARI Abdessamad-Réda, pour nous avoir fait l'honneur d'accepter de juger notre travail.

- *Mes remerciements également à :*

  MM. : Noureddine CHERCHALI, Samir BENBELKACEM, Toufik LARROUM, Elhocine BOUTELLAA, Khalas HAMMOUCHE, Hakim TAHI, Yassine MERAIHI, Hatem AMRANI et Atika BADAOUI, SALAH MAHFOUD.

  Qui ont contribué de près ou de loin à l'aboutissement de ce projet.

- *Mes collègues : Halim Amrani, Boussad Belakebi, Abdelkader Bellarbi, Mohamed Boudefla Mabrouk Boumaraf, Amar Badreddine Cherchali, Mustapha Mellahi, Idir Oudjoudi et Fayçal Zerarga.*

Que toutes ces personnes, soient remerciées du fond du cœur.

Dédicace

Je dédie ce modeste travail à

Mes parents

Mes frères et sœurs

Mes amis.

Résumé

La composition des services Web définit une nouvelle forme d'interopérabilité entre les systèmes d'information et les applications distribuées et hétérogènes. Cette nouvelle forme offre des perspectives prometteuses en termes de collaboration et de coopération pour différentes organisations commerciales et industrielles.

Cependant, les services Web participant dans une composition, sont implémentés par différents fournisseurs qui ont des contextes différents impliquant des interprétations différentes, ce qui augmente les hétérogénéités entre les services, notamment, les hétérogénéités sémantiques de données échangées entre les services qui peuvent générer des dysfonctionnements ou des résultats imprévisibles pendant l'exécution de la composition. Dans ce mémoire, nous nous intéressons au problème d'interopérabilité sémantique. Nous traitons, particulièrement, le problème d'hétérogénéité sémantique et contextuelle de données échangées entre les services Web, participant et le service Web composite (la composition elle-même).

Notre travail s'inscrit autour d'une solution de médiation de données basée sur le contexte afin de détecter et résoudre automatiquement les conflits sémantiques et contextuels de données échangées dans une composition. Dans notre proposition, nous allons se baser sur les ontologies afin de représenter les contextes locaux et de décrire la sémantique globale des données qui circulent dans une composition.

Mots clés :

Services Web, Composition de services Web, Ontologie, Services Web sémantique, Contexte, Interopérabilité sémantique, Médiation.

Abstract

The composition of Web services defines a new form of interoperability between information systems and heterogeneous distributed applications. This new form offers promising prospects in terms of collaboration and cooperation for various commercial and industrial organizations.

However, Web services participating in a composition, are implemented by different providers that have several contexts involving different interpretations which increases the heterogeneities between services, especially, the semantic heterogeneity of data exchanged between services that can generate faults or unpredictable results during the execution of the composition. In this Thesis we focus on the problem of the semantic interoperability, specifically, we treat the problem of heterogeneity semantic and contextual of data exchanged between Web services, participant and composite Web Service (the composition).

Our work is about mediation solution for data based on the context to detect and resolve automatically the semantic and contextual conflicts of data exchanged in a composition. In our proposal, we based on ontologies to represent the local contexts and describe the global semantics of data circulate in a composition.

Keywords:

Web Services, Composition of Web Services, Ontology, Semantic Web Services, Context, Semantic Interoperability, Mediation.

ملخص

تركيبة خدمات الويب تحدد شكلا جديدا للتوافقية بين نظم المعلومات والتطبيقات الموزعة وغير المتجانسة. وهذا الشكل الجديد يوفر عدة تطلعات من حيث التعاون والتنسيق بين مختلف المنظمات التجارية والصناعية. غير أن، خدمات الويب المتشاركة في التشكيل، المطبقة من قبل الممولين التي تتضمن سياقات و تفسيرات مختلفة تزيد من التباين بين الخدمات.خاصة اختلاف معنى البيانات المتبادلة بين خدمات الويب، هذا التباين بإمكانه إحداث خلل وضيفي وإعطاء نتائج غير متوقعة خلال تنفيذ التركيب.

في هذه المذكرة ركزنا اهتمامنا على مشاكل التباين في المعني و بالأخص معالجة مشكل عدم التجانس ما بين البيانات المتبادلة بين خدمات الويب، المشاركة والناتجة عن التركيب. في عملنا هذا ركزنا اهتمامنا حول إيجاد وساطة للبيانات على أساس المعنى ليتم تسهيل اكتشاف وحل التباين في المعنى آليا، وهذا بين البيانات المتبادلة في تركيب خدمات الويب.اقتراحنا مبني على استخدام تكنولوجية الانطولوجيا لتمثيل معاني البيانات المتبادلة بين خدمات الويب.

الكلمات المفتاحية :

خدمات الويب.تكوين خدمات ويب، الانطولوجيا، خدمات الويب الدلالية، السياق، الدلالات التوافقية، الوساطة.

Table des matières

Liste des figures

Liste des tableaux

Liste d'abréviations

BPEL	Business Process Execution Language for Web Services
HTTP	HyperText Transfer Protocol
OCML	Operational Conceptual Modeling Language
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services Web
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RPC	Remote Procedure Call
SAWSDL	Semantic Annotations for WSDL and XML Schema
SOAP	Simple Object Access Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WS	Service Web
WS-CDL	Web Service Choreography Description Language
WSCI	Web Service Choregraphy Interface
WSDL	Web Service Description Language
WSDL-S	Web Service Description Language-Semantic
WSFL	Web Service Flow Langage
WSMO	Web Service Modeling Ontology
XLANG	XML Business Process Language
XML	eXtensible Markup Language
XSD	XML Schema

Introduction

Introduction

Contexte de travail

Aujourd'hui, le besoin des entreprises en termes de coopération et de collaboration devient de plus en plus indispensable pour faire face aux nouvelles contraintes imposées par le marché. Ceci a un impact direct sur les systèmes d'informations (SI). En effet, les entreprises coopèrent et se communiquent dans le but de réaliser certaines tâches communes à travers les interactions entre des applications distribuées qui participent à mettre en œuvre les Systèmes d'Information.

Dans cet usage, l'interopérabilité apparait comme un enjeu fondamental qui concerne la capacité de deux applications ou plus à coopérer et à échanger de l'information [Mellal 07]. Notons qu'il existe trois types d'interopérabilités: l'interopérabilité organisationnelle, l'interopérabilité technique et l'interopérabilité sémantique. L'interopérabilité organisationnelle offre une définition claire des rôles et des responsabilités de chaque partenaire. L'interopérabilité technique facilite et garantit les échanges des données. L'interopérabilité sémantique permet une compréhension unique et commune des diverses informations échangées. Ceci permet d'éviter les problèmes de malentendu ou d'incompréhension des demandes ou réponses formulées entre les partenaires [Blanc 06].

Assurer l'interopérabilité entre les applications distribuées et réparties est un grand défi, du fait que ces applications ont été généralement conçues de manière indépendante et par différents partenaires pour divers buts et grâce à plusieurs méthodes et technologies.

Afin de renforcer l'interopérabilité, les services Web ont récemment émergé avec l'appariation et l'évolution d'Internet. Ceux-ci définissent un nouveau paradigme de développement des interactions entre les applications distribuées et hétérogènes. Ils sont considérés comme une technologie prometteuse qui rend l'interopérabilité plus simple à travers l'utilisation des standards du Web. En effet, les services Web encapsulent les hétérogénéités des applications distribuées sous forme des composants logiciels techniquement interopérables se basant sur des protocoles et langages standards.

Grâce à l'utilisation des protocoles et langages standards basés sur le langage XML (eXtensible Markup Language), les services Web restent indépendants des systèmes d'exploitation et des environnements de développement. Cet effet facilite les interactions entre

les services par le biais de la composition. Ainsi, cette nouvelle forme d'interopérabilité offre des perspectives prometteuses en termes de collaboration et de coopération pour différentes organisations commerciales et industrielles.

Une composition de services Web consiste à combiner les fonctionnalités de plusieurs services Web au sein d'un même processus métier pour répondre à des demandes complexes qu'un seul service ne peut satisfaire [Aît-Bachir 08]. Malgré que les services Web soient conçus pour résoudre les problèmes d'interopérabilité notamment l'interopérabilité technique, ils ne sont conçus initialement pour répondre aux exigences d'échange sémantique (interopérabilité sémantique). Par conséquent, les hétérogénéités sémantiques sont gérées de manière manuelle durant la phase de conception d'une composition.

Afin de pallier à ce problème, des solutions et des approches ont été proposées d'une part pour décrire la sémantique des services Web de manière formelle et compréhensible par les machines. Elles se basent sur les ontologies (technologies proposés par la communauté du Web sémantique) qui permet d'expliciter la sémantique des fonctionnalités offertes par les services Web de façon plus formelle. Nous sommes alors dans le cas des services Web sémantiques. Et d'une autre part pour résoudre les hétérogénéités par des mécanismes de médiation basés sur la sémantique. Ces médiations exploitent les informations sémantiques décrites les services Web sémantiques afin de réconcilier les conflits sémantiques entre les services Web engagés dans une composition.

Problématique

Dans le cadre de ce mémoire, nous traitons la problématique de l'interopérabilité sémantique au niveau des services Web engagés dans une composition dans un contexte ouvert et dynamique propre à chaque service. Le contexte est défini comme « tout élément interne ou externe, relatif à l'application, à l'utilisateur, ou même complètement extérieur, qui pourrait modifier le déroulement d'une interaction » [Dey et al 01].

Nous aborderons dans notre travail, les hétérogénéités des données échangées entre les services Web. Nous nous focaliserons, particulièrement, sur les hétérogénéités sémantiques et contextuelles de ces données. Notre proposition consiste à fournir une solution de médiation pour réconcilier les conflits sémantiques et contextuels et permettre des échanges consistants entre les services participant à une composition. Notre solution de médiation se base sur les ontologies qui représentent les contextes locaux et la sémantique globale de données échangées entre les services.

Organisation du mémoire

Ce mémoire est structuré de la manière suivante :

Le chapitre I est consacré à l'étude des services Web et leur composition pour mieux comprendre les concepts de base de cette technologie. Ce chapitre se divise en deux parties : nous présentons dans la première partie les services Web, leur architecture, leur modèle d'interaction et leurs standards de base. Dans la deuxième partie de ce chapitre, nous aborderons la composition des services Web, ses différents types, les langages de composition et les étapes de composition.

Le chapitre II présente un état de l'art sur les ontologies et les services Web sémantiques. Il se divise aussi en deux parties: la première partie est consacrée à étudier les ontologies et leurs apports. La deuxième partie présente quelques approches de réalisation des services Web sémantiques proposées dans la littérature. Une comparaison entre ces différentes approches est effectuée pour sélectionner la plus appropriée à notre travail.

Le chapitre III présente un état de l'art sur les hétérogénéités entre les services Web dans une composition et les solutions de médiation correspondantes. Ensuite, nous nous focaliserons sur les hétérogénéités des données échangées entre les services ainsi que les approches de médiation proposées dans la littérature pour résoudre ces hétérogénéités. Une synthèse sera effectuée pour clarifier l'avantage et la nécessité d'une approche sémantique orientée contexte pour notre problématique.

Le chapitre IV décrit notre contribution. Il s'agit de proposer une solution de médiation de données orientée contexte en se basant sur les travaux de Mrissa [Mrissa 07]. Cette solution se résume aux points suivants: Intégration du contexte de données de la composition elle-même. Proposition d'une solution pour résoudre les hétérogénéités entre les éléments du contexte eux-mêmes en incluant le contexte de donnée de la composition. Proposition d'une annotation plus standardisée du fichier WSDL en utilisant l'annotation SAWSDL du W3C. Définition des fonctions de conversions (atomiques et composées) associées à chaque propriété sémantique du contexte. Proposition d'un algorithme de détection des conflits contextuels de données dans une composition qui prend en considération le contexte des données du service Web composite (la composition en elle-même).

Le chapitre V consiste à valider notre travail par la réalisation d'une composition réelle contenant des hétérogénéités sémantiques (de type contextuel) des données échangées entre les services Web, participant et la composition.

Deux outils ont été développés, l'un pour aider les fournisseurs des services Web ainsi que les concepteurs de composition à annoter les fichiers WSDL par la recommandation SAWSDL. Le second outil permet de détecter les hétérogénéités contextuelles des données nuisant à la bonne exécution de la composition.

Chapitre I

I. Composition des services Web

I.1 Introduction

Aujourd'hui, les services Web définissent un nouveau paradigme de développement des applications distribuées. Ce sont des composants logiciels offrant aux utilisateurs des fonctionnalités bien particulières. Ces composants-la sont techniquement interopérables, modulaires, auto-contenus et auto-descriptifs. Ils peuvent être publiés, localisés et invoqués par d'autres applications ou d'autres services Web. Se basant sur des langages et des protocoles standards d'internet par souci d'interopérabilité. Cependant, les fonctionnalités offertes par les services Web sont relativement simples, alors que les besoins des utilisateurs sont de plus en plus complexes, à tel point qu'un seul service Web ne peut pas les satisfaire. L'implication de la composition des services Web, dans le but de créer un nouveau service Web dit composite offre des nouvelles perspectives qui répondent aux exigences complexes des utilisateurs.

Ce chapitre est consacré à étudier les services Web et leur composition, qui constituent le cœur de notre problématique. Il se divise en deux parties, nous présentons dans la première partie les services Web par quelques définitions et caractéristiques ainsi que leur architecture et leur modèle d'interactions afin de montrer les avantages des services Web par rapport à d'autres technologies similaires, ensuite nous nous intéressons à étudier les langages et les protocoles permettant d'implémenter les services Web. Ces protocoles constituent des standards qui augmentent l'interopérabilité entre les applications et les services ceci constitue un grand avantage. Dans la deuxième partie de ce chapitre, nous abordons la composition des services Web, en clarifiant dans un premier temps ce qu'est une composition de services Web ainsi que ses différents types. Dans un deuxième temps, les langages de composition les plus

utilisés et les plus normalisés seront examinés et nous terminons par l'illustration des étapes de composition.

I.2 Introduction aux services Web

I.2.1 Définition d'un service Web

Le consortium W3C[1] définit un service Web comme étant une application ou un composant logiciel identifié par un URI (Uniform Resource Identifier), dont ses interfaces et ses liens peuvent être décrits en XML[2] (*eXtensible Markup Language*), sa définition peut être découverte par d'autres services Web et il peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant des protocoles Internet standards (Cf. [Bhiri 05]).

I.2.2 Caractéristiques des services Web

D'après [Boudali 07], et la définition de W3C que nous avons citée ci-dessus, plusieurs auteurs définissent les services web par des caractéristiques technologiques distinctives. Les plus importantes parmi elles sont:

- **URI**: L'URI est un identifiant d'un contenu sur le Web comme un document tel qu'un texte, audio, vidéo ou bien une page Web. Le service Web est un composant logiciel accessible par un URI spécifique.
- **Ses interfaces et ses liaisons sont publiées, localisées et invoquées via le langage XML:** Parmi les principales tâches d'un service Web il y a la publication de ces interfaces dans un registre, la localisation en interrogeant ce registre qui l'héberge et l'invocation par un ou plusieurs autres services Web après sa localisation. Ces tâches sont réalisées en utilisant le langage XML.

[1] W3C: World Wide Web Consortium, http://www.W3.org.

[2] XML: ce langage est présenté dans la section I.2.5.1 du ce chapitre.

- **Capacité d'interagir avec des composants logiciels via le langage XML utilisant des protocoles Internet standards:** Un service Web est créé pour être interrogé par d'autres logiciels, contrairement à une page Web ou à une autre application qui ne l'utilise pas. L'interopérabilité est basée sur l'utilisation du XML et des protocoles Internet standards, tels que HTTP, SMTP et FTP.
- **Composante logicielle légèrement couplée à interaction dynamique**: Un service Web avec son client (un client de service Web peut être un autre service Web, un logiciel ou bien une application web) qui l'invoque, et qui sont indépendants l'un de l'autre. Si une modification survient sur un service Web, le client n'a pas besoin de connaître la machine, le langage de programmation, le système d'exploitation ou autres paramètres de son environnement d'implantation, afin d'établir à nouveau une communication avec le service Web. Le client possède une fonctionnalité qui consiste à faire une localisation et une invocation du service Web, au moment de l'exécution du programme de service Web, de manière automatique.

Ces caractéristiques mettent en valeur les avantages principaux d'un service Web, à savoir :

- Interface décrite d'une manière interprétable par les machines, qui permet aux applications clientes d'accéder aux services de manière automatique.
- Utilisation de langages et de protocoles indépendants des plateformes d'implantation, qui renforcent l'interopérabilité entre les services.
- Utilisation des normes actuelles du Web, qui permettent la réalisation des interactions entre les services Web faiblement couplés et favorisent aussi l'interopérabilité.

I.2.3 Modèle d'interaction des services Web

Les interactions entre les services Web impliquent trois participants [Hubert et al 03]: le fournisseur de services, l'annuaire de services et le client du service (Comme illustré sur la Figure I-1). Cette interaction s'effectue à travers les trois opérations suivantes: la publication

de la description des services (*publish*), la recherche et la découverte de la bonne description du service (*find*) et l'association et/ou l'invocation du service basée sur sa description (*bind*).

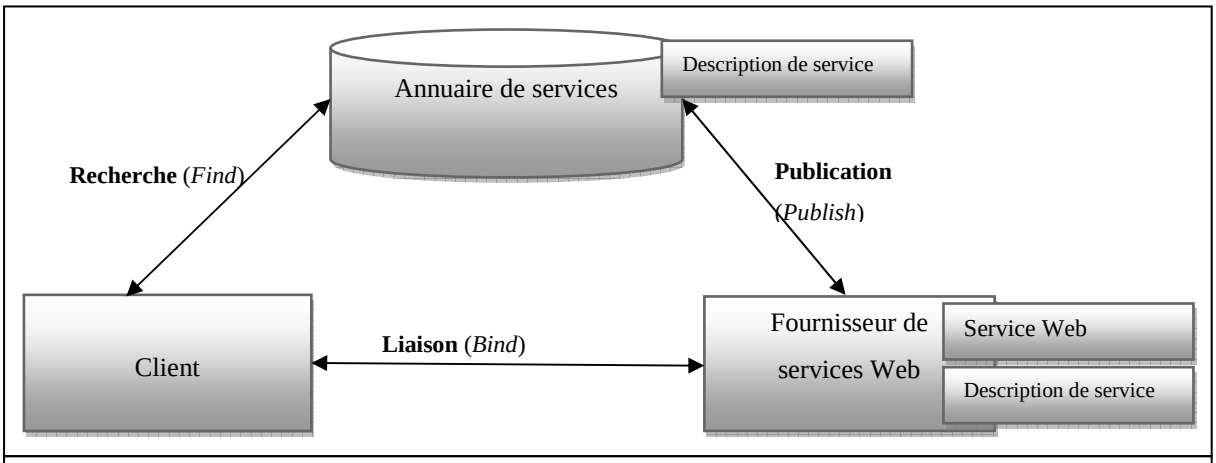

Figure I-1: modèle d'interactions des services Web [Hubert et al 03].

Afin d'assurer la collaboration et l'interaction entre les applications, chacun des participants de ce modèle d'interaction présente une facette métier et une facette d'architecture technique:

- **Le fournisseur**: correspond au propriétaire du service du point de vue métier, et du point de vue architecture technique ; il est constitué par la plate-forme d'accueil du service.
- **Le client**: d'un point de vue métier correspond au demandeur du service, et d'un point de vue architecture technique, il est constitué par l'application de recherche et d'invocation d'un service. L'application cliente peut être elle-même un service Web.
- **L'annuaire**: correspond à un registre de descriptions des services offrant des facilités de publication de services pour les fournisseurs ainsi que des facilités de recherche pour les clients.

I.2.4 Architecture d'un service Web

L'architecture standard d'un service Web est implémentée à l'aide des diverses technologies et standards (UDDI, WSDL, SAOP et HTPP, présentés dans la section I.2.5 de ce chapitre), organisés en quatre couches. Chacune d'elles répond à des préoccupations fonctionnelles différentes telles que la publication, la description, la messagerie et le transport, comme illustré sur la Figure I-2 [Hubert et al 03].

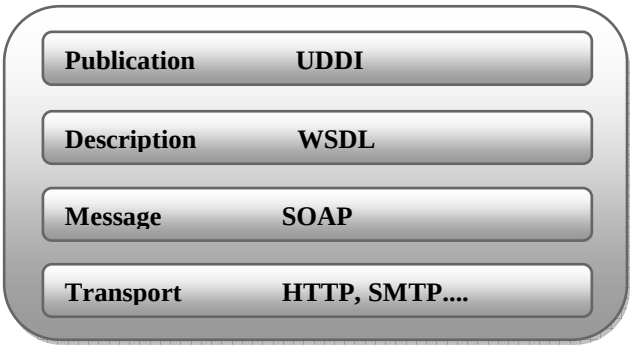

Figure I-2: cadre architectural standard des services Web.

Dans un souci d'interopérabilité, les différentes couches de l'architecture d'un service Web s'interfacent avec des standards, comme suit:

- **La couche de publication**: repose sur le protocole UDDI (*Universal Description, Discovery and Integration*), qui assure le regroupement, le stockage et la diffusion des descriptions des services Web.
- **La couche description**: est prise en charge par le langage WSDL (*Web Service Description Language*) [Christensen et al 01], qui décrit les fonctionnalités fournies par le service Web, les messages reçus et envoyés pour chaque fonctionnalité, ainsi que le protocole utilisé pour la communication.
- **La couche message**: la couche message utilise des protocoles reposants sur le langage XML, car sa syntaxe unique résout les conflits syntaxiques lors de l'encodage des données. Actuellement, SOAP (*Simple Object Access Protocol*) est le protocole le plus utilisé pour cette couche.
- **La couche transport**: le protocole le plus utilisé dans cette couche est l'HTTP (*Hyper Text Transfert Protocol*). Cependant, d'autres protocoles peuvent être utilisés, tels que le SMTP (*Simple Mail Transfer Protocol*) ou le FTP (*File Transfer Protocol)*, permettant ainsi aux services Web de rester indépendants du mode de transport utilisé.

I.2.5 Standards du service Web

Le concept des services web s'articule actuellement autour des plusieurs standards**,** nous présentons dans cette section les trois standards de base, à savoir, SOAP (*Simple Object Access Protocol*), WSDL (*Service web Description Language*) et UDDI (*Universal*

Description, Discovery and Integration). Ces trois standards sont basés sur le langage XML (*eXtansible Markup Language*) dont nous allons présenter le rôle dans les services web, ensuite nous nous intéresserons aux trois standards cités ci-dessus.

I.2.5.1 XML (eXtansible Markup Language)

XML[3], le langage fondateur des services Web, est le résultat de la coopération d'un grand nombre d'entreprises et chercheurs partenaires du W3C. C'est un langage de description et d'échange de documents structurés. Il permet de décrire la structure logique des documents à l'aide des balises qui permettent de marquer les éléments qui la composent, et les relations entre eux [Hubert et al 03]. Ce langage possède plusieurs avantages, dont nous allons citer quelques uns:

- Une syntaxe unique qui permet son adoption par divers systèmes.
- Une structure arborescente qui facilite sa lisibilité.
- Une grande extensibilité, car il n'impose aucune restriction d'utilisation en dehors de sa syntaxe.

Donc grâce à XML les services Web restent indépendants des systèmes d'exploitation et des plateformes de développement, ce qui facilite les interactions entre les applications et les services Web.

I.2.5.2 SOAP (Simple Object Access Protocol)

SOAP est un protocole standardisé par le Consortium W3C, qui assure des appels de procédures à distance RPC (*Remote Procedure Call*), s'appuyant principalement sur le protocole HTTP et le langage XML. Il assure l'interaction entre services Web en transportant les paquets de données encapsulés sous forme de texte structuré en format XML (Cf. [Hubert et al 03]). SOAP forme la couche inférieure de la pile standard des protocoles des services Web (Cf. la Figure I-2), fournissant un Framework pour l'échange de messages sur lequel les services Web peuvent se baser.

[3] XML: l'annexe A présente ce langage plus en détails.

I.2.5.2.1 Structure du message SOAP

Les messages échangés lors de l'utilisation du protocole SOAP sont basés sur le langage XML. Ils sont composés de deux parties, l'en-tête de protocole de transport et l'enveloppe SOAP (Cf. la Figure I-3).

L'en-tête du protocole de transport: qui dépend de protocole de transport utilisé, par exemple si le protocole HTTP est utilisé, l'en-tête contient :

- la version de protocole HTTP utilisée,
- la date de génération de message SOAP,
- le type d'encodage du contenu (généralement de type XML).

L'enveloppe SOAP: la partie principale d'un message SOAP est l'enveloppe (symbolisée par la balise *enveloppe*), cette dernière est subdivisée en deux sous-parties: la partie en-tête (*Header*) et la partie corps du message (*Body*).

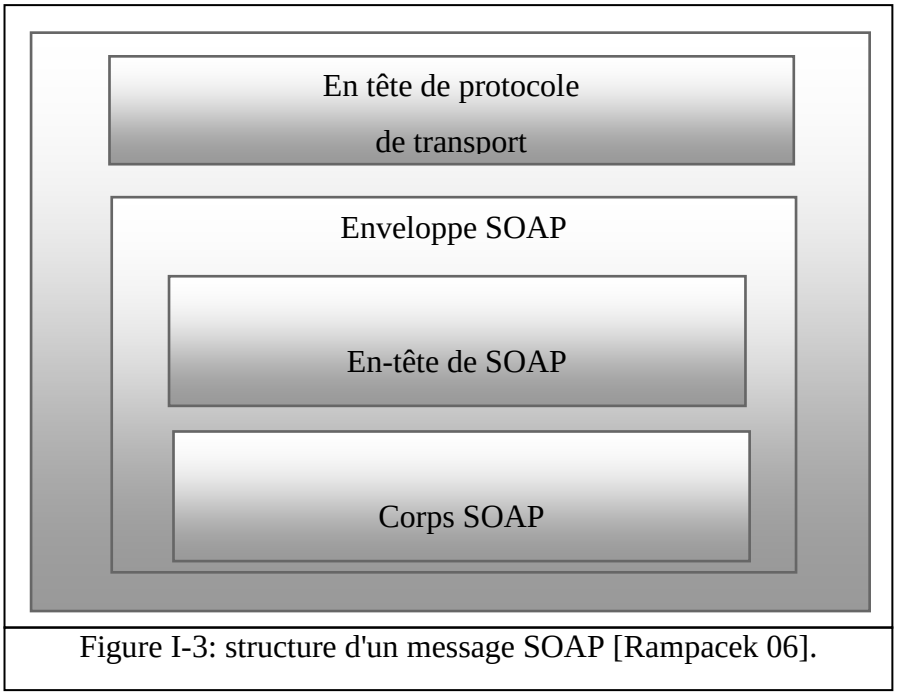

Figure I-3: structure d'un message SOAP [Rampacek 06].

L'en-tête du message SOAP: la partie en-tête SOAP (SOAP *Header*) est optionnelle et extensible (Cf.[Rampacek 06]). Les balises XML qui permettent de symboliser cette partie sont: <*env:Header*> et </*env:Header*>. Ces balises peuvent être complétées par des attributs permettant de définir le domaine de noms du service Web. En fait, l'en-tête permet principalement d'ajouter des informations sur le comportement des différents nœuds intermédiaires, lors de traitement du message.

Un nœud est un intermédiaire SOAP, incluant le récepteur et l'émetteur SOAP, désignable depuis un message SOAP. Son rôle est de traiter l'en-tête, ensuite de transférer le résultat (le message SOAP modifié) à un autre intermédiaire (qui peut être le récepteur final).

Les principaux attributs des éléments qui forment le bloc d'en-tête sont :

- ***env:role*:** permet d'indiquer à quel nœud la fonction décrite est destinée.
- ***env:mustUnderstand*:** c'est une valeur booléenne, elle permet de préciser que le traitement devient obligatoire pour un nœud intermédiaire, par exemple, pour un calcul très long, il peut être utile d'envoyer un e-mail à chaque étape.
- ***env:relay*:** cet attribut permet de relayer un message à un autre nœud si le premier nœud n'est pas capable de le traiter.

Le corps du message SOAP: l'élément corps de message SOAP contient des données spécifiques à l'application. Les balises symbolisant cette partie sont: *<env:Body>*et *</env:Body>*. Les données doivent donc être sérialisées selon l'encodage XML. En plus, des données de cette partie peuvent transporter un type spécial comme: les messages d'erreurs (SOAP *Fault*).

I.2.5.3 WSDL (Web Service Description Language)

WSDL [Christensen et al 01], est un standard du W3C, qui permet de définir une syntaxe XML pour décrire les méthodes et les paramètres des services Web invocables par le biais de messages au format SOAP. Il permet de définir qu'est-ce qu'un service Web est capable de faire, son emplacement et comment l'invoquer. De même que SOAP, WSDL est un langage indépendant des protocoles de transport de messages [Izza 06].

I.2.5.3.1 Eléments de document WSDL

D'une manière générale, Les éléments qui composent un document WSDL sont les suivants [Lopez-velasco 08]:

- les opérations proposées par le service Web,
- les données et messages échangés lors de l'appel d'une opération,
- le protocole de communication,
- les ports d'accès au service.

Dans un document WSDL, il existe une séparation entre deux niveaux indépendants (le niveau abstrait et le niveau concret). Le niveau abstrait regroupe les informations qui peuvent être réutilisées (c'est à dire, ils ne sont pas spécifiques à un service). Tandis que, le niveau concret est constitué de la description des protocoles d'accès au service Web (informations particulières à un service).

Le niveau abstrait: ce niveau décrit les informations propres aux méthodes proposées par le service, ainsi que les informations traitant des messages et des données échangés lors de l'invocation du service. Si deux services proposent les mêmes méthodes, le niveau abstrait de description WSDL peut être réutilisé. Ce niveau est composé des informations suivantes :

- **Les types de données**: le document WSDL permet de décrire les types de données échangées. WSDL supporte les types élémentaires (tels que les entiers, les chaînes de caractères, etc.) et les types complexes. Dans le cas où les données échangées possèdent une structure particulière (c'est à dire un type complexe), il est possible de les décrire par l'intermédiaire d'un XSD (XML *schema*).
- **Les messages (les données):** un message correspond aux données qui sont véhiculées selon les méthodes invoquées. Chaque opération du service possède deux définitions de message: la première correspond à la requête (*Input*) et la seconde correspond à la réponse (*Output*). La description d'un message contient le nom de l'élément en paramètre d'entrée ou de sortie selon le type du message.
- **Les opérations:** une opération représente une unité de travail, c'est-à-dire, une méthode proposée par le service Web décrit. Chaque opération est identifiée par son nom.

Le niveau concret: ce niveau décrit la manière dont le client accède à un service Web en particulier, il est non réutilisable (propre à un service unique). Les informations décrites dans le niveau concret sont les suivantes :

- **Le protocole de communication:** cette description permet de définir le protocole à utiliser pour l'appel des méthodes du service.
- **Les ports d'accès au service**: dans un document WSDL, l'accès au service est défini par une collection de ports d'accès. Chaque port représente la localisation

du service par son URI. Un même service peut être accessible sur plusieurs ports différents.

I.2.5.3.2 Structure d'un document WSDL

La structure de document WSDL se distingue selon les deux versions de WSDL, WSDL1.1[4] et WSDL 2.0[5], la plupart des descriptions des services Web actuelles sont décrites par la première version, tandis que, la deuxième version 2.0 est en phase de standardisation par W3C. Nous ne présentons ici que la version WSDL1.1.

Une description WSDL est un document XML composé d'un élément racine (*definitions*) et de cinq sous-éléments obligatoires (types, *message*, *portType*, *binding* et *service*). La Figure I-4 illustre la structure générale d'une description WSDL et établit les liens entre les éléments XML et les catégories d'informations définies dans la section précédente (Cf. la section I.2.5.3.1 du ce chapitre).

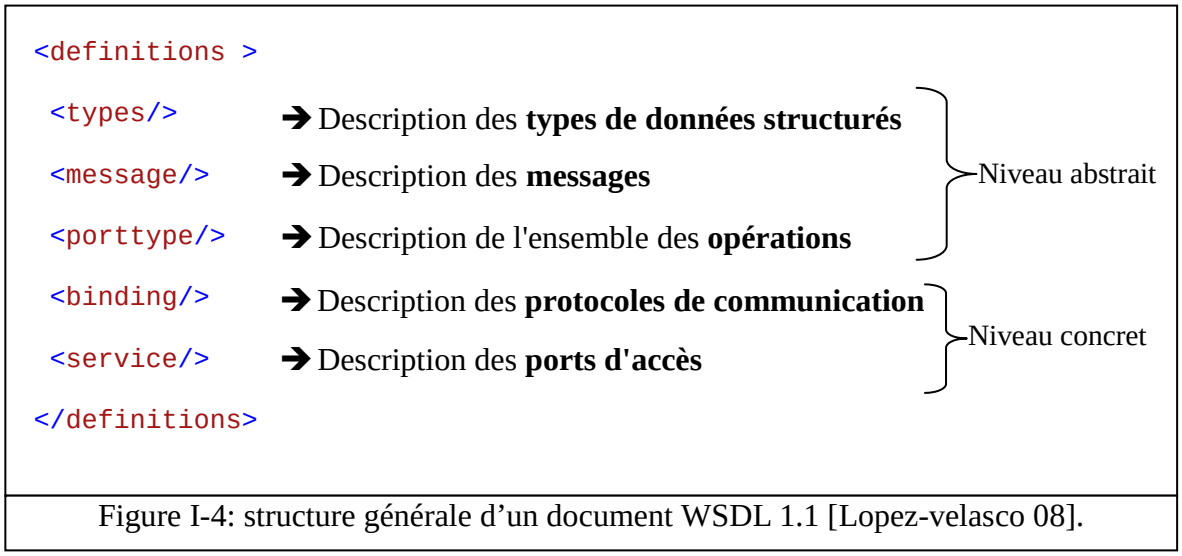

Figure I-4: structure générale d'un document WSDL 1.1 [Lopez-velasco 08].

L'élément *definitions* : est la racine du document WSDL, il contient les autres sous-éléments (*types*, *message*, *portType*, *binding*, *service*) et les espaces de noms qui permettent de connaître la version de SOAP, les définitions de schéma XML utilisées dans le document

[4] WSDL1.1: http://www.w3.org/TR/wsdl.

[5] WSDL2.0: http://www.w3.org/TR/wsdl20.

et d'autres espaces de noms utiles, comme l'espace de nom pour le fournisseur du service Web.

L'élément *types* : il contient les définitions des types de données appliqués aux messages échangés. Afin de garantir une interopérabilité maximale, et une grande indépendance au niveau des plateformes, WSDL utilise XSD en tant que système de types de données.

L'élément *message* : il introduit les types de messages supportés par le service et décrit les données transmises lors des appels des méthodes du service Web.

L'élément *portType* : cet élément décrit l'ensemble des opérations proposées par le service Web.

L'élément *binding*: il définit les protocoles de communication et les spécifications des formats de données pour les ensembles d'opérations (*portType*).

L'élément *service*: il décrit la collection des ports d'accès au service. Cet élément permet de localiser le service Web et de faire des appels aux méthodes disponibles.

I.2.5.4 UDDI (Universal Description, Discovery and Integration)

L'UDDI [Hubert et al 03], est un standard résultant d'une initiative d'un regroupement d'entreprises (Ariba[6], IBM[7], Microsoft[8]), il constitue un référentiel commun d'entreprises visant à établir un format d'annuaire des services Web. UDDI permet de décrire et de localiser une entreprise (ou toute unité organisationnelle) ou un service, à partir des interfaces qui sont personnalisables [Izza 06]. L'annuaire UDDI est consultable sous trois facettes (Comme illustré sur la Figure I-5), qui sont bien détaillées dans [Hubert et al 03]:

[6] Ariba :http://www.Ariba.com

[7] IBM: http://www.ibm.com

[8] Microsoft: http://www.microsoft.com

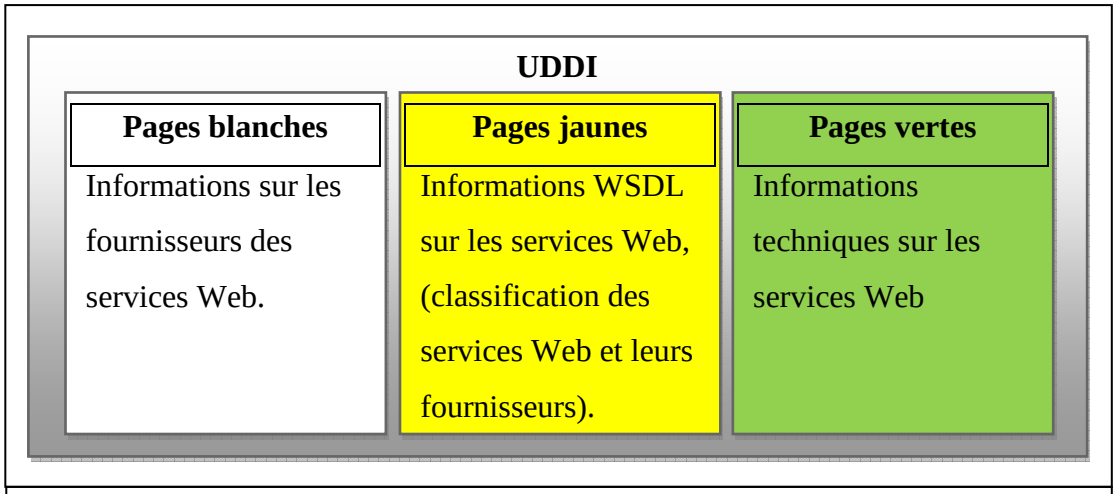

Figure I-5: les trois facettes de l'annuaire UDDI [Hubert et al 03].

- **Pages blanches:** elles comportent des informations sur les fournisseurs des services telles que le nom et les coordonnées du fournisseur.
- **Pages jaunes:** elles comportent la description WSDL des services Web déployés par les fournisseurs ainsi que les informations permettant de les classifier par catégorie.
- **Pages vertes**: elles comportent les informations techniques détaillées sur les services fournis, telles que les descriptions des services Web et les informations de liaison sur les services.

I.2.5.4.1 Modèle d'information d'UDDI

Le modèle d'informations d'UDDI comporte cinq structures de données principales (Comme illustré sur la Figure I-6), définies sous forme XSD (XML *schema*) [Hubert et al 03]. Ces structures de données sont les suivantes:

- ***BusinessEntity*:** contient les informations sur le fournisseur offrant le service dans l'annuaire UDDI. Les quatre autres structures de données sont référenciées via cette partie.
- ***BusinessService* :** contient le nom et une description du service Web proposé.
- ***BindingTemplate* :** contient les informations concernant le lieu d'hébergement du service Web. Il est possible d'enregistrer de multiples *BindingTemplate* pour le même service Web, afin de définir différents points d'accès.
- ***tModel* :** contient les informations concernant le mode d'accès au service Web. C'est un mécanisme d'échange de métadonnées relatives au service, tel qu'une description du service Web ou un pointeur vers le fichier WSDL décrivant le

service. *tModel* peut être considéré comme étant un point d'accès alternatif aux types de données contenues dans UDDI.

- ***PublishAssertion* :** met en correspondance deux ou plusieurs structures *BusinessEntity*.

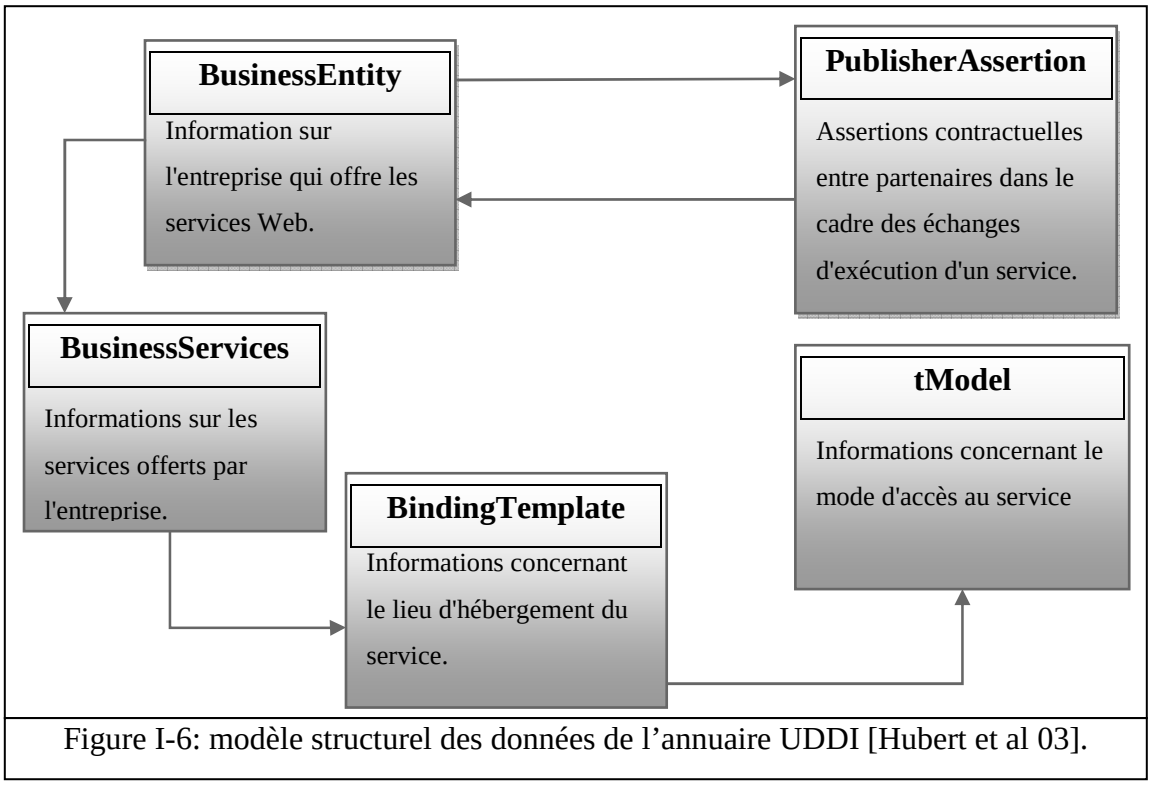

Figure I-6: modèle structurel des données de l'annuaire UDDI [Hubert et al 03].

I.3 Composition des services Web

I.3.1 Définition d'une composition

Une composition de services Web est constituée de plusieurs services qui interagissent les uns avec les autres [Aît-Bachir 08], afin d'offrir de nouvelles fonctionnalités qu'un seul service ne pourrait pas les offrir.

La composition de services Web est étudiée selon deux points de vue, un point de vue global des services Web composés, le terme employé pour cela est la chorégraphie [Bhiri 05] et un point de vue local ou privé des services Web composite, on parle alors de terme d'orchestration [Dumez 10].

I.3.2 Types de composition de services Web

La composition des services web peut se faire de deux manières différentes qui sont l'orchestration et la chorégraphie [Dumez 10] et [Lopez-velasco 08]:

I.3.2.1 Orchestration

L'orchestration de services Web résulte un nouveau service Web dit service Web composé, qui peut être défini comme l'agrégation de plusieurs autres services Web atomiques ou composés. Ce service composé contrôle la collaboration entre les services Web engagés dans la composition, tel qu'un chef d'orchestre [Dumez 10].

L'orchestration de services Web exige de définir l'enchaînement des services Web selon un canevas prédéfini, et de les exécuter selon un script d'orchestration. Ces derniers (le canevas et le script) décrivent les interactions entre services Web en identifiant les messages, et en spécifiant la logique et les séquences d'invocation. Le module exécutant le script d'orchestration de services Web est appelé un moteur d'orchestration (Cf. la Figure I-7). Ce dernier est une entité logicielle qui joue le rôle d'intermédiaire entre les services, en les appelants suivant le script d'orchestration [Lopez-velasco 08].

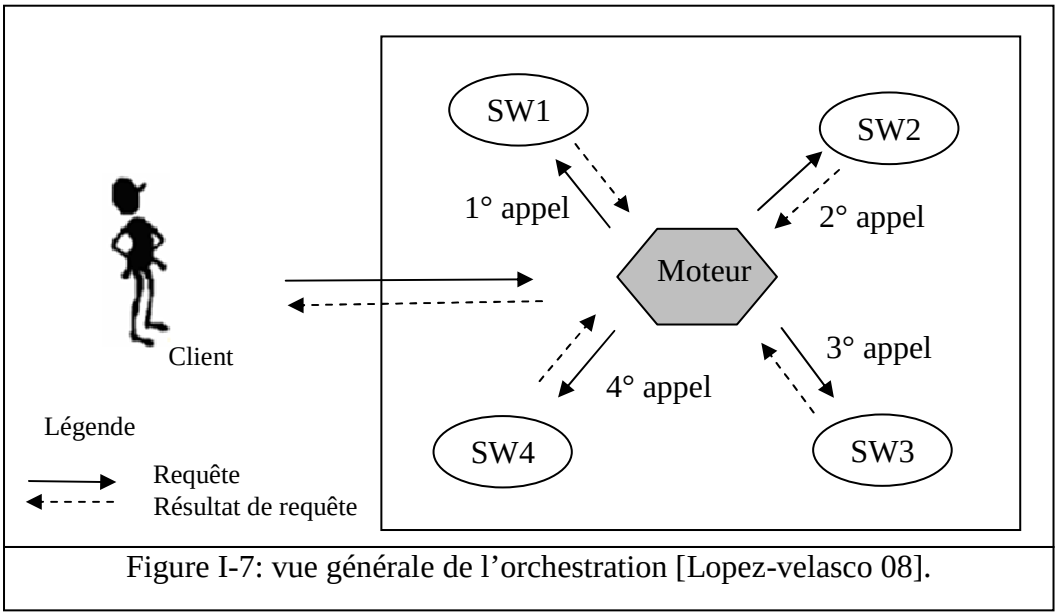

Figure I-7: vue générale de l'orchestration [Lopez-velasco 08].

I.3.2.2 Chorégraphie

La chorégraphie de services Web est une généralisation de l'orchestration qui consiste à concevoir une coordination décentralisée des services Web. Dans une chorégraphie, les interactions de type pair-à-pair (P2P) sont décrites dans un langage de description de chorégraphie(CDL). Les services suivent alors le scénario global de composition sans point de contrôle central [Dumez 10].

La chorégraphie est aussi appelée composition dynamique. En effet, l'exécution n'est pas régie de manière statique comme dans une composition de type orchestration. Dans une

chorégraphie (Cf. la Figure I-8), à chaque pas de l'exécution, le service Web choisit le service Web qui lui succède et implémente ainsi une partie de la chorégraphie. La composition de type chorégraphie n'est pas connue, ni décrite à l'avance [Lopez-velasco 08].

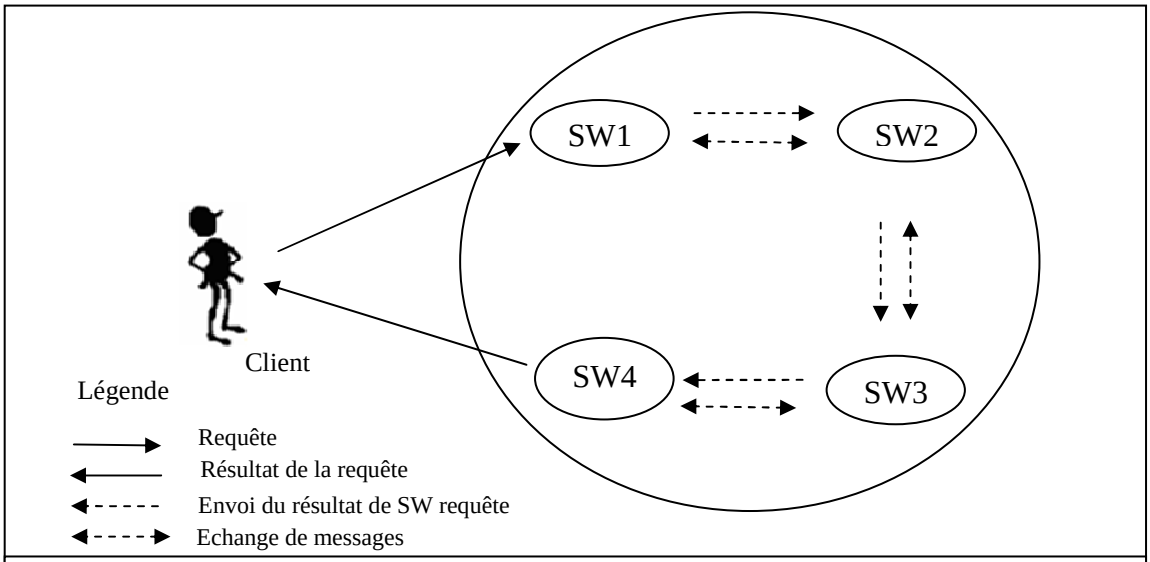

Figure I-8: vue générale de la chorégraphie [Lopez-velasco 08].

I.3.2.3 Processus métiers abstraits et exécutables

Une composition de services Web peut aussi être décrite en termes d'un processus exécutable ou abstrait [Duarte-Amaya 07]:

I.3.2.3.1 Processus abstraits

Les processus abstraits spécifient des échanges de messages publics entre les différentes parties. La description n'inclut pas les détails des flux des différents échanges provenant des différents partenaires (services engagés dans la composition) et le processus ainsi décrit n'est pas exécutable. On se focalise sur la vue protocolaire permettant de réaliser une abstraction du processus métier [Rampacek 06].

Un processus métier abstrait n'est pas exécutable, il est utilisé principalement pour deux raisons :

- Décrire le comportement d'un service sans savoir exactement dans quel processus métier il va intervenir.
- Définir les protocoles de collaboration entre plusieurs partenaires et décrire le comportement externe de chacun d'eux.

Les processus abstraits sont généralement utilisés comme des modèles pour aider à la définition des processus exécutables.

I.3.2.3.2 Processus exécutables

Les processus exécutables représentent la nature et l'ordre des différents échanges entre les différentes parties: ils définissent le processus métier lui-même. Les processus exécutables sont directement exécutés par un moteur d'orchestration [Rampacek 06], en effet, ce type de processus est modélisé par une orchestration. Un processus métier exécutable compose un nouveau service Web à partir d'un ensemble de services Web existants.

À l'opposé des processus abstraits, les processus exécutables décrivent le fonctionnement interne du processus. Le cheminement interne et les conditions sont donc visibles dans la description de ces processus. Les deux principaux avantages de posséder la description interne du processus métier sont:

- Un moteur(ou un serveur) peut directement exécuter ces processus.
- L'évaluation des conditions peut-être connue et donc n'est plus vue comme un choix indéterministe par les autres partenaires.

I.3.3 Etapes de composition

De manière générale, trois étapes sont nécessaires pour le bon déroulement d'une composition, à savoir l'étape de découverte de services Web, de spécification de processus métier et d'exécution de la composition [Mrissa 07]:

L'étape de découverte: cette étape à pour objectif de trouver les servies Web pertinents aux besoins de la composition.

L'étape de spécification de processus métier: dans cette étape on spécifie une composition par un processus métier qui gère les échanges de messages et les structures de contrôle nécessaires, comme les boucles et les opérations conditionnelles, on parle dans cette étape sur le processus métier exécutable qui est modélisé par une orchestration, et sur le processus métier abstrait qui est modélisé par une chorégraphie.

L'étape de l'exécution de la composition: est l'étape d'invocation des services Web engagés dans une composition, en d'autre terme l'étape d'exécution de processus métier.

I.3.4 Langages de composition des services Web

Il existe deux catégories des langages de composition des services Web, une pour décrire les compositions de type d'orchestration, parmi les langages proposés dans cette catégorie nous citons: XLANG (*XML Business Process Language*), WSFL (*Web Services Flow Language*) et BPEL4WS[9] (*Business Process Execution Language for Web Services*) [Rampacek 06]. L'autre pour décrire les compositions de type de chorégraphie, parmi les langages de chorégraphie nous citons: WSCI (*Web Service Choregraphy Interface*) et WS-CDL (*Web Service Choregraphy Description Language*) [Lopez-velasco 08].

Nous étudions dans cette section le langage BPEL4WS [Rampacek 06] qui décrit une composition de type orchestration, il résulte la fusion de deux langages WSFL et XLANG. De plus, il est le langage le plus utilisé par les industriels, et il est en phase de devenir un standard.

I.3.4.1 BPEL4WS (Business Process Execution Language for Web Services)

BPEL[10] est un langage de composition de services Web de type orchestration, il est le fruit de travail de collaboration des acteurs majeurs: Sun, IBM et Microsoft. Il est issu de la fusion de deux langages WSFL et XLANG, combinant les avantages du langage structuré par bloc XLANG avec ceux du langage de processus basé sur les processus métier WSFL. En effet, BPEL est un langage d'orchestration basé sur XML, il est utilisé de façon majoritaire par les acteurs des technologies basées sur les services Web [Rampacek 06]:

- *Microsoft*: l'utilise dans sa plateforme *Biztalk 2004* (serveur d'exécution de processus métier).
- *Oracle*: l'utilise comme langage de description et d'exécution de processus métier dans son serveur *Oracle BPEL*.

[9] BPEL4WS: appelé aussi BPEL.

[10] BPEL: la syntaxe BPEL présenté plus en détaille en Annexe B.

- *IBM*: l'utilise également dans sa plateforme d'exécution de services web *WebSphere*[11].
- *ActiveBpel*[12]: un moteur d'exécution de services web Open Source, est également basé sur l'exécution de processus métier décrit en BPEL.

I.3.4.1.1 Avantages et fonctionnalités du BPEL

Le langage BPEL présente plusieurs avantages dont nous citons quelques uns:

- BPEL est basé sur XML, et il supporte les protocoles standards des services web comme SOAP, WSDL, UDDI (Cf. la section I.2.5 de ce chapitre).
- Sa spécification a été faite par plusieurs acteurs industriels majeurs (Cf. la section I.3.4.1 de ce chapitre), ce qui la rendue plus finie, plus complète et moins ambiguë. Les objectifs sont donc fixés de manière claire et précise et chaque élément est bien détaillé.
- Les éléments syntaxiques de BPEL permettent de modéliser les processus abstraits et les processus exécutables (voir la différence entre les deux processus dans la section I.3.2.3 de ce chapitre).
- Tous leurs éléments présentent des mécanismes de compensation : très utiles lors d'une interaction longue dans le temps ou pour implémenter un mécanisme de transaction.
- le processus permettant le parallélisme présente également des liens de synchronisations.

I.3.4.1.2 Syntaxe du langage BPEL

Une composition des services Web spécifié en langage BPEL est composée de quatre parties qui sont:

- **Les attributs supérieurs**

[11] WebSphere: http://www-01.ibm.com/software/websphere.

[12] ActiveBpel: http://www.ActiveBpel.com

Cette première partie permet la déclaration des informations concernant le processus métier comme son nom, les espaces de nom supportés, les documents WSDL ou schémas XML importés par le processus métier.

- **Lien de partenaire:**

Cette deuxième partie permet la déclaration des *partnerlinks*: ces derniers définissent une liaison entre les ensembles d'opérations des services invoqués par le service Web composite décrit en BPEL, et un nom de liaison bien précis du service Web engagé dans la composition. Ces liaisons sont appelées *partner* et permettent ainsi d'identifier facilement les différents services Web utilisés dans cette composition (processus métier).

- **Les variables:**

Cette troisième partie permet de déclarer des variables, utilisant les types importés de descriptions WSDL associées à la spécification BPEL. Ces variables seront utilisées dans la partie description comportementale du processus métier, pour maintenir un état au niveau du service Web, et ainsi assurer des liaisons de données entre deux opérations.

- **Les activités:**

Les activités décrivent le comportement du processus métier lui-même en utilisant différents opérateurs. Elles peuvent être basiques ou complexes. Les activités complexes sont composées d'autres activités basiques ou complexes.

Les deux tableaux Tableau I-1 et Tableau I-2 illustrent les différentes activités basiques et complexes du langage BPEL.

Les activités basiques	Rôles
<***invoke***>	Invoque une opération dans un service web.
<***receive***>	Attend un message d'une source externe.
<***reply***>	Répond à une source externe.
<***assign***>	Attend un certain moment.
<***wait***>	Copie les données d'une place à l'autre.
<***throw***>	Lance une erreur d'exécution.
<***terminate***>	Termine l'instance de service en entier.
<***empty***>	Ne fait rien.

Tableau I-1: activités basiques du langage BPEL.

Les activités complexes	Rôles
< ***sequence***>	Définit un ordre d'exécution.
<***switch***>	Exprime L'acheminement conditionnel des tâches.
<***while***>	Exprime la boucle.
<***pick***>	Attend l'arrivée d'un événement.
<***flow***>	Exprime l'acheminement parallèle des tâches.

Tableau I-2: activités complexes du langage BPEL.

I.4 Conclusion

Les services Web sont une technologie récente basée sur des langages et des protocoles standards. Ils ont été créés afin de renforcer l'interopérabilité entre les applications distribuées et hétérogènes, et ce de manière indépendante des plateformes d'exécution et des langages de programmation utilisés. D'une manière générale, un seul service ne répond pas aux exigences complexes des utilisateurs, le besoin de combiner et de réutiliser les services Web existants afin d'offrir des fonctionnalités plus efficaces s'avère indispensable, ceci à cause de plusieurs raisons dont la diminution du coût et du temps de développement.

Malgré les apports manifestes des services Web au niveau d'interopérabilité des applications distribuées et hétérogènes, les services Web ignorent toujours l'aspect sémantique dans leur description. En effet, les descriptions WSDL décrivent seulement le coté syntaxique du service Web. Par conséquent, les hétérogénéités, précisément les hétérogénéités sémantiques entre les services Web engagés dans une composition sont gérées de manière manuelle lors de la conception de processus métier. Afin de parer à ce manque la communauté du Web sémantique a introduit des solutions pour ajouter la dimension sémantique dans la description WSDL associée au service Web.

Le chapitre suivant est consacré à étudier ces solutions basées sur les ontologies afin de décrire la sémantique des services Web.

Chapitre II

II. Ontologies et services Web sémantiques

II.1 Introduction

Bien que les services Web soient des composants logiciels techniquement interopérables, ils n'ont pas été conçus initialement pour répondre aux exigences d'échanges sémantiques. En effet, le langage WSDL fournit seulement une description syntaxique d'un service Web. Cependant, il néglige complètement l'aspect sémantique du service. Par conséquent, les conflits sémantiques sont gérés de manière manuelle durant la phase de conception d'un processus métier. Pour atteindre l'interopérabilité sémantique, les services Web doivent être capables de prendre en charge la signification de données qu'ils échangent, de les interpréter correctement et de les décrire de manière sémantiquement explicite et compréhensible par les machines.

Pour remédier à ce problème, des solutions basées sur les ontologies ont été proposées par la communauté du Web sémantique pour décrire explicitement la sémantique des fonctionnalités de services Web. La convergence des technologies de services Web et de l'ontologie a donné naissance aux services Web sémantiques.

Les services Web sémantiques sont des services Web simples enrichis par des descriptions supplémentaires qui sont explicitées dans des ontologies. Ainsi, cette nouvelle forme de la sémantique devient explicite et compréhensible par les machines. L'objectif de services Web sémantiques est de faciliter l'automatisation des tâches telles que la découverte, la sélection, la composition et l'invocation des services.

Dans notre cas, nous nous intéressons à étudier les différentes approches proposées pour réaliser les services Web sémantiques. Ainsi, nous présentons dans ce chapitre les ontologies et leurs apports aux services Web. Ensuite, une étude sur les services Web sémantique est effectuée.

II.2 Ontologies

Les ontologies sont un sujet de recherche populaire dans diverses communautés particulièrement dans l'ingénierie des connaissances, la recherche d'information, les systèmes d'information coopératives et le Web sémantique.

Récemment, elles ont été introduites pour formaliser les connaissances dans les systèmes experts. Elles définissent les primitives indispensables pour leur représentation, ainsi que leur sémantique dans un contexte particulier.

Dans la première partie du chapitre nous abordons la technologie des ontologies. Nous commençons par une définition des ontologies, leurs composantes, leurs typologies. Aussi, nous passons en revue les différentes méthodes et outils et langages de représentation des ontologies.

II.2.1 Définition de l'ontologie

Dans la littérature, il existe une multitude de définitions pour la notion d'ontologie. Cependant, la définition qui est la plus connue et la plus citée, d'après [Izza 06], est celle de Gruber: "une ontologie est une spécification explicite d'une conceptualisation" (Cf. [Gruber 92]).

D'une manière générale, une ontologie est vue comme un ensemble de concepts permettant de modéliser un ensemble de connaissances dans un domaine donné. Un concept peut présenter plusieurs sens thématiques. Les concepts sont liés entre eux par des relations sémantiques, des relations de composition et d'héritage. Dans ce que suit nous détaillons la structuration des ontologies et leurs constituants de base.

II.2.2 Structuration des ontologies

Les ontologies sont basées sur l'utilisation de certains nombres de composantes de base: concepts, relations, fonctions, axiomes et instances. Nous présentons plus en détail la définition et les propriétés de chaque composante.

II.2.2.1 Composantes des ontologies

Les connaissances traduites par une ontologie sont véhiculés à l'aide d'un certain nombre de composantes ou briques. Elles sont principalement des concepts, des relations, des fonctions, des axiomes et des instances [Gruber 92] et [Izza 06].

- **Les concepts**, aussi appelés termes ou classes de l'ontologie, constituent les objets de base manipulés par les ontologies. Ils correspondent aux abstractions pertinentes du domaine du problème.
- **Les relations** traduisent les interactions existantes entre les concepts. Des relations comme les relations de spécialisation (subsomption), les relations de composition, les relations d'instanciation, etc. sont définies entre les concepts.
- **Les fonctions** sont des cas particuliers de relations dans lesquelles le $n^{ième}$ élément de la relation est défini de manière unique à partir des $(n-1)^{ième}$ éléments précédents.
- **Les axiomes** permettent de modéliser des assertions toujours vraies, à propos des abstractions du domaine traduites par l'ontologie. Ils permettent de combiner des concepts, des relations et des fonctions pour définir des règles d'inférences. Ces axiomes peuvent intervenir dans la déduction ainsi que dans la définition des concepts et des relations.
- **Les instances** (encore appelée individus) constituent la définition extensionnelle de l'ontologie. Ils représentent des éléments singuliers véhiculant les connaissances à propos du domaine du problème.

II.2.2.2 Propriétés des concepts

Un concept est généralement défini par un terme, une intention et une extension. Le terme (le nom) correspond à l'identité du concept. L'intention (la notion) du concept contient la sémantique du concept exprimée en termes de propriétés, d'attributs, de règles et de contraintes. L'extension du concept regroupe les objets manipulés à travers le concept; ces objets sont appelés instances du concept [Bachimont 00].

Il est possible d'associer aux concepts un certain nombre de propriétés qui peuvent porter aussi bien sur l'extension que sur l'intention. Les principales propriétés sont énumérées comme suit [Furst 02]:

- **La généricité**: un concept est générique s'il n'admet pas d'extensions.
- **L'identité**: un concept porte une propriété d'identité si cette propriété permet d'identifier les instances de ce concept.
- **La rigidité:** un concept est rigide si toute instance de ce concept est aussi une instance de toutes les classes de l'ontologie.
- **L'anti-rigidité**: un concept est anti-rigide si toute instance de ce concept est définie par son appartenance à l'extension d'un autre concept.

En plus de ces propriétés intrinsèques, nous avons fait référence à des propriétés inter-concepts. Celles-ci portent sur des propriétés extrinsèques aux concepts. Il s'agit principalement de [Furst 02]:

- **L'équivalence**: deux concepts sont équivalents s'ils ont la même extension.
- **La disjonction**: deux concepts sont disjoints si leurs extensions sont disjointes.
- **La dépendance** : un concept est dépendant d'un deuxième concept si les instances de premier concept sont dépendantes des instances du deuxième.

II.2.2.3 Propriétés des relations

Tout comme pour les concepts, il existe aussi pour les relations un certain nombre de propriétés. Elles sont principalement les propriétés intrinsèques, les propriétés interrelations, et les propriétés liant une relation et des concepts [Furst 02] et [Izza 06].

Les propriétés intrinsèques à une relation permettent de décrire une relation. Nous distinguons principalement :

- **Les propriétés algébriques** : symétrie, réflexivité, transitivité.
- **La cardinalité** : nombre de participations possibles d'une instance d'un concept dans une relation.

Les propriétés interrelations portant sur plusieurs relations, qui peuvent être:

- **L'incompatibilité** : deux relations sont incompatibles si elles ne peuvent pas lier les mêmes instances d'un concept.
- **L'inverse** : deux relations binaires sont inverses l'une de l'autre si la liaison est faite dans les deux sens.

Les propriétés liant une relation et des concepts sont:

- **Le lien relationnel** : il existe un lien relationnel entre une relation R et deux concepts C1 et C2 si, pour tout couple d'instances des concepts C1 et C2, il existe une relation de type R qui lie les deux instances de C1 et C2.
- **La restriction de relation** : pour tout concept de type C1 et toute relation de type R liant C1, les autres concepts liés par la relation sont d'un type imposé.

II.2.3 Typologie des ontologies

Les ontologies peuvent être classifiées selon quatre catégories [Psyché et al 03] et ceci en fonction de (Cf. la Figure II-1) :

- ✓ l'objet de conceptualisation,
- ✓ le niveau de détail,
- ✓ le niveau de complétude,
- ✓ et le niveau de formalisme.

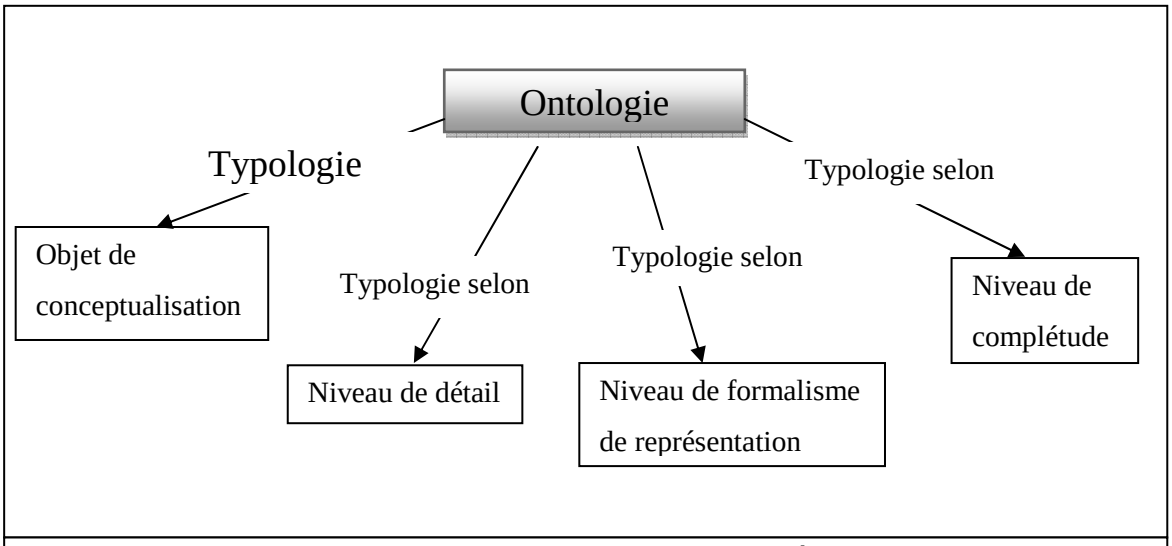

Figure II-1: typologie d'ontologies [Psyché et al 03]

Une autre classification de Guarino mentionnée dans [Mellal 07] permet de définir quatre catégories d'ontologies, à savoir (Cf. la Figure II-2) :

- ✓ les ontologies de haut niveau,
- ✓ les ontologies de domaine,
- ✓ les ontologies de tâche,
- ✓ les ontologies d'application.

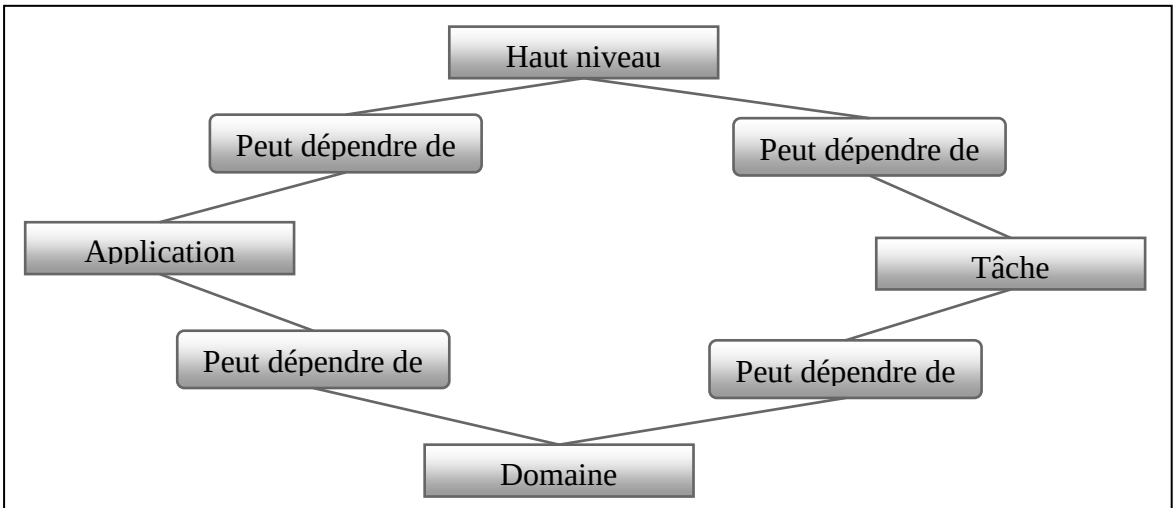

Figure II-2: typologie d'ontologie selon Guarino [Mellal 07].

II.2.4 Construction des ontologies

La construction des ontologies est une tâche très complexe, impliquant plusieurs acteurs, diverses connaissances et formalismes. Afin de maîtriser cette complexité, la mise en place d'une méthodologie peut se révéler nécessaire. L'utilisation d'une méthodologie a pour objectif de rationaliser et d'optimiser le processus de développement dans le sens où il permet de réduire les délais et les coûts ainsi que l'amélioration de la qualité du processus et du produit de développement.

À l'heure actuelle, nous pouvons recenser dans la littérature une multitude de méthodologies de construction [Bala 07]. Cependant, il n'existe pas encore de consensus en matière de normes de construction. Et par conséquent, il n'existe pas encore de méthodes universellement reconnue pour la construction d'ontologies [Uschold et al 96]. Ceci relève beaucoup plus du savoir-faire que de l'ingénierie. En dépit de cela, de nombreux critères et principes permettant de guider la construction d'ontologies ont été proposés. Certains sont décrits dans les prochaines sections.

II.2.4.1 Cycle de développement ontologique

Le développement des ontologies doit s'appuyer sur les mêmes principes que ceux appliqués en génie logiciel. Elles doivent avoir, donc, un cycle de vie qui nécessite d'être spécifié.

Quelque soit les méthodologies de construction utilisées, le cycle de vie ontologique comprend les étapes suivantes [Furst 02] et [Izza 06]:

- Evaluation des besoins.
- Construction (conceptualisation, ontologisation et opérationnalisation).
- Diffusion.
- Utilisation (Cf. la Figure II-3).

Après chaque utilisation significative, l'ontologie et les besoins sont réévalués. Ensuite, l'ontologie peut être étendue et, si nécessaire, en partie reconstruite. L'étape d'ontologisation peut être complétée par une étape d'intégration.

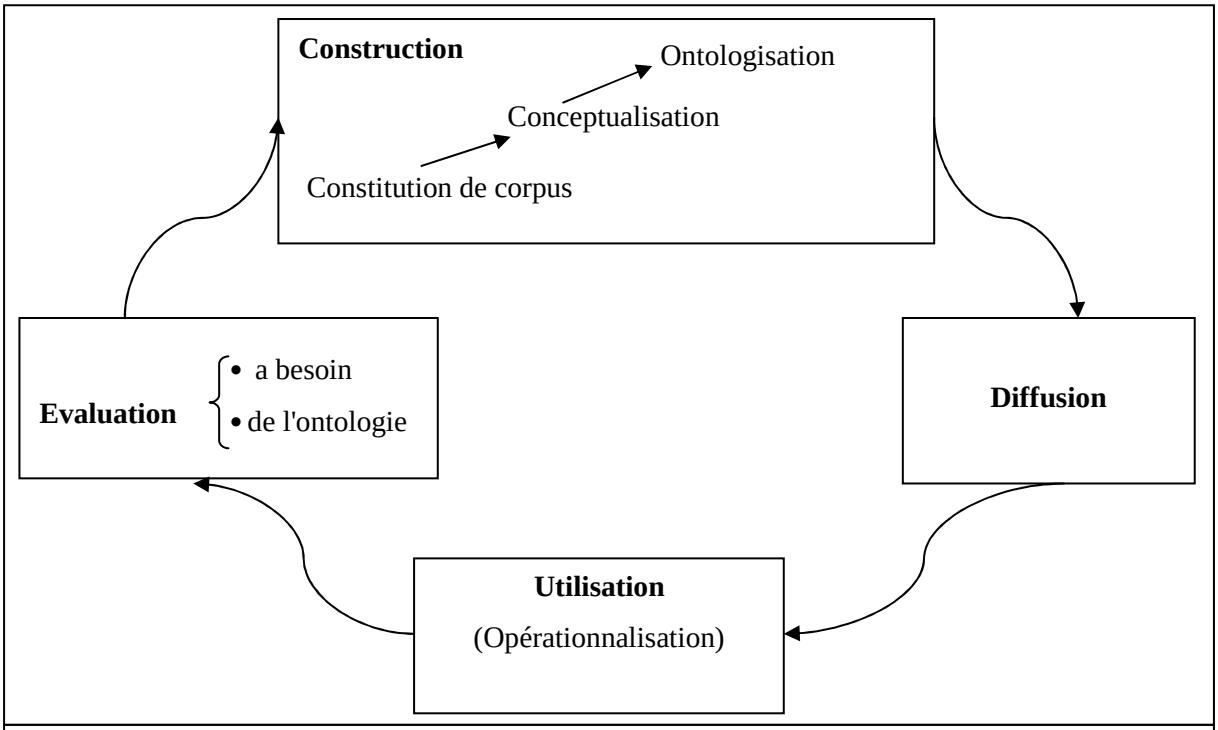

Figure II-3: cycle de vie d'une ontologie.

II.2.4.2 Principes pour la construction des ontologies

Plusieurs principes ont été proposés dans la littérature afin de mieux guider la construction des ontologies. Il s'agit essentiellement, des principes de Gruber [Gruber 93] :

- **Clarté et objectivité** : l'ontologie devrait fournir des définitions claires et objectives pour les termes, indépendantes de tout choix d'implémentation.
- **Cohérence** : consistance des axiomes afin de pouvoir formuler, par la suite, des inférences cohérentes.
- **Extensibilité**: c'est à dire la possibilité d'étendre l'ontologie sans modification.

II.2.4.3 Méthodologies de construction des ontologies

Une méthodologie étant considérée comme ensemble de principes de construction systématiquement reliés, appliqués avec succès par un auteur (ou plusieurs) dans la construction d'ontologies. Selon [Bala 07], il existe une trentaine de méthodologies de développement d'ontologies.

Ces méthodologies permettent la construction d'ontologies :

- à partir du début,
- par intégration ou fusion avec d'autres ontologies,
- par réingénierie,

- par construction collaborative,
- et l'évaluation des ontologies construites.

Ici, nous ne citons que la méthodologie d'Uschold [Uschold et al 96] qui nous parait la plus représentative.

II.2.4.3.1 La méthodologie d'Uschold

La méthodologie de d'Uschold [Uschold et al 96], est composée de quatre principales étapes (Cf. la Figure II-4):

- Identifier le but et la portée de l'ontologie.
- Construire l'ontologie.
- Evaluer l'ontologie.
- Et documenter l'ontologie.

Le processus de construction repose sur la capture et le code de connaissances, et éventuellement leur intégration à d'autres ontologies.

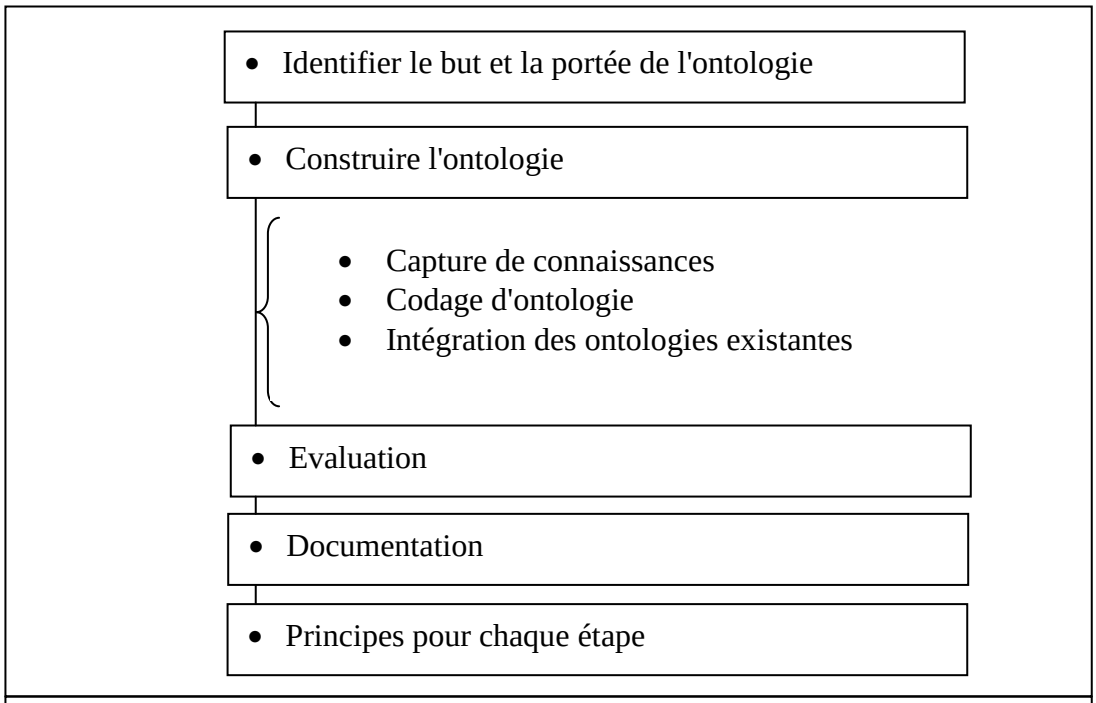

Figure II-4: la méthode d'Uschold[Uschold et al 96].

II.2.4.4 Langages de présentation des ontologies

Il existe de nombreux langages informatiques, plus ou moins récents, spécialisés dans la création et la manipulation des ontologies comme RDF [Manola et al 99], RDFS [Guha 04], OWL [Deborah et al 04], etc. Nous en décrivons quelques-uns dans la suite:

- **RDF**

RDF (*Resource Description Framework*) [Manola et al 99]**,** est un modèle de graphe destiné à décrire, de façon formelle, les ressources Web et leurs métadonnées et permettre le traitement automatique de telles descriptions.

- **RDFS**

RDFS (*Resource Description Framework Schema*) [Guha 04]**,** est un langage extensible qui permet la représentation des connaissances. Il appartient à la famille des langages du Web sémantique publiés par le W3C. Il fournit des éléments de base pour la définition d'ontologies ou de vocabulaires destinés à structurer des ressources RDF.

- **OWL**

OWL (*Web Ontology Language*) [Deborah et al 04], est apparu plus tard. C'est une expression XML fondé sur une syntaxe RDF. Il fournit les moyens pour définir des ontologies Web structurées. Il se différencie du couple RDF / RDFS par le fait que c'est un langage d'ontologies, contrairement à RDF. Si RDF et RDFS apportent à l'utilisateur la capacité de décrire des classes et des propriétés, OWL intègre, en plus, des constructeurs de comparaison des propriétés et des classes: identité, équivalence, cardinalité, symétrie, transitivité, disjonction, etc. Ainsi, OWL offre aux machines une plus grande capacité d'interprétation du contenu Web que RDF et RDFS, grâce à un vocabulaire plus large et à une vraie sémantique formelle.

II.2.4.5 Environnements et outils de développement d'ontologie

Les éditeurs d'ontologie constituent des outils nécessaires aidant à la construction des ontologies. Il existe différents éditeurs d'ontologie, les plus connus sont :

ONTOLINGUA [13]: le serveur Ontolingua est le plus connu des environnements de construction des ontologies. Il consiste en un ensemble d'outils et de services qui supportent la construction en coopération des ontologies, par des groupes séparés géographiquement. Le langage ontologique utilisé dans Ontolingua est KIF [Genesereth 05].

[13] ONTOLINGUA: http://www.ksl.stanford.edu/software/ontolingua/.

PROTEGE [14]: est un environnement graphique de développement d'ontologie le plus utilisé pour construire les ontologies avec multiples langages. C'est un logiciel libre d'utilisation qui peut être modifié par l'utilisateur. Il est alors possible de réaliser des modules additionnels (plugins) pour modifier ou compléter ce logiciel. Il regroupe aujourd'hui une communauté d'utilisateurs assez importante et constitue une référence pour beaucoup d'autres outils. Protégé est un éditeur ontologique pour les différents langages à savoir RDF, RDFS et OWL.

II.3 Services Web sémantiques

Les services Web sémantiques sont des services Web simples dont la description WSDL est augmentée par des descriptions supplémentaires. Ces descriptions sont explicitées dans des ontologies à part, afin que d'autres applications ou d'autres services Web puissent comprendre non seulement la syntaxe des fonctionnalités offertes mais aussi la sémantique des ces derniers qui renforce l'interopérabilité. En effet, actuellement, les services Web sont décrits par le langage WSDL qui permet de définir les opérations et les fonctionnalités offertes par le service de manière seulement syntaxique, Cependant, lorsqu'on veut automatiser les différents aspects liés aux services Web comme la composition, l'invocation et la sélection, les clients de ces derniers (applications et services Web) ne peuvent pas avoir les sémantiques des fonctionnalités offertes. Dans ce cas le développeur qui veut interroger un service Web via son application doit tout d'abord avoir une connaissance préalable du sens de la syntaxe en le gérant manuellement.

Les services Web sémantiques sont des services Web décrits de telle sorte qu'un client (applications ou autres services Web) peut interpréter leurs fonctionnalités offertes. Pour permettre cela, la description syntaxique du service Web doit être augmentée en information sémantique et exploitable par machine (application ou service) et cela fait par la technologie d'ontologique.

Dans cette partie, nous présentons, proposées dans la littérature, quelques approches de réalisation des services Web sémantiques avec une brève comparaison entre ces différentes approches afin de ressortir l'approche la plus adéquate pour notre problématique.

[14] PROTEGE: http://protege.stanford.edu/.

II.3.1 Approches pour la réalisation des services Web sémantique

Les approches de réalisation des services Web sémantique selon [Mrissa 07] se divisent en deux catégories, la première catégorie consiste à développer un langage complet qui décrit le service Web et sa description sémantique dans un seul bloc, on peut citer dans cette catégorie les approches OWL-S [Martin et al 04] et WSMO [Lausen et 05]. La deuxième catégorie consiste à annoter les langages existants avec la description sémantique parmi les approches qui s'inscrivent dans cette catégorie WSDL-S [Akkiraju et al 05] et SAWSDL [Farrell et al 07]. Nous ne présenterons dans la suite que les quatre approches suivantes OWL-S, WSMO, WSDL-S et SAWSDL.

II.3.1.1 OWL-S (Web Ontology Language for Services Web)

OWL-S (*Web Ontology Language for Services Web*) [Martin et al 04] est une ontologie dédiée à la description sémantique de services Web [Martin et al 04]. OWL-S se base sur le langage standard OWL [Deborah et al 04] (Cf. la section II.2.4.4 de chapitre II) qui permet de représenter des ontologies de façon standardisées sur le web. L'objectif de OWL-S est de formaliser de façon non ambiguë les services Web de manière à ce qu'un agent logiciel (une application ou un autre service web considère un client de service) puisse exploiter automatiquement les informations concernant ces services Web.

OWL-S décrit la sémantique d'un service Web à l'aide de trois classes *ServiceProfile, ServiceModel* et *ServiceGrounding* comme illustré sur la Figure II-5.

- *ServiceProfile*: exprime ce que le service Web propose.
- *ServiceModel*: exprime le fonctionnement du service Web.
- *ServiceGrounding*: exprime comment le service Web peut être utilisé.

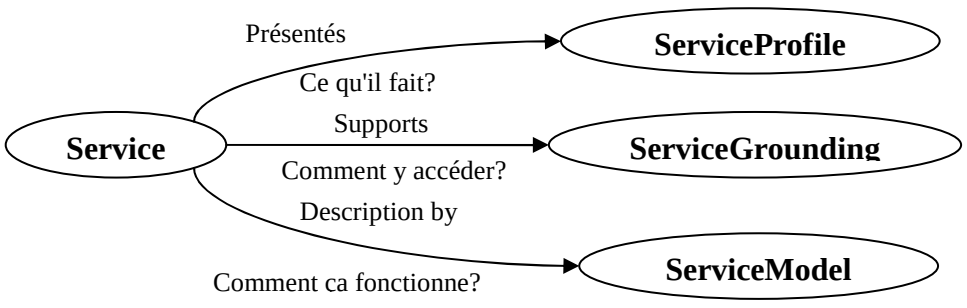

Figure II-5: ontologie OWL-S [Martin et al 04].

- ***ServiceProfile*:** la classe *ServiceProfile* d'un service OWL-S spécifie trois informations pour exprimer ce que le service Web propose.
    - Nom du service Web, contacts et description textuelle du service : le nom du service est utilisé comme identificateur du service et les informations contacts et la description textuelle sont destinées aux utilisateurs humains.
    - Description fonctionnelle du service : elle spécifie ce que le service Web exige en termes d'entrées (inputs) et de résultats en sortie (outputs), elle spécifie aussi les préconditions et les effets du service Web.
    - Classification taxinomique.
- ***ServiceModel* :** la classe *ServiceModel* décrit le fonctionnement du service Web. Ceci est fait en exprimant les transformations faites par le service Web sur les données (input à output), et transformation d'état (préconditions et effets).

Les services Web peuvent être modélisés avec OWM-S en tant que processus grâce à la classe *Process*. La classe *Process* est une sous-classe de la classe *ServiceModel*. Pour décrire un processus, on spécifie les entrées et sorties et ses états. Les transitions d'un état à un autre décrites par les préconditions et les effets de chaque processus.

La classe *Process* distingue trois types de processus à savoir :

- Les processus atomiques (*AtomicProcess*): exécutable en une seule étape, un processus atomique ne peut pas être décomposé de façon plus profonde. Il est invoqué directement par l'utilisateur du service.
- Les processus composites (*CompositeProcess*): un processus composite est constitué par l'assemblage d'autres processus (composite ou non composite). Les processus composites associent des processus à l'aide de structures de contrôle permettant de décrire leur logique d'exécution. Les structures de contrôle sont les suivantes :
    - ✓ séquence (*Sequence*) représente une suite ordonnée de processus,
    - ✓ exécutions concurrentes de processus sont décrites par *Split*,
    - ✓ synchronisation peut être décrite par *Split+Join*,
    - ✓ exécution de processus sans ordre particulier avec *Unordered*,
    - ✓ le choix est décrit par *Choice*,
    - ✓ les branchements conditionnels du type si/alors/sinon sont décrits par *If-Then-Else*,
    - ✓ *Repeat, Iterate et Repeat-Unti* permettent d'effectuer des itérations.

- Les processus simples (*SimpleProcess*): les processus simples ne sont pas invocables comme les processus atomiques, mais, leurs exécutions s'effectuent en une seule étape. Les processus simples sont employés comme éléments d'abstraction. Un processus simple peut être employé pour fournir une vue d'un certain processus atomique, ou une représentation simplifiée d'un certain processus composé.

- ***ServiceGrounding*:** la classe *ServiceGrounding* d'un service OWL-S définit les détails techniques permettant d'accéder au service Web. Les deux premières classes *ServiceProfile* et *ServiceModel* d'une description OWL-S représentent la forme abstraite d'un service Web. La *ServiceGrounding* est la forme concrète d'une représentation abstraite, elle fournit les détails concrets d'accès au service Web, tels les protocoles, les URIs, les messages envoyés.

II.3.1.2 WSMO (Web Service Modeling Ontology)

WSMO (*Web Service Modeling Ontology*) [Lausen et 05] est une architecture conceptuelle visant à expliciter la sémantique des services Web. Elle est organisée en quatre éléments principaux comme présenté sur la Figure II-6 :

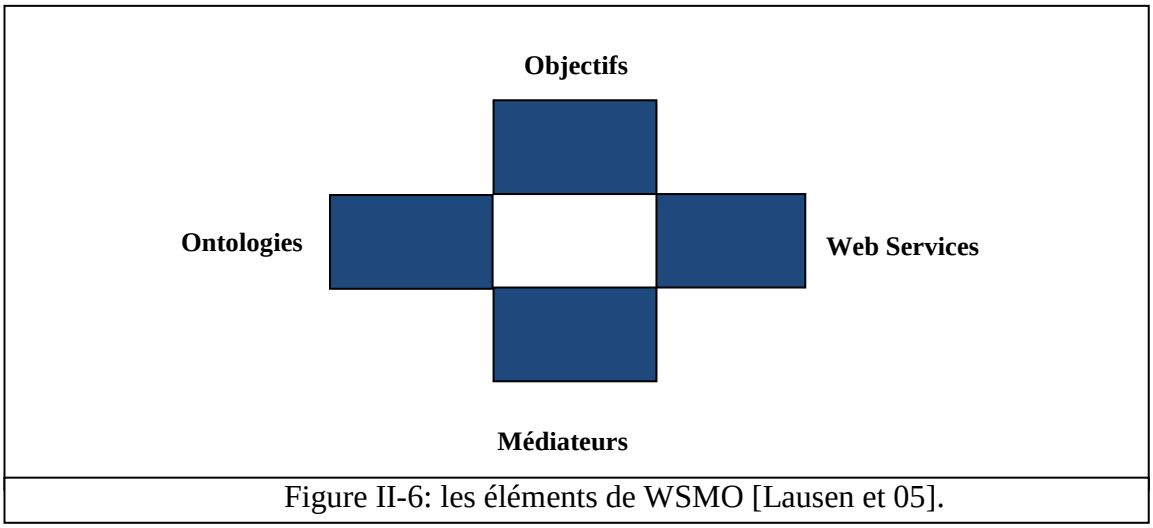

Figure II-6: les éléments de WSMO [Lausen et 05].

- **Les services Web**: sont définis comme des entités qui fournissent une fonctionnalité. Une description est associée à chaque service, dans le but de décrire sa fonctionnalité, son interface, et ses détails internes.
- **Les objectifs**: servent à décrire les souhaits des utilisateurs en termes de fonctionnalités requises. Les objectifs sont une vue orientée utilisateur du processus d'utilisation des

services Web, ils sont une entité à part entière dans le modèle WSMO. Un objectif décrit la fonctionnalité, les entrées, les sorties, les préconditions et les postconditions d'un service Web.

- **Les médiateurs**: sont utilisés pour résoudre de nombreuses incompatibilités, telles que les incompatibilités de données dans le cas où les services Web utilisent différents terminologies, les incompatibilités de processus dans le cas de la combinaison de services Web, et les incompatibilités de protocoles lors de l'établissement des communications.
- **Les ontologies**: fournissent la terminologie de référence aux autres éléments des WSMO, afin de spécifier le vocabulaire du domaine de connaissance d'une manière interopérable par les machines.

II.3.1.3 WSDL-S (Web Service Description Language-Semantic)

WSDL-S (*Web Service Description Language-Semantic*) [Akkiraju et al 05] est un langage WSDL augmenté de sémantique, cette sémantique est ajoutée en deux étapes, La première consiste à faire référence, dans la partie définition WSDL à une ontologie dédiée au service à publier; La deuxième consiste à annoter les opérations de définition WSDL de sémantique (Cf. la Figure II-7).

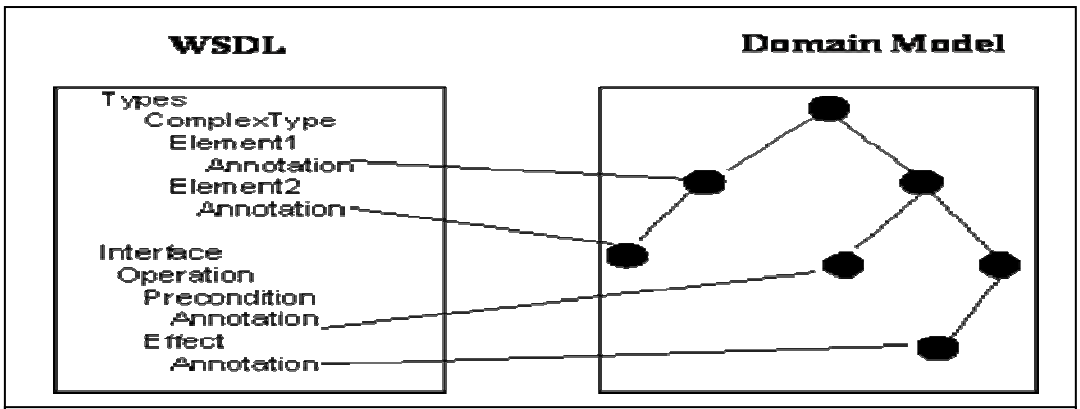

Figure II-7: WSDL augmenté de la sémantique [Akkiraju et al 05].

WSDL-S distingue quatre modèles sémantiques [Kopecký 05] à savoir:

- ***InputSemantic***: le sens des paramètres d'entrée.
- ***OutputSemantic***: le sens des paramètres de sortie.
- ***Precondition***: un ensemble d'états sémantiques qui doivent être vrais afin d'invoquer une opération avec succès.

- ***Effect***: un ensemble d'états sémantiques qui doivent être vrais après qu'une opération réalise son exécution.

WSDL-S utilise l'extensibilité de WSDL et fournit les cinq éléments suivants:

- ***ModelReference***: Un attribut pour lequel la valeur d'URL désigne un concept dans le modèle sémantique des services Web et spécifie qu'il y a une correspondance entre le propriétaire de l'attribut et le concept référencé.
- ***SchemaMapping***: un attribut pour lequel l'URL indique une correspondance entre les concepts d'ontologie.
- ***Precondition* et *Effect***: Ils sont utilisés pour spécifier les préconditions et les postconditions d'une opération donnée. Les conditions sont spécifiées par une référence au modèle sémantique (réutilisation de l'attribut *ModelReference*).
- **Category**: un attribut qui fournit un pointeur vers certaine catégorie de taxonomie.

II.3.1.4 SAWSDL (Semantic Annotations for WSDL and XML Schema)

SAWSDL (*Semantic Annotations for WSDL and XML Schema*), est la suite de WSDL-S [Farrell et al 07]. Il a été conçu pour spécifier les éléments de WSDL à l'aide des ontologies de manière plus standardisée. Le mécanisme d'annotation de SAWSDL est décrit de manière indépendante de tout langage de représentation d'ontologies.

SAWSDL propose deux sortes d'annotations sémantiques: une pour identifier les concepts sémantiques référencés par l'attribut *modelReference* et une seconde pour faire le lien entre les concepts et le fichier WSDL, représenté par les attributs *liftingSchemaMapping* et *loweringSchemaMapping*. L'attribut *modelReference* permet d'annoter tous les éléments de langage WSDL, à une ou plusieurs URI référençant des ontologies qui explicitent la sémantique aux éléments annotés. Ce type d'annotation est utilisé dans divers aspects liés aux services Web telles que la recherche, la sélection, la composition et l'invocation. Dans le cas où les concepts sémantiques associés aux éléments WSDL sont explicités dans le format XML, le code peut directement être intégré dans le fichier.

L'approche SAWSDL permet d'ajouter facilement une description sémantique au langage WSDL. L'avantage de cette initiative par rapport au WSDL-S est la standardisation

ainsi qu'elle permet d'annoter les WSDL existants sans trop alourdir le fichier de description initiale.

II.3.2 Comparaison des approches

Nous résumons, dans le Tableau II-1, les principales caractéristiques des principales approches de la réalisation des services Web sémantiques présentées précédemment. Les critères de comparaison sont :

- Niveau d'abstraction: qui permet de préciser le niveau d'abstraction (conceptuel, technique) associée à l'approche.
- Niveau d'ouverture: qui permet de représenter le degré de standardisation permis par l'approche.
- Description sémantique: qui permet de préciser la manière avec laquelle les services Web sont décrits sémantiquement.

Les critères	OWL-S	WSMO	WSDL-S	SAWSDL
Niveau d'abstraction	Conceptuel (basée sur une ontologie générique de services Web)	conceptuel (basée sur MOF)	conceptuel (basé sur l'annotation sémantique de WSDL)	conceptuel (basé sur l'annotation sémantique de WSDL et BPEL)
Niveau d'ouverture (Standardisation)	très forte (basé sur OWL, WSDL)	faible	forte (basée sur WSDL, mais WSDL-S n'est pas standard)	forte (basée sur WSDL et BPEL)
Description sémantique	Ontologie générique OWL-S	Ontologie WSMO	Annotation des Fichiers WSDL	Annotations des fichiers WSDL et BPEL

Tableau II-1: principales caractéristiques des approches de réalisation des services Web sémantiques.

L'analyse du Tableau II-1 permet de retenir certains points importants qui sont:

- OWL-S: constitue une ontologie générique de services Web, permettant principalement de décrire la sémantique des services Web dans un cadre général. OWL-S est basée sur le standard OWL et compatible avec les autre standards notamment WSDL, et ce grâce au *ServiceGrounding* qui permet de définir des mappings entre OWL-S et WSDL.
- WSMO: constitue une approche basée sur des fondements conceptuels pour la réalisation des services Web sémantiques, elle est basée aussi sur des langages propriétaires. L'une des limites les plus importantes est le fait que cette approche ignore les

standards existants, ce qui suppose que des mappings doivent être définis afin de garantir l'intégration avec les standards existants.

- WSDL-S, SAWSDL: l'idée de ce type d'approches est l'enrichissement sémantiques des langages existants. Ces approches sont avantageuses dans la mesure où elles exploitent les éléments d'extensibilité des langages existants. SAWSDL est très avantageant en termes de standardisation, de simplicité et de la manière d'annotation qui n'alourdit pas le fichier de description.

II.4 Conclusion

Les ontologies sont une solution idéale pour modéliser la connaissance de manière plus formelle. Elles ont été utilisées pour expliciter la sémantique des fonctionnalités offertes par les services de telle sorte qu'une machine (application ou autres services Web) puisse comprendre leurs significations. Les services Web sémantiques ont été conçus pour automatiser les divers aspects liés aux services Web telle que la composition. En effet, sans descriptions sémantiques, les hétérogénéités entre les services Web sont gérées de façon manuelle. Dans ce cas, les développeurs ont besoin des connaissances préalables pour réaliser un processus métier où les services Web sont hétérogènes.

Dans le chapitre suivant, nous abordons le problème d'hétérogénéité entre les services Web engagés d'une composition. Nous citerons les travaux qui ont été proposés dans la littérature afin de résoudre ces derniers par la médiation, nous concentrerons plus précisément sur l'hétérogénéité entre les données échangées dans une composition en étudiant les différentes approches de médiations de données proposées.

Chapitre III

III. Médiations de services Web

III.1 Introduction

Un déroulement d'une composition de services Web conduira la plupart du temps à des échecs à cause des hétérogénéités entre les services Web engagés dans la composition. Une médiation de services Web est nécessaire pour résoudre les hétérogénéités présentes entre ces derniers afin de permettre des interactions réussies. Dans ce chapitre, nous étudions les hétérogénéités entres les services Web et les types de médiations de services Web, ensuite nous détaillerons les types d'hétérogénéités de données échangées entre les services Web et les approches de médiation proposées dans la littérature pour résoudre ces hétérogénéités. De plus, nous réalisons une synthèse sur ces approches pour clarifier la nécessité d'une approche sémantique orientée contexte pour résoudre notre problématique.

III.2 Hétérogénéités dans une composition de services Web

Les services Web n'ont pas initialement été réalisés pour interagir les uns avec les autres ; en effet, de nombreuses hétérogénéités apparaissent au niveau des fonctionnalités souhaitées, au niveau de données échangées entre les services Web composés, ainsi qu'au niveau comportemental des services Web. Ces conflits peuvent générer des dysfonctionnements pendant les trois étapes de composition (la découverte, la spécification de processus métier et l'exécution de composition):

- **Découverte** : les fonctionnalités sélectionnées du service Web ne correspondent pas toujours aux besoins de la composition ; en effet, la plupart des fonctionnalités offertes par les services Web sont décrites de manière hétérogène.
- **Spécification de processus métier:** plusieurs hétérogénéités apparaissent dans cette étape, généralement de type comportemental tel que :
    - incompatibilités entre les types d'interactions différents (requête simple, requête avec réponse, ...etc.).

- échanges des messages selon différentes séquences.
- incompatibilités des protocoles de transport.
- incompatibilités entre les degrés de sécurité et les propriétés transactionnelles.

- **Exécution de la composition:** les données échangées entre les services Web engagés dans une composition peuvent être gênés par des hétérogénéités qui empêchent l'exécution de la composition, en effet, les données transportées entre les services Web peuvent présenter des hétérogénéités d'ordre syntaxique, structurel, sémantique, et même contextuelle.

III.3 Médiations entre services Web

La résolution des hétérogénéités s'appuie généralement sur des techniques de médiation, visant à résoudre les conflits entre deux acteurs au minimum.

Dans le domaine de services Web, la fonction d'un médiateur consiste à résoudre les hétérogénéités entre les services Web. Il existe trois types de médiation pour résoudre ces hétérogénéités qui sont: la médiation de données, la médiation de fonctionnalité, et la médiation de processus métier [Cabral et a 05]:

La médiation des fonctionnalités: ce niveau de médiation concerne la correspondance entre les fonctionnalités demandées par le client et celles fournies par le service Web, ce type de médiation est utilisé dans l'étape de découverte.

La médiation de processus métier: ce niveau de médiation a pour objectif de résoudre l'hétérogénéité comportementale entre les services Web au sein de la composition, ce type de médiation concerne l'étape de spécification de processus métier.

La médiation de données: ce niveau de médiation concerne l'adaptation de données échangées entre les services Web, ce niveau permet de résoudre l'hétérogénéité liée aux données à savoir l'hétérogénéité : syntaxique, structurelle, sémantique, et contextuelle.

Dans la section suivante, nous nous focaliserons particulièrement sur la médiation de données qui est le cœur de notre problématique.

III.4 Médiation de données échangées entre services Web

L'idée principale de la médiation de données échangées entre les services Web est que tout envoi ou réception de données entre les services Web doit passer par un tiers qui joue le

rôle de médiateur (Comme illustré sur la Figure III-1). En d'autres termes, si un service Web envoie une donnée vers un autre service Web, la donnée passe par le médiateur avant que ce dernier ne la transmette à sa destination, après traitement si nécessaire [Lopez-velasco 08].

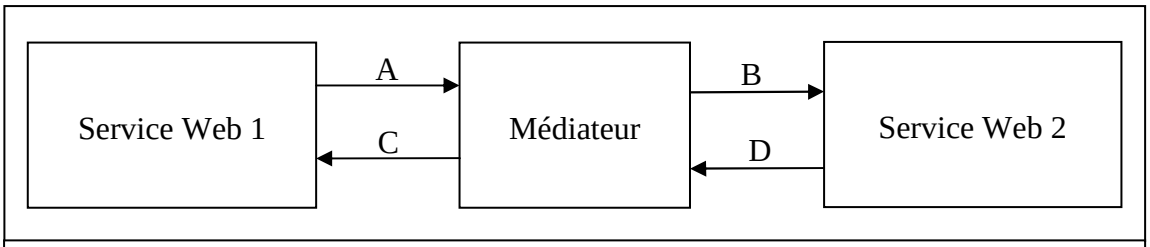

Figure III-1: principe d'un médiateur de données.

Le rôle de médiateur de données est la résolution des hétérogénéités (syntaxique, structurelle, sémantique et contextuelle) liées aux données, c'est-à-dire de réconcilier la conversation initiée entre les services Web engagés dans une composition.

III.4.1 Hétérogénéités de données échangées entre services Web

Comme nous l'avons mentionné précédemment, on distingue quatre niveaux d'hétérogénéités de données échangées entre services Web: syntaxique, structurel, sémantique (Cf. [Mrissa 07]) et contextuel (Cf. [Sattanathan 06]). Ces hétérogénéités s'appliquent à la fois aux attributs d'une donnée, en effet, une donnée possède trois attributs essentiels (Cf. la Figure III-2), qui sont : le nom de donnée, le type de donnée et la valeur de donnée.

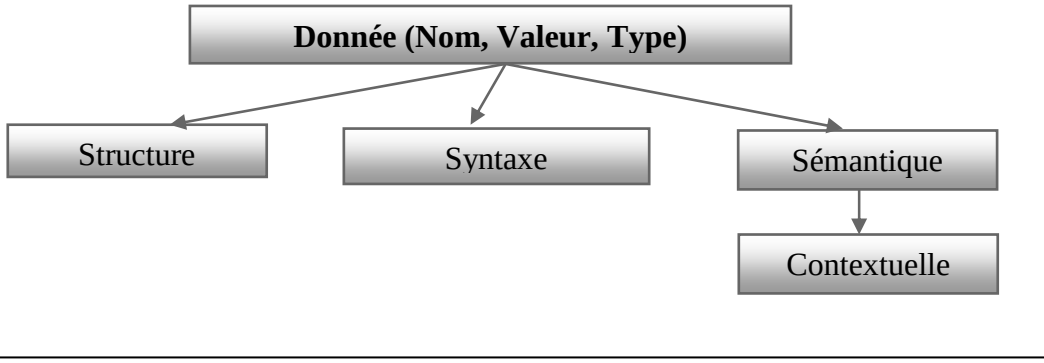

Figure III-2: les attributs d'une donnée et leurs hétérogénéités.

Certaines hétérogénéités sont résolues par l'utilisation des langages liés aux services Web, comme: XML, XML schéma, et les langages ontologiques comme OWL et RDF. D'autres hétérogénéités nécessitent un niveau de médiation pour les réconcilier et gérer leurs conflits, de la manière suivante:

Hétérogénéité syntaxique: cette hétérogénéité concerne les différences syntaxiques entres les langages de représentation des données. Cependant, cette hétérogénéité est résolue par l'utilisation du langage XML qui exige une syntaxe unifié pour décrire le nom, le type et la valeur de la donnée.

Hétérogénéité structurelle: cette hétérogénéité concerne les différences dans la représentation des données au niveau schéma et au niveau de structure de données. La solution pour cette hétérogénéité est de traiter chaque attribut de donnée que nous voulons décrire à part, par exemple, le nom d'une donnée est stocké dans le cas général dans une ontologie unique, donc sa structure est représentée sous la forme d'un triplet RDF ou OWL selon le langage ontologique utilisé. Le type d'une donnée est décrit par le langage XSD (XML schema). La valeur d'une donnée est une instanciation du XML schema, et sa structure est unique, seules les valeurs des instances changent pendant l'exécution.

Hétérogénéité sémantique: cette hétérogénéité concerne le sens véhiculé par la donnée. La solution adoptée pour ce problème est d'utiliser des ontologies partagées entre les services Web. Ces ontologies permettent de fixer le sens du nom d'une donnée utilisée dans la composition. La spécification XML schema permet de résoudre les conflits sémantiques liés aux types de données, en fait les significations des différents types de données sont décrites de manière explicite par cette dernière. Pour la sémantique des valeurs, la plupart des travaux existants (Cf.[Cabral et a 05] et [Haller et al 05]) considèrent que le sens lié au nom de donnée est suffisant pour interpréter correctement la valeur de ces données.

Hétérogénéité contextuelle: cette hétérogénéité concerne l'interprétation des données par rapport aux contextes où les données sont placées. L'hétérogénéité du niveau contextuel est un cas particulier de celle du niveau sémantique, seul l'attribut valeur peut poser des problèmes, en effet, la signification d'une valeur peut changer selon un contexte précis.

Dans la section suivante, nous présentons un état de l'art sur les solutions qui ont été proposées pour résoudre ces hétérogénéités. Nous présentons les approches de médiations de données structurelles, de médiations de données sémantiques et de médiations données orientées contexte. Le niveau syntaxique n'est pas traité puisque l'usage de langage XML ne nécessite pas un autre niveau de médiation.

III.4.2 Approches de médiations de données échangées entre services Web

Dans cette section, nous présentons les approches et les solutions de médiations qui ont pour but la résolution des hétérogénéités entre les données échangées entre services Web. Ces approches sont classées selon le niveau d'hétérogénéité prise en charge par ces dernières. En effet, des approches structurelles, des approches orientées sémantique et des approches sémantiques orientées contexte sont présentées. Pour l'hétérogénéité syntaxique, le langage XML assez suffisant pour gérer ce niveau, par conséquent, ce niveau ne nécessite pas une médiation supplémentaire. Notons que dans la littérature, il existe plusieurs travaux dans chaque catégorie des approches[Spencer et al 04], [Radetzki et al 04] et [Bowers et al 04] qui s'inscrivent dans la catégorie des approches de médiations de données de type structurel et [Cabral et a 05] et [Haller et al 05] dans la catégorie des approches de médiations de données de type sémantique, sauf que pour les approches de type sémantique orienté contexte, à notre connaissance, il n’existe qu'un seul travail qui est celui de [Mrissa 07]. Pour chaque catégorie, nous étudions un seul travail, en terminant par une synthèse des approches.

III.4.2.1 Approches de médiation de données structurelles

Dans [Spencer et al 04], les auteurs proposent une approche de médiation structurelle pour l'intégration des services Web. Ils considèrent que les données échangées entre les services Web sont sémantiquement compatibles grâce à l'utilisation de langage de description sémantique OWL-S (ce langage a été présenté dans le chapitre I section III.2.1), en effet, selon les auteurs, les données même sémantiquement compatibles, peuvent suivre des structures différentes. Il y a donc un besoin de réconcilier les représentations de ces données au niveau structurel. Le médiateur proposé dans ce travail est une file d'inférence qui utilise des règles logiques de transformation de données, il permet d'identifier les tâches suivantes:

- conversation et correspondances entre les types de données,
- réconciliation et restructuration de l’arborescence au niveau du schéma des messages,
- et conversation entre les concepts des ontologies.

Dans leur architecture globale, (Cf. la Figure III-3), les auteurs utilisent un compilateur de règles. Ce compilateur raisonne à partir des ontologies OWL-S, des descriptions WSDL

fournies par les services Web engagées dans la composition et des ontologies référencées par ces derniers, aussi que les règles de conversions entre les types de données. La somme de ces spécifications est utilisée pour générer les règles logiques de transformations. Une fois que les règles logiques ont été générées, un médiateur sous forme de file d'inférence est inséré entre les services Web engagées dans la composition. Ce médiateur reçoit les données du premier service Web émetteur, et envoie le résultat au service Web récepteur après réconciliation.

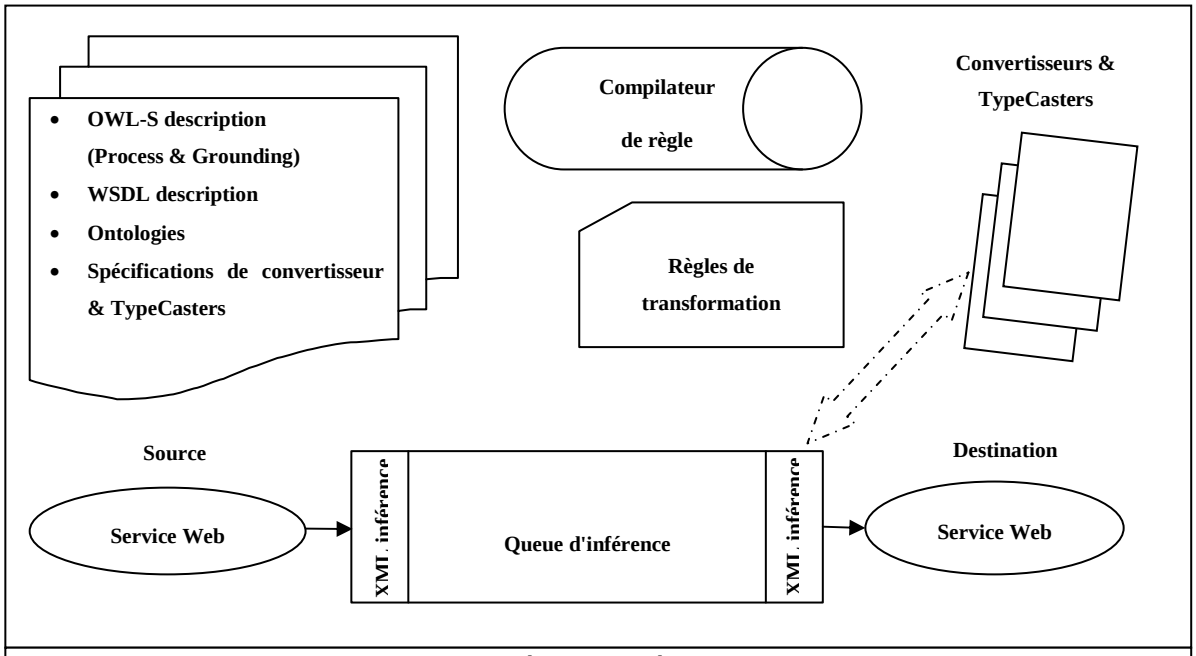

Figure III-3: architecture globale du système de médiation structurelle [Spencer et al 04].

III.4.2.2 Approches de médiation de données sémantiques

La plateforme d'IRS-III[15] contient trois types de médiateurs permettant de résoudre les hétérogénéités entre les services Web, qui sont : médiateur de but, médiateur de données et médiateur de processus (comme illustré sur la Figure III-4). Cabral et Domingue travaillent sur cette plateforme de composition. Ils proposent dans [Cabral et a 05] une solution de médiation de données au niveau sémantique.

[15] IRS-III: Plateforme de composition des services Web

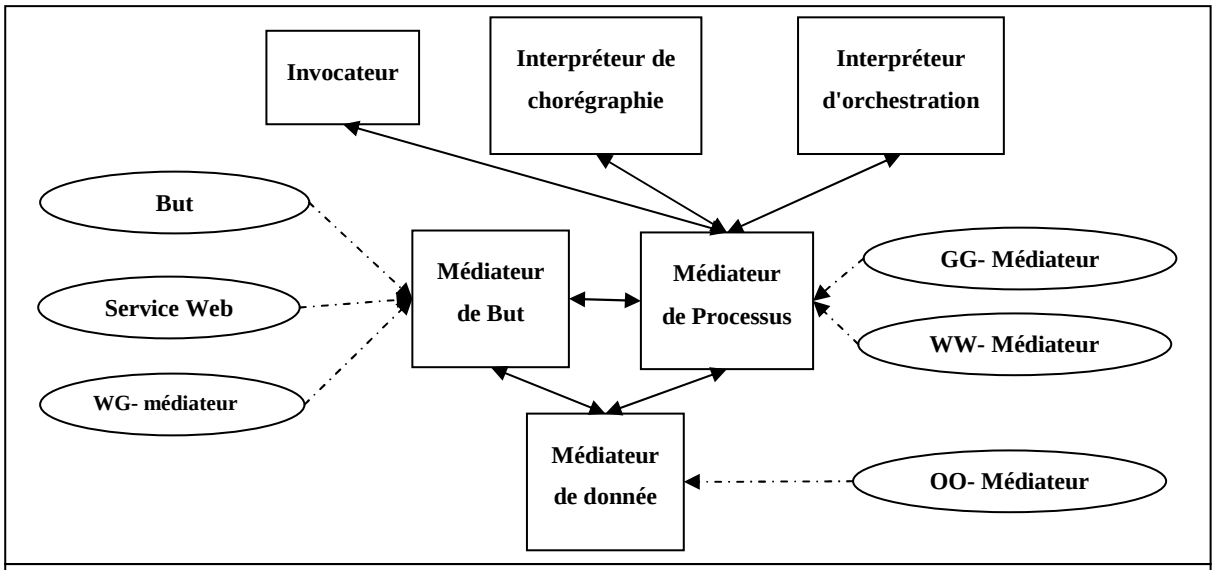

Figure III-4: framework de médiation d'IRS-III [Cabral et al 05].

À l'opposition de travail de Spencer et Liu (Cf. la section précédente), Cabral et Domingue supposent que l'enrichissement sémantique de services Web par des ontologies de domaines ne correspond qu'à une seule structure de données, donc par cette supposition les auteurs ignorent les incompatibilités structurelles et travaillent seulement sur la médiation sémantique. Le rôle de médiateur de données sémantiques proposé par les auteurs est d'effectuer la correspondance entre les termes des ontologies de domaines fournies avec les services Web.

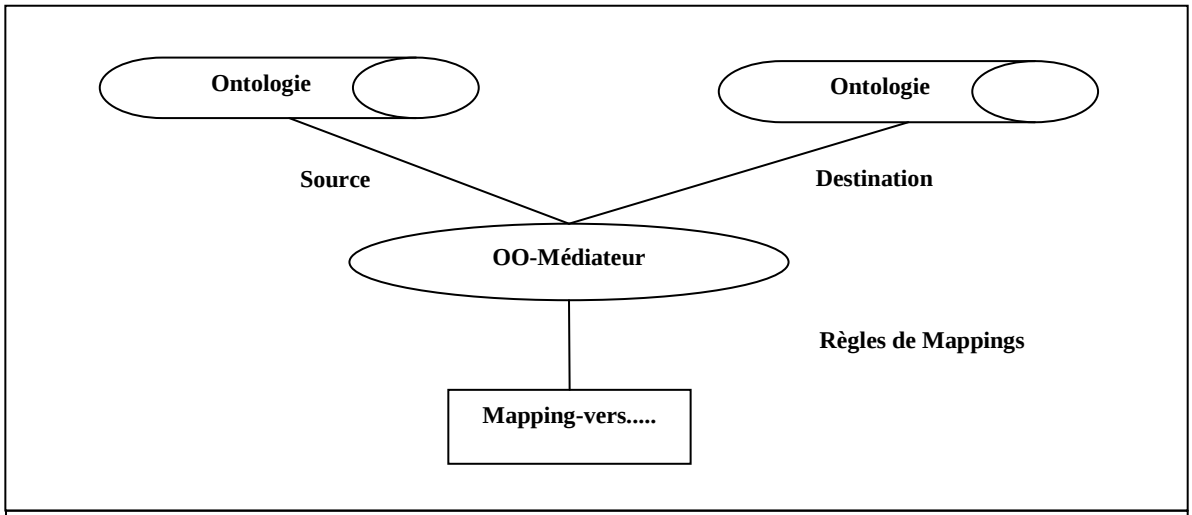

Figure III-5: médiation de données sémantique dans IRS-III [Cabral et al 05].

Les auteurs basent leur médiateur sémantique de données sur le langage OCML (*Operational Conceptual Modeling Language*) pour établir des correspondances entre les termes des ontologies de domaines fournies avec les services Web. Ces correspondances sont

stockées ensuite dans une ontologie instantanée, grâce à cette dernière le médiateur peut convertir les données d'une ontologie vers une autre.

III.4.2.3 Approches de médiation de données orientées contexte

Dans cette section nous présentons les approches de médiation de données basées sur le contexte, avant de détailler ces approches, nous devons présenter tout d'abord quelques notions liées au contexte de données comme la propriété sémantique et la valeur sémantique, ainsi que les modèles de représentations de contexte et enfin nous terminons par une vue plus détaillée sur ces approches.

III.4.2.3.1 Notion du contexte dans le Web service

Le contexte est utilisé souvent pour personnaliser et adapter les systèmes et les applications aux différents environnements. Sa définition de manière générale étant « tout élément interne ou externe, relatif à l'application, à l'utilisateur, ou même complètement extérieur, qui pourrait modifier le déroulement d'une interaction » [Dey et al 01].

Mrissa dans [Mrissa 07] a adapté cette définition au domaine lié au service Web et plus particulièrement aux données échangées entre services Web composés, « Le contexte d'une donnée englobe tout élément interne ou externe, relatif à la donnée ou même complètement extérieur, qui est nécessaire à l'interprétation correcte de la donnée » [Mrissa 07]. Dans le domaine de composition de services Web, deux contextes entrent en jeu lors d'un échange de données. Le premier est le contexte de données de service Web émetteur qui est lié aux environnements de service dans lequel les données ont été modélisées, alors que, le deuxième est le contexte de données de service destinataire lié aux environnements dans lequel les données doivent être interprétées. Par conséquent, la médiation de données de type contextuel consiste à transformer la donnée transmise du contexte dans lequel elle à été modélisée vers le contexte dans lequel elle doit être interprétée.

III.4.2.3.2 Notion de propriété sémantique

Un contexte est un ensemble d'éléments décrivant les divers aspects et caractéristiques sémantiques qui lui sont associés. Ces éléments sont appelés propriétés sémantiques, chaque propriété possède un nom et une valeur.

La Figure III-6 illustre le concept de date et son contexte qui comporte deux propriétés sémantiques, la première propriété sémantique est le format de date et sa valeur "dd/MM/yyyy", et la deuxième est le type de calendrier lui correspondant et sa valeur "Hijri",

cette propriété sémantique signifie que le calendrier est islamique (hégirien). On peut dire que le concept date est interprété sémantiquement dans ce contexte comme une date hégirien sous la forme "dd/MM/yyyy".

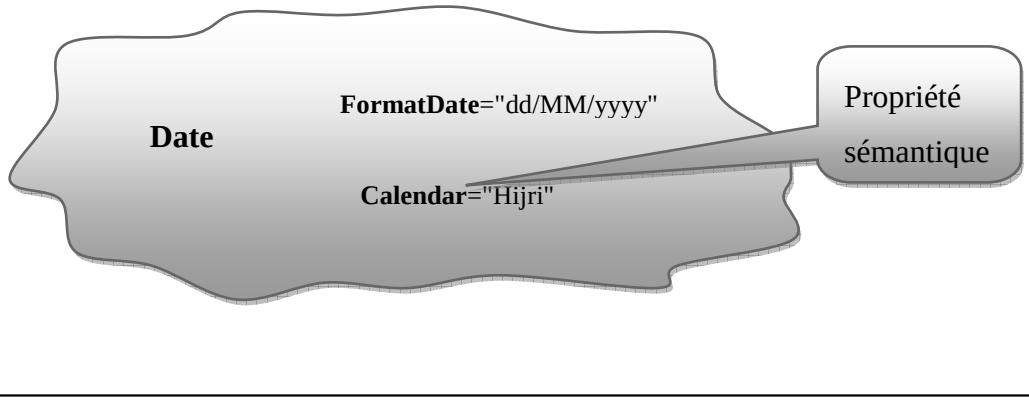

Figure III-6: exemple de concept de date et son contexte.

III.4.2.3.3 Notion de valeur sémantique

Le concept de valeur sémantique est introduit dans les travaux Sciore [Sciore et al 94]. Une valeur sémantique est une valeur simple associée à un contexte, le contexte (Cf. section III.4.2.3.1) d'une donnée est modélisé par un ensemble fini et récursif de méta-attributs qui peuvent contenir des valeurs différentes et se compose d'un schéma et d'une spécification. Le schéma décrit la structure du contexte, alors que la spécification spécifie les valeurs prises par une partie ou l'ensemble des méta-attributs dans un contexte précis. Par exemple, la valeur "10/02/1432" et le contexte (*FormatDate="dd/MM/yyyy" et Calendar="Hijri"*), Ce qui nous permet d'interpréter la valeur "10/02/1432" comme étant le dix du deuxième mois (Safar) de l'année mille quatre cent trente-deux du calendrier islamique.

III.4.2.3.4 Modèles de représentation du contexte

Selon [Chelbabi 06] (Cf. [Strangand et al 04]) six modèles ont été proposés dans la littérature pour représenter le contexte, à savoir: modèle Clé-Valeur, modèle Schéma Markup, modèle graphique, modèle logique, modèle orienté objet et modèle ontologique. Chaque modèle a des inconvénients et des avantages, certains modèles sont utilisés car ils sont adaptés à des domaines spécifiques et d'autres pour leurs simplicité et uniformité. Le modèle le plus simple à utiliser en termes de structure de donnée est le modèle Clé-Valeur mais son utilisation est très limitée. Quand les systèmes ont besoin d'un haut degré de formalisme

comme les systèmes basés logique, le modèle logique (Faits, expressions et règles) est convenable pour définir le contexte de ce dernier.

Le modèle graphique, le modèle orienté objet et le modèle Schéma Markup ont été adoptés dans certains travaux pour représenter le contexte (Cf. [Strangand et al 04]). Ainsi que le modèle ontologique (les ontologies ont été présentées dans la section II.2 du chapitre II) a été adopté pour décrire le contexte dans différents systèmes, effectivement, l'ontologie permet de décrire différents concepts et leurs relations. Elle est caractérisée par un formalisme hautement expressif et la possibilité d'utiliser des techniques de raisonnement.

III.4.2.3.5 Approche de Mrissa [Mrissa 07]

L'approche de [Mrissa 07], est la seule solution dans la littérature, à notre connaissance, qui a traité une telle problématique. L'auteur a proposé une approche de médiation de données basée sur le contexte. L'idée originale de son travail qui concerne le contexte de donnée est inspirée de celle proposée par la communauté de base de données (Cf.[Sciore et al 94]). Cette approche permet de résoudre les problèmes d'hétérogénéités sémantiques de données échangées entre les services Web au sein d'une composition en intégrant le contexte de données lors de l'exécution de la composition. Afin d'atteindre son objectif, l'auteur se base sur la notion de médiateur sémantique basé sur le contexte. Le rôle d'un médiateur de contexte dans le cadre de son travail est de résoudre les conflits sémantiques liés aux contextes des données. Le médiateur proposé par l'auteur est un service Web généré et inséré automatiquement pendant l'étape d'exécution de la composition. Son fonctionnement est réalisé en quatre étapes comme illustré sur la Figure III-7.

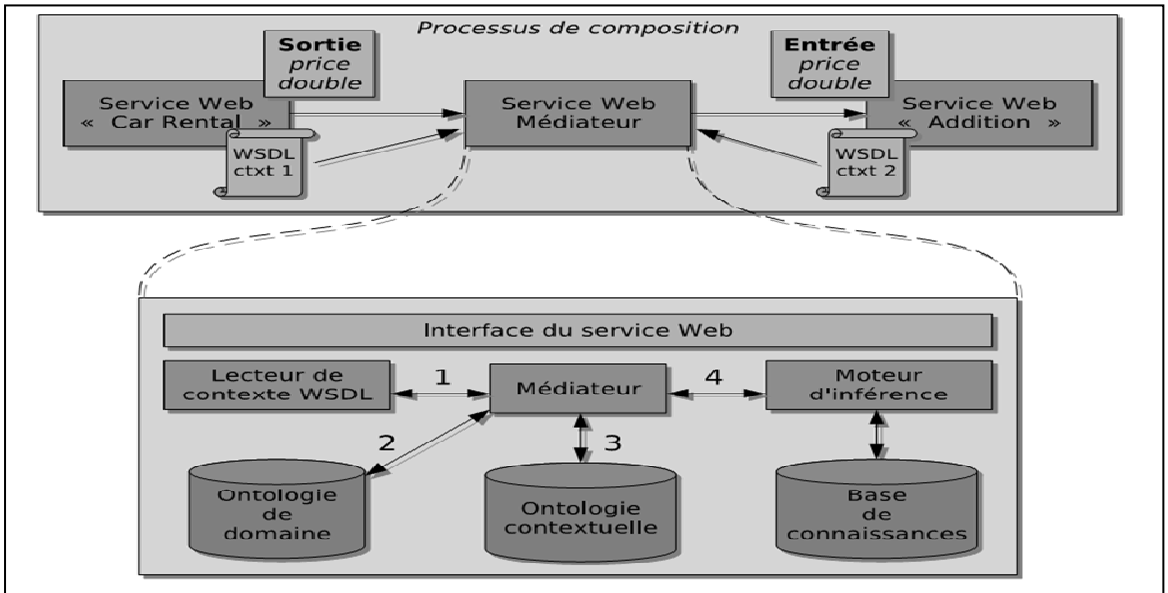

Figure III-7: fonctionnement du médiateur dans une composition [Mrissa 07].

Le service médiateur récupère les descriptions WSDL annotées des deux services Web (émetteur et récepteur), Ensuite, il vérifie par l'intermédiaire des ontologies du domaine que les concepts utilisés par les deux services sont équivalents. Cette vérification d'équivalence consiste à déterminer si les contextes de données des paramètres de sortie du premier service correspondent aux contextes de données des paramètres d'entrée du second service. Si les concepts sont équivalents, l'exécution de la composition se fait de manière normale sinon pour chaque service, une structure arborescente du contexte de l'objet sémantique (présenté dans la suite de cette section) est construite à l'aide des ontologies contextuelles. Après la construction de l'objet sémantique, le service Web médiateur convertit les données du premier service dans une représentation contextuelle requise par le second service à l'aide du moteur d'inférence.

Après la clarification du rôle de médiateur de contexte, nous présentons les deux étapes préalables à cette approche:

1. L'intégration du contexte dans la description des services : afin de remplir cet objectif, l'auteur propose :
   - ✓ Un modèle de représentations des données échangées entre les services : Pour représenter le contexte de la donnée, l'auteur introduit la notion d'objet sémantique (Cf. la Figure III-8), un objet sémantique S attaché à la donnée représentée comme un quadruplet composé: d'un concept c, d'une valeur v, d'un type t et d'un contexte C: S=(c, v, t, C).
     - **Le concept** : Cet élément représente le concept auquel l'objet sémantique fait référence.
     - **La valeur** : Cet élément représente la valeur de l'objet sémantique.
     - **Le type** : Cet élément représente le type de la valeur décrite.
     - **Le contexte** : Cet élément représente le contexte de l'objet sémantique. Le contexte est composé par des modifieurs (propriété sémantique, voir la section III.4.2.3.2 du chapitre III). Un modifieur a la capacité de modifier la signification de l'objet sémantique (la donnée) auquel il est associé. Un

modifieur est un objet sémantique, mais il est associé à un autre objet sémantique.

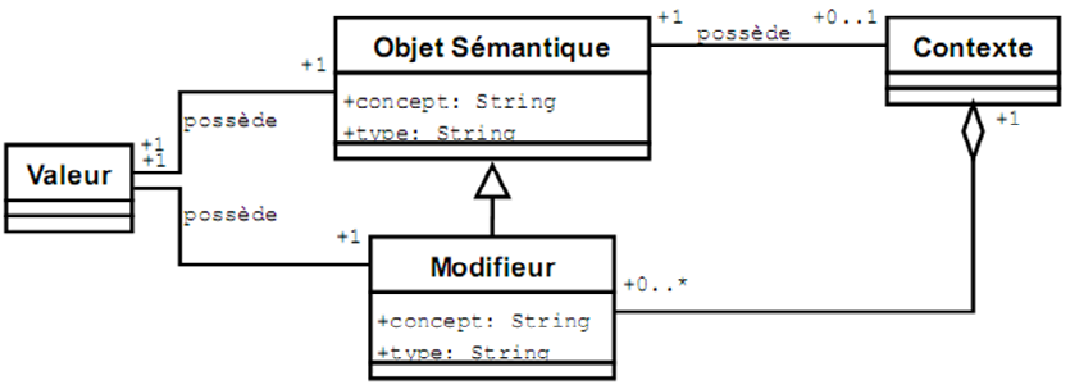

Figure III-8: représentation UML de l'objet sémantique [Mrissa 07].

- ✓ L'utilisation des ontologies contextuelles couplées à des ontologies de domaine, afin de décrire le contexte de données de manière explicite.
- ✓ L'annotation des descriptions de services: afin de pouvoir exploiter les ontologies contextuelles associées aux données par le médiateur, l'auteur propose une annotation du langage de description des services WSDL par des métadonnées. L'annotation proposée repose sur l'ajout d'un attribut contexte au sein de l'élément représentant les paramètres (la partie "*part*" dans le document WSDL).

2. L'intégration des médiateurs dans la composition de services : pour remplir cet objectif, l'auteur propose trois étapes:

Etape 1 : la détection des hétérogénéités sémantiques dans la composition : Un algorithme proposé par l'auteur analyse la composition pour localiser les flux de données.

Etape 2 : la génération des services Web médiateurs nécessaires : dans cette étape, un service Web médiateur est automatiquement généré et déployé pour chaque flux de données où l'algorithme de la première étape aurait détecté des hétérogénéités sémantiques éventuelles.

Etape 3 : l'insertion des services Web médiateurs dans la composition : dans cette étape, les services web médiateurs sont intégrés dans la composition, en insérant des invocations aux services Web médiateurs dans le processus métier (la composition) original.

III.4.3 Synthèse sur les approches de médiations de données

Une composition des services Web est toujours confrontée par de nombreux conflits, qui empêchent son bon déroulement et son exécution induira des résultats imprévisibles.

Parmi ces conflits, on trouve les hétérogénéités des données, échangées entre les services Web participants dans une composition, quelque soit leurs types structurelle, sémantique et contextuelle.

Plusieurs travaux ont été proposés dans la littérature pour résoudre telle problématique. En effet, Spencer [Spencer et al 04] travaille sur les hétérogénéités structurelles de données, ils supposent que le niveau sémantique est compatible grâce à l'utilisation des langages issus de technologies des Web sémantique comme le langage de description sémantique OWL-S (Cf. la section II.3.1.1 du chapitre II), la médiation proposée dans telles approches est une médiation totalement structurelle.

Alors que, cette supposition reste ignorée par d'autres travaux comme celle de [Cabral et a 05] qui suppose que le niveau sémantique est plus important que celui de données et il associe à chaque concept sémantique une représentation structurelle unique grâce à l'utilisation des ontologies globales.

Les correspondances et les conversions entre les concepts utilisés dans la composition de services Web qui sont représentées de manière formelle dans des ontologies globales sont établies par le langage de description OCML. Les approches de médiations sémantiques sont très répandues pour certains domaines spécifiques où les fournisseurs de services Web développent leurs services Web spécifiquement pour ce domaine. Dans ce cas, il est très simple de faire un accord entres tous les fournisseurs sur une représentation commune des connaissances de ce domaine sous forme d'une ontologie globale (de domaine), ceci est très difficile dans le contexte d'Internet où les contextes sont ouverts.

Les approches de médiations de données sémantiques orientées contexte ont été proposées pour balayer ce manque, en effet, l'adoption d'une approche orientée contexte n'impose pas aux fournisseurs d'adapter leurs sémantiques locales aux sémantiques globales, aussi d'autres avantages sont cités dans le chapitre IV, la seule approche citée dans cette catégorie est le travail de Mrissa.

L'approche de Mrissa est basée sur un modèle de représentation de données (objet sémantique) très puissant, dont la donnée est décrite de manière explicite. Cette approche utilise la notion des ontologies contextuelles pour représenter le contexte local de donnée et une ontologie de domaine pour expliciter les concepts utilisés dans les services. La seule hétérogénéité résolue dans ce travail est celle de type de valeur entre deux contextes. Cependant, les hétérogénéités entre deux contextes peuvent être de type sémantique, structurelle et de valeur (Cf. Chapitre II). Le deuxième point que l'auteur n'a pas pris en

considération est le contexte de service Web composite lui-même, seuls les contextes des services Web engagés dans la composition sont pris en charge. Or, la prise en charge de contexte de service Web composite (processus métier lui-même) donne des perspectives importantes comme la personnalisation de la composition selon le profil (le contexte) d'utilisateur. Nous proposons une solution de ce problème dans le chapitre IV. L'annotation de fichier WSDL par les informations contextuelles dans cette approche n'est pas standardisée, cela crée une sorte de répétitions et redondances de données dans le fichier WSDL surtout dans le cas où le fichier WSDL contient un nombre important de messages, nous proposons une annotation standard pour ce manque en utilisant le langage de description sémantique le plus standard à ce jour, SAWSDL (Cf. la section II.3.1.4 du chapitre II).

III.5 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur les hétérogénéités qui peuvent gêner le bon déroulement d'une composition de services Web de manière générale. Les solutions de médiations concernent chaque type d'hétérogénéité sont exposées. Nous avons tardé sur les hétérogénéités des données échangées entre les services Web engagées dans une composition qui est le cœur de notre problématique et les solutions de médiations concernant ces hétérogénéités.

Les hétérogénéités concernent les données que nous avons citées se divisent en quatre types (syntaxique, structurelle, sémantique et contextuelle). Par conséquent, les solutions de médiations sont classées selon les hétérogénéités prise en charge par ces dernières. En effet, des approches de médiations structurelles, sémantiques et orientées contexte ont été proposées ; nous avons mis en évidence la nécessité et les avantages des approches sémantiques orientées contexte par rapport aux deux premiers types surtout dans le contexte ouvert comme Internet.

Cependant, peu de travaux sont basés sur ce type d'approche, nous pouvons dire que la seule approche orientée contexte proposée est celle de Mrissa. L'auteur dans cette approche s'inspire dans son travail concernant le contexte de celui proposé par la communauté de base de données ; malgré les avantages indéniables de son travail, nous constatons plusieurs points faibles qui sont cités dans la section III.4.3.

Dans le chapitre suivant, nous proposons notre approche de médiation sémantique orientée contexte, qui est fondée sur le travail de Mrissa en améliorant et augmentant plusieurs aspects de son travail.

Chapitre IV

IV. Vers une approche de médiation de données orientée contexte

IV.1 Introduction

Nous présentons dans ce chapitre notre approche de détection et de résolution des hétérogénéités sémantiques, liées aux contextes des données échangées entre services Web. Notre travail est une approche de médiation de donnée orientée contexte. Elle consiste en l'insertion des fonctions de conversions atomiques et composites, implémentées par des services Web qui jouent le rôle des médiateurs contextuels.

Notre proposition est une amélioration de l'approche de [Mrissa 07], qui se base sur l'idée de séparation du contexte local des services Web et la sémantique globale de la composition des services Web. Les aspects de notre amélioration sont résumés dans ce qui suit :

- Intégration du contexte des données de service Web composite (la composition elle-même), qui est complètement ignoré dans l'approche de Mrissa. Celle-ci ne modalise que les données des services Web engagés dans la composition. Alors que l'intégration du contexte des données de la composition nous offre des perspectives importantes sur la personnalisation de la composition selon le profil (contexte) d'utilisateur.
- Proposition d'une solution pour résoudre les hétérogénéités entre les éléments de contextes eux-mêmes.
- Proposition d'une annotation standardisée du fichier WSDL par les descriptions contextuelles, dont leurs sémantiques sont décrites explicitement dans des ontologies contextuelles couplées avec une ontologie globale.
- Définition des fonctions de conversions atomiques associées à chaque propriété sémantique du contexte, appliquées sur les valeurs de ces dernières. L'avantage de cette solution, d'une part est la composition des fonctions de conversions atomiques, lorsque les concepts ont plusieurs propriétés sémantiques comme le cas général. Et

d'autre part obtenir une fonction composite associée au concept nécessitant la conversion sans aucun redéveloppement. Ces fonctions atomiques et composites sont implémentées par des services Web médiateurs appelés des médiateurs contextuels.

- Proposition d'un algorithme de détection des conflits contextuels de données gênant le bon déroulement de la composition, qui prend en considération le contexte des données de service Web composite (la composition en elle-même). Une fois ces conflits détectés, l'insertion des médiateurs contextuels déjà développée est faite de manière manuelle.

Dans ce chapitre, nous présentons notre approche de médiation des données échangées entre services Web orientée contexte. Nous commençons tout d'abord avec un exemple d'illustration, qui clarifie la nécessité d'utilisation de contexte pour modéliser la sémantique locale des données d'un service Web. Cet exemple montre clairement les hétérogénéités contextuelles gênant le bon déroulement de la composition. Ensuite nous présentons les ontologies contextuelles et la manière d'extraction du contexte à partir d'une ontologie contextuelle.

Les différentes hétérogénéités entre les éléments du contexte ainsi que la solution proposée pour les réconcilier sont abordées dans la section IV.4. Les fonctions de conversions contextuelles (atomiques et composites), proposées afin de convertir les valeurs des concepts, ainsi que les valeurs des propriétés sémantiques formant le contexte, sont abordées dans la section IV.5. Ensuite dans la section IV.6 nous présentons notre méthode d'annotation des fichiers WSDL pour référencer les ontologies contextuelles couplées avec l'ontologie globale. Enfin un algorithme de détection des hétérogénéités contextuelles dans une composition et la médiation proposée sont donnés dans la section IV.7.

IV.2 Exemple d'illustration

Afin de montrer le besoin d'utilisation de contexte, pour représenter la sémantique locale des données, on va considérer un exemple de composition réelle. Dans cette dernière les hétérogénéités des données liées aux contextes sont manifestes. Aussi les services Web engagées dans la composition et le service Web composite (la composition elle-même) et leurs contextes sont illustrés.

IV.2.1 Processus métier (la composition)

Considérons un développeur algérien qui veut implémenter un service Web de réservation de billet d'avion pour la ligne Algérie-Arabie Saoudite. Ce service de réservation

fournira, en plus des informations du voyage (destination, durée du voyage, type d'avion, prix du billet, etc.), les informations de prévisions de météo de la destination. Ce service est nommé *FlightWeatherWS* (le service web composite comme illustré sur la Figure IV-3).

Le développeur décide d'implémenter ce service par la composition de deux services existants nommés respectivement *ReservationWS* et *WeatherWS*. Le premier est développé par une agence algérienne de réservation et de planification des vols de la ligne Algérie-Arabie Saoudite. Ce service a comme paramètre d'entrée les informations du voyage (ville de départ, destination, dates de départ et de retour, ...etc.) et comme paramètre de sortie les informations sur les vols disponibles (ville de départ, destination, dates de départ et de retour, numéro de vol, prix du billet, ...etc.). Le deuxième est un service Web Saoudien qui permet d'offrir les prévisions de météo des villes saoudiennes. Ce service a comme paramètre d'entrée la date et la ville choisie et sa météo comme paramètre de sortie.

Par souci de simplicité, les services Web sont schématisés comme des boites noires, comme illustrés sur les figures Figure IV-1, Figure IV-2 et Figure IV-3. Les messages d'entrée (*Inputs*) sont en couleur bleue et les messages de sortie (*Outputs*) en couleur verte. Les détails des ces services Web sont décrits dans leurs fichiers WSDL fourni dans l'annexe C.

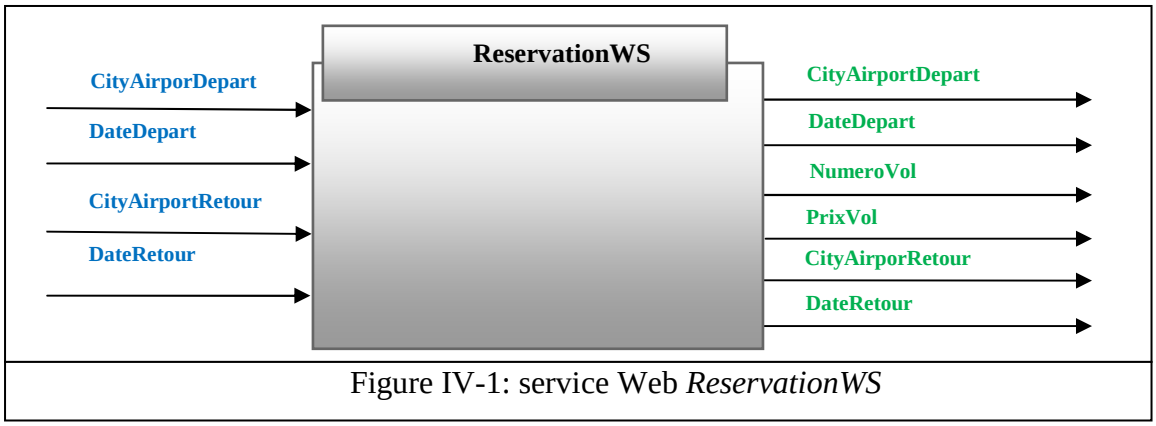

Figure IV-1: service Web *ReservationWS*

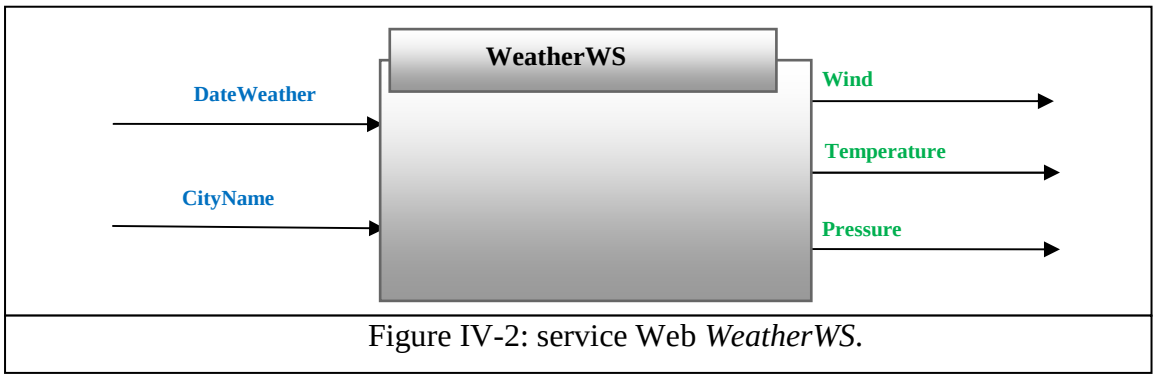

Figure IV-2: service Web *WeatherWS*.

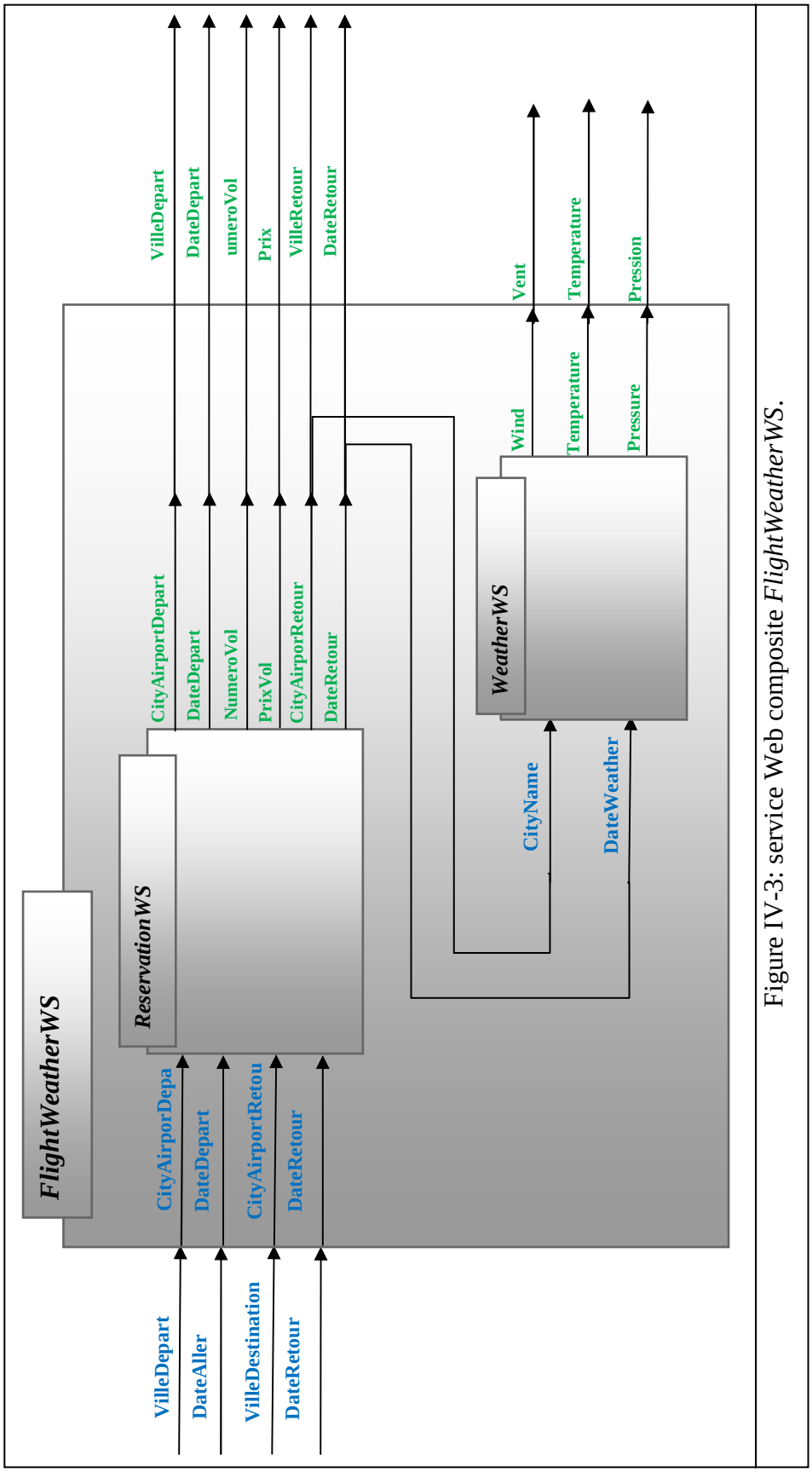

Figure IV-3: service Web composite *FlightWeatherWS*.

La composition développée par le concepteur est de type orchestration (voir les différents types de compositions dans la section I.3.2 du chapitre I).

Le développeur définit la logique de Workflow[16] de cette composition (le service Web *FlightWeatherWS* voir Figure IV-3) à l'aide de l'outil graphique *Oracle BPEL designer*[17]. Par conséquence, le langage de composition utilisé est le langage BPEL (Le langage BPEL a été présenté dans la section I.3.4.1 du chapitre I).

Notons que l'outil graphique de composition *OracleBPEL designer*, ne signale aucun conflit dans le processus de composition.

IV.2.2 Illustration des hétérogénéités contextuelles

Dans le chapitre III, nous avons mentionné que l'hétérogénéité contextuelle concerne les différentes interprétations de données par rapport à leurs contextes. En effet, le niveau contextuel d'hétérogénéité est un cas particulier de niveau sémantique d'hétérogénéité.

Les trois services sont développés par des fournisseurs différents, ce qui implique que leurs données sont placées dans des contextes différents. Par conséquent, ils sont interprétés de manières distinctes, comme illustré dans le tableau ci-dessous.

Services Web	**FlightWeatherWS**	**ReservationWS**	**WeatherWS**
Format de date	*yyyy-MM-dd*	*yyyy-MM-dd*	*dd-MM-yyyy*
Calendrier	*Gregorien*	*Gregorien*	*Hijri*
Devise	*DZD*	*DZD*	*Nul*
Pays	*DZ*	*DZ*	*Nul*
Facteur de multiplication	*1*	*1*	*1*
Température	*Celsius*	*Nul*	*Fahrenheit*
Vent	*KMH*	*Nul*	*KMH*
Pression	*hPa*	*Nul*	*hPa*

Tableau IV-1: différents attributs formant les contextes des données des trois services de notre exemple d'illustration.

[16] Workflow: processus métier

[17] Oracle BPEL designer: présenté dans la section 1.

La somme de ces attributs forme le contexte de données d'un service web. La Figure IV-4 illustre le contexte de service Web composite *FlightWeatherWS* extrait à partir de Tableau IV-1.

Généralement le contexte de données n'est pas décrit dans le fichier WSDL et les outils de composition actuels ne peuvent pas détecter les conflits liés aux contextes de données et ne disposent pas des mécanismes pour les résoudre. Sans réconciliation, ces conflits produisent des erreurs ou des échecs pendant l'étape d'exécution de la composition.

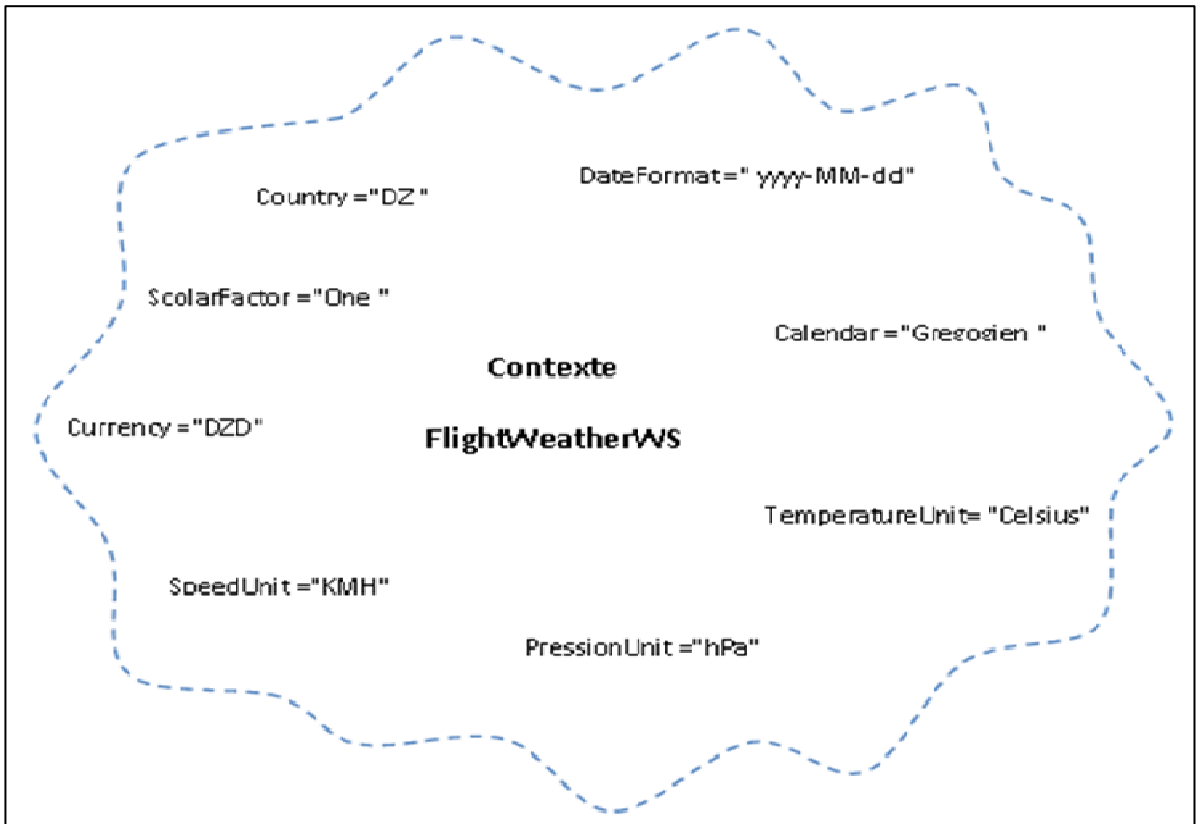

Figure IV-4: contexte de données du service Web composite FlightWeatherWS.

Effectivement, si par exemple la date "2011-06-20" envoyée par le service web *ReservationWS* au service Web *WeatherWS* comme illustré sur la Figure IV-5. Le premier service considère selon son contexte que cette date est le 20 juin de l'année 2011 du calendrier grégorien. Alors que, le deuxième service attend une date sous la forme "dd-MM-yyyy" de calendrier islamique. En effet, le deuxième service doit recevoir la date "18-07-1432" selon son contexte, qui correspond à la même date "2011-06-20" en calendrier grégorien. Cependant, sans mécanisme de médiation contextuelle le service Web *WeatherWS* consomme la date comme elle est modélisée par le premier service, ce qui engendre des erreurs inattendues.

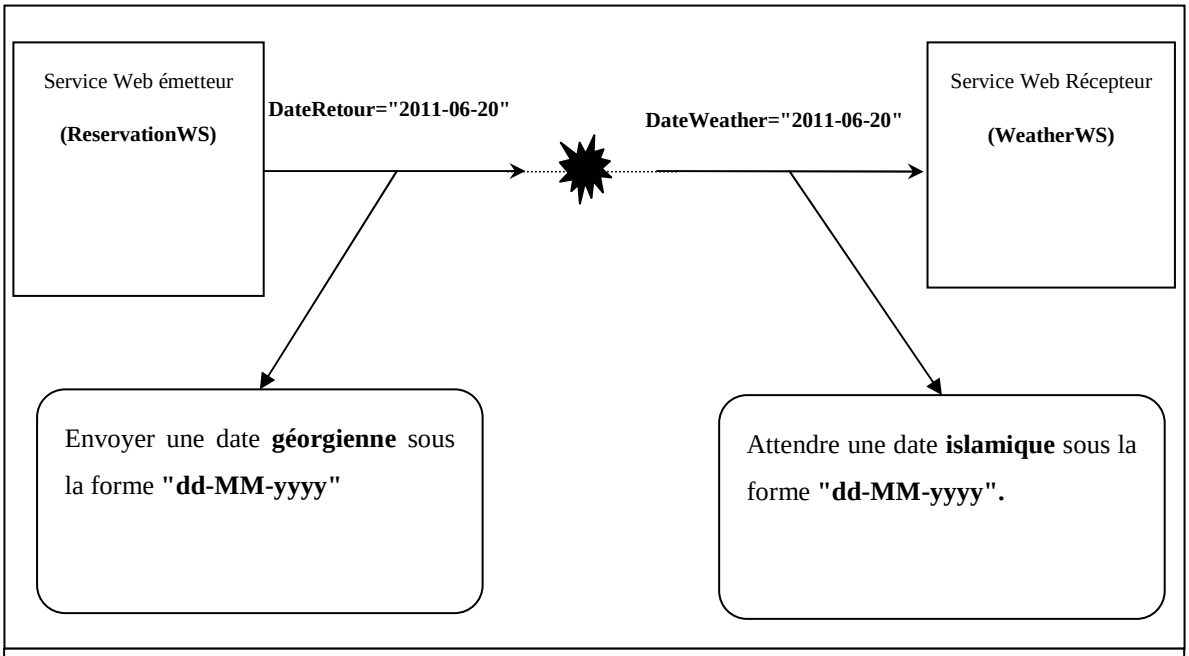

Figure IV-5: conflit contextuel entre les deux dates *DateRetour* et *DateWeather*.

IV.3 Ontologies contextuelles de données échangées entre les services Web

Les informations supplémentaires formant le contexte de données ne sont pas explicitées dans les fichiers WSDL. Plusieurs modèles ont été proposés dans la littérature pour représenter le contexte de données (Cf. section III.4.2.3.4 du chapitre III). Le modèle le plus adaptatif à notre problématique est le modèle ontologique. Ce choix se justifie par le fait que les ontologies sont considérées comme étant le modèle de représentations le plus expressif. Elles permettent de représenter la sémantique existante entre les différents concepts (les données qui forment le contexte) et leurs relations.

Nous avons détaillé, dans le chapitre III, les ontologies et leurs apports au domaine de service Web sémantique. La Figure IV-6 illustre une représentation graphique de l'ontologie contextuelle du service Web composite *FlightWeatherWS*, qui décrit de manière explicite le contexte de données. Les concepts sont présentés par des rectangles noirs. Chaque concept a au moins une propriété sémantique[18] qui est aussi un concept, et l'instance du dernier concept forme la valeur de la propriété sémantique. Ces dernières sont présentées par des rectangles en couleur rouge.

[18] : Appelé aussi modifieur dans certain article, elle est présentée dans la section III.4.2.3.3.

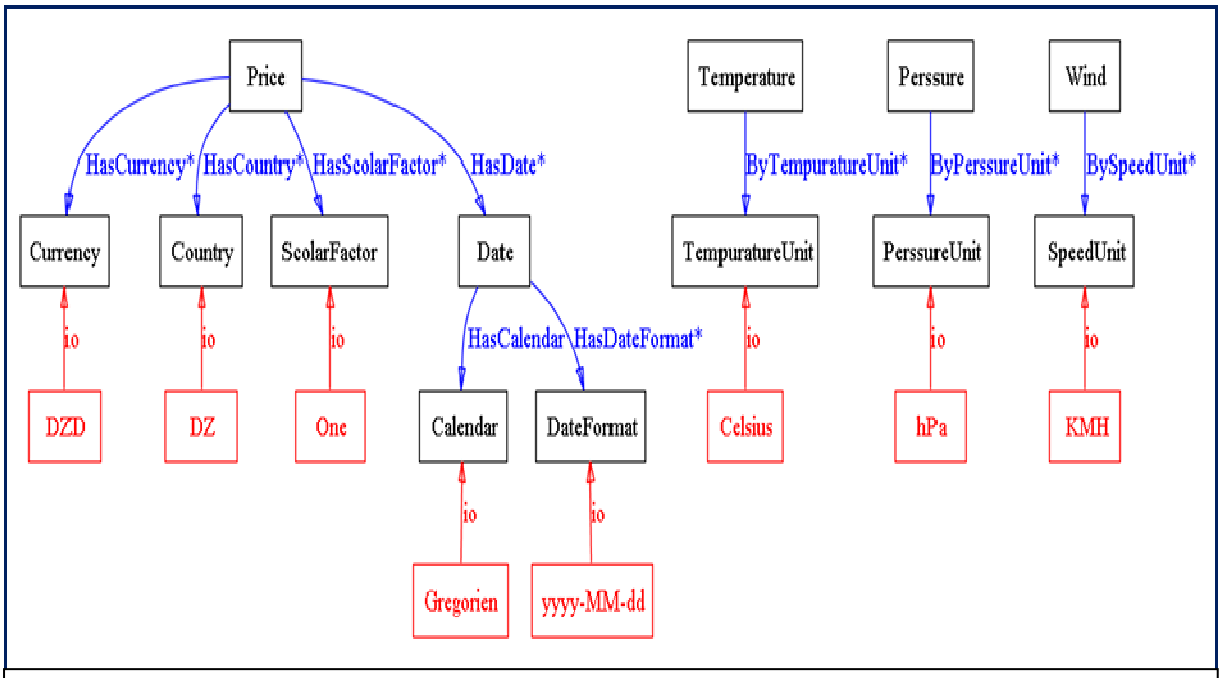

Figure IV-6: ontologie contextuelle associée au service composite *FlightWeatherWS*.

À partir de l'ontologie contextuelle associée aux concepts de service Web, on peut former le contexte de n'importe quelle donnée.

Par exemple, le contexte de concept *Date* se compose en deux propriétés sémantiques qui sont le type de calendrier avec sa valeur "*Gregorien*" et le format de date avec sa valeur "yyyy-MM-dd". Alors, le contexte du concept *Date* est:

Date={Calendar="Gregorien"),(DateFormat="yyyy-MM-dd")}.

IV.4 Hétérogénéités entre deux différents contextes

Lors de l'échange de données entre les services Web deux contextes entrent en jeu, celui du service émetteur où la donnée a été modélisée et celui du service récepteur où la donnée doit être interprétée.

Le problème d'hétérogénéité se pose aussi entre les attributs (les propriétés sémantiques) de ces deux contextes, car ils ont été modalisés par des fournisseurs différents. En effet, trois types d'hétérogénéité peuvent être présentés entre les attributs des deux contextes à savoir: l'hétérogénéité sémantique, structurelle et de valeur, comme illustrées sur la Figure IV-7.

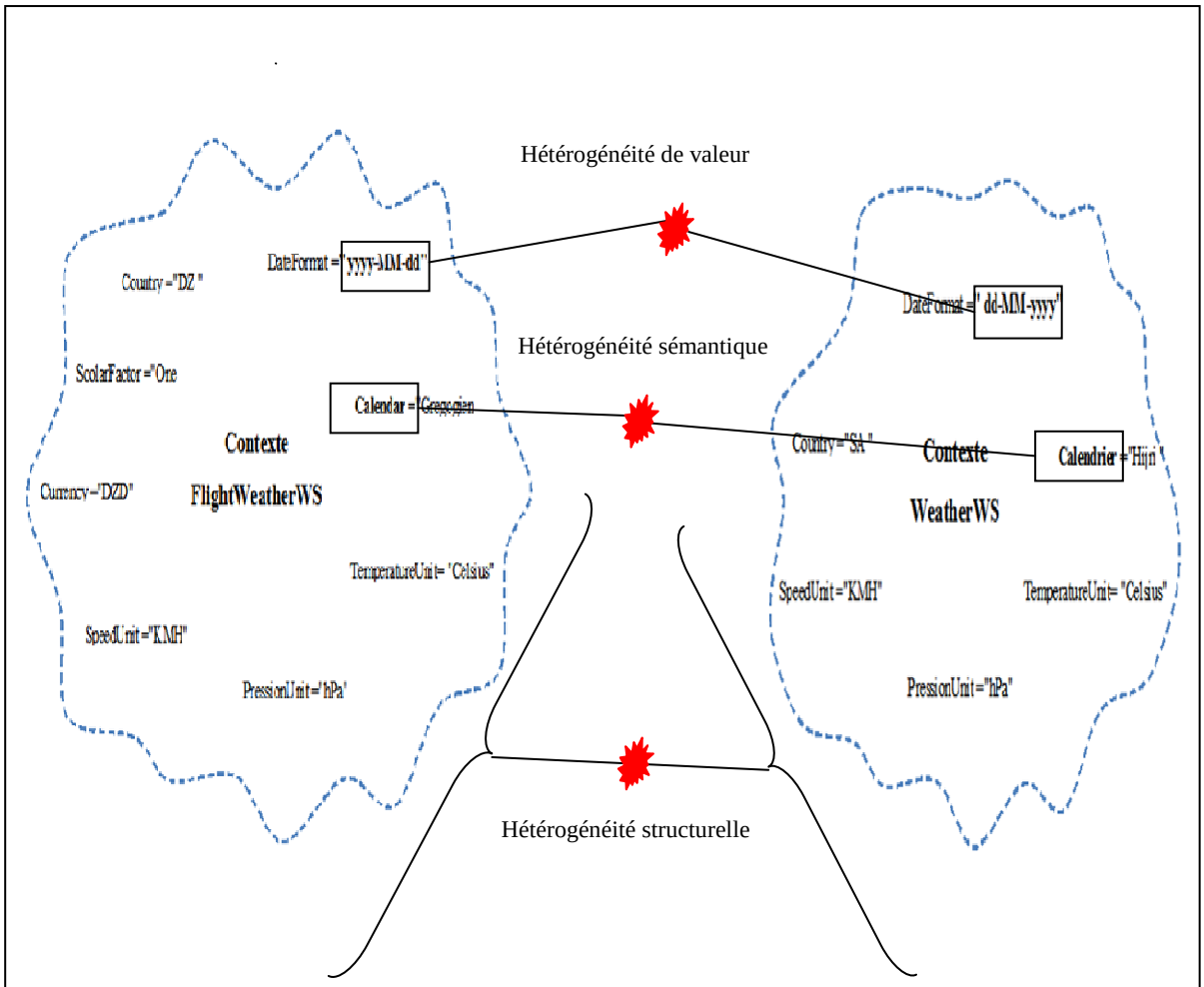

Figure IV-7: différentes hétérogénéités entre deux contextes.

Notre solution est la suivante:

L'hétérogénéité sémantique : l'hétérogénéité sémantique concerne la signification des vocabulaires utilisés pour décrire les propriétés sémantiques. Pour notre exemple, le fournisseur de service Web composite *FlightWeatherWS* utilise pour décrire le type de calendrier utilisé le nom **Calendrier**, alors que, le fournisseur de service Web *WeatherWS* utilise le vocabulaire **Calendar** (un mot en anglais). Notre solution pour telle hétérogénéité sémantique est de mettre des correspondances entre ces termes, ces correspondances sont inscrites dans l'ontologie globale. Notons que, le contexte de service est explicité dans une ontologie contextuelle comme nous avons proposé dans la section IV.3. Donc il faut juste ajouter une correspondance entre le terme concerné dans l'ontologie contextuelle et l'ontologie globale, comme illustré dans la Figure IV-8.

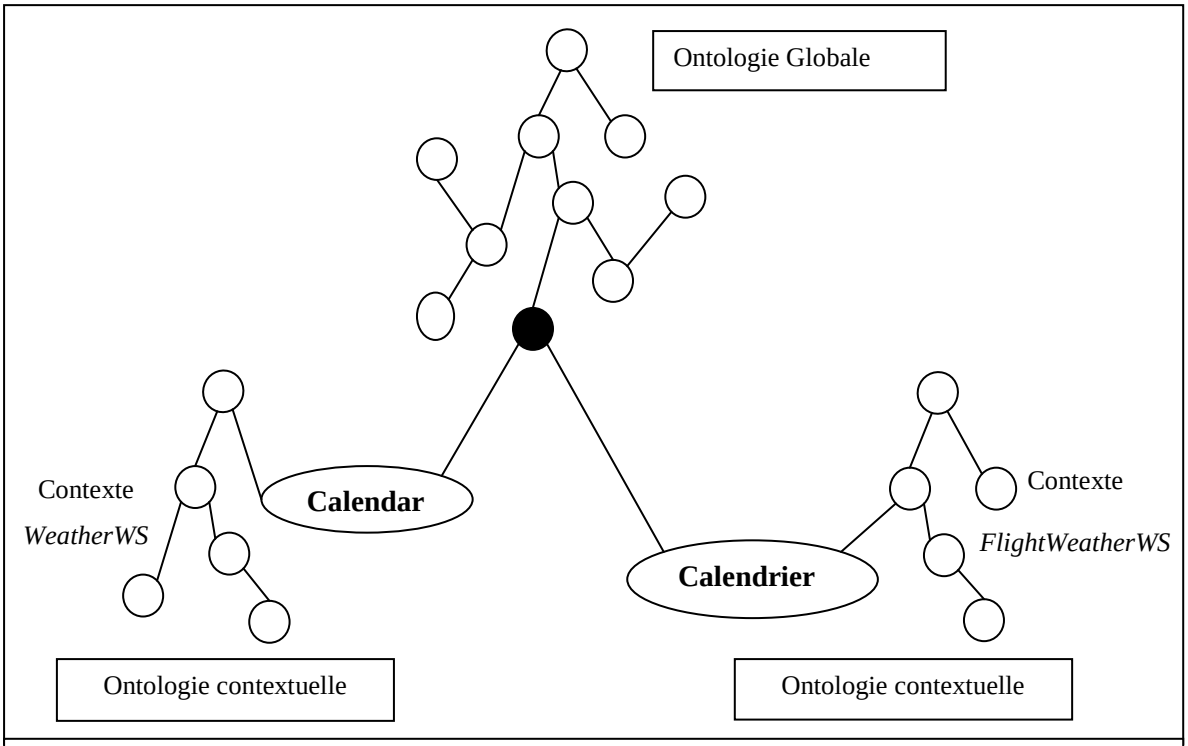

Figure IV-8: les correspondances sémantiques entre les attributs de deux contextes.

L'hétérogénéité structurelle: les propriétés sémantiques de services Web sont aussi confrontées par des conflits structurels. Effectivement, un service Web qui offre les informations de météo des villes gratuitement n'a pas besoin de décrire les propriétés sémantiques associées au concept prix comme le service Web *WeatherWS*. Alors que le service *FlightWeatherWS* décrit les propriétés sémantiques associées au concept prix comme la devise et le pays. En plus, il décrit les propriétés sémantiques associées aux informations de météo comme l'unité de mesure de température et l'unité de mesure de la pression, comme illustré sur la Figure IV-7.

Nous supposons dans notre travail que l'enrichissement sémantique par les ontologies contextuelles couplées par l'ontologie globale ne correspondent qu'à une représentation structurelle, idée inspirée à partir du travail de [Cabral et a 05].

L'hétérogénéité de valeur : l'hétérogénéité de valeur concerne les valeurs de propriétés sémantiques elles-mêmes. Cette hétérogénéité n'est pas un problème en elle-même, elle est le résultat de l'utilisation de l'approche contextuelle, qui offre la liberté aux fournisseurs de modéliser le contexte local des données.

Dans notre illustration, on prend l'exemple simple sur la température échangée entre le service Web *WeatherWS* et le service Web composite *FlightWeatherWS* (Cf. la Figure IV-7).

Pour le premier service la température est une valeur mesurée en degrés Fahrenheit, par contre le deuxième service consomme cette valeur en degrés Celsius.

Afin de résoudre telle hétérogénéité, il suffit de décrire explicitement les propriétés sémantiques formant les contextes de données de services Web. Ainsi que les types des valeurs associées à ces derniers et les implications des fonctions de conversions sur ces valeurs.

Dans notre cas, si on veut convertir une valeur de température mesurée en degrés Fahrenheit à une valeur mesurée en degrés Celsius, nous utilisons la fonction suivante: $\mathrm{T}(°\mathrm{Celsius}) = 5/9\ (\mathrm{T}(°\ \mathrm{Fahrenheit}) - 32)$.

IV.5 Fonctions de conversions contextuelles

Dans le chapitre III, nous avons clarifié la notion de valeur sémantique. En effet, les données échangées entre les services Web sont considérées comme des valeurs sémantiques, car elles sont des valeurs simples associées à des concepts et interprétées dans des contextes précisés. Si le contexte de service émetteur diffère de celui de service récepteur lors de l'échange de données entre les services, les valeurs sémantiques doivent convertir leurs valeurs simples (interprétées dans le contexte de service émetteur) vers d'autres valeurs simples interprétées dans le contexte de service récepteur,

Les fonctions qui changent les valeurs sémantiques sont appelées les fonctions de conversions contextuelles, puisqu'elles changent l'interprétation contextuelle de ces valeurs. Par exemple la valeur sémantique **50** du concept *Temperature,* envoyée par le service Web *WeatherWS* vers le service Web composite *FlightWeatherWS,* nécessite une conversion contextuelle. Car le contexte de donnée associé au concept *Temperature* du premier service (*TemperatureUnit=Fahrenheit*) est différent du contexte de donnée associé au même concept *Temperature* de deuxième service (*TemperatureUnit=Celsius*).

Dans notre travail, nous définissons pour chaque propriété sémantique une fonction de conversion atomique. Par exemple, la fonction atomique associée à la propriété sémantique *TemperatureUnit*, pour convertir l'unité de mesure de température, est :

$$\textbf{Fonction}_{\textbf{Temperature}} = \mathbf{T}\ (°\mathbf{Celsius}) = \mathbf{5/9}\ (\boldsymbol{T}(°\ \mathbf{Fahrenheit}) - \mathbf{32}).$$

Donc, si on applique cette fonction sur la valeur sémantique **50** qui est mesurée en °Fahrenheit Celsius du concept *Temperature* on obtient la valeur **10** mesuré en ° Celsius**,** qui est consommée par le service *FlightWeatherWS*.

Certains concepts sont décrits par plusieurs propriétés sémantiques, comme le concept *DateRetour* qui décrit par le format de date et le type de calendrier :

Date= {Calendar="Gregorien"), (DateFormat="yyyy-MM-dd")}.

Dans ce cas, la fonction appliquée sur la valeur de ce concept est appelée fonction de conversion composite, car elle est composée de deux fonctions atomiques **Fonction** $_{\textbf{Calendar}}$ et **Fonction** $_{\textbf{DateFormat}}$:

Fonction $_{\textbf{Date}}$ **= Fonction** $_{\textbf{DateFormat}}$ **(Fonction** $_{\textbf{Calendar}}$ **(Date)).**

Ces fonctions sont implémentées comme des services Web médiateurs appelés médiateurs contextuels et sont insérés aux points où les conflits contextuels ont été détectés. Cela pour réconcilier les hétérogénéités liées aux contextes. La détection automatique des conflits contextuels et l'insertion des ces médiateurs contextuels sont présentés dans la section IV.7.

IV.6 Annotation contextuelle de fichier WSDL

Les interfaces des services Web sont décrites par le langage WSDL de manière syntaxique. Afin de faciliter l'interopérabilité sémantique entre les services Web, l'annotation sémantique est utilisée pour établir les correspondances entre les éléments de fichier WSDL et les concepts des ontologies. En effet, plusieurs approches ont été proposées pour ajouter les informations sémantiques aux fichiers WSDL (Cf. la section II.3.1 du chapitre II). Pour notre problématique, les informations sémantiques sont les informations formant les contextes associées aux concepts utilisés par les services, qui se trouvent dans des ontologies contextuelles et l'ontologie globale.

Nous avons choisi pour notre problématique l'annotation SAWSDL (Cf. la section II.3.1.4 du chapitre II), ce choix a été justifié dans la section chapitre II. Effectivement, SAWSDL est un standard du W3C qui fourni une extension d'attribut appelé *modelReference*, pour spécifier les correspondances entre les éléments de langage WSDL y compris les messages d'entrées (*Inputs*) et de sorties (*Outputs*) et les concepts des ontologies. Nous utilisons cette extension d'attribut pour associer les messages décrits dans le fichier WSDL

avec les URIs des concepts. Ces derniers sont explicités soit dans les ontologies contextuelles et/ou dans l'ontologie globale.

Nous avons utilisé deux manières d'annotations: l'annotation globale et l'annotation locale. En effet, dans notre annotation la valeur de l'attribut *modelReference* contient un seul URI ou deux URIs séparés par un espace blanc. Dans le premier cas l'URI est pointé vers le concept de l'ontologie globale, ce cas est utilisé pour le concept qui n'a pas de contexte local. Tandis que, dans le deuxième cas le premier URI est pointé vers le concept de l'ontologie globale et le deuxième URI pointé vers le concept de l'ontologie contextuelle. Ce deuxième cas est utilisé quand le concept a une sémantique globale, explicitée dans une ontologie globale, et une interprétation locale explicitée dans une ontologie contextuelle.

```
<wsdl:definitions targetNamespace="WeatherWS"
            :
            :
xmlns:sawsdl="http://www.w3.org/ns/sawsdl" />">
<wsdl:types>
 <s:schema elementFormDefault="qualified" targetNamespace="WeatherWS">
  <s:element name="GetWeather">
   <s:complexType>
    <s:sequence>
     <s:element maxOccurs="1" minOccurs="0" type="s:string" name="CityName"
        sawsdl:modelReference="http://Global#City" />
     <s:element maxOccurs="1" minOccurs="0" type="s:string" name="DateWeather"
        sawsdl:modelReference="http://Global#Date  http://WeatherWS#Date" />
    </s:sequence>
   </s:complexType>
  </s:element>
      :
      :
  <s:complexType name="WeatherInfo">
   <s:sequence>
    <s:element maxOccurs="1" minOccurs="1" type="s:double" name="Wind "
    sawsdl:modelReference="http://Global#Wind  http://WeatherWS#Wind"/>
    <s:element maxOccurs="1" minOccurs="1" type="s:double" name="Temperature"
     sawsdl:modelReference="http://Global#Temperature  http://WeatherWS#Temperature"/>
    <s:element maxOccurs="1" minOccurs="1" type="s:double" name="Pressure"
     sawsdl:modelReference="http://Global#Perssure  http://WeatherWS#Perssure" />
   </s:sequence>
  </s:complexType>
 </s:schema>
</wsdl:types>
    :
    :
```

Figure IV-9: annotation contextuelle de fichier WSDL de service Web *WeatherWS*.

La Figure IV-9 présente la manière de notre annotation de fichier WSDL de service Web *WeatherWS* en utilisant la recommandation SAWSDL. Les éléments annotés sont clarifiés en vert foncé. Chaque élément du fichier WSDL a un attribut *modelReference* qui est

pointé vers un concept ontologique. Par exemple les éléments *DateWeather* et *CityName* correspondent aux concepts ontologiques *Date* et *City* respectivement. L'élément *DateWeather* a une interprétation locale explicitée par l'ontologie contextuelle référencée par l'URI *« http:// WeatherWS# »*, et une sémantique globale explicitée par l'ontologie globale référencé par l'URI *« http:// Global#»***.** Tandis que l'élément *CityName* n'a pas un contexte local, dans ce cas il est référencé seulement par l'ontologie globale d''URI *« http:// Global#»*.

IV.7 Médiation contextuelle

Les étapes précédentes sont des étapes préalables à notre approche de médiation sémantique orientée contextuelle. L'objectif principal de notre travail est la détermination automatique et la réconciliation des conflits liés aux contextes de données échangées entre les services Web, par l'insertion du médiateur contextuel dans les points où les conflits contextuels ont été détectés.

Dans notre travail les médiateurs contextuels sont des services Web simples, nous associons à chaque fonction de conversion (Cf. la section IV.5 du ce chapitre) atomique ou composite un médiateur contextuel. Dans notre exemple, le médiateur contextuel associé à la fonction atomique Fonction $_{\textbf{Temperature}}$ est MédiateurContexteul $_{\textbf{Temperature}}$, qui consomme une valeur de température en °Fahrenheit et la convertit en valeur exprimée en °Celsius, ce médiateur implémente la fonction $\mathrm{T}\,(°\mathrm{C}) = 5/9\,(T(°\,\mathrm{F}) - 32)$, comme illustré sur la Figure IV-10.

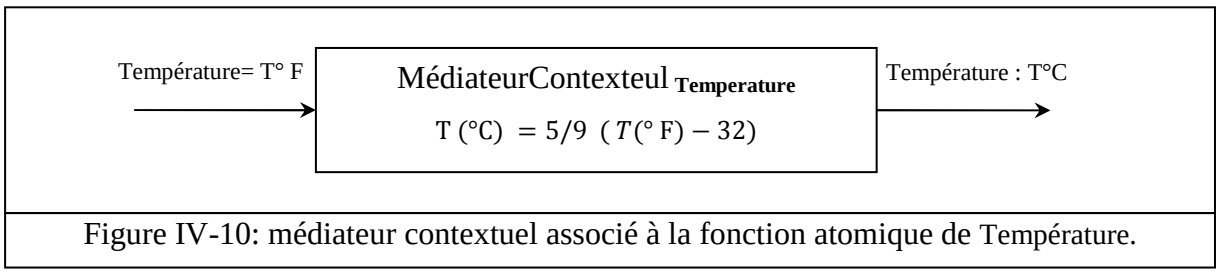

Figure IV-10: médiateur contextuel associé à la fonction atomique de Température.

Les services Web médiateurs associés aux fonctions de conversions sont implémentés à part, de manière manuelle. Ils peuvent déjà exister en ligne comme le médiateur contextuel associé à la fonction de conversion de date grégorienne en date islamique. La génération automatique des médiateurs contextuels, à partir de fonction de conversion atomique ou composite, est l'une des perspectives de ce travail.

Dans la section IV.7.1, nous proposons un algorithme permettant de localiser les points des hétérogénéités contextuelles dans une composition de services Web. La section IV.7.2 clarifie notre proposition d'insertion des médiateurs contextuels dans les points détectés.

IV.7.1 Détermination des hétérogénéités contextuelles

Avant d'insérer les médiateurs contextuels, afin de réconcilier les conflits qui gênent le bon déroulement de l'exécution, il est nécessaire de localiser les points où doivent être insérés ces médiateurs. Ce sont les points critiques où il y a un transfert de données confrontés par des conflits contextuels.

Les hétérogénéités contextuelles se présentent lorsque une donnée échangée entre deux services Web et leurs contextes sont différents, comme explicité dans la section IV.2.2 du chapitre IV. Dans notre exemple de composition, les points critiques sont:

1. lorsque la date envoyée par le service *ReservationWS* au service Web *WeatherWS*,
2. lorsque la température envoyée par le service Web composite *WeatherWS* au service Web composite *FlightWeatherWS*.

La Figure IV-11 illustre les points critiques dans notre exemple d'illustration.

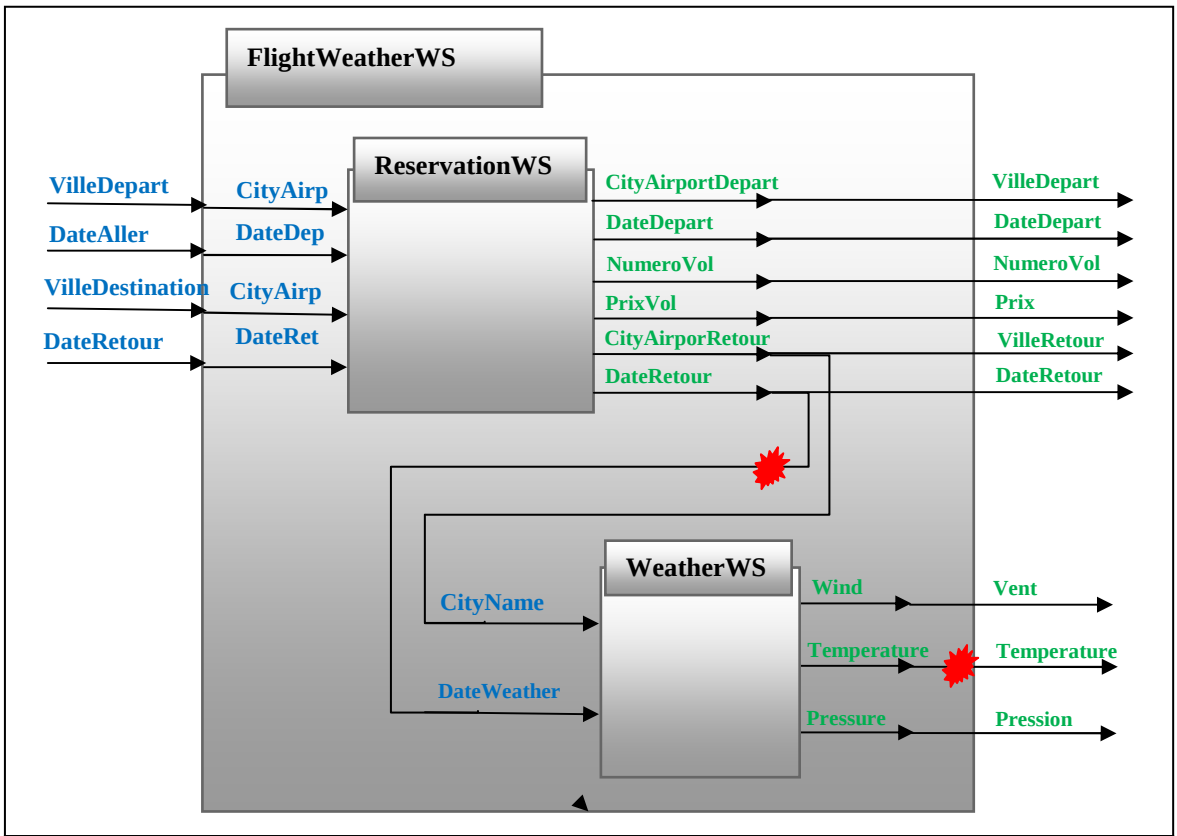

Figure IV-11: les points critiques dans notre exemple de composition.

L'algorithme qui permet de détecter les points critiques dans une composition est présenté sur la Figure IV-12. Cet algorithme analyse la spécification de la composition faite par le langage BPEL. Le programme extrait toutes les activités où il y a des transferts des données, ces activités sont appelés *Assign* dans le langage BPEL. Pour chaque activité, le concept de service Web (émetteur) et le concept de service Web (récepteur) sont extraits.

Entrer: BPEL process **Proc**,
Ensemble de fichiers WSDL annotés **WS = {ws},**
Ensemble des ontologies contextuelles **OntoC = {ontoC}**,
Ontologie Global **OntoG ;**
Sortie: Ensemble des conflits contextuel **CC = {cc}**;
Ensemble des noms des fonctions de conversions atomiques **FonAt= {fonAt};**
CC = ∅
FonAt = ∅
Pour chaque **Assign** dans **Proc**
WS_E ← ObtenirServiceWebEmetteur(**Assign**, **Proc**, **WS**)
WS_R ← ObtenirServiceWebRecepteur(**Assign**, **Proc**, **WS**)
Concept_e ← ObtenirConceptEmeteur(**WS_E**, **Assign, OntoG**)
Concept_r ← ObtenirConceptRecepteur(**WS_R**, **Assign, OntoG**)
Si Concept_e ⇔ Concept_r
C← **Concept_e**
Ctxt_E ← ObtenirContext(**WS_E**,**C**, **OntoC**)
Ctxt_R← ObtenirContext(**WS_R**, **C**, **OntoC**)
Pour chaque Propriété Sémantique **PS** dans **Ctxt_E**
VPS_E ← ObtenirValeurProrieteSémantique(**C**, **PS**, **Ctxt_E**)
VPS_R ← ObtenirValeurProrieteSémantique(**C**, **PS**, **Ctxt_R**)
Si VPS_E ≠ **VPS_R**
Alors cc ← ObtenirConflitContextuel(**C**, **PS**, **Ctxt_E**, **Ctxt_E**,**VPS_E, VPS_R**)
CC ← **CC** ∪ {**cc**}
fonAtom ←ObtenirNomFonctionAtomique(**C, PS, VPS_E, VPS_R)**
FonAtom= FonAtom ∪ { **fonAtom**}
Return CC

Figure IV-12: algorithme de détection des conflits contextuel dans une composition.

À partir de fichiers WSDL associés aux services Web concernés et annotés par les attributs contextuels, l'algorithme vérifie si le contexte associé au concept de service Web émetteur correspond au contexte associé au concept de services Web récepteur. Les contextes de donnés sont explicités dans des ontologies contextuelles couplées par l'ontologie globale. Et si ce n'est pas le cas, l'algorithme donne ce point comme un point critique, et donne aussi les conflits contextuels entre ces deux concepts. Par conséquence, le programme affiche les noms des fonctions de conversions atomiques et composites nécessaires pour implémenter les médiateurs contextuels qui doivent être insérés par la suite.

IV.7.2 Insertion des médiateurs contextuels

Une fois que les points critiques sont détectés, ainsi que les conflits contextuels et les noms des fonctions de conversions par l'algorithme proposées dans la section IV.7.1. Il ne reste plus que de mettre à jour la composition, en ajoutant des invocations aux médiateurs contextuels associés aux fonctions de conversions atomiques et composites. Ces dernières sont implémentées comme des services Web, comme illustré dans Figure IV-13.

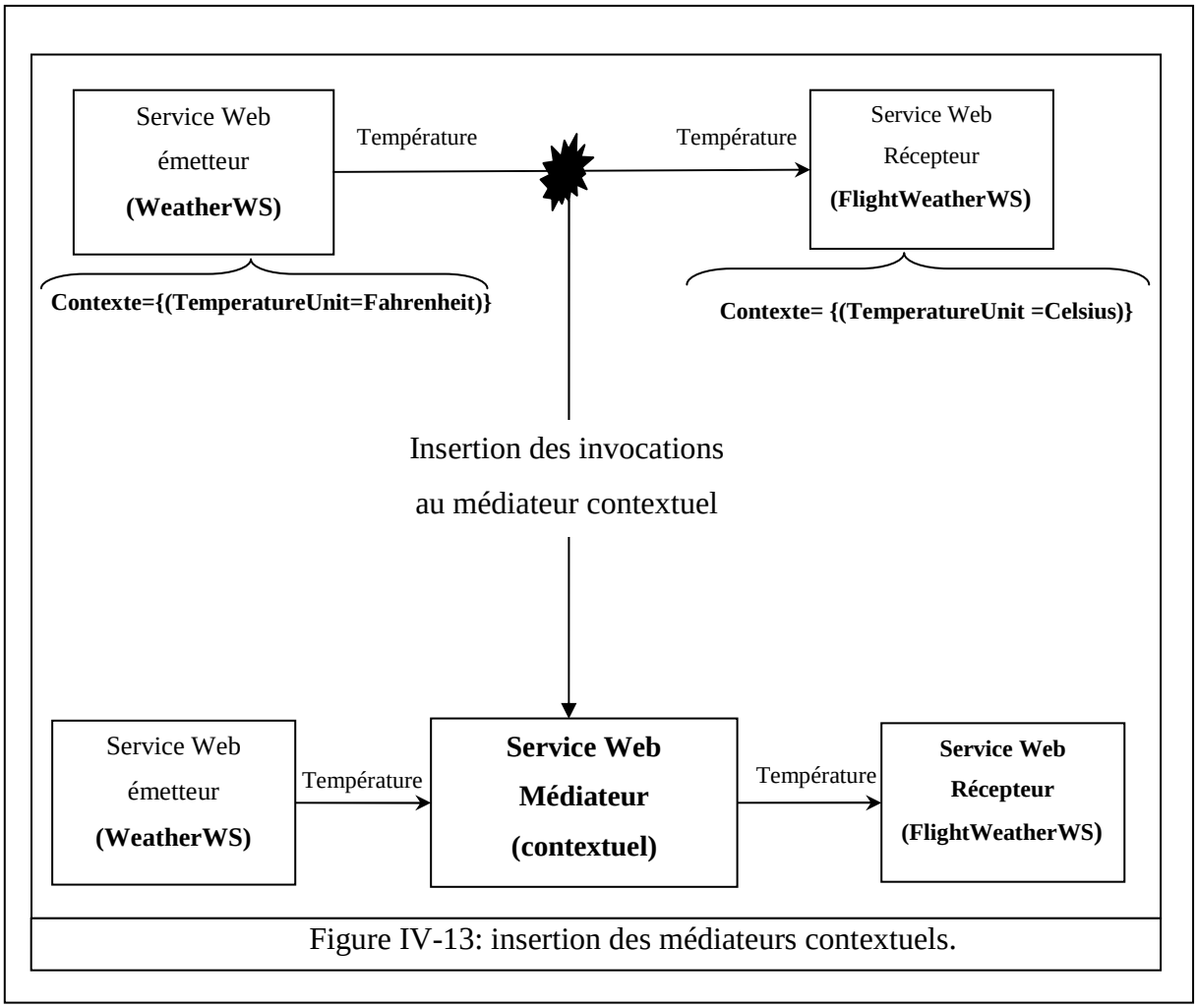

Figure IV-13: insertion des médiateurs contextuels.

Dans notre travail nous supposons que ces médiateurs contextuels sont déjà implémentés ou bien sont disponibles en ligne (Internet). La mise à jour de la composition est faite de manière manuelle par un outil de composition (ex: *Oracle BPEL designer*). L'insertion automatique et la génération des services Web médiateurs n'est pas l'objet de notre travail, et peut être considérée comme une perspective de ce travail.

IV.8 Démarche globale de processus de médiation

En résumant, dans la Figure IV-14, la démarche globale suivie dans notre approche. Les fournisseurs fourniront les services Web. Ils construisent les ontologies contextuelles explicitant les sémantiques locales de leurs services Web. Les fournisseurs annotent les descriptions WSDL des services Web par un outil d'annotation (ex : SAWSDLAT[19]). Ensuite le développeur spécifie et annote son service Web composite. Il analyse la composition par un outil de détection des hétérogénéités contextuelles (ex : CDHT[20]), dans la troisième étape. S'il existe d'hétérogénéités le développeur insert les services Web médiateurs (Etape quatre) par un outil de composition dans les points détectés et le processus de médiation revient à l'étape deux. Dans le cas où toutes les hétérogénéités sont résolues le processus terminera son exécution par le déploiement de service Web composite (la composition) dans un serveur d'orchestration (étape 5).

[19] : Outil d'annotation développé dans le chapitre V, selon la recommandation SAWSDL.

[20] : Outil de détection des hétérogénéités contextuelles développé dans le chapitre V.

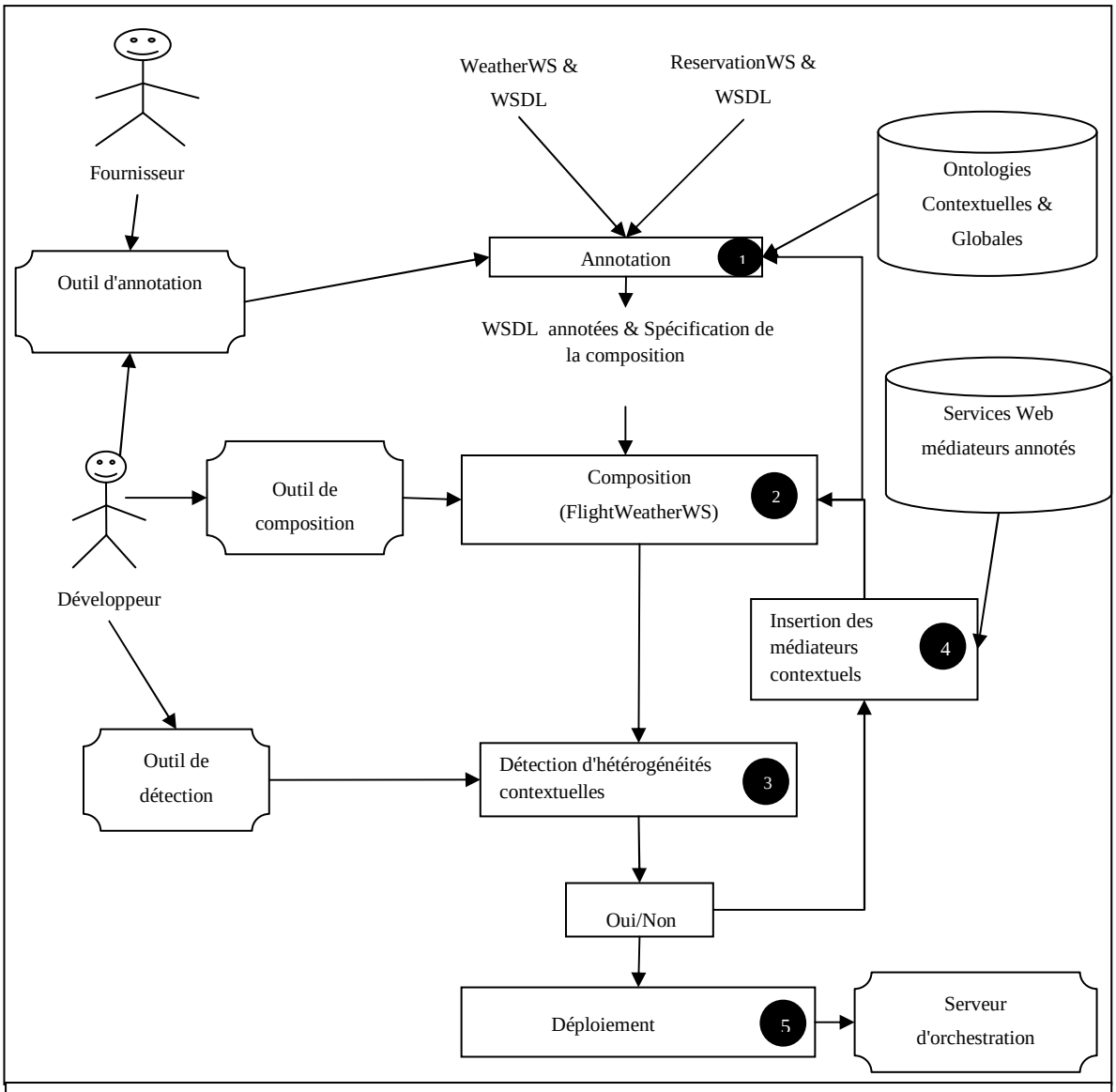

Figure IV-14: démarche globale de détection et de réconciliation des hétérogénéités.

IV.9 Comparaison entre notre approche et l'approche de Mrissa

Notre approche se différencie de celle de Mrissa en plusieurs points que nous avons améliorés dans notre travail. Nous avons résumé ces différents points et améliorations dans le Tableau IV-2. Nous présentons une comparaison sur les aspects suivants :

- Modèle de représentation de contexte.
- La sémantique locale et globale des services Web.
- Les services web concernés.
- Les fonctions de conversions.
- Les hétérogénéités prises en charge.

- La génération des services Web médiateur (automatique ou manuelle).
- L'étape de détection et d'insertion de médiateur contextuels (exécution ou conception).

Approche	Mrissa	Notre approche
Modèle de représentation de contexte	Modèle ontologique	Modèle ontologique
Sémantique locale & globale	Ontologie contextuelle & domaine	Ontologie contextuelle & globale
Services Web concernés	Services Web engagées dans la composition	Services Web engagées & la composition elle-même
Fonctions de conversions	Basé sur l'objet sémantique	Basé sur les propriétés sémantiques formant le contexte
Nature de fonctions de conversions	Conversions des éléments de l'objet sémantiques	Fonctions atomiques et composite associées aux propriétés sémantiques
Hétérogénéités entre les contextes eux-mêmes	Ne sont pas prise en charge	Prise en charge
Génération des médiateurs contextuels	Automatiquement	manuellement
Insertion des médiateurs contextuels	Automatiquement	manuellement
Etape d'insertions des médiateurs	*Run time*[21]	*Design time* [22]

Tableau IV-2: différences entre notre approche et le travail de Mrissa.

IV.10 Conclusion

Dans ce chapitre, nous avons présenté notre approche de médiation orientée contexte de données échangées entre services Web engagées dans une composition. Nous avons illustré les hétérogénéités sémantiques et contextuelles qui gênent le bon déroulement d'une composition par un exemple d'illustration réel de planification de voyage de vol. Nous avons amélioré dans notre proposition le travail de [Mrissa 07] dans plusieurs points, essentiellement: L'intégration de contexte de services Web composite (la composition elle-même). La résolution des hétérogénéités entre les éléments du contexte eux-mêmes en incluant le contexte de donnée de la composition. L'utilisation d'une annotation plus standardisée du fichier WSDL en utilisant l'annotation SAWSDL du W3C. La définition des

[21] : Etape d'exécution

[22] : Etape de conception

fonctions de conversions (atomiques et composées) associées à chaque propriété sémantique du contexte. Et la proposition d'un algorithme de détection des conflits contextuels de données dans une composition qui prend en considération le contexte des données du service Web composite (la composition en elle-même).

Dans le chapitre suivant, nous validons notre approche par la réalisation d'un prototype de l'exemple présenté dans la section IV.2 de ce chapitre et le développement de deux outils d'aide. Un pour annoter les ficher WSDL par la recommandation SAWSDL. Et l'autre, pour faciliter la détection des hétérogénéités contextuelles dans une composition afin d'insérer de médiateurs contextuels pour les réconcilier.

Chapitre V

V. Prototype

V.1 Introduction

Ce chapitre est consacré à la validation de notre approche (vue précédemment dans le chapitre IV) qui consiste à réaliser la composition des services Web de l'exemple d'illustration présenté dans la section IV.2 du chapitre IV. En effet, cette composition est gênée par des hétérogénéités contextuelles des données échangées entre les services Web participants et le service Web composite (la composition elle-même). Ces hétérogénéités ont été présentées et clarifiées dans le précédent chapitre.

Pour détecter les hétérogénéités contextuelles dans une composition des services Web, nous avons développé deux principaux outils. L'outil « SAWSDLAT » (*SAWSDL Annotation Tool*) utilisé pour annoter les fichiers WSDL associés aux services Web et ceci selon notre manière d'annotation présentée dans la section IV.6 du chapitre IV. Notons que la recommandation SAWSDL du W3C a été respectée. L'autre outil appelé « CDHT » (*Contextuals Heterogeneities Detection Tool*) utilisé pour détecter les hétérogénéités contextuelles dans une composition. Cet outil est une implémentation de l'algorithme de détection des hétérogénéités contextuelles présentée dans la section IV.7.1 du chapitre IV.

En résumé, la validation de notre approche passe essentiellement par les étapes suivantes : il s'agit d'implémenter les services Web engagés dans la composition, d'établir la spécification BPEL de la composition et de construire les ontologies contextuelles et l'ontologie globale.

Nous présentons, aussi, les deux outils SAWSDLAT (*SAWSDL Annotation Tool*) et CDHT (*Contextuals Heterogeneities Detection Tool*) ainsi que les langages et les APIs utilisés.

V.2 Implémentation de la composition

V.2.1 Développement des services Web

Les deux services Web (*WeatherWS* et *ReservationWS*) participants dans la composition de notre exemple illustré dans la section IV.2 du chapitre IV sont développés dans l'environnement .Net[23].Ces deux services ont été implémentés en utilisant le langage orienté objet C Sharp[24].

Ces services sont hébergés dans deux serveurs Web différents (IIS5.1 et IIS7.1)[25]. Les fichiers WSDL qui les décrivent sont aussi publiés avec leurs services. Notons que nous pouvons utiliser un annuaire UDDI des services Web afin de publier les fichiers WSDL.

À titre indicatif, Nous illustrons le fichier WSDL associé au service Web *WeatherWS* dans la Figure V-1.Alors que, les fichiers WSDL associé aux autres services (les services Web participant dans la composition et le service Web composite) sont présentés dans l'annexe C.

Les fonctionnalités de service Web *WeatherWS* sont décrites par les éléments abstraits et concrets du langage WSDL. Parmi ces derniers, nous citons *Element*, *Message*, *Operation Service* et *Binding*. Pour souci de lisibilité, nous ne présentons dans la Figure V-1 que les éléments d'entrées (*Input*s) et de sorties (*Outputs*). Notons que ces éléments (paramètres) sont décrits syntaxiquement dans le fichier WSDL (seulement le nom et son type) ; par la suite, ces paramètres (*Inputs* et *Output*) seront annotés à l'aide de notre outil d'annotions (SAWSDLAT) présenté dans la section V.3 du ce chapitre.

Les paramètres d'entrés (*Inputs*) de ce service sont :

- La ville (*CityName*) de type chaine de caractères.
- La date (*DateWeather*) de type chaine de caractères.

Et ses paramètres de sorties (*Outputs*) sont :

- La vitesse de vent (*Wind*) de type réel.
- La température (*Temperature*) de type réel.

[23] L'environnement de développement .Net : Visual studio 2008

[24] C Sharp (C#) : langage de programmation orienté objet créé par Microsoft.

[25] IIS : serveur Web créé par Microsoft.

- La pression (*Pressure*) de type réel.

```
<wsdl:definitions targetNamespace="WeatherWS"
            :
            :
xmlns:sawsdl="http://www.w3.org/ns/sawsdl" />">
<wsdl:types>
 <s:schema elementFormDefault="qualified"
targetNamespace="WeatherWS">
  <s:element name="GetWeather">
   <s:complexType>
    <s:sequence>
     <s:element maxOccurs="1" minOccurs="0"
        type="s:string"  name="CityName"/>
     <s:element maxOccurs="1" minOccurs="0"
        type="s:string" name="DateWeather" />
    </s:sequence>
   </s:complexType>
  </s:element>
      :
      :
  <s:complexType name="WeatherInfo">
   <s:sequence>
    <s:element maxOccurs="1" minOccurs="1"
                type="s:double" name="Wind "/>
    <s:element maxOccurs="1" minOccurs="1"
                type="s:double" name="Temperature"/>
    <s:element maxOccurs="1" minOccurs="1"
                type="s:double" name="Pressure" />
   </s:sequence>
  </s:complexType>
 </s:schema>
</wsdl:types>
    :
    .
```

Les éléments *Input* du service

- Le nom de la ville
- La date

Les éléments *Outputs* du service

- La vitesse de vent
- Le degré de température.
- la pression

Figure V-1: éléments des messages (*Inputs* et *Outputs*) du service *WeatherWS*.

V.2.2 Spécification BPEL de la composition

La spécification BPEL de la composition *FlightWeatherWS* est réalisée par l'intermédiaire d'un outil graphique *Oracle BPEL Designer* de composition des services Web. Cet outil permet de créer une composition des services Web en langage BPEL. La spécification est hébergée dans un serveur d'orchestration *Oracle BPEL*[26], comme un service Web à part entière. Sa description WSDL est publiée aussi dans le même serveur d'orchestration *Oracle BPEL*.

[26] : *Oracle BPEL*: moteur d'exécution des services Web composite.

Ainsi, nous proposons un schéma global (Cf. la Figure V-2) résumant les technologies et les outils utilisés afin d'implémenter et d'héberger les services Web d'une part, et ceux utilisés afin de spécifier et d'orchestrer la composition, d' autre part.

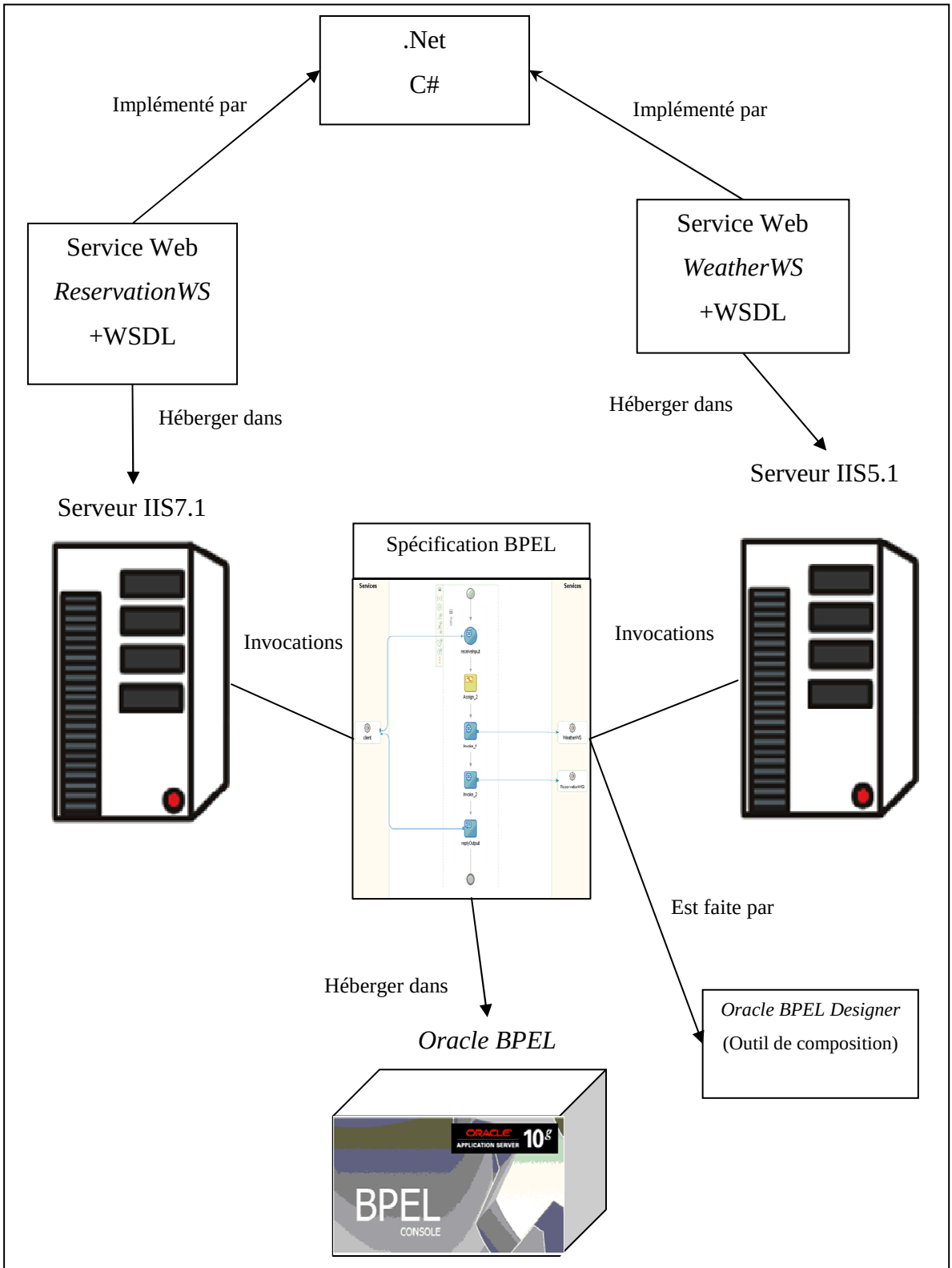

Figure V-2: illustration des outils utilisés pour implémenter la composition et les services.

V.2.3 Construction des ontologies contextuelles et de l'ontologie globale

Dans cette section, nous développons les ontologies contextuelles et l'ontologie globale de notre exemple d'illustration à l'aide de l'outil graphique *Protégé*[27] (Cf. la section II.2.4.5 du chapitre II). Protégé est un environnement graphique de développement des ontologies (Cf. la Figure V-3). Il regroupe aujourd'hui une communauté d'utilisateurs assez importante et il constitue une référence pour beaucoup d'autres outils.

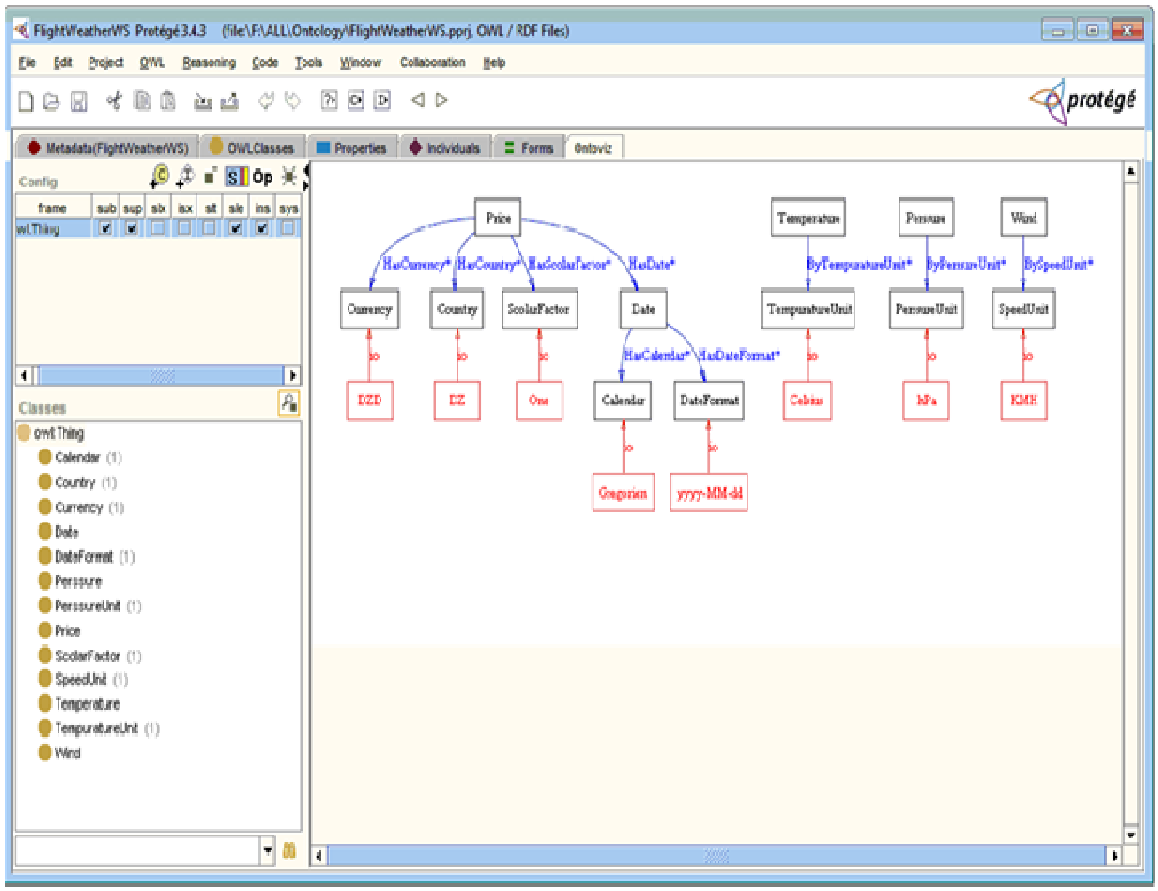

Figure V-3: interface graphique de l'éditeur protégé.

Protégé est un éditeur ontologique pour plusieurs langages à savoir RDF, RDFS, OIL et OWL (ces langages ont été présentés dans la section II.2.4.4 du chapitre II). Dans notre cas, nos ontologies sont représentées en langage OWL (*Web Ontology Language*). Le langage OWL est fondé sur une syntaxe RDF et suit le format standard d'XML. Il intègre des

[27] Protégé : http://protege.stanford.edu

constructeurs de comparaison des propriétés et des classes comme l'équivalence, la cardinalité, la symétrie, la transitivité, la disjonction, etc. Ceci dans le but de pouvoir raisonner sur les classes et les instances de l'ontologie (Cf. la section II.2.4.4 du chapitre II).

Nous illustrons dans cette section l'ontologie contextuelle associée au service Web *WeatherWS* en langage OWL (Cf. la Figure V-4) à titre indicatif. Cependant, les autres ontologies contextuelles associées aux services Web participants dans notre composition et celle de service Web composite ainsi que l'ontologie globale sont présentées en langage OWL en annexe D.

L'ontologie contextuelle de service Web *WeatherWS* composée de quatre parties principales :

- La première partie : consiste à la déclaration des espaces de nom utilisés dans la suite de fichier.
- La deuxième partie : consiste à la déclaration des concepts (classes) utilisés par le service Web. Ces concepts ont une sémantique locale explicitée dans l'ontologie contextuelle. Parmi les concepts utilisés par le service *WeatherWS*, nous citons : *Temperature*, *TempuratureUnit*, *Pressure*, *PessureUnit*, *Date*, *DateFormat*, etc.
- La troisième partie : consiste à la déclaration des relations entres les concepts, ici, les relations sont de type fonctions[28]. Nous citons par exemple la relation *HasDateFormat* entre les deux concepts *Date* et *DateFormat* et la relation *ByTempuratureUnit* entre les deux concepts *Temperature* et *TempuratureUnit.*
- La quatrième partie : consiste à la déclaration des instances (individus) des concepts explicitées.

Ces quatre parties sont bien clarifiées sur la Figure V-4 .

[28] Voir la différence entre fonction et relation dans la section II.2.2.1 du chapitre II.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  ..............
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
 xml:base="http://WeatherWS">
<owl:Class rdf:ID="SpeedUnit"/>
 <owl:Class rdf:ID="Wind"/>
 <owl:Class rdf:ID="TempuratureUnit"/>
 ..............
<owl:Class rdf:ID="Calendar"/>
 <owl:Class rdf:ID="Temperature"/>
 <owl:Class rdf:ID="DateFormat"/>
 <owl:FunctionalProperty rdf:ID="ByTempuratureUnit">
  <rdfs:domain rdf:resource="#Temperature"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#TempuratureUnit"/>
 </owl:FunctionalProperty>
..............
  <owl:FunctionalProperty rdf:ID="HasDateFormat">
  <rdfs:domain rdf:resource="#Date"/>
  <rdfs:range rdf:resource="#DateFormat"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 </owl:FunctionalProperty>
  </owl:FunctionalProperty>
</owl:FunctionalProperty>
 </owl:FunctionalProperty>
 <Calendar rdf:ID="Hijri"/>
 <SpeedUnit rdf:ID="KMH"/>
 <TempuratureUnit rdf:ID="Celsius"/>
 <PerssureUnit rdf:ID="InHg"/>
 <DateFormat rdf:ID="dd-MM-yyyy"/>
</rdf:RDF>
```

La première partie (Espace de noms)

La deuxième partie (Les classes)

La troisième partie (Les relations)

La quatrième partie (Les instances)

Figure V-4: illutration de l'ontologie contextuelle de service *WeatherWS* en format XML

V.3 Outil d'annotation contextuelle SAWSDLAT

Nous avons développé l'outil appelé SAWSDLAT (*SAWSDL Annotation Tool*) qui permet d'annoter les fichiers WSDL des services Web selon notre manière d'annotation

globale et locale, décrite dans la section IV.6 du chapitre IV, en respectant la recommandation SAWSDL du W3C.

En fait, la description WSDL est très complexe, elle est basée sur le langage XML qui nécessite des outils et des APIs pour le manipuler. De ce fait, l'outil SAWSDLAT permet l'ajout, la suppression et la modification des annotations des fichiers WSDL. Il aide les utilisateurs[29] d'annoter les fichiers WSDL associés aux services Web par des descriptions supplémentaires selon notre méthode. Ces descriptions sont explicitées dans les ontologies contextuelles et les ontologies globales.

L'outil SAWSDLAT a été implémenté en utilisant le langage de programmation orienté objet *JAVA*[30] sous l'environnement de développement graphique *NetBeans*[31]. Ici, nous avons utilisé l'API *WSDL4J*[32]. Cette API permet de manipuler les fichiers WSDL en langage *JAVA*. Elle permet de stocker, de modifier et de sauvegarder le fichier WSDL. Elle permet aussi d'accéder aux éléments (*Service, Port, Operation, Message*, etc.) qui composent un fichier WSDL.

Dans notre cas, nous avons utilisé cet API pour parcourir les éléments WSDL afin de remplir les champs de l'interface graphique de l'outil SAWSDLAT (Cf. la Figure V-5). L'outil permet de visualiser sur son interface le nom du service, les noms des ports, les noms des opérations fournies par le service Web, les messages et leurs types (Input et Output) proposés par chaque opération et les éléments correspondant aux messages. Nous citons aussi les types (chaine de caractères, entier, réel, …etc.) de l'élément final à annoter (Cf. la Figure V-5).

Pour l'annotation contextuelle, nous avons développé deux méthodes, l'une nommée SAWSDLG (*SAWSDL Global*) et l'autre nommée SAWSDCL (*SAWSDL Local*).La première pour ajouter, modifier et/ou supprimer les URIs des concepts de l'ontologie globale correspondant aux éléments que nous souhaitons annoter. Et la deuxième pour ajouter, modifier et/ou supprimer les URIs des concepts explicités dans les ontologies contextuelles associées aux services Web selon la recommandation SAWSDL standardisée par W3C.

[29] Utilisateurs: peuvent être des fournisseurs des services Web ou des concepteurs de la composition

[30] JAVA : http://sun.com

[31] NetBeans : http://www.NetBeans.com

[32] WSDL4J: http://sourceforge.net/projects/wsdl4j.

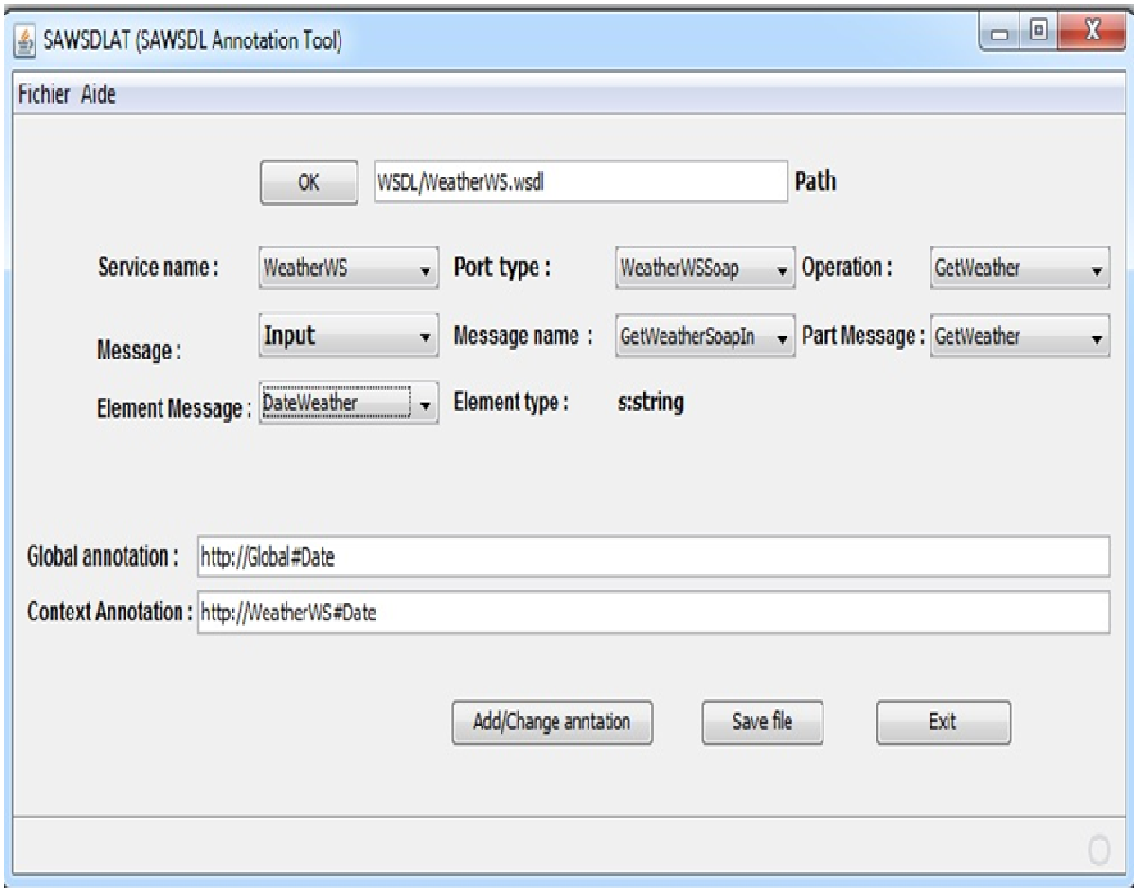

Figure V-5: outil d'annotation contextuelle SAWSDLAT.

L'exemple illustré sur la Figure V-5 montre l'annotation de fichier WSDL de service Web *WeatherWS*. La procédure suivie est la suivante :

- Nous indiquons le chemin ou bien l'URI de ce fichier dans le champ nommé *Path*. Une fois validé par le bouton OK, tous les éléments de fichier WSDL sont affichés dans les champs correspondent. Alors, en choisissant le type de message une liste des éléments de ce message est affichée.
- Nous sélectionnons par exemple dans cette liste l'élément *DateWeather*. sachant que cet élément a une sémantique globale et locale explicitée le concept *Date* respectivement dans l'ontologie globale référencée par l'URI (*http://Global#Date*), et dans l'ontologie contextuelle référencée par l'URI (*http://WeatherWS#Date*). Dans l'ontologie contextuelle (*http://WeatherWS*) le concept *Date* a deux propriétés sémantiques qui sont : le type de calendrier utilisé (*Calendar=Hijri*) c'est-à-dire un calendrier islamique et le format de date (*DatFormat=dd-MM-yyyy*) comme nous les avons présentés dans la section IV.3 du chapitre IV.

- Enfin, nous sauvegardons les modifications et les annotations dans le fichier WSDL par le bouton *Add/Change*.

Ces annotations contextuelles permettent d'expliciter de manière formelle la sémantique locale des concepts utilisés par les services et de faciliter par la suite, la médiation contextuelle ainsi que l'automatisation de détection des hétérogénéités contextuelles.

V.4 Outil de détection des hétérogénéités contextuelles CHDT

Nous avons développé un autre outil qui permet la détection des hétérogénéités contextuelles dans une composition de services Web. Cet outil nommé CHDT (*Contextuals Heterogeneities Detection Tool*) est une implémentation de l'algorithme de détection des hétérogénéités contextuelles présenté dans la section IV.7.1 du chapitre IV.

Il est développé en utilisant le langage de programmation orienté objet *JAVA* sous l'environnement de développement *NetBeans*. Nous avons utilisé deux APIs, *WSDL4J* pour extraire et manipuler les éléments de langage WSDL des services Web et *JENA*[33] pour raisonner et manipuler les ontologies.

L'outil CHDT analyse la spécification de la composition faite en BPEL, afin d'extraire les flux de données dans les activités (appelées *Assing* dans le langage BPEL) entre les services Web engagés dans la composition. Pour chaque activité, le programme extrait les deux variables de flux de données (l'élément d'entrée et l'élément de sortie) avec leurs services et leurs fichiers WSDL, ensuite il crée un modèle WSDL en mémoire. Il extrait les URIs pour les deux ontologies contextuelles et globale, avec l'API *JENA* qui permet de manipuler (raisonner, lire, écrire, modifier) une ontologie, le programme construit trois modèles ontologiques en mémoire (deux pour les ontologies contextuelles associées aux éléments examinés, et l'autre pour l'ontologie globale). Ensuite le programme fait le raisonnement et la comparaison entre ces trois modèles ontologiques selon l'algorithme illustré dans la section IV.7.1 du chapitre IV. Les points (les activités) critiques où il existe des hétérogénéités contextuelles sont affichés avec leurs contextes. Les indications sur les conversions entre les propriétés sémantiques nécessaires à la médiation contextuelle sont aussi données.

[33] JENA : http://jena.sourceforge.net.

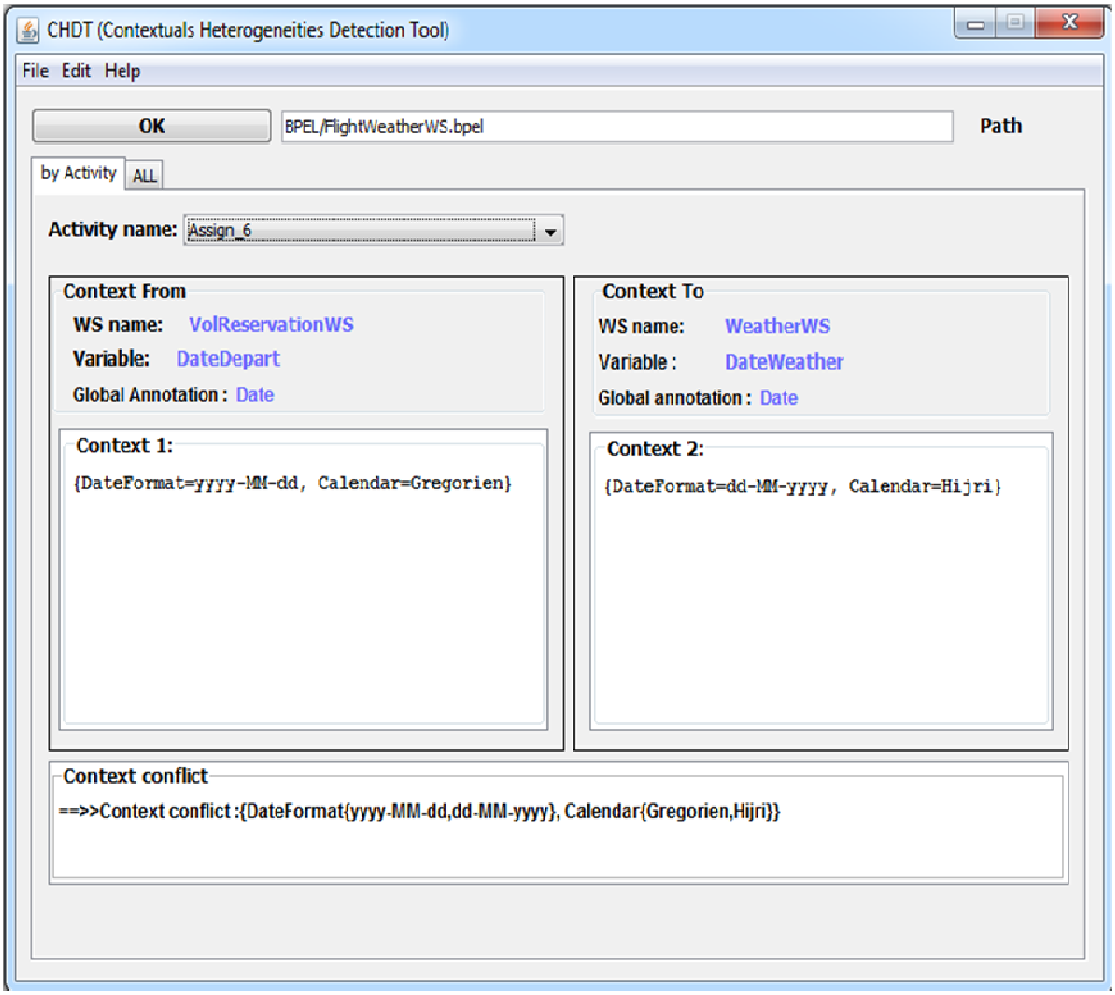

Figure V-6: outil de détection des hétérogénéités contextuelles CHDT.

La Figure V-6 montre l'interface graphique de l'outil CHDT. Il est composé de deux onglets. Le premier onglet affiche la liste des activités, les conflits pour chaque activité et les conversions nécessaires. Le deuxième onglet affiche tous les conflits, comme illustré sur la Figure V-7.

Afin d'analyser et détecter les conflits contextuels l'outil CDHT nécessite que l'emplacement de fichier BPEL de la composition soit saisi dans le champ éditable nommé *Path*. Dans la Figure V-6 nous avons spécifié le fichier BPEL de notre composition (ici, le fichier de l'emplacement *BPEL/FlightWeathrWS.bpel*). Après l'exécution (en cliquant sur le bouton *OK*), le programme analyse la composition et pour chaque activité sélectionnée par l'utilisateur, il nous donne les conflits contextuels dans le premier onglet. En parallèle, il affiche toutes les activités qui ont des conflits contextuels dans le deuxième onglet comme illustré sur la Figure V-7. Dans notre cas, l'outil détecte deux hétérogénéités contextuelles, une dans l'activité nommé *Assine_6* et l'autre des l'activité *Assign_14*.

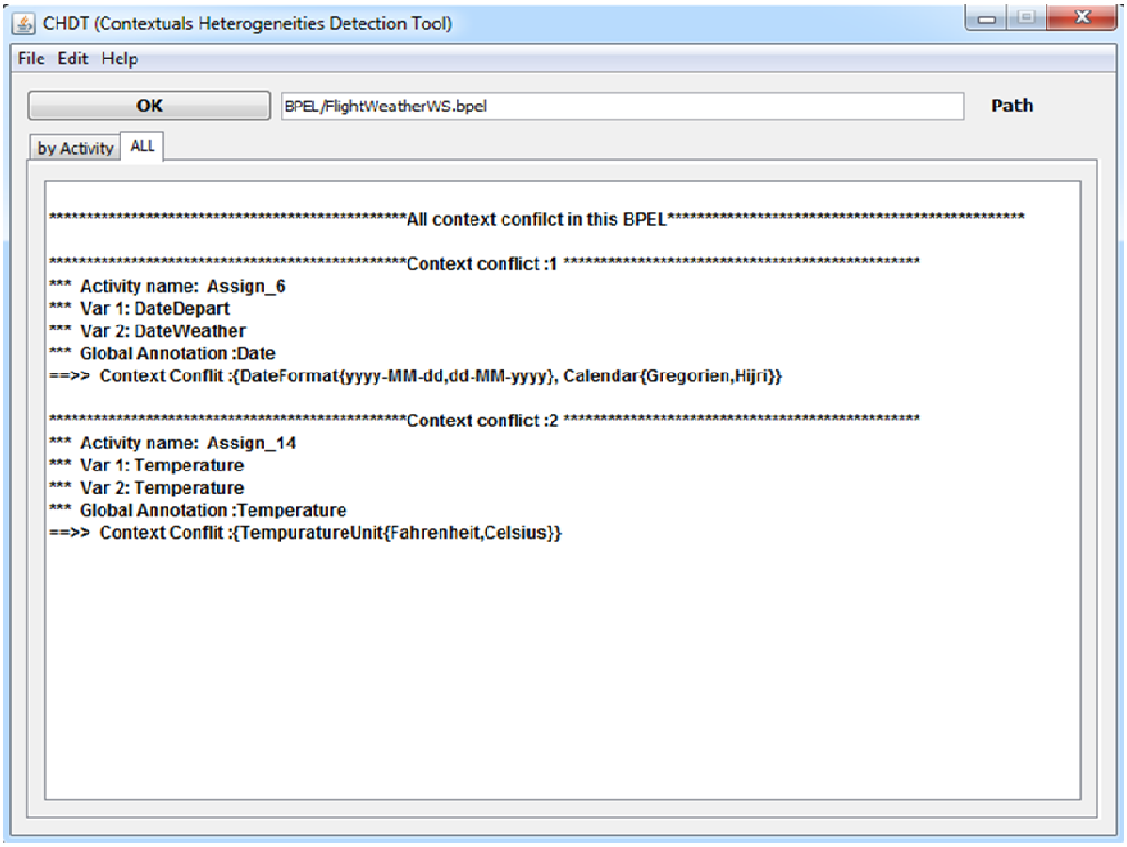

Figure V-7: onglet affiche toutes les hétérogénéités contextuelles dans la composition

Notre proposition de conversion et d'insertion des médiateurs contextuels dans les activités critiques est présentée dans la section IV.7.2 du chapitre précédent.

V.5 Conclusion

Afin de valider notre approche présentée dans le chapitre IV, nous avons réalisé la composition de l'exemple d'illustration donnée de ce même chapitre. Ensuite, nous avons développé deux outils d'aide, l'un appelé SAWSDLAT qui facilite aux fournisseurs et aux développeurs des services Web l'annotation des fichiers WSDL par les descriptions contextuelles explicitées dans les ontologies contextuelles et globales, selon notre manière proposée dans la IV.6 du chapitre IV. Et l'autre appelé CHDT qui facilite aux développeurs des processus métiers (composition) la détection des hétérogénéités contextuelles qui gênent le bon déroulement de la composition. Ces deux outils servent à faciliter la médiation contextuelle en insérant des services Web médiateurs associés à des fonctions de conversions (atomiques et composites) comme nous les avons présentées dans la section IV.5 du chapitre IV. Dans ce chapitre, nous avons aussi présenté les outils et les environnements de

développement des services Web, l'outil de spécification de la composition, les serveurs Web pour les déployer, l’éditeur ontologique utilisé pour développer les ontologies contextuelles et globales ainsi que les APIs et le langage utilisé pour développer les deux outils d’aide.

Conclusion et perspectives

Conclusion et perspectives

Conclusion

Aujourd'hui, l'interopérabilité est devenue un domaine de recherche fondamental des systèmes d'information distribués et hétérogènes. Les services Web sont considérés comme une solution potentielle aux problèmes d'interopérabilités. Ils définissent un nouveau paradigme de développement des interactions entre des applications distribuées de manière à ce qu'elles restent indépendantes des environnements et des plateformes d'exécution d'une part et aussi des choix des langages de développement et technologies d'implémentations utilisés d'une autre part. La composition de services Web en particulier permet de combiner plusieurs fonctionnalités des services Web afin de répondre aux exigences qu'un seul service ne peut satisfaire.

Malgré les avantages indéniables qu'apportent les services Web au niveau des problèmes d'interopérabilité, leur composition est souvent confrontée à plusieurs hétérogénéités. Nous pouvons citer l'hétérogénéité de type syntaxique, structurelle, sémantique et contextuelle des données échangées entre service Web. En effet, les services Web n'ont pas été conçus initialement pour répondre aux exigences d'échanges sémantiques. Par conséquence, ces hétérogénéités (notamment sémantique et contextuelle) sont gérées de manière manuelle pendant l'étape de conception d'une composition.

Afin de remédier à cette lacune, plusieurs approches ont été proposées. D'une part, pour l'enrichissement de la description des services Web par la sémantique (généralement explicitée dans des ontologies) et d'autre part pour la médiation qui est nécessaire afin de réconcilier les hétérogénéités entre les services Web. La médiation est classée selon le niveau d'hétérogénéité pris en charge. Les approches de médiations contextuelles de données sont favorisées pour plusieurs raisons, essentiellement :

- Elles n'imposent pas aux fournisseurs d'adapter leur sémantique locale avec la sémantique globale de la composition.
- Les tâches de détection et de résolution des hétérogénéités ne sont pas confiées aux concepteurs de la composition. Ces tâches sont prises en charge par le système et effectuées de manière automatique ce qui diminue les erreurs possibles lors de la composition.

Cependant, peu de travaux se sont basés sur de telles approches de médiation. On peut dire que le seul travail basé sur ces approches est celui proposé par [Mrissa 07] qui nécessite lui aussi plusieurs améliorations.

Dans le but de contribuer dans ce domaine, nous avons proposé une approche de médiation de données orientée contexte afin de réconcilier les hétérogénéités, notamment les hétérogénéités sémantiques et contextuelles, des données échangées entre les services Web participant dans une composition.

Notre proposition est une amélioration de plusieurs aspects présentés dans [Mrissa 07]. Ceci concerne les points suivant :

- Intégration de contexte de données de la composition (le service Web composite). L'objectif est de pouvoir détecter et résoudre automatiquement les hétérogénéités entre les services Web participant dans la composition d'une part, les services Web participant et la composition elle-même d'autre part.
- Proposition d'une solution pour résoudre les hétérogénéités entre les éléments de deux contextes différents avec prise en charge de contexte de données de la composition.
- Annotation des fichiers WSDL des services Web par les sémantiques explicitées dans les ontologies contextuelles couplées avec l'ontologie globale. Ceci en utilisant une annotation plus standardisée pour ne pas alourdir les fichiers WSDL.
- Définition des fonctions de conversions (atomiques et composites) associées aux propriétés sémantiques formant le contexte. Ces dernières permettent de convertir les valeurs des propriétés sémantiques et les données échangées entre les services.
- Proposition d'un algorithme de détection des hétérogénéités contextuelles de données dans une composition, en prenant en considération le contexte des données de la composition elle-même.

Perspectives

Ce travail nous a permis de dégager plusieurs perspectives, à savoir:

L'intégration de contexte de données de la composition nous permettra de personnaliser la composition selon le contexte d'utilisation, en modélisant le contexte de données de l'utilisateur final (profil d'utilisateur).

L'implémentation et l'insertion des services Web médiateurs associés aux fonctions de conversions contextuelles (atomiques et composites) sont effectués de manière manuelle. Nous proposons de générer et d'insérer automatiquement ces services médiateurs à partir des fonctions de conversions et des contextes de données des services Web.

La détection des hétérogénéités contextuelles proposée dans notre travail est effectuée pendant l'étape de conception (Design-time). Pour une médiation plus dynamique qui permet aux fournisseurs de changer leur contexte sans informer préalablement les clients, nous envisageons que la détection soit effectuée pendant l'étape d'exécution (Run-time).

Nous proposons d'utiliser le contexte de données dans une approche de découverte pour aider les concepteurs à désélectionner les services Web, qui ne sont pas pertinents par rapport aux besoins de la composition.

Bibliographie

Bibliographie

[Aît-Bachir 08] A. Aît-Bachir,ArchiMed un canevas pour la détection et la résolution des incompatibilités des conversations entre services web,Université Joseph Fourier-Grenoble 1,thése de doctorat ,2008.

[Akkiraju et al 05] R.Akkiraju, J.Farrell,J.A.Miller, M.Nagarajan,M. Schmidt, A.Sheth, K.Verma ,Web Service Semantics - WSDL-S, Rapport téchnique, IBM,http://www.w3.org/Submission/WSDL-S/, 2005.

[Bachimont 00] B.Bachimont, Engagement Sémantique et Engagement Ontologique : Conception et Réalisation D'ontologies En Ingénierie Des Connaissances, chapter 19, pages 305–324.Eyrolles, 2000.

[Bala 07] M. Bala, Interopérabilité Sémantique des Systèmes d'Information Distribués ,thèse de magister, Institut National de Formation en Informatique INI Alger, 2007.

[Bhiri 05] S. Bhiri, Approche Transactionnelle pour Assurer des Compositions Fiables de Services Web ,Université Henri Poincaré – Nancy 1, thése de doctorat , 2005.

[Blanc 06] S. Blanc, Contribution à la caractérisation et à l'évaluation de l'interopérabilité pour les entreprises collaboratives, thèse de doctorat , UNIVERSITE BORDEAUX 1, 2006.

[Boudali 07] F. Boudali, Publication et découverte des web services pour le domaine du e-learning, mémoire de magister,INI Alger, 2007.

[Bowers et al 04] S. Bowers and B. Ludascher. An ontology-driven framework for data transformation in scientific workflows, Springer, 2004.

[Cabral et a 05] L. Cabral and J. Domingue, Mediation of semantic web services in irs-iii ,In First International Workshop on Mediation in Semantic Web Services (MEDIATE 2005),Amsterdam ,2005.

[Chelbabi 06] M. Chelbabi,Découverte de Services Web Sémantiques : une Approche basée sur le Contexte, mémoire de magister, INI Alger , 2006.

[Christensen et al 01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, Web Services Description Language (WSDL) 1.1, rapport téchnique,W3C,http://www.w3.org/TR/wsdl, 2001.

[Deborah et al 04] F. Deborah,P. McGuinness, H M. Frank ,Owl web ontology language, rapport technique, W3C, http://www.w3.org/TR/owl-features/ , 2004.

[Dey et al 01] A. K. Dey, G. D. Abowd et D. Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction Journal, 2001.

[Duarte-Amaya 07] H. Duarte-Amaya,Canevas pour la composition de services web avec propriétés transactionnelles,UNIVERSITE JOSEPH FOURIER, thése de doctorat , 2007.

[Dumez 10] C. Dumez,Approche dirigée par les modéles pour la spécification, la vérification formelle et la mise en oeuvre de services Web composés,Université de Technologie de Belfort-Montbéliard ,thése de doctorat,2010.

[Farrell et al 07] J. Farrell et H. Lausen, SAWSDL : Semantic Annotations for WSDL and XML Schema,rapport téchnique, W3C,http://www.w3.org/TR/sawsdl/, 2007.

[Furst 02] F. Furst. L'ingénierie ontologique, rapport technique, Institut de recherche en Informatique de Nantes, 2002.

[Genesereth 05] Michael R. Genesereth,KIF:Knowledge Interchange Format,rapport téchnique, site web: http://logic.stanford.edu/kif/dpans.html, 2005.

[Gruber 92] T R. Gruber, A Translation Approach to Portable Ontology Specifications, raport téchnique KSL 92-71, Knowledge Systems Laboratory,Université de Stanford.1992.

[Gruber 93] T R.Gruber et R. Poli,Toward Principles for the Design of Ontologies Used for Knowledge Sharing, rapport téchnique KSL 93-04, Knowledge Systems Laboratory,Université de Stanford.1993.

[Guha 04] R.V. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, rapport téchnique, http://www.w3.org/TR/rdf-schema/,2004.

[Hachliche et al 08] K.Hachliche, I.Khalfaoui, Conception et réalisation d'un système pour la composition de services web. INI Alger, mémoire de fin d'étude, 2008.

[Haller et al 05] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. Wsmx - a semantic service-oriented architecture, IEEE Computer Society, 2005.

[Hubert et al 03] K. Hubert ,M. Valérie , LES WEB SERVICES,Edition DUNOD, 2003.

[Hunter 01 et al 01] D.Hunter ,C.Cagle, D.Gibbons, N.Ozu, J.Pinnock, P.Spence,Initiation à XML avec trois études de cas détaillées,Editions Eyrolles, 2001.

[Izza 06] S.Izza, Integration des systèmes d'information industriels, Ecole Nationale Supérieure des Mines de Saint-Etienne, thèse de doctorat, 2006.

[Kopecký 05] J. Kopecký ,Aligning WSMO and WSDL-S , rapport téchnique , http://www.wsmo.org/TR/d30/v0.1/20050805/ , 2005.

[Lausen et 05] H. Lausen,A. Polleres,D. Roman,Web Service Modeling Ontology (WSMO), rapport téchnique ,http://www.w3.org/Submission/WSMO/ , 2005.

[Lopez-velasco 08] C. Lopez-velasco ,Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation ,thèse de doctorat ,UNIVERSITE JOSEPH FOURIER, 2008.

[Manola et al 99] Frank Manola et Eric Miller, RDF: Resource Description Framework,rapport téchnique, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2009.

[Martin et al 04] D. Martin,M. Bursteinal,J. Hobbs et al, Owl-s : Semantic markup for web services,rapport téchnique, W3C, http://www.w3.org/Submission/OWL-S/, 2004.

[Mellal 07] N. Mellal, Réalisation de l'interopérabilité sémantique des systèmes, basée sur les ontologies et les flux d'information, thèse de doctorat, Université de Savoie, 2007.

[Mrissa 07] M. Mrissa,Médiation Sémantique Orientee Contexte pour la Composition de Services Web,Université Claude Bernard Lyon I,thèse de doctorat ,2007.

[Psyché et al 03] V. Psyché, O. Mendes et J. Bourdeau. Apport de l'ingénierie ontologique aux environnements de formation à distance. Revue STICEF, 10, 2003.

[Radetzki et al 04] U. Radetzki and A. B. Cremers. Iris : A framework for mediator-based composition of service-oriented software. IEEE Computer Society, 2004.

[Rampacek 06] S.Rampacek,Sémantique, interactions et langages de description des services web complexes ,thèse de doctorat ,Université de reims Champagne-Ardenne ,2006.

[Sattanathan 06] S. Sattanathan, N. C. Narendra,Z. Maamar, Ontologies for Specifying and Reconciling Contexts of Web Services,The First International Workshop on Context for Web Services,Electronic Notes in Theoretical Computer Science,2006.

[Sciore et al 94] E.Sciore,M.Siegel,and A.Rosenthal,Using semantic values to facilitate interoperability among heterogeneous information systems, ACM Trans, Database Syst,1994.

[Spencer et al 04] B.Spencer et S.Liu, Inferring data transformation rules to integrate Semantic Web Services, ISWC, Springer,2004.

[Strangand et al 04] T. Strangand and L. Popien, A context modeling survey,In WorkshopProceedings,First International Workshop on Advanced Context,Modelling,Reasoning And Management at UbiComp,2004.

[Uschold et al 96] M. Uschold et M. Gruninger, Ontologies: principles, methods, and applications, Knowledge Engineering Review, 11(2):93–155, 1996.

Annexes

Annexes

Annexe A: langage XML

I. Introduction

Le langage XML (Extensible Markup Language) est un langage de balisage générique. Il sert essentiellement à stocker/transférer des données de type texte Unicode structurées en champs arborescents. Ce langage est qualifié d'extensible car il permet à l'utilisateur de définir les balises des éléments. L'objectif initial est de faciliter l'échange automatisé de contenus entre systèmes distribués hétérogènes, donc il permet de renforcer interopérabilité.

Ce langage possède plusieurs avantages comme:

- Une syntaxe unique qui permet son adoption par divers systèmes d'exploitation.
- Une structure arborescente qui facilite sa lisibilité.
- Une grande extensibilité, car il n'impose aucune restriction d'utilisation en dehors de sa syntaxe.

Nous présentons dans cette annexe ce langage en plus de détails, nous nous basons essentiellement sur [Hunter 01 et al 01] et [Hachliche et al 08].

II. Structure d'un document XML

Un document XML est composé de trois parties comme illustré dans la figure suivante:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- '''Commentaire''' -->
<élément-document xmlns="http://exemple.org/" xml:lang=";fr">
  <élément>Texte</élément>
  <élément>élément répété</élément>
  <élément>
    <élément>Hiérarchie récursive</élément>
  </élément>
  <élément>
    Texte avec<élément>un élément</élément>inclus
  </élément>
  <élément/>
  <!-- élément vide -->
  <élément attribut="valeur"></élément>
</élément-document>
```

- **Un prologue:** Il contient des déclarations, sa présence est facultative.
- **Un arbre d'éléments:** contient le contenu de l'élément.
- **Les commentaires:** leur présence est facultative, les commentaires peuvent apparaître dans la partie prologue ou dans l'arbre d'éléments.

1. Le prologue

Le prologue contient une déclaration XML, des instructions de traitement et une déclaration de type de document.

a. Déclaration XML

La déclaration XML apporte des informations sur la version du langage utilisé, le codage des caractères et la complétude du document.

Les déclarations ont la forme suivante:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Elle indique au processeur qui va traiter le document :

- La version du langage XML utilisé dans le document est la version 1.0.
- Le codage de caractères utilisé dans le document suit la norme *UTF-8*.
- Le non existence de déclarations externes, car l'attribut *standalone* a la valeur *yes*, le processeur[34] considère que toutes les déclarations nécessaires au traitement du document y sont incluses. Si cet attribut a la valeur *no*, le processeur doit rechercher des déclarations dans d'autres fichiers pour pouvoir traiter convenablement le document.

b. Instructions de traitement

Les instructions de traitement sont facultatives, elles ne font pas partie du document. Leur contenu est simplement transmis à une application en vue de déclencher certains traitements. Elles peuvent aussi apparaître dans le corps du document.

[34] Une application qui traite le document XML.

2. L'arbre d'éléments

Les éléments sont les objets les plus importants des documents XML. En effet fondamentalement, les documents XML sont des hiérarchies strictes d'éléments. Ainsi il existe toujours un (et un seul) élément père appelé élément *racine*, qui contient tout les autres éléments.

Les éléments les plus fins, dits éléments terminaux de l'arbre, pourront être des paragraphes textuels, des entités graphiques, des liens. Tous ces éléments forment ensemble une hiérarchie unique, un arbre.

Chaque élément d'un document XML se compose d'une balise d'ouverture, d'un contenu d'élément, et d'une balise de clôture, à l'exception des éléments vides.

```
<nom> contenu de l'élément </nom>
```

c. Attributs

Les attributs sont simplement des paires nom/valeur pour décrire certaines propriétés de l'élément, les attributs sont placés immédiatement après le nom de l'élément et doivent être séparés de ce dernier d'un caractère d'espacement.

Dans l'exemple suivant l'élément rapport est un élément non vide comportant deux attributs langue, date-modif.

```
<mémoire langue= "''FR''" date-modif="''15.01.2011''"/>
```

d. Contenu d'un élément

Un élément peut contenir d'autres éléments, des données des références à des entités, des sections littérales et des instructions de traitements. Il peut aussi être vide. Soulignons aussi que rien n'interdit de faire figurer une instance d'un élément à l'intérieur d'une autre instance du même type d'élément.

3. Commentaires

Des commentaires peuvent être inclus dans un document. Ils sont encadrés par les marques de début et de fin de commentaire :

```
<!-- '''Commentaire''' -->
```

Le corps d'un commentaire peut contenir n'importe quel caractère à l'exception de la chaine double-tiret.

III. Espace de noms

XML définit un système permettant de créer des balises modulaires, c'est-à-dire pouvoir donner la possibilité d'utiliser des balises provenant de différents langages à balise au sein d'un même document grâce à la notion d'espace de noms.

La définition d'un espace de nom permet d'associer toutes les balises d'un langage à un groupe afin d'être capable de mêler différents langages à balise dans un même document XML.

IV. XML Schéma

Les schémas XML permettent de décrire des classes de documents XML ; ils apportent une modularité qui permet de définir des structures très génériques en les spécialisant par des mécanismes d'héritage et de surcharge. Cela se fait via des composants de modélisation qui permettent de définir les contraintes et les relations de dépendances des constituants d'un document XML : les types de données, les éléments et leurs contenus et les attributs et leurs valeurs. Les schémas permettent également la gestion des valeurs par défaut des attributs et des éléments.

V. Traitement de données XML

Toute application qui doit traiter des entités XML est interfacée avec un parseur à travers une API. Il existe deux API standards, qui correspondent à deux modèles de traitement différents des documents. Dans le premier modèle, le parseur lit le flot de données XML en entrée, et reconnaît et interprète les marques de balisage au fur à mesure qu'il les rencontre. Chaque construction reconnue est immédiatement signalée et passée à l'application. L'API standard qui implémente ce modèle et le rend disponible aux applications s'appelle SAX (Simple API for XM).

Ce modèle d'API est simple, efficace et adapté à des traitements adaptatifs tels que :

- Vérification syntaxique de conformité et/ ou de validité d'un document.
- Expansion des noms qualifiés définis dans des domaines nominaux.
- Fragmentation d'un document en sous arbres en vue de son indexation et/ou de son stockage dans un système de base de données.

- Remplacement des références à des entités par leur valeur.
- Transcodage des caractères.

Dans le second système le parseur compile l'ensemble du document et en construit une représentation interne sous forme de bosquet, c'est-à-dire un graphe composé d'arbres, et d'arcs reliant certains nœuds d'un arbre à un autre. L'API propose alors à l'application un ensemble de fonctions pour naviguer dans ce bosquet. L'API standard qui implémente ce modèle est le DOM (Document Object Model).

Ce modèle plus gourmand en ressources, est en revanche parfaitement adapté à la programmation d'applications interactives, dans lesquelles la totalité du bosquet d'un document est exposé à l'application, et, à travers elle, à l'utilisateur final.

Annexe B: langage de composition BPEL

I. Introduction

BPEL est un langage standard pour l'orchestration de services web proposé par IBM, Microsoft et BEA. Il résulte de la fusion de deux langages concurrents:

- WSFL (Web Service Flow Language) proposé par IBM
- et XLANG (XML Business Process Language) de Microsoft.

BPEL est un langage basé sur XML pour spécifier le comportement d'un processus métiers (la composition). En effet, ce langage permet de définir les processus métiers qui externalisent leurs fonctionnalités comme des services web [Hubert et al 03] et [Rampacek 06].

II. Syntaxe du langage BPEL

Une composition des services Web faite en BPEL est composée de quatre parties qui sont:

- **Les attributs supérieurs**

Cette première partie permet la déclaration des informations concernant le processus métier (la composition) comme son nom, les espaces de noms supportés, les documents WSDL ou schémas XML importés par le processus métier, comme illustré sur la figure suivante:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<process name="FlightWeatherWS"
     targetNamespace="http://xmlns.oracle.com/FlightWeatherWS"
     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:ns1="VolReservationWS"
     xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:ns2="WeatherWS"
     xmlns:client="http://xmlns.oracle.com/FlightWeatherWS"
     xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
     xmlns:ora="http://schemas.oracle.com/xpath/extension"
```

- **Lien de partenaire:**

Cette deuxième partie permet la déclaration des *partnerlinks* (les services Web engagés dans la composition): ces derniers définissent une liaison entre les ensembles d'opérations des services invoqués par le service Web composite décrit en BPEL, et un nom de liaison bien précis du service Web engagé dans la composition.

Ces liaisons sont appelées *partner* et permettent ainsi d'identifier facilement les différents services Web participants dans cette composition. La figure suivante illustre cette deuxième partie.

```
<partnerLinks>
  <partnerLink name="client" partnerLinkType="client:FlightWeatherWS"
         myRole="FlightWeatherWSProvider"
         partnerRole="FlightWeatherWSProvider"/>
  <partnerLink name="VolReservationWS" myRole="VolReservationWSSoap_Role"
         partnerRole="VolReservationWSSoap_Role"
         partnerLinkType="ns1:VolReservationWSSoap_PL"/>
  <partnerLink myRole="WeatherWSSoap_Role" name="WeatherWS"
         partnerRole="WeatherWSSoap_Role"
         partnerLinkType="ns2:WeatherWSSoap_PL"/>
 </partnerLinks>
```

- **Les variables**:

Cette troisième partie permet de déclarer des variables, utilisant les types importés de descriptions WSDL associées à la spécification BPEL. Ces variables seront utilisées dans la partie description comportementale du processus métier, pour maintenir un état au niveau du service Web, et ainsi assurer des liaisons de données entre deux opérations. La déclaration de cette troisième partie faite de la manière suivante:

```
<variables>
  <variable name="inputVariable"
       messageType="client:FlightWeatherWSRequestMessage"/>
  <variable name="outputVariable"
       messageType="client:FlightWeatherWSResponseMessage"/>
  <variable name="Invoke_1_GetBillet_InputVariable"
       messageType="ns1:GetBilletSoapIn"/>
  <variable name="Invoke_1_GetBillet_OutputVariable"
       messageType="ns1:GetBilletSoapOut"/>
  <variable name="Invoke_2_GetWeather_InputVariable"
       messageType="ns2:GetWeatherSoapIn"/>
  <variable name="Invoke_2_GetWeather_OutputVariable"
       messageType="ns2:GetWeatherSoapOut"/>
 </variables>
```

- **Les activités:**

Les activités décrivent le comportement du la composition elle-même en utilisant différents opérateurs. Elles peuvent être basiques ou complexes. Les activités complexes sont composées d'autres activités basiques ou complexes.

La figure suivante illustre quelque activité utilisée dans notre travail.

```
 <assign name="Assign_15">
  <copy>
   <from part="parameters"
       query="/ns2:GetWeatherResponse/ns2:GetWeatherResult/ns2:Pressure"
       variable="Invoke_2_GetWeather_OutputVariable"/>
   <to variable="outputVariable" part="payload"
     query="/client:FlightWeatherWSProcessResponse/client:Pression"/>
  </copy>
 </assign>
 <reply name="replyOutput" partnerLink="client"
     portType="client:FlightWeatherWS" operation="process"
     variable="outputVariable"/>
</sequence>
```

Les deux tableaux suivant résument les différentes activités basiques et complexes du langage BPEL.

Les activités basiques	Rôles
<invoke>	Invoque une opération dans un service web.
<receive>	Attend un message d'une source externe.
<reply>	Répond à une source externe.
<assign>	Attend un certain moment.
<wait>	Copie les données d'une place à l'autre.
<throw>	Lance une erreur d'exécution.
<terminate>	Termine l'instance de service en entier.
<empty>	Ne fait rien.
Les activités complexes	**Rôles**
< sequence>	Définit un ordre d'exécution.
<switch>	Exprime L'acheminement conditionnel des tâches.
<while>	Exprime la boucle.
<pick>	Attend l'arrivée d'un événement.
<flow>	Exprime l'acheminement parallèle des tâches.

Annexe C: fichiers WSDL annotés des services Web

Dans cette annexe, nous présenterons le fichier WSDL de service Web composite (la composition) *FlightWeatherWS* et les fichiers WSDL des deux services Web *ReservationWS* et *WeatherWS* de notre exemple présenté dans la section IV.2 du chapitre IV en format texte, nous n'illustrerons ici que les éléments des messages annotés par les descriptions contextuelles et les descriptions globales. Les annotations sont illustrées en couleur vert foncé.

1. Fichier WSDL du service Web composite FlightWeatherWS

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:client="http://xmlns.oracle.com/FlightWeatherWS"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://xmlns.oracle.com/FlightWeatherWS" name="FlightWeatherWS"
targetNamespace="http://xmlns.oracle.com/FlightWeatherWS">
 <types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
elementFormDefault="qualified" targetNamespace="http://xmlns.oracle.com/FlightWeatherWS">
      <element name="FlightWeatherWSProcessRequest">
        <complexType>
          <sequence>
            <element name="VilleDepart" sawsdl:modelReference="http://Global#City
http://FlightWeatherWS#City" type="string"/>
            <element name="DateAller" sawsdl:modelReference="http://Global#Date
http://FlightWeatherWS#Date" type="string"/>
            <element name="VilleDestination" sawsdl:modelReference="http://Global#City
http://FlightWeatherWS#City" type="string"/>
            <element name="DateRetour" sawsdl:modelReference="http://Global#Date
http://FlightWeatherWS#Date" type="string"/>
          </sequence>
        </complexType>
      </element>
      <element name="FlightWeatherWSProcessResponse">
        <complexType>
          <sequence>
            <element name="VilleDepart" sawsdl:modelReference="http://Global#City
http://FlightWeatherWS#City" type="string"/>
            <element name="DateAller" sawsdl:modelReference="http://Global#Date
http://FlightWeatherWS#Date" type="string"/>
            <element name="VilleDestination" sawsdl:modelReference="http://Global#City
http://FlightWeatherWS#City" type="string"/>
            <element name="DateRetour" sawsdl:modelReference="http://Global#Date
http://FlightWeatherWS#Date" type="string"/>
            <element name="NumeroVol" type="int"/>
            <element name="Prix"
sawsdl:modelReference="http://Global#Price http://FlightWeatherWS#Price" type="double"/>
```

```
            <element name="Vent" sawsdl:modelReference="http://Global#Wind
http://FlightWeatherWS#Wind" type="double"/>

            <element name="Temperature" sawsdl:modelReference="http://Global#temperature
http://FlightWeatherWS#Temperature" type="double"/>

            <element name="Pression" sawsdl:modelReference="http://Global#Perssure
http://FlightWeatherWS#Perssure" type="double"/>

          </sequence>

        </complexType>

      </element>

    </schema>

       </types>

      <message name="FlightWeatherWSRequestMessage">

          .

          .

          .

          .
```

2. Fichier WSDL du service Web ReservationWS

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions targetNamespace="VolReservationWS"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="VolReservationWS" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">

 <wsdl:types>

  <s:schema elementFormDefault="qualified" targetNamespace="VolReservationWS">

   <s:element name="GetBillet">

    <s:complexType>

     <s:sequence>

      <s:element maxOccurs="1" minOccurs="0" name="CityAirportDepart"
sawsdl:modelReference="http://Global#City" type="s:string"/>

      <s:element maxOccurs="1" minOccurs="0" name="DateDepart"
sawsdl:modelReference="http://Global#Date http://VolReservationWS#Date" type="s:string"/>

      <s:element maxOccurs="1" minOccurs="0" name="CityAirportDestination"
sawsdl:modelReference="http://Global#City" type="s:string"/>

      <s:element maxOccurs="1" minOccurs="0" name="DateRetour"
sawsdl:modelReference="http://Global#Date http://VolReservationWS#Date" type="s:string"/>

     </s:sequence>

    </s:complexType>

   </s:element>

   <s:element name="GetBilletResponse">

    <s:complexType>

     <s:sequence>

      <s:element maxOccurs="1" minOccurs="1" name="GetBilletResult" type="tns:BilletAvion"/>

     </s:sequence>

    </s:complexType>

   </s:element>
```

```
   <s:complexType name="BilletAvion">
    <s:sequence>
     <s:element maxOccurs="1" minOccurs="0" name="CityAirportDepart"
sawsdl:modelReference="http://Global#City" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="0" name="DateDepart"
sawsdl:modelReference="http://Global#Date http://VolReservationWS#Date" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="0" name="CityAirportDestination"
sawsdl:modelReference="http://Global#City" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="0" name="DateRetour"
sawsdl:modelReference="http://Global#Date http://VolReservationWS#Date" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="1" name="NumeroVol" type="s:int"/>
     <s:element maxOccurs="1" minOccurs="1" name="PrixVol"
sawsdl:modelReference="http://Global#Price http://VolReservationWS#Price" type="s:double"/>
    </s:sequence>
   </s:complexType>
  </s:schema>
 </wsdl:types>
 <wsdl:message name="GetBilletSoapIn">
  <wsdl:part name="parameters" element="tns:GetBillet">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="GetBilletSoapOut">
  <wsdl:part name="parameters" element="tns:GetBilletResponse">
  </wsdl:part>
      </wsdl:message>
         .
         .
```

3. Fichier WSDL du service web WeatherWS

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="WeatherWS"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="WeatherWS" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
 <wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="WeatherWS">
   <s:element name="GetWeather">
    <s:complexType>
     <s:sequence>
      <s:element maxOccurs="1" minOccurs="0" name="CityName"
sawsdl:modelReference="http://Global#City" type="s:string"/>
      <s:element maxOccurs="1" minOccurs="0" name="DateWeather"
sawsdl:modelReference="http://Global#Date http://WeatherWS#Date" type="s:string"/>
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:element name="GetWeatherResponse">
```

```
    <s:complexType>
     <s:sequence>
      <s:element maxOccurs="1" minOccurs="1" name="GetWeatherResult" type="tns:WeatherInfo"/>
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:complexType name="WeatherInfo">
    <s:sequence>
     <s:element maxOccurs="1" minOccurs="1" name="Wind"
sawsdl:modelReference="http://Global#Wind http://WeatherWS#Wind" type="s:double"/>
     <s:element maxOccurs="1" minOccurs="1" name="Temperature"
sawsdl:modelReference="http://Global#Temperature http://WeatherWS#Temperature"
type="s:double"/>
     <s:element maxOccurs="1" minOccurs="1" name="Pressure"
sawsdl:modelReference="http://Global#Perssure http://WeatherWS#Perssure" type="s:double"/>
    </s:sequence>
   </s:complexType>
  </s:schema>
 </wsdl:types>
 <wsdl:message name="GetWeatherSoapOut">
  <wsdl:part name="parameters" element="tns:GetWeatherResponse">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="GetWeatherSoapIn">
  <wsdl:part name="parameters" element="tns:GetWeather">
    .
    .
    .
    .
```

Annexe D: ontologies contextuelles et ontologie globale

Dans cette annexe, nous présenterons les ontologies contextuelles et l'ontologie globale.

1. Ontologie contextuelle du service Web composite *FlightWeatherWS*

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://FlightWeatherWS#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xml:base="http://FlightWeatherWS">
 <owl:Ontology rdf:about=""/>
 <owl:Class rdf:ID="SpeedUnit"/>
<owl:Class rdf:ID="DateFormat"/>
 <owl:Class rdf:ID="TempuratureUnit"/>
 <owl:Class rdf:ID="Perssure"/>
<owl:Class rdf:ID="Calendar"/>
<owl:Class rdf:ID="PerssureUnit"/>
<owl:Class rdf:ID="Country"/>
<owl:Class rdf:ID="Price"/>
 <owl:Class rdf:ID="Date"/>
 <owl:Class rdf:ID="Currency"/>
<owl:Class rdf:ID="ScolarFactor"/>
<owl:Class rdf:ID="Unit"/>
<owl:Class rdf:ID="Temperature"/>
 <owl:Class rdf:ID="Wind"/>
 <owl:ObjectProperty rdf:ID="BySpeedUnit">
  <rdfs:range rdf:resource="#SpeedUnit"/>
  <rdfs:domain rdf:resource="#Wind"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasDateFormat">
  <rdfs:range rdf:resource="#DateFormat"/>
  <rdfs:domain rdf:resource="#Date"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasCountry">
  <rdfs:range rdf:resource="#Country"/>
```

```
  <rdfs:domain rdf:resource="#Price"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasScolarFactor">
  <rdfs:domain rdf:resource="#Price"/>
  <rdfs:range rdf:resource="#ScolarFactor"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasCurrency">
  <rdfs:domain rdf:resource="#Price"/>
  <rdfs:range rdf:resource="#Currency"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="ByPerssureUnit">
  <rdfs:domain rdf:resource="#Perssure"/>
  <rdfs:range rdf:resource="#PerssureUnit"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="ByTempuratureUnit">
  <rdfs:domain rdf:resource="#Temperature"/>
  <rdfs:range rdf:resource="#TempuratureUnit"/>
 </owl:ObjectProperty>
 <owl:FunctionalProperty rdf:ID="HasCalendar">
  <rdfs:domain rdf:resource="#Date"/>
  <rdfs:range rdf:resource="#Calendar"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 </owl:FunctionalProperty>
 <owl:InverseFunctionalProperty rdf:ID="HasDate">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Date"/>
  <rdfs:domain rdf:resource="#Price"/>
 </owl:InverseFunctionalProperty>
 <ScolarFactor rdf:ID="One"/>
 <Currency rdf:ID="DZD"/>
 <Country rdf:ID="DZ"/>
 <PerssureUnit rdf:ID="hPa"/>
 <DateFormat rdf:ID="yyyy-MM-dd"/>
 <SpeedUnit rdf:ID="KMH"/>
 <TempuratureUnit rdf:ID="Celsius"/>
 <Calendar rdf:ID="Gregorien"/>
</rdf:RDF>
```

2. Ontologie contextuelle de service Web *ReservationWS*

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
```

```
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://VolReservationWS#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
 xml:base="http://VolReservationWS">
<owl:Ontology rdf:about="file:/F:/ALL/Ontologie/VolReservationWS/VolReservationWS.owl"/>
<owl:Class rdf:ID="Country"/>
<owl:Class rdf:ID="Perssure"/>
<owl:Class rdf:ID="Date"/>
<owl:Class rdf:ID="Calendar"/>
<owl:Class rdf:ID="Wind"/>
<owl:Class rdf:ID="Temperature"/>
<owl:Class rdf:ID="DateFormat"/>
<owl:Class rdf:ID="Currency"/>
<owl:Class rdf:ID="Unit"/>
<owl:Class rdf:ID="TempuratureUnit"/>
<owl:Class rdf:ID="ScolarFactor"/>
<owl:Class rdf:ID="PerssureUnit"/>
<owl:Class rdf:ID="Price"/>
<owl:Class rdf:ID="SpeedUnit"/>
<owl:ObjectProperty rdf:ID="BySpeedUnit">
  <rdfs:domain rdf:resource="#Wind"/>
  <rdfs:range rdf:resource="#SpeedUnit"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="MeasuredBy"/>
 <owl:ObjectProperty rdf:ID="ByTempuratureUnit">
  <rdfs:domain rdf:resource="#Temperature"/>
  <rdfs:range rdf:resource="#TempuratureUnit"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="hastest">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="ByPerssureUnit">
  <rdfs:domain rdf:resource="#Perssure"/>
  <rdfs:range rdf:resource="#PerssureUnit"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasDateFormat">
  <rdfs:domain rdf:resource="#Date"/>
  <rdfs:range rdf:resource="#DateFormat"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasCurrency">
  <rdfs:range rdf:resource="#Currency"/>
```

```
  <rdfs:domain rdf:resource="#Price"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasScolarFactor">
  <rdfs:range rdf:resource="#ScolarFactor"/>
  <rdfs:domain rdf:resource="#Price"/>
 </owl:ObjectProperty>
 <owl:FunctionalProperty rdf:ID="HasCalendar">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Date"/>
  <rdfs:range rdf:resource="#Calendar"/>
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:ID="HasCountry">
  <rdfs:range rdf:resource="#Country"/>
  <rdfs:domain rdf:resource="#Price"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 </owl:FunctionalProperty>
 <owl:InverseFunctionalProperty rdf:ID="HasDate">
  <rdfs:domain rdf:resource="#Price"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Date"/>
 </owl:InverseFunctionalProperty>
 <Currency rdf:ID="EURO"/>
 <Calendar rdf:ID="Gregorien"/>
 <SpeedUnit rdf:ID="KMH"/>
 <ScolarFactor rdf:ID="One"/>
 <DateFormat rdf:ID="yyyy-MM-dd"/>
 <Country rdf:ID="DZ"/>
 <owl:AllDifferent/>
 <PerssureUnit rdf:ID="InHg"/>
 <TempuratureUnit rdf:ID="Celsius"/>
</rdf:RDF>
```

3. Ontologie contextuelle du service Web *WeatherWS*

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://WeatherWS#"
  xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
```

```
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
xml:base="http://WeatherWS">
<owl:Ontology rdf:about="file:/F:/ALL/Ontologie/WeatherWS/WeatherWS"/>
<owl:Class rdf:ID="ScolarFactor"/>
<owl:Class rdf:ID="SpeedUnit"/>
<owl:Class rdf:ID="Wind"/>
<owl:Class rdf:ID="TempuratureUnit"/>
<owl:Class rdf:ID="Currency"/>
<owl:Class rdf:ID="Price"/>
<owl:Class rdf:ID="Perssure"/>
<owl:Class rdf:ID="Country"/>
<owl:Class rdf:ID="Date"/>
<owl:Class rdf:ID="PerssureUnit"/>
<owl:Class rdf:ID="Unit"/>
<owl:Class rdf:ID="Calendar"/>
<owl:Class rdf:ID="Temperature"/>
<owl:Class rdf:ID="DateFormat"/>
<owl:FunctionalProperty rdf:ID="ByTempuratureUnit">
 <rdfs:domain rdf:resource="#Temperature"/>
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:range rdf:resource="#TempuratureUnit"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="HasDate">
 <rdfs:domain rdf:resource="#Price"/>
 <rdfs:range rdf:resource="#Date"/>
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="HasCountry">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:domain rdf:resource="#Price"/>
 <rdfs:range rdf:resource="#Country"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="HasDateFormat">
 <rdfs:domain rdf:resource="#Date"/>
 <rdfs:range rdf:resource="#DateFormat"/>
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="BySpeedUnit">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:range rdf:resource="#SpeedUnit"/>
 <rdfs:domain rdf:resource="#Wind"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="HasScolarFactor">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:range rdf:resource="#ScolarFactor"/>
```

```
  <rdfs:domain rdf:resource="#Price"/>
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:ID="HasCurrency">
  <rdfs:domain rdf:resource="#Price"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Currency"/>
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:ID="HasCalendar">
  <rdfs:domain rdf:resource="#Date"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Calendar"/>
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:ID="ByPerssureUnit">
  <rdfs:domain rdf:resource="#Perssure"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#PerssureUnit"/>
 </owl:FunctionalProperty>
 <Calendar rdf:ID="Hijri"/>
 <SpeedUnit rdf:ID="KMH"/>
 <Country rdf:ID="SA"/>
 <ScolarFactor rdf:ID="One"/>
 <owl:AllDifferent/>
 <Currency rdf:ID="SAR"/>
 <TempuratureUnit rdf:ID="Celsius"/>
 <PerssureUnit rdf:ID="InHg"/>
 <DateFormat rdf:ID="dd-MM-yyyy"/>
</rdf:RDF>
```

4. Ontologie globale de la composition

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://www.owl-ontologies.com/Global.owl#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xml:base="http://www.owl-ontologies.com/Global.owl">
 <owl:Ontology rdf:about=""/>
 <owl:Class rdf:ID="Weather">
  <owl:equivalentClass> <owl:Class rdf:ID="meteo"/></owl:equivalentClass>
 </owl:Class>
```

```
 <owl:Class rdf:ID="Prix">
  <owl:equivalentClass> <owl:Class rdf:ID="Price"/> </owl:equivalentClass>
 </owl:Class>
 <owl:Class rdf:ID="Humidite">
  <owl:equivalentClass> <owl:Class rdf:ID="Humidity"/> </owl:equivalentClass>
 </owl:Class>
 <owl:Class rdf:ID="Country"/> <owl:Class rdf:ID="Visibilite">
  <owl:equivalentClass> <owl:Class rdf:ID="Visibility"/>
</owl:equivalentClass>
 </owl:Class>
 <owl:Class rdf:ID="Currency"/>
 <owl:Class rdf:about="#Visibility">
 <owl:equivalentClass rdf:resource="#Visibilite"/>
</owl:Class><owl:Class rdf:about="#Price">
<owl:equivalentClass rdf:resource="#Prix"/> </owl:Class>
<owl:Class rdf:ID="City">
<owl:equivalentClass>
<owl:Class rdf:ID="Ville"/>
</owl:equivalentClass> </owl:Class>
 <owl:Class rdf:ID="Date"/> <owl:Class rdf:ID="Wind">
 <owl:equivalentClass> <owl:Class rdf:ID="Vent"/>
 </owl:equivalentClass></owl:Class>
 <owl:Class rdf:ID="Temperature"/>
 <owl:Class rdf:about="#meteo">
<owl:equivalentClass rdf:resource="#Weather"/></owl:Class>
 <owl:Class rdf:ID="Pressure"> <owl:equivalentClass> <owl:Class rdf:ID="Pression"/>
</owl:equivalentClass>   </owl:Class> <owl:Class rdf:about="#Humidity"> <owl:equivalentClass
rdf:resource="#Humidite"/> </owl:Class> <owl:Class rdf:about="#Ville"> <owl:equivalentClass
rdf:resource="#City"/>   </owl:Class> <owl:Class rdf:about="#Pression"> <owl:equivalentClass
rdf:resource="#Pressure"/> </owl:Class> <owl:Class rdf:about="#Vent"> <owl:equivalentClass
rdf:resource="#Wind"/>   </owl:Class> <owl:ObjectProperty rdf:ID="HasPurssure">
  <rdfs:range rdf:resource="#Pressure"/>
  <rdfs:domain rdf:resource="#Weather"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasWind">
  <rdfs:domain rdf:resource="#Weather"/>
  <rdfs:range rdf:resource="#Wind"/>
  <owl:inverseOf rdf:resource="#HasWind"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="WeatherOfCity">
  <rdfs:range rdf:resource="#City"/>
  <rdfs:domain rdf:resource="#Weather"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="WeatherOnDate">
  <owl:inverseOf rdf:resource="#WeatherOnDate"/>
  <rdfs:range rdf:resource="#Date"/>
```

```
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
  <rdfs:domain rdf:resource="#Weather"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasTemperature">
  <rdfs:domain rdf:resource="#Weather"/>
  <rdfs:range rdf:resource="#Temperature"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasCurrency">
  <rdfs:domain rdf:resource="#Price"/>
  <rdfs:range rdf:resource="#Currency"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasVisibility">
  <rdfs:domain rdf:resource="#Weather"/>
  <rdfs:range rdf:resource="#Visibility"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasCity">
  <rdfs:range rdf:resource="#City"/>
  <rdfs:domain rdf:resource="#Country"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="HasHumidity">
  <owl:inverseOf rdf:resource="#HasHumidity"/>
  <rdfs:range rdf:resource="#Humidity"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
  <rdfs:domain rdf:resource="#Weather"/>
 </owl:ObjectProperty>
 <Currency rdf:ID="EURO"/> <Currency rdf:ID="DZD"/>
 <Currency rdf:ID="USD"/> <Currency rdf:ID="SAR"/>
</rdf:RDF>
```

Annexe E: spécification BPEL de la composition

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<process name="FlightWeatherWS"
     targetNamespace="http://xmlns.oracle.com/FlightWeatherWS"
     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20"
     xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:ns1="VolReservationWS"
     xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:ns2="WeatherWS"
     xmlns:client="http://xmlns.oracle.com/FlightWeatherWS"
     xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
     xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc"
>
 <!--
///////////////////////////////////////////////////////////////////////////////////////////////
      List of services participating in this BPEL process
///////////////////////////////////////////////////////////////////////////////////////////////
 -->
 <partnerLinks>
  <!--
   The 'client' role represents the requester of this service.
  -->
  <partnerLink name="client" partnerLinkType="client:FlightWeatherWS"
         myRole="FlightWeatherWSProvider"
         partnerRole="FlightWeatherWSProvider"/>
  <partnerLink name="VolReservationWS" myRole="VolReservationWSSoap_Role"
         partnerRole="VolReservationWSSoap_Role"
         partnerLinkType="ns1:VolReservationWSSoap_PL"/>
  <partnerLink myRole="WeatherWSSoap_Role" name="WeatherWS"
         partnerRole="WeatherWSSoap_Role"
         partnerLinkType="ns2:WeatherWSSoap_PL"/>
 </partnerLinks>
 <!--
///////////////////////////////////////////////////////////////////////////////////////////////
////////
   VARIABLES
///////////////////////////////////////////////////////////////////////////////////////////////
///////
 -->
 <variables>
  <!-- Reference to the message passed as input during initiation -->
```

```
  <variable name="inputVariable"
       messageType="client:FlightWeatherWSRequestMessage"/>
  <!-- Reference to the message that will be returned to the requester-->
  <variable name="outputVariable"
       messageType="client:FlightWeatherWSResponseMessage"/>
  <variable name="Invoke_1_GetBillet_InputVariable"
       messageType="ns1:GetBilletSoapIn"/>
  <variable name="Invoke_1_GetBillet_OutputVariable"
       messageType="ns1:GetBilletSoapOut"/>
  <variable name="Invoke_2_GetWeather_InputVariable"
       messageType="ns2:GetWeatherSoapIn"/>
  <variable name="Invoke_2_GetWeather_OutputVariable"
       messageType="ns2:GetWeatherSoapOut"/>
 </variables>
 <!--

////////////////////////////////////////////////////////////////////////////////////////////
////////
   ORCHESTRATION LOGIC
   Set of activities coordinating the flow of messages across the
   services integrated within this business process

////////////////////////////////////////////////////////////////////////////////////////////
////////
 -->
 <sequence name="main">
  <!-- Receive input from requestor. (Note: This maps to operation defined in
FlightWeatherWS.wsdl) -->
  <receive name="receiveInput" partnerLink="client"
       portType="client:FlightWeatherWS" operation="process"
       variable="inputVariable" createInstance="yes"/>
  <!-- Generate reply to synchronous request -->
  <assign name="Assign_1">
   <copy>
    <from variable="inputVariable" part="payload"
       query="/client:FlightWeatherWSProcessRequest/client:VilleDepart"/>
    <to part="parameters" query="/ns1:GetBillet/ns1:CityAirportDepart"
      variable="Invoke_1_GetBillet_InputVariable"/>
   </copy>
  </assign>
  <assign name="Assign_2">
   <copy>
    <from variable="inputVariable" part="payload"
       query="/client:FlightWeatherWSProcessRequest/client:DateAller"/>
    <to part="parameters" query="/ns1:GetBillet/ns1:DateDepart"
      variable="Invoke_1_GetBillet_InputVariable"/>
   </copy>
  </assign>
```

```
<assign name="Assign_3">
   <copy>
    <from variable="inputVariable" part="payload"
       query="/client:FlightWeatherWSProcessRequest/client:VilleDestination"/>
    <to part="parameters" query="/ns1:GetBillet/ns1:CityAirportDestination"
      variable="Invoke_1_GetBillet_InputVariable"/>
   </copy>
  </assign>
  <assign name="Assign_4">
   <copy>
    <from variable="inputVariable" part="payload"
       query="/client:FlightWeatherWSProcessRequest/client:DateRetour"/>
    <to part="parameters" query="/ns1:GetBillet/ns1:DateRetour"
      variable="Invoke_1_GetBillet_InputVariable"/>
   </copy>
  </assign>
  <invoke name="Invoke_1" partnerLink="VolReservationWS"
      portType="ns1:VolReservationWSSoap" operation="GetBillet"
      inputVariable="Invoke_1_GetBillet_InputVariable"
      outputVariable="Invoke_1_GetBillet_OutputVariable"/>
  <assign name="Assign_5">
   <copy>
    <from part="parameters"
       query="/ns1:GetBilletResponse/ns1:GetBilletResult/ns1:CityAirportDestination"
       variable="Invoke_1_GetBillet_OutputVariable"/>
    <to part="parameters" query="/ns2:GetWeather/ns2:CityName"
      variable="Invoke_2_GetWeather_InputVariable"/>
   </copy>
  </assign>
  <assign name="Assign_6">
   <copy>
    <from part="parameters"
       query="/ns1:GetBilletResponse/ns1:GetBilletResult/ns1:DateDepart"
       variable="Invoke_1_GetBillet_OutputVariable"/>
    <to part="parameters" query="/ns2:GetWeather/ns2:DateWeather"
      variable="Invoke_2_GetWeather_InputVariable"/>
   </copy>
  </assign>
  <invoke name="Invoke_2" partnerLink="WeatherWS" portType="ns2:WeatherWSSoap"
      operation="GetWeather"
      inputVariable="Invoke_2_GetWeather_InputVariable"
      outputVariable="Invoke_2_GetWeather_OutputVariable"/>
  <assign name="Assign_7">
   <copy>
    <from part="parameters"
```

```
      query="/ns1:GetBilletResponse/ns1:GetBilletResult/ns1:CityAirportDepart"
      variable="Invoke_1_GetBillet_OutputVariable"/>
    <to variable="outputVariable" part="payload"
      query="/client:FlightWeatherWSProcessResponse/client:VilleDepart"/>
  </copy>
 </assign>
 <assign name="Assign_8">
  <copy>
   <from part="parameters"
      query="/ns1:GetBilletResponse/ns1:GetBilletResult/ns1:DateDepart"
      variable="Invoke_1_GetBillet_OutputVariable"/>
    <to variable="outputVariable" part="payload"
      query="/client:FlightWeatherWSProcessResponse/client:DateAller"/>
  </copy>
 </assign>
 <assign name="Assign_9">
  <copy>
   <from part="parameters"
      query="/ns1:GetBilletResponse/ns1:GetBilletResult/ns1:CityAirportDestination"
      variable="Invoke_1_GetBillet_OutputVariable"/>
    <to variable="outputVariable" part="payload"
      query="/client:FlightWeatherWSProcessResponse/client:VilleDestination"/>
  </copy>
 </assign>
 <assign name="Assign_10">
  <copy>
   <from part="parameters"
      query="/ns1:GetBilletResponse/ns1:GetBilletResult/ns1:DateRetour"
      variable="Invoke_1_GetBillet_OutputVariable"/>
    <to variable="outputVariable" part="payload"
      query="/client:FlightWeatherWSProcessResponse/client:DateRetour"/>
  </copy>
 </assign>
 <assign name="Assign_11">
  <copy>
   <from part="parameters"
      query="/ns1:GetBilletResponse/ns1:GetBilletResult/ns1:NumeroVol"
      variable="Invoke_1_GetBillet_OutputVariable"/>
    <to variable="outputVariable" part="payload"
      query="/client:FlightWeatherWSProcessResponse/client:NumeroVol"/>
  </copy>
 </assign>
 <assign name="Assign_12">
  <copy>
   <from part="parameters"
      query="/ns1:GetBilletResponse/ns1:GetBilletResult/ns1:PrixVol"
```

```
        variable="Invoke_1_GetBillet_OutputVariable"/>
     <to variable="outputVariable" part="payload"
       query="/client:FlightWeatherWSProcessResponse/client:Prix"/>
    </copy>
   </assign>
   <assign name="Assign_13">
    <copy>
     <from part="parameters"
        query="/ns2:GetWeatherResponse/ns2:GetWeatherResult/ns2:Wind"
        variable="Invoke_2_GetWeather_OutputVariable"/>
     <to variable="outputVariable" part="payload"
       query="/client:FlightWeatherWSProcessResponse/client:Vent"/>
    </copy>
   </assign>
   <assign name="Assign_14">
    <copy>
     <from part="parameters"
        query="/ns2:GetWeatherResponse/ns2:GetWeatherResult/ns2:Temperature"
        variable="Invoke_2_GetWeather_OutputVariable"/>
     <to variable="outputVariable" part="payload"
       query="/client:FlightWeatherWSProcessResponse/client:Temperature"/>
    </copy>
   </assign>
   <assign name="Assign_15">
    <copy>
     <from part="parameters"
        query="/ns2:GetWeatherResponse/ns2:GetWeatherResult/ns2:Pressure"
        variable="Invoke_2_GetWeather_OutputVariable"/>
     <to variable="outputVariable" part="payload"
       query="/client:FlightWeatherWSProcessResponse/client:Pression"/>
    </copy>
   </assign>
   <reply name="replyOutput" partnerLink="client"
       portType="client:FlightWeatherWS" operation="process"
       variable="outputVariable"/>
  </sequence>
</process>
```