UNIVERSITE DJILALI BOUNAAMA-KHEMIS MILIANA

FACULTE DE SCIENCES ET DE LA TECHNOLOGIE

DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE

MEMOIRE

Pour l'obtention de diplôme

MASTER EN INFORMATIQUE

SPECIALITE : « Ingénierie de logiciels »

Présenté par

BAOUCHE Ikbal et RAHALI Mohamed

Une approche dirigée par les modèles pour la modélisation des services web composés

Soutenue publiquement le 13/06/2017 devant le jury composé de :

Président de jury :	Mr A. Khalfi Université de Khemis Miliana
Encadrant :	Mme H. Hachichi Université de Khemis Miliana
Examinateur	Mr O. Harbouche Université de Khemis Miliana
	Mr D. Bahloul Université de Khemis Miliana

Année Universitaire 2016/2017

Mes remerciements

Que toutes les personnes qui nous ont aidés durant l'élaboration de ce travail trouvent dans ces lignes l'expression de nos reconnaissances et nos profondes gratitudes. Nous tiens remercier notre encadreuse Mme. Hiba HACHICHI qui nous toujours prodigué encouragements et conseils et nous ont réservé un temps précieux tout au long de notre projet.

Nous tiens aussi à exprimer nos gratitudes et reconnaissances à tous les membres de jury d'avoir accepté d'évaluer notre travail.

Nous remercions vivement tous ceux qui nous ont assistés à développer nos compétences dans diverses disciplines tout au long du cursus universitaire.

Nous portons toutes nos gratitudes aux enseignants de l'université Djilali Bounaama à Khemis Miliana, pour leur dévouement et leur assistance tout au long de nos études universitaires.

Une pensée particulière va à toutes les personnes qui ont contribuées de près ou de loin à l'élaboration de ce travail.

Dédicaces

À nos parents pour tout le bien qu'ils n'ont cessé de nous prodiguer depuis notre naissance et pour tous les sacrifices qu'ils ont consentis pour nous.

À tous nos amis et tous ceux qu'ont été là pour nous dans les bons mais aussi dans les mauvais moments,

À nos collègues,

*J*e dédie ce travail

ملخص

في أيامنا هذه أصبحت خدمات الويب كثيرة الاستخدام خصوصا من قبل الشركات وذلك من أجل عرض أعمالهم وبياناتهم عبر شبكة الإنترنت. تعتبر عملية تركيب خدمات الويب من أكثر المواضيع التي جلبت اهتمام الباحثين، حيث أنها تمكن من معالجة المشاكل المعقدة بالاعتماد على خدمات ويب بسيطة أخرى وذلك من خلال التعاون فيما بينهم،

العمل المقدم في هذه الورقة هو مساهمة في مجال الهندسة بالنماذج (IDM). الهدف الرئيسي هو تطبيق تحويل النماذج بهدف تكوين وصف متكامل وصحيح لبنية نظام تركيب خدمات الويب. استخدمنا لغة نمذجة بيانية من نوع (UML الرسم البياني بالفئات) من اجل انشاء مخطط مسبق يضمن التركيب الصحيح لخدمات الويب.

العمل المقدم في هذا الإطار يتكون من منهجية وأداة لتحويل ملفات وصف خدمات الويب (wsdl) الى رسم بياني بالفئات، هذا التحويل يعتبر في نفس الوقت نمذجة لعملية تركيب خدمات الويب. الأداة منجزة بلغة السي شارب (C-Sharp) في وسط التطوير الخاص بميكروسوفت(visual studio) .

Résumé

De nos jours, les services web sont devenus très utilisés notamment par les entreprises pour rendre accessible leurs métiers et leurs données via le Web. La composition des services web est un

sujet qui suscite l'intérêt des chercheurs, elle offre la possibilité de traitement de problèmes complexes même avec des services simples existants tout en coopérant entre eux.

Le travail présenté dans ce mémoire est une contribution dans le domaine de l'Ingénierie Dirigée par les Modèles (IDM). Son principal objectif est l'application des techniques de la transformation de modèles à fin de modéliser la structure d'un système de composition des services web. Nous allons faire usage d'un langage de modélisation graphique de type diagramme de class UML pour obtenir des modèles de composition afin de garantir une composition des services web correcte.

Le travail présenté dans ce cadre consiste en une approche et un outil pour la transformation des fichiers WSDL en un modèle UML (diagramme de classe), cette transformation est au même temps considérer comme une modélisation d'une composition de services web. L'outil est conçu en langage C-Sharp sous l'environnement de développement Microsoft visual studio.

Abstract

Nowadays, web services have become widely used by companies to make their businesses and data accessible via the Web. The composition of web services is a subject that arouses the interest of

researchers; it offers the possibility of handling complex problems even with existing simple services while cooperating with each other.

The work presented in this thesis is a contribution in the field of Model Driven Engineering (MDE). Its main objective is the application of techniques of model transformation in order to model the structure of web services composition system. We will make use of a graphical modeling language of UML class diagram type to obtain composition models in order to guarantee a correct web services composition.

The work presented in this context consists of an approach and a tool for the transformation of WSDL files into a UML model (class diagram), this transformation is at the same time considered as a modeling of a composition of web services. The tool designed in C-Sharp under the Microsoft Visual Studio development environment.

Table des matières

Table des figures

Introduction générale

Au cours des dernières décennies, l'évolution des systèmes logiciels a conduit à une croissance continue du niveau de complexité des réseaux et des protocoles. L'intégration des applications et leur interopérabilité, la flexibilité de leur mise en œuvre et de leur reconfiguration sont apparue comme des exigences essentielles pour les solutions technologiques et les styles architecturaux modernes. Ces exigences ont conduit à l'émergence de méthodes et d'approches qui prennent en charge la conception des systèmes d'une manière efficace et permettent en même temps un haut degré de modularité et de réutilisation des composants. L'évolution des styles architecturaux a soutenu cette tendance en donnant lieu à des approches axées sur les services.

L'architecture orientée services (SOA pour Service Oriented Architecture), en tant que paradigme architectural, constitue un accomplissement de telles approches en permettant l'exposition, l'interconnexion et la réutilisation d'applications à base de services et ce, en dépit de l'hétérogénéité des domaines, des technologies et des fournisseurs impliqués. L'une des mises en œuvre d'applications à base d'architecture orientée services la plus utilisée est la technologie des services Web. Elle se concentre sur la définition d'un système logiciel comme une application distribuée qui implique une composition d'agents logiciels indépendants et interconnectés. Ces agents fournissent leurs fonctionnalités sous forme de services disponibles sur le Web et collaborent ensemble dans le but de réaliser un ensemble d'objectifs offrant des capacités à tendance commerciale.

Dans la technologie des services Web, l'infrastructure de base est constituée des standards : WSDL (Web Service Description Language), UDDI (Universal Description, Discovery and Integration) et SOAP (Simple Object Access Protocol). Les Web services permettent aux entreprises de rendre accessibles leur information et expertise via Internet. Une fois le service Web développé, l'entreprise publie la description de son interface en utilisant WSDL. Du point de vue de l'utilisateur, cette description constitue la seule information disponible concernant le service. Les utilisateurs du service Web accèdent à la description WSDL via des registres spécifiques tels qu'UDDI. SOAP est le protocole standard utilisé pour interagir avec les services Web. Ce protocole décrit entre autres le format des messages échangés.

Un service Web présente la particularité d'offrir des fonctionnalités spécifiques à une tâche et, la plupart du temps, ne peut répondre aux exigences des usagers. Afin de fournir à ces utilisateurs des

services personnalisés, il est parfois nécessaire d'organiser des services de bases existantes et de les combiner, offrant ainsi de nouvelles fonctionnalités. Dans ce contexte, deux principaux axes ont reçu une grande attention de la part des organisations de la recherche et de l'industrie. Le premier concerne la sélection des services dont les fonctionnalités répondent aux besoins des utilisateurs. Ce processus, qui est appelé sélection des services Web, est réalisé en utilisant un ensemble de critères. Le second axe de recherche concerne le processus de combinaison des services sélectionnés (appelés services composants) en un service composite. Le processus réalisant cette tâche est appelé composition des services Web. En conséquence, la tendance actuelle est d'exprimer la logique d'un service Web composite en utilisant un langage de modélisation des processus métiers adapté aux services Web. Un certain nombre de langages de ce type ont émergé tels que BPML (Business Process Modeling Language), BPEL (Business Process Execution Language) et WSCI (Web Service Choreography Interface). Parmi ces langages, BPEL a été désigné comme le langage standard pour la description de la composition des services Web sous forme de processus métier.

L'inconvénient majeur de ces langages de composition est qu'ils s'appuient sur des concepts purs de programmation et ont été conçus pour l'implémentation et l'exécution des services composites sans prendre en considération l'étape de la spécification. La spécification constitue une étape cruciale dans le cycle de vie de n'importe quelle application et est particulièrement importante parce que :

- Elle renforce la compréhension globale du système.
- Elle permet de se détacher de l'implémentation et d'obtenir des modèles abstraits plus clairs.
- Lorsque le modèle est suffisamment expressif et précis, il peut servir de base pour l'implémentation par génération automatique de code à partir du modèle de composition.

La technologie des services web soulève des problèmes de recherche très intéressants Parmi lesquels nous pouvons citer :

- La modélisation : comment spécifier la syntaxe et la sémantique d'un service sans ambiguïté ?
- la composition : comment assembler plusieurs services atomiques et composites pour un travail particulier ?
- la vérification : comment valider la composition des services web ?

Dans ce travail nous nous intéressons aux deux premiers axes de recherche à savoir la modélisation, et la composition des services web.

L'ingénierie dirigée par les modèles (IDM) est une approche de développement logicielle qui se concentre sur la réalisation de modèles abstraits plutôt que sur des concepts algorithmiques. Elle offre les moyens pour concevoir des modèles, effectuer des raisonnements sur ces modèles, vérifier leur bon fonctionnement, et générer le code à partir des modèles. Ces derniers sont une manière intuitive et naturelle permettant de représenter l'état et le comportement d'un système, quel que soit son degré de complexité et à chaque étape de son cycle de vie. La transformation de modèles et le cœur de l'IDM, Elle a apporté plusieurs améliorations significatives dans le développement des systèmes logiciels complexes en fournissant des moyens permettant de passer d'un niveau d'abstraction à un autre ou d'un espace de modélisation à un autre.

UML est considéré de nos jours comme un standard pour la modélisation orientée objet des systèmes complexes. Les diagrammes structurels permettent de modéliser l'aspect statique du système à concevoir comme les diagrammes de classes. UML peut être étendu et personnalisé par l'utilisation du mécanisme des profils ce qui nous encourager d'utiliser un profil UML-S (UML pour l'ingénierie des services) pour adapter les modèles UML à un domaine ou à notre problématique.

Ce mémoire est organisé Comme suit :

Chapitre1 : Le chapitre 1 a pour objectif d'éclaircir tous les concepts relatifs à la SOA. Dans la première partie nous présentons en premier lieu la notion de service et d'architecture orientée service en identifiant l'origine des termes avec leurs principales définitions. Dans la seconde partie, nous abordons une implémentation bien particulière de la SOA qui est l'architecture orientée service Web (WSOA). Nous y définissons le concept de services Web, la composition de services Web et nous y présentons brièvement les différents standards qui encadrent les services Web. Dans la troisième partie Nous abordons également le sujet de la composition des services web. Enfin, nous présentons un aperçu sur les différentes approches de modélisation et de composition des services web proposées dans la littérature.

Chapitre 2 : Dans ce chapitre, nous présentons les concepts de base de l'Ingénierie Dirigée par les Modèles (IDM) avec ses différentes variantes, nous nous intéressons en particulier à l'Architecture Dirigée par les Modèles (ADM) et la transformation de modèles.

Chapitre 3 : Ce chapitre est consacré à la présentation de langage de modélisation UML-S et ses concepts de base.

Chapitre 4 : Ce chapitre est consacré à la mise en œuvre de l'approche suivi. Nous proposons également une étude de cas où nous utilisons notre outil pour permettre de bien illustrer l'approche de modélisation de la composition des services web.

Chapitre 1

Les Services Web

1. Introduction

Au cours des quatre dernières décennies, les architectures logicielles ont tenté de faire face à des niveaux croissants de complexité logicielle. Mais le niveau de complexité continue d'augmenter, et les architectures traditionnelles (comme l'architecture client-serveur, l'architecture a plusieurs niveaux et les standards basées RPC comme le CORBA) semblent atteindre la limite de leur capacité à traiter le problème. Dans le même temps, les besoins traditionnels des organisations informatiques persistent, La nécessité de répondre rapidement aux nouvelles exigences de l'entreprise, la nécessité de réduire continuellement le coût de l'informatique pour l'entreprise, et la capacité d'absorber et d'intégrer de nouveaux partenaires commerciaux et de nouveaux ensembles de clients. Au cours des dernières années l'industrie de Logiciels a fourni plusieurs architectures informatiques conçues pour permettre un traitement entièrement distribué, des langages de programmation conçus pour s'exécuter sur n'importe quelle plate-forme, réduisant considérablement les calendriers d'implémentation et une myriade de produits de connectivité conçus pour permettre une meilleure et plus rapide intégration des applications. Cependant, la solution complète continue de s'échappe. Maintenant, l'architecture orientée services (SOA) est promue dans l'industrie comme la prochaine étape évolutive dans l'architecture logicielle pour aider les organisations de la technologie d'information (Information technology) à répondre à leurs défis toujours plus complexe. Vous constaterez qu'une SOA, du moins pour l'instant, est la meilleure base sur laquelle une organisation informatique peut prendre ses actifs existants à l'avenir ainsi que Construire ses nouveaux systèmes d'application**.**

2. L'architecture orienté service (SOA)

2.1. Définition

Bien qu'il n'existe pas de définition standard ou officielle pour la SOA, e. En effet plusieurs définitions ont été proposées, nous avons constaté que la définition d'IBM de SOA (Donnée ci-dessous) adéquate dans le contexte de cette section :

« L'architecture orientée services (SOA) est une approche architecturale informatique centrée sur l'entreprise qui prend en charge l'intégration de votre taches en tant que des taches liées, répétables ou des services ».

Il est important de noter que SOA n'est pas un produit mais un style architectural **[1]**.

M. Dodani dans son travail intituler 'From objects to services : A journey in search of component reuse nirvana' en 2004 à donner une autre définition d l'SAO :

"L'architecture orientée service est un paradigme qui permet l'intégration d'applications et de ressources de manière flexible en :

(1) représentant chaque application ou ressource sous la forme d'un service exposant une interface standardisée.

(2) Permettant à un service d'échanger des informations structurées (messages, documents, "objets métier").

(3) Coordonnant et organisant les services afin d'assurer qu'ils puissent être invoqués et utilisés de manière efficace."**[2]**

Il n'y a pas une définition exact de l'architecture orienté service, mais toutes les définitions sont d'accord que SOA est un paradigme destiné à résoudre les problèmes d'hétérogénéité et interopérabilité des logiciels qui constituent le système d'information.

nous pouvons affirmer d'une manière générale que l'architecture orientée services est un style architectural pour la conception, le développement, le déploiement et la gestion de systèmes logiciels distribués qui délivre des fonctionnalités d'application sous forme de services interopérables, soit à l'utilisateur final ou à d'autres services. Les systèmes qui sont construits en se basant sur les caractéristiques SOA sont appelés systèmes orientés services **[3]**.

2.2. Service

Le service est la brique de base de la SOA. Il représente une entité logicielle fonctionnelle déployée et invocable à distance. Il est difficile de trouver une définition exacte du terme « Service » parce que plusieurs définitions existent **[3]**.

Un service est défini comme une entité qui représente certaines fonctionnalités (application fonctionnelle, transaction commerciale, un service du système de base, etc.) exposée en tant que composant d'un processus métier. Un service est défini à l'aide d'une interface qui expose les fonctionnalités et masque les détails de mise en œuvre sous- jacents. Le service doit être sans état pour assurer qu'il ne dépend pas de l'état ou du contexte d'autres services. Les services sont appelés

via des protocoles de communication bien définis qui mettent l'accent sur l'interopérabilité et la transparence de localisation **[2]**.

Un service en SOA est une pièce exposée de fonctionnalité avec trois propriétés :

1) le contrat d'interface au service est indépendant de la plateforme.
2) le service peut être situé et invoqué dynamiquement.
3) le service est autonome. Autrement dit, le service gère son propre État.

2.3. Caractéristiques de l'architecture SOA

L'architecture orienté service architecture est caractérisée par :

- Un couplage faible entre les services : implique qu'un service n'appelle pas directement un autre service.
- La réutilisation de service.
- L'indépendance par rapport aux aspects technologiques : c-à-d les services avec cette architecture sont indépendants de ses plates-formes.
- La découverte des services disponibles.
- La mise à l'échelle est rendue possible grâce à la découverte et à l'invocation des nouveaux services lors de l'exécution **[4]**.

2.4. Avantages et inconvénients des SOAs

- Les Avantages :

- **Réutilisabilité de service :** En SOA, une application est générée par l'assemblage des petites fonctionnalités, autonomes et faiblement couplés .Par conséquent, les services peuvent être réutilisés dans plusieurs applications indépendantes de leurs interactions avec d'autres services.
- **La fiabilité :** Les applications basées sur SOA sont plus fiables car les petits services indépendants sont plus faciles à tester et à déboguer par rapport à de gros blocs de code.

- **Maintenabilité :** Comme un service est une entité indépendante, il peut être facilement mis à jour ou maintenu sans avoir à se préoccuper d'autres services. De grandes applications complexes peuvent ainsi être gérées facilement.
- **Localisation Indépendante :** Les services sont généralement publiés dans un répertoire où les consommateurs peuvent les consulter. Cette approche permet à un service de changer sa position à tout moment. Cependant, les consommateurs sont toujours en mesure de localiser leur service demandé par l'annuaire chercher.
- **l'évolutivité et la disponibilité :** Plusieurs instances d'un seul service peuvent s'exécuter simultanément sur différents serveurs. Cela augmente l'évolutivité et la disponibilité du service.
- **Amélioration de la qualité du logiciel :** Puisque les services peuvent être réutilisés, il n'y a pas de possibilité de fonctionnalités redondantes. Cela permet de réduire les erreurs dues à des données incohérentes et améliore ainsi la qualité du code.
- **Indépendance de la plate-forme :** SOA facilite le développement d'un produit complexe en intégrant différents produits de différents fournisseurs indépendamment de la plate-forme et de la technologie.
- **Augmentation de la productivité :** Les développeurs peuvent réutiliser les applications héritées existantes et créer des fonctionnalités supplémentaires sans avoir à développer la totalité de la chose à partir de zéro. Cela augmente la productivité des développeurs et, en même temps, réduit considérablement le coût de développement d'une application.

- Défis :

- **Défi de performance :** Chaque fois qu'un service interagit avec un autre service, une validation complète de chaque paramètre d'entrée a lieu. Cela augmente le temps de réponse et la charge de la machine, ce qui réduit les performances globales.
- **Gestion des services complexes :** Le service doit s'assurer que les messages ont été délivrés en temps favorable. Mais comme les services continuent d'échanger des messages pour effectuer des tâches, le nombre de ces messages peut aller en millions, même pour une seule application. Cela pose un grand défi à la gestion d'une énorme population de services.

- **Coût d'investissement élevé :** La mise en œuvre de la SOA nécessite un investissement important au moyen de la technologie (infrastructures technologique), le développement et les ressources humaines **[5]**.

2.5. l'SOA et les services web

Il faut ici faire la distinction entre services web et SOA qui sont deux concepts orthogonaux pourtant très complémentaires. Un dessin ou modèle SOA peut se construire sans les services web et services web implique pas par lui-même la conformité aux principes SOA. Mais les services web sont basés sur les standards préférables de se rendre compte de la SOA.

Donc les SOAs peuvent être implémentées via des services web, mais les services web ne sont pas nécessairement requis pour implémenter SOA.

3. Les services web

3.1. Définition

Depuis les dernières années, l'utilisation de Services web a connu une popularité grandissante. Ces services sont très utilisés notamment par les entreprises pour rendre accessible leurs métiers ou leurs données via le web. Les services web ont été proposés initialement par IBM et Microsoft, puis en partie standardisés par le W3C (le consortium du World Wide Web). Plusieurs définitions des services web ont été proposées dans la littérature :

D'un point de vue technique, un service Web est une entité logicielle offrant une ou plusieurs fonctionnalités allant des plus simples aux plus complexes. Ces entités sont publiées, découvertes et invoquées à travers le web grâce à l'utilisation d'internet comme infrastructure de communication ainsi que de formats de données standardisés comme XML **[6]**. Cette définition met en évidence uniquement le standard XML, donc n'importe quelle application orientée Internet qui garantit ces caractéristiques et qui utilise les technologies basées sur XML est aussi un Service web.

Selon IBM : « Les services web sont la nouvelle vague des applications web. Ce sont des applications modulaires, auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le web. Les services web effectuent des actions allant de simples requêtes à des processus métiers complexes. Une fois qu'un service web est déployé, d'autres applications (y

compris des services web) peuvent le découvrir et l'invoquer ». Les deux premières définitions affirment que les services web sont accessibles par d'autres à travers le web en utilisant des protocoles et des formats standards, mais elles ne mettent pas en évidence les technologies utilisées pour mettre en œuvre un service web **[7]**.

Selon le W3C : « Un service web est un système logiciel identifié par une URI et conçu pour supporter l'interaction interopérable de machine à machine sur un réseau. Il possède une interface décrite dans un format exploitable par la machine, c.à.d. décrite en WSDL (web Services Description Language). D'autres systèmes interagissent avec le service web d'une façon prescrite par sa description en utilisant des messages SOAP (Simple Object Access Protocol), typiquement en utilisant HTTP (HyperText Transfer Protocol) avec une sérialisation XML en même temps que d'autres normes du Web ». Cette définition met l'accent sur les standards de l'Internet et l'interface ouverte qui permet les invocations des services. Cependant, elle n'est pas encore suffisamment précise parce qu'elle ne mentionne pas la découverte de services web (UDDI) **[8]**.

Une définition plus raffinée des services web est fournie par le dictionnaire Webopedia3 qui définit un service web comme « une manière standardisée d'intégration des applications basées sur le Web en utilisant les standards ouverts XML, SOAP, WSDL, UDDI et les protocoles de transport de l'Internet. XML est utilisé pour représenter les données, SOAP pour transporter les données, WSDL pour décrire les services disponibles, et UDDI pour lister les fournisseurs de services et les services disponibles » **[9]**.

3.2. L'architecture des services web

Dans cette section, nous allons présenter l'architecture de base des services web tel que proposée par IBM. Cette architecture comporte trois entités : le fournisseur de service, L'annuaire de services et le client ou utilisateur du service :

- **Le Fournisseur du service :** qui publie son service en fournissant la description au format WSDL dans l'annuaire de services et réalise les opérations propres au service. C'est donc le propriétaire du service web. Il représente l'environnement d'hébergement et d'exécution du service. Il est constitué de trois couches de bases :

✓ La couche de données : contient les différentes bases de données utilisées par le service.

- ✓ La couche applicative : c'est la plateforme de développement qui assure l'exécution du service web.
- ✓ La couche de description : elle expose les fonctionnalités du service via un fichier WSDL.

- **L'Annuaire :** qui, d'une part reçoit et enregistre les descriptions de services publiées par les fournisseurs, et d'autre part reçoit et répond aux recherches de services lancées par les clients. C'est donc un registre de description qui offre aux fournisseurs le moyen de publier et d'indexer leurs services web sur le réseau. Il permet également aux clients de rechercher ces services selon plusieurs critères.
- **Le Client :** qui obtient la description du service grâce à l'annuaire et utilise le service.

Dans l'architecture des services web, le Client est défini comme le consommateur du service. Il peut accéder à ce dernier en échangeant avec le fournisseur des messages SOAP. Cet échange de messages se fait généralement à l'aide des protocoles Internet standards tels que HTTP, SMTP, etc. Techniquement, le Client peut être une simple application Windows ou web, comme il peut être un autre service web.

Par conséquent, un service web est toujours accompagné d'une description fournissant aux applications les informations nécessaires à son utilisation. Les services sont implantés par les fournisseurs qui mettent à disposition les descriptions de services sous forme de fichiers WSDL. Ces descriptions sont centralisées et stockées dans des annuaires. La notion d'annuaire est comparable aux annuaires téléphoniques que nous utilisons pour accéder à des personnes ou des services. Les applications clientes envoient des requêtes aux annuaires pour sélectionner les services, de la même manière que nous recherchons un numéro dans un annuaire téléphonique. Elles téléchargent ensuite les descriptions des services sélectionnés, et les invoquent directement **[10]**.

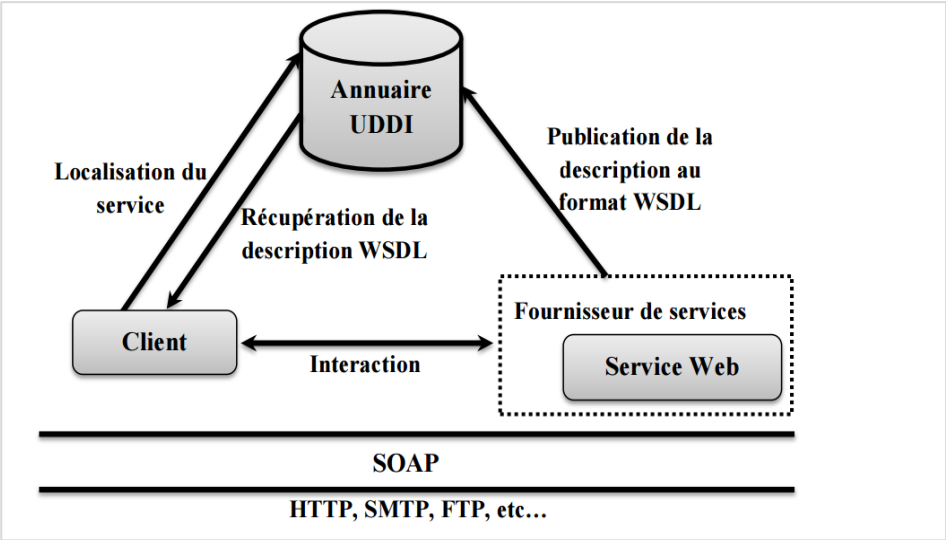

Figure 1.1 Architecture de Base de Service web [10]

3.3. Langages et protocoles utilisés par les services web

L'architecture fonctionnelle des services web que nous avons présenté précédemment montre l'utilisation de nombreux langages et protocoles durant le déploiement et l'invocation des services web. L'atout principal des protocoles SOAP et UDDI ainsi que du langage WSDL est de se reposer sur le langage XML (eXtensible Markup Language). Cette même architecture montre aussi que les services web se basent sur l'utilisation des normes actuelles d'Internet comme le protocole HTTP.

3.3.1. Le langage XML (eXtensible Markup Language)

XML est un format universel d'échanges de données, très souple, dérivé de SGML (ISO 8879). Il joue également un rôle de plus en plus important dans l'échange d'un large éventail de données sur le Web et ailleurs.

XML constitue la technologie de base des architectures Web services ; c'est un facteur important pour contourner les barrières techniques. XML est un standard qui permet de décrire des documents structurés transportables sur les protocoles d'Internet. En effet, il apporte à l'architecture des Web services l'extensibilité et la neutralité vis à vis des plates-formes et des langages de développement **[11].**

La technologie des services web a été conçue pour fonctionner dans des environnements totalement hétérogènes. Cependant, l'interopérabilité entre les systèmes hétérogènes demande des

mécanismes puissants de correspondance et de gestion des types de données des messages entre les différents participants (clients et fournisseurs). C'est une tâche où les schémas de type de données XML s'avèrent bien adaptés. C'est pour cette raison que la technologie des services web est essentiellement basée sur XML ainsi que les différentes spécifications qui tournent autour (les espaces de nom, les schémas XML, et les schémas de Type) **[12]**.

Parmi les spécifications XML, nous soulignons :

- ✓ XSD (XML Schema) : c'est un langage qui sert à décrire formellement un vocabulaire **[13]**.
- ✓ XSLT (Extensible Stylesheet Language Transformations) : est utilisé pour transformer un document XML basé sur un certain schéma en un autre document XML qui peut être un document lui-même basé sur un autre schéma **[14]**.
- ✓ XPath (XML Path Language) : fournit une syntaxe d'expressions utilisées pour créer des chemins de localisation **[15]**.

3.3.2. Le SOAP (Simple Object Access Protocol)

SOAP est un protocole pour l'échange d'informations structurées avec les services Web, recommandé par le W3C. SOAP est le successeur de XML-RPC]. Il est basé sur XML pour le format des messages et il s'appuie généralement sur des protocoles de la couche application pour le transport des messages (RPC, HTTP). SOAP forme la couche inférieure de la pile des protocoles des services Web, fournissant un Framework pour l'échange de messages sur lequel les services Web peuvent se baser. Un message SOAP est structuré en trois parties : un en-tête (facultative) et un corps à l'intérieur d'une enveloppe (voir figure 1.2).

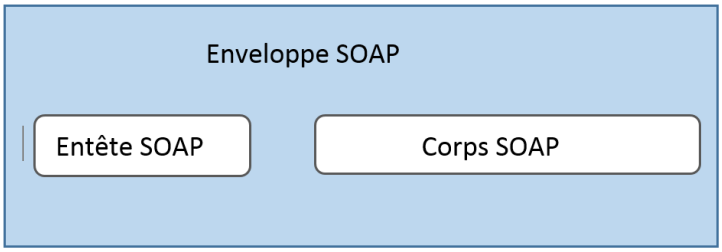

Figure 1.2 Structure d'un message SOAP

SOAP présente l'avantage d'être suffisamment polyvalent pour utiliser différents protocoles de transport. En effet, bien que HTTP soit le protocole par défaut, il est tout à fait possible d'utiliser

HTTPS (identique à HTTP mais avec chiffrement) ou SMTP (protocole de messagerie). En revanche, SOAP peut être considérablement plus lent que ses concurrents (ex : CORBA) du fait de la verbosité de XML **[16]**.

3.3.3. WSDL (Web Service Description Language)

Le langage WSDL (Web Service Description Language) proposé par Ariba, IBM et Microsoft auprès du W3C est un format de description des Web services fondé sur XML. Il permet de décrire de façon précise les Web services en incluant des détails tels que les protocoles, les serveurs, les ports utilisés, les opérations pouvant être effectuées, les formats des messages d'entrée et de sortie ainsi que les exceptions pouvant être renvoyées. Il permet la représentation d'un Web Service de manière plus abstraite pour la réutilisation.

Il est devenu une recommandation du W3C depuis le 26 Juin 2007. Un document WSDL contient des informations opérationnelles concernant le service. La définition du service marquée par la balise est la racine de tout document WSDL. Elle peut contenir les attributs précisant le nom du service et les espaces de nommage. Un document WSDL contient les entités suivantes :

- **Types :** précise les types de données complexes, pour lequel WSDL emplois XML Schéma.
- **Message :** l'abstraction décrivant les données échangées entre Web services.
- **Opération** : l'abstraction décrivant une action implémentée par un Web service.
- **Type de port :** un ensemble d'opérations implémenté par une terminaison donnée.
- **Liaison (binding) :** un protocole concret d'accès à un port et un format de spécification des messages et des données pour ce port.
- **Port :** un point de terminaison identifié de manière unique par la combinaison d'une adresse Internet et d'une liaison.
- **Service :** une collection de ports. Il est important de mentionner que le document WSDL est divisé en deux parties : l'interface du service et son implémentation. L'interface du service est la partie réutilisable de la définition du service, elle peut être référencée par de multiples implémentations du service. Elle est composée des balises : Types, Message, Opération et Liaison. Alors que la partie implémentation, décrite par les balises Port et Service, est unique

et présente une terminaison pour invoquer le Web service. De plus, chaque document WSDL peut être documenté grâce à une balise bien que cet élément soit facultatif **[17]**.

La figure 1.3 offre une représentation concise de la spécification WSDL.

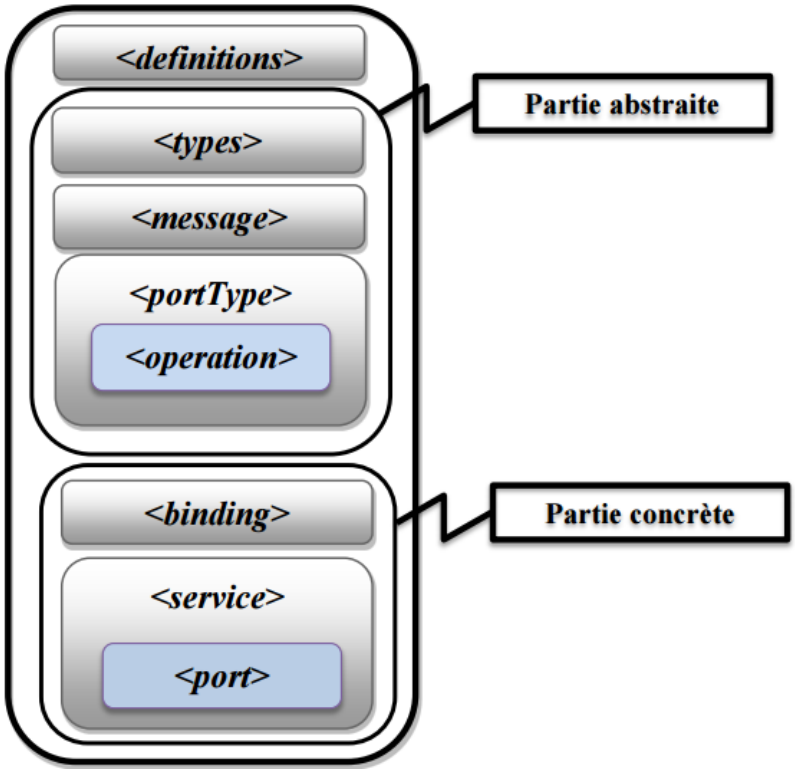

Figure 1.3 Structure d'une description WSDL 1.1 [10]

Pour plus d'illustration, la figure 1.4 représente la description WSDL du service simple **« HelloService »**. Ce dernier offre une fonction (opération) appelée **« sayHello »**. La fonction reçoit un message **« SayHelloRequest »** avec une chaine de caractères représentant le paramètre d'entrée, et retourne un message **« SayHelloResponse »** qui contient une autre chaine de caractères. Par exemple, si vous passez le mot «world» comme paramètre, le service renvoie le message d'accueil, **« Hello, world ! »**.

```
<? Xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
 targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <message name="SayHelloRequest">
 <part name="firstName" type="xsd:string"/>
 </message>
        <message name="SayHelloResponse">
        <part name="greeting" type="xsd:string"/>
 </message>
 <portType name="Hello_PortType">
        <operation name="sayHello">
                <input message="tns:SayHelloRequest"/>
                <output message="tns:SayHelloResponse"/>
        </operation>
 </portType>
 <binding name="Hello_Binding" type="tns:Hello_PortType">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="sayHello">
          <soap:operation soapAction="sayHello"/>
          <input>
                <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="urn:examples:helloservice"
                use="encoded"/>
        </input>
        <output>
                <soap:body
```

```
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="urn:examples:helloservice"
                use="encoded"/>
        </output>
      </operation>
    </binding>
    <service name="Hello_Service">
        <documentation>WSDL File for HelloService</documentation>
        <port binding="tns:Hello_Binding" name="Hello_Port">
            <soap:address
            location="http://localhost:8080/soap/servlet/rpcrouter"/>
        </port>
     </service>
    </definitions>
```

Figure 1.4 Exemple d'un document wsdl [10]

La version 1.0 de l'interface WSDL a été créée en 2000 par IBM, Microsoft et Ariba, ensuite le consortium W3C a publié la version WSDL 1.1 en mars 2001 comme une note. En juin 2007, la version WSDL 2.0 a été créée comme une recommandation W3C (C.à.d. norme). WSDL 2.0 a apporté quelques changements à la version WSDL 1.1, ils sont résumés comme suit :

- <**portType**> est devenu <**interface**>**.**
- <**port**> est remplacé par <**endpoints**>.
- <**message**> est supprimé et l'élément <**xsd : element**> est utilisé pour spécifier les données échangées.

3.3.4. UDDI

Est un élément du groupe de normes qui constituent la pile de services Web. La spécification UDDI définit une méthode standard pour la publication et la découverte des composants logiciels basés sur le réseau d'une architecture orientée services (SOA). Son développement est mené par le consortium OASIS de fournisseurs de logiciels d'entreprise et de clients **[18]**.

Le but fonctionnel d'un registre UDDI est la représentation de données et de métadonnées sur les services Web. Un registre, qu'il soit utilisé sur un réseau public ou au sein de l'infrastructure interne d'une organisation, offre un mécanisme normalisé pour classer, cataloguer et gérer les services Web, afin qu'ils puissent être découverts et consommés par d'autres applications. Dans le cadre d'une stratégie généralisée d'indirection entre les applications basées sur les services, l'UDDI offre plusieurs avantages aux responsables informatiques à la fois en temps de conception et en temps d'exécution, y compris l'augmentation de la réutilisation du code et l'amélioration de la gestion de l'infrastructure **[18]** :

- ✓ Publier des informations sur les services Web et les règles de catégorisation spécifiques à une organisation.
- ✓ Trouver des services Web (au sein d'une organisation ou à travers des limites organisationnelles) qui répondent à des critères donnés.
- ✓ Déterminer les protocoles de sécurité et de transport pris en charge par un service Web donné et les paramètres nécessaires pour invoquer le service.
- ✓ Offrir un moyen d'isoler les applications (et de fournir un basculement et un routage intelligent) contre les défaillances ou les changements dans les services invoqués.

Les données stockées dans l'annuaire sont structurées en trois catégories. **Les pages blanches** comprennent la liste des entreprises ainsi que des informations associées à ces dernières. Nous y retrouvons donc des informations comme le nom de l'entreprise, ses coordonnées, la description de l'entreprise mais également l'ensemble de ses identifiants. **Les pages jaunes** recensent les services Web de chacune des entreprises sous le standard WSDL. **Les pages vertes** fournissent des informations techniques précises sur les services fournis. Typiquement on indiquera dans ces informations les adresses Web des services et les moyens d'y accéder.

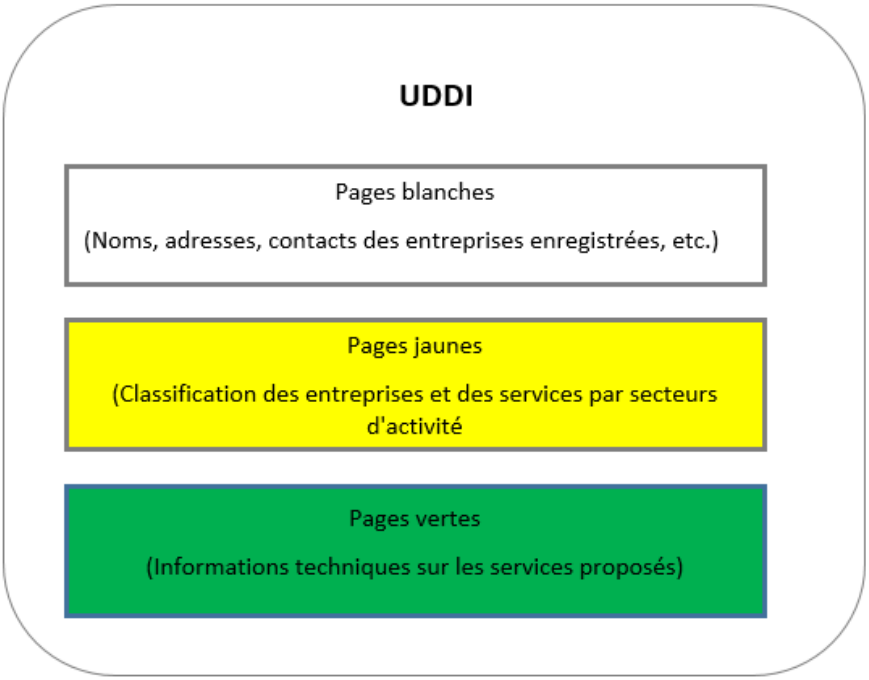

Figure 1.5 Le contenu d'un annuaire UDDI

Le protocole d'utilisation de l'UDDI offre trois fonctions de base :

- **Publish** : permet de publier un nouveau service.
- **find** : permet d'interroger l'annuaire UDDI.
- **bind** : permet d'établir une connexion entre l'application cliente et le service.

4. La composition des services web

Depuis les dernières années, l'utilisation de services web a connu une popularité grandissante. Ces services sont très utilisés notamment par les entreprises pour rendre accessible leurs métiers ou leurs données via le Web. Les services web, tels qu'ils sont présentés, sont conceptuellement limités à des fonctionnalités relativement simples qui sont modélisées par une collection d'opérations **[19]**. Cependant, il est nécessaire de construire de nouvelles applications par composition de services afin de répondre à des exigences plus complexes.

La composition de services web est considérée comme l'une des motivations les plus

importantes de cette technologie. Elle détient un potentiel énorme dans la réorganisation et l'intégration des applications d'entreprise. Malheureusement, les technologies actuelles basées sur WSDL, UDDI et SOAP offrent des solutions pour la description, la publication, la découverte et l'interopérabilité des services web, mais ne permettent pas d'effectuer leur composition qui reste une tache très complexe.

Dans cette section, nous présentons la composition des services web et ses concepts adjacents.

4.1. Définition

Une composition de services Web est constituée de plusieurs services qui interagissent les uns avec les autres, afin d'offrir de nouvelles fonctionnalités qu'un seul service ne pourrait pas les offrir. La composition permet de combiner des services pour former un nouveau service dit composé ou composite. L'exécution d'un service composé implique des interactions avec des services partenaires en faisant appel à leurs fonctionnalités. Le but de la composition est avant tout la réutilisation de services (simples ou composés) et de préférence sans aucune modification de ces derniers **[4]**.

4.2. Approche de réalisation d'une composition

Il existe des différentes approches pour la réalisation d'une composition de services Web. Afin de choisir parmi ces approches, le concepteur de systèmes à base de services Web doit prendre en compte différents paramètres. Dans ce qui suit, nous allons présenter deux classifications de ces différentes approches qui se placent selon deux points de vue différents. La première classification distingue entre une composition de type **orchestration** et une composition de type **chorégraphie**. La deuxième classification distingue quant à elle entre une composition **statique** et une composition **dynamique**.

4.2.1. L'orchestration

L'orchestration des 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 **[16].**

L'orchestration de service représente un processus d'entreprise exécutable centralisé unique (orchestrateur) qui coordonne l'interaction entre les différents services. L'orchestrateur est

responsable de l'appel et de la combinaison des services. La relation entre tous les services participants est décrite par un seul point d'extrémité (c'est-à-dire le service composé). L'orchestration comprend la gestion des transactions entre services individuels.

L'orchestration utilise une approche centralisée pour la composition du service. Et Parmi les langages d'orchestration, WS-BPEL 2.0 (ou BPEL) semble avoir gagné le consensus. BPEL est un langage exécutable standardisé par l'OASIS et basé sur XML.

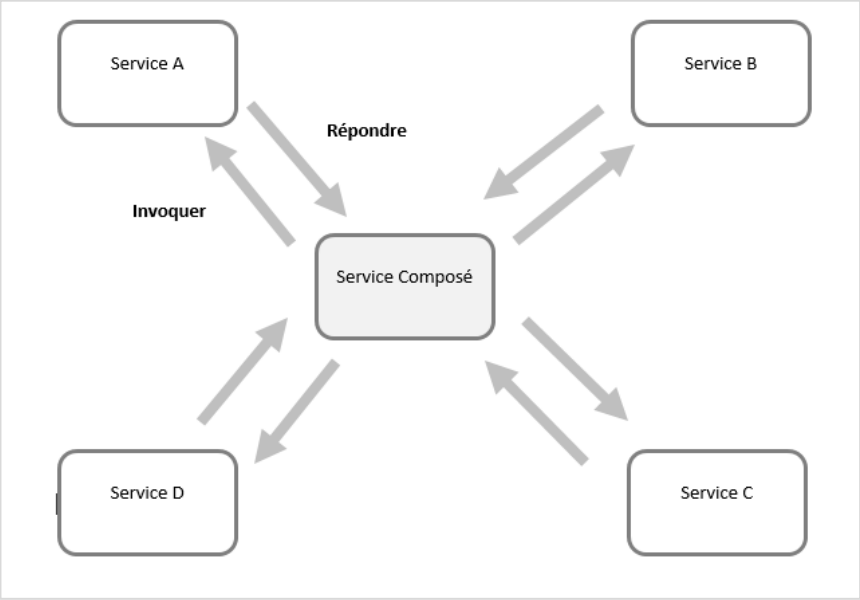

Figure 1.6 L'orchestration

En résumé, l'orchestration :

- Définit un seul maître de contrôle pour tous les aspects d'une composition.
- Mappage facile à l'SOA.
- Est généralement plus simple pour commencer, Mais souvent plus difficile à étendre à des processus plus complexes.
- Représente l'état de la pratique et appliqué par la majorité des outils **[20]**.

4.2.2. La chorégraphie

La chorégraphie de services est une généralisation de l'orchestration qui consiste à concevoir une coordination décentralisée des applications. 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 **[16]**.

La chorégraphie est une description globale des services participants, définie par l'échange de messages, les règles d'interaction et les accords entre deux ou plusieurs points finaux. La chorégraphie utilise une approche décentralisée pour la composition du service.

La chorégraphie diffère d'une orchestration par rapport où la logique qui contrôle les interactions entre les services.

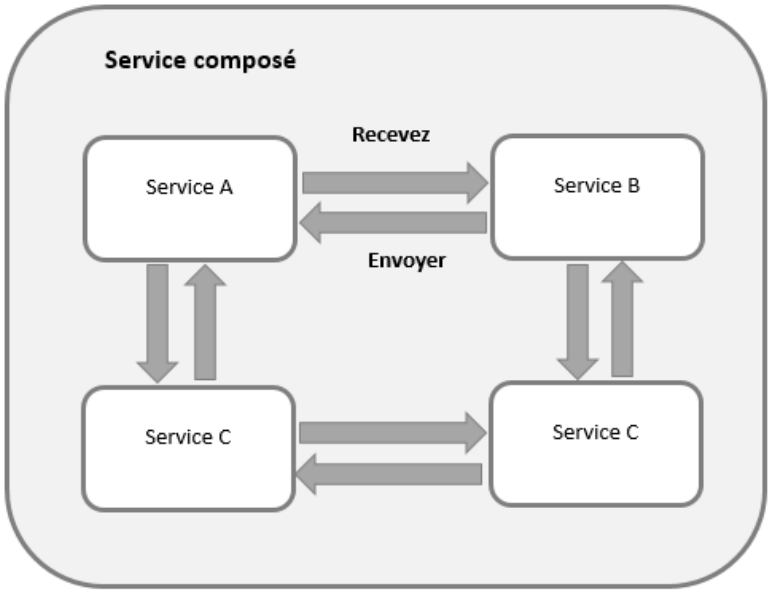

Figure 1.7 Chorégraphie

4.2.3. Composition statique et composition dynamique

Afin de répondre à une requête complexe du client, les services Web disponibles peuvent être combinés pour créer un nouveau service Web composite. Cette combinaison peut être réalisée de façon statique ou dynamique.

Dans l'approche statique, l'agrégation des services Web est réalisée à la phase de conception du service composite. Les services Web sont déterminés, agrégés et reliés entre eux, pour être par la suite déployés. Ce type de composition est plus approprié aux environnements stables où les services préalablement identifiés par l'utilisateur ne sont pas susceptibles de changer à court terme. Néanmoins, la modification d'un service composant ou sa substitution par un autre suscite également des modifications au niveau de la conception du service composite.

Contrairement à la composition statique où le nombre de services fournis est limité et les services à composer sont spécifiés au préalable, la composition dynamique, elle, est initiée par une requête de l'utilisateur. Elle permet de découvrir, sélectionner et combiner dynamiquement les services à partir des annuaires de services Web, au moment de l'exécution de ladite requête, tout en tenant compte des contraintes de l'utilisateur. Plusieurs approches ont adopté ce type de composition.

L'approche dynamique répond à des besoins de flexibilité et d'adaptation. Elle permet en effet de générer des plans de compositions adaptés à chaque requête à partir des services disponibles au moment de la composition. L'approche dynamique présente plusieurs avantages par rapport à l'approche statique. En outre le fait qu'elle est plus flexible et tient compte des changements qui peuvent survenir aux moments de l'exécution, elle demeure moins restrictive puisqu'elle permet de s'adapter à de nouvelles exigences de services. La composition statique, par contre, considère que les fonctions du service ne changent pas et que les services sont toujours disponibles. Toutefois, la réalisation de la composition dynamique dans un environnement où le nombre de services fournis est en constante évolution n'est pas une simple tâche **[21]**.

4.3. Langages utilisés pour la composition des services web

4.3.1. Le langage XLANG

(Web Services for Business Process Design) Créé par Microsoft, le XLANG est une extension de WSDL. Elle fournit en même temps un modèle pour une orchestration des services et des contrats de collaboration entre celles-ci. XLANG a été conçus avec une base explicite de théorie de calcul **[22].**

4.3.2. WSFL

(Web Service Flow Language) est un langage basé sur XML pour la description des services web composite. WSFL considère deux types de compositions de services web :

1) Le premier type indique le modèle approprié d'utilisation d'une collection de services web, de telle manière que la composition résultante décrive comment réaliser le but particulier d'un business, typiquement, le résultat est une description d'un processus métier.
2) Le deuxième type indique le modèle d'interaction d'une collection de services web. Dans ce cas, le résultat est une description de l'interaction globale associée **[23]**.

4.3.3. Langage WSCI

WSCI : (Services Web Chorégraphie Interface) est un langage de description basé sur XML dont l'objectif est de décrire les messages échangés entre le service web participant à des interactions de chorégraphie et les autres services de cette chorégraphie **[24]**.

4.3.4. Le langage WSCL

WSCL : (Web Services Conversation Language) permet de définir les conversations au niveau des entreprises ou les processus publics pris en charge par un service Web. WSCL spécifie les documents XML échangés et le séquencement autorisé de ces échanges de documents. Les définitions de conversation WSCL sont elles-mêmes des documents XML et peuvent donc être interprétées par des infrastructures de services Web et des outils de développement **[25]**.

4.3.5. Le langage BPEL

Plusieurs langages de compositions de services ont été proposés dans la littérature, le langage BPEL (Business Process Exécution Langage) est l'un des plus importants formalismes. Il est largement utilisé dans le cadre de la mise en œuvre d'une architecture orientée services.

BPEL (connu aussi sous le nom BPEL4WS ou WSBPEL) représente la convergence de deux langages, WSFL (Web Services Flow Languages) et de XLANG. Il combine les deux approches et fournit un vocabulaire riche, basé sur XML, pour la description des processus métiers, BPEL permet de modéliser les processus métiers en termes d'orchestration.

Langage BPEL se caractérise par rapport aux autres langages par sa gestion des exceptions (fautes et événements), l'exécution parallèle des activités et la synchronisation des flots, son mécanisme de compensation, et sa gestion de la corrélation des messages.

Notons qu'un processus BPEL est directement exécutable par un moteur d'orchestration de BPEL (ex. activeBPEL, Oracle BPEL Process Manager,…). BPEL s'appuie sur le langage WSDL puisque chaque processus BPEL est exposé comme un service Web via une interface WSDL. BPEL utilise les opérations, les données ainsi que les liens des partenaires décrits dans son interface WSDL. Ce dernier décrit aussi tous les éléments nécessaires à un processus BPEL pour interagir avec ses partenaires, à savoir, l'adresse des partenaires, les protocoles de communication et les opérations disponibles.

En outre, BPEL fait appel aux standards WSDL, SOAP, UDDI, mais aussi aux autres standards de services Web (Figure 1.8) tels que :

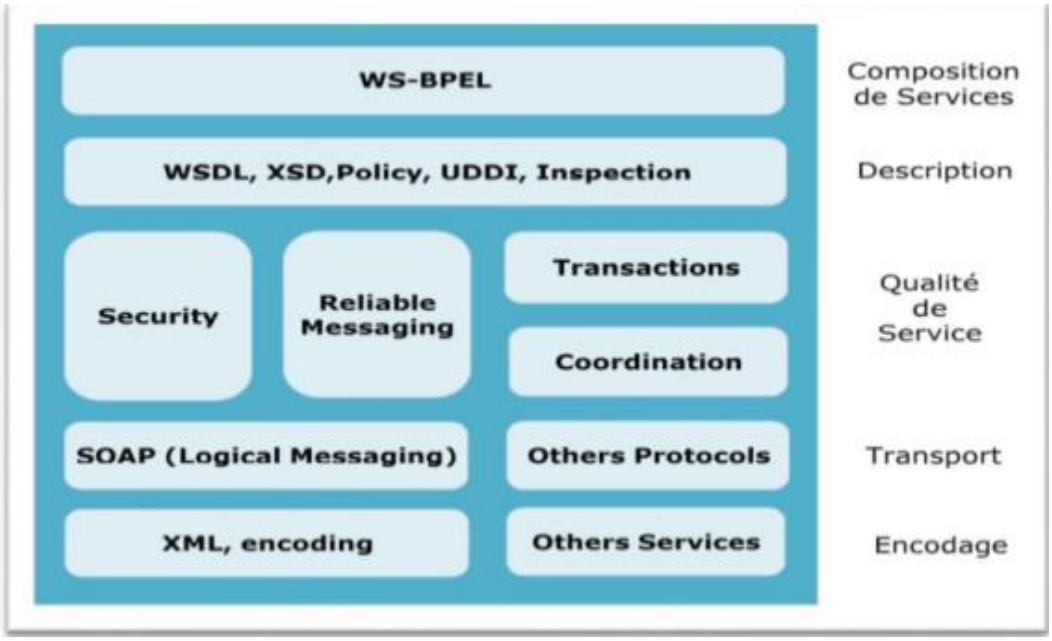

Figure 1.8 BPEL dans l'architecture des services Web [4]

WS-Security : qui est une extension de SOAP permettant de sécuriser les échanges de messages

WS-Reliable Messaging : qui est un protocole permettant aux messages d'être délivrés de manière fiable entre différentes applications réparties, dans le cas de pannes de composants, de réseaux ou de systèmes **[26].**

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.
- ✓ Sa spécification a été faite par plusieurs acteurs industriels majeurs, 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.
- ✓ Le processus permettant le parallélisme présente également des liens de synchronisations.

5. Approches de modélisation et de composition des services web (Approches dirigées par les modèles)

Comme présenté précédemment, plusieurs langages de composition de services web ont été proposés au cours de ces dernières années. Il s'agit malheureusement de langages textuels exécutables conçus pour satisfaire à la phase d'implémentation d'un service web composé et qui négligent de fait l'étape de spécification qui est importante, car elle facilite la compréhension globale du système ainsi que la tâche de développement. De plus, l'analyse formelle des langages proposés n'est pas possible à cause de leur manque de formalisme.

Donc, il est indispensable d'utiliser des techniques et des modèles permettant la spécification formelle de la composition de services web, dans le but de s'assurer du bon fonctionnement des services avant de les implémenter. Différents formalismes, pour la modélisation et la spécification de la composition de services, ont été proposés. Nous pouvons citer : les diagrammes d'activité UML (diagrammes d'activité et diagramme de classe), les réseaux de Petri et les automates **[4].**

5.1. Les approches Semi-Formelles

5.1.1. UML

UML (en anglais Unified Modeling Language ou « langage de modélisation unifié ») est un langage de modélisation graphique à base de pictogrammes. Il est apparu dans le monde du génie logiciel, dans le cadre de la « conception orientée objet ». Couramment utilisé dans les projets logiciels, il peut être appliqué à toutes sortes de systèmes ne se limitant pas au domaine informatique **[27]**.

UML, en particulier le diagramme d'activités et le diagramme de classe, ont été adapté pour la modélisation de la composition de services. Le diagramme de classes UML permettant de décrire la structure statique de la composition. Il modélise les différentes parties du système sous forme de classes et leurs relations sous forme d'héritage, d'associations ou d'agrégation. Dans le digramme d'activité L'exécution d'un service est généralement représentée par une action, c'est à dire une étape de l'activité. La transition vers cette activité modélise alors l'appel au service et inversement celle en sortie indique la fin de l'exécution du service.

5.2. Les approches Formelles

5.2.1. Réseaux de Petri

Les réseaux de Petri constituent une approche bien établie pour la modélisation de processus. Un réseau de Petri est un graphe dirigé, connecté et biparti, c'est à dire composé de deux types de nœuds. Les nœuds représentent soit des places, soit des transitions. Les places peuvent éventuellement contenir des jetons, qui correspondent généralement aux ressources disponibles **[28].**

Les RdPs constituent un Framework pour la modélisation de la composition de services. La facilité de la compréhension, l'utilisation de ses notations graphiques, ainsi que sa puissance de vérification et d'analyse formelles à travers un ensemble de bases et techniques mathématiques, justifient le choix de l'utilisation de cet outil pour des problématiques de modélisation et de vérification de la composition de services **[29].**

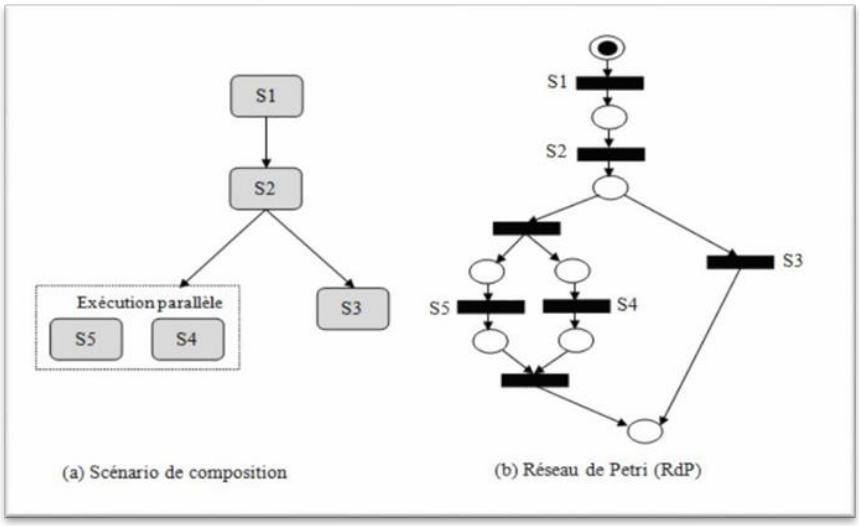

Figure 1.9 Exemple de modélisation par réseau de Pétri d'un service composé [4]

Dans la famille des approches formelle Il existe d'autres alternatives de modélisation, on cite les suivants :

- ❖ L'algèbre de processus
- ❖ Language Z
- ❖ LOTOS

6. Discussion

L'architecture orienté service présente actuellement au cœur des entreprises dans le cadre d'une collaboration inter-entreprises de la architecture SAO pour rendre accessible par le réseau leurs données et leur savoir-faire. L'élaboration de tell architecture et ces implémentations appuyez sur des technologies clés. La technologie primordiale et la plus commune de réalisation de SOA c'est la WSOA (Web Service Oriented Architecture). Cette technologie émergente facilite également les collaborations inter-entreprises. Ce processus d'intégration et de combinaison de plusieurs services Web est couramment appelé la composition de services.

Plusieurs langages textuels ont été proposés Pour la mise en œuvre de la description et l'exécution de tell collaboration (composition) .Ces langages sont conçus pour satisfaire à la phase d'implémentation d'un service Web composé mais ils ne prennent pas en compte l'étape de spécification qu'elle est très importante.

Pour se détacher de l'implémentation l'étape de spécification nous permet de réaliser des modèles abstraits plus clairs et modélise précisément le comportement du système, aidant à la compréhension globale du système et s'assurer bien aux clients que la réalisation sera conforme à leurs exigences. Afin de répondre à ce besoin, d'autres travaux avec une toute autre approche ont vu le jour. Ces travaux suivent les principes de l'ingénierie dirigée par les modèles (IDM) pour obtenir le service Web composé.

L'ingénierie dirigée par les modèles (IDM) est une approche de développement, très populaire dans l'industrie logicielle, qui se concentre sur la réalisation de modèles abstraits plutôt que sur des concepts de programmation ou algorithmiques. La phase de spécification est donc particulièrement importante dans une approche IDM. Les principes d'IDM sont tout à fait applicables au développement de services Web composés puisqu'il s'agit de composants logiciels.

7. Conclusion

Dans ce chapitre, nous avons présenté d'une manière générale tous les aspects fondamentaux de l'architecture orientée services et nous avons discuté la nécessité d'une telle technologie et est des divers concepts qui ont contribué à son évolution. La présentation de l SOA menée a mise en évidence deux aspects principaux : les technologies de réalisation de l'SOA et les implémentations de l'SOA. Nous avons également décrit l'architecture fondamentale des services web ainsi que les langages et les protocoles de base permettant leur déploiement. Ensuit nous avons abordé le sujet de la composition de services web et les langages dédiés à l'implémentation d'un tel concept. Après, nous avons présenté un survol sur les différentes approches de modélisation et de composition des services web existantes dans la littérature. Enfin, nous avons discuté l'orientation vers l'IDM et l'adoption de ses principes au cœur de processus de composition.

Étant donné que dans notre travail, nous allons appliquer ces principes, le chapitre suivant sera consacré à présenter les fondements de l'IDM.

Chapitre 2

Ingénierie Dirigée par les Modèles

1. Introduction

L'ingénierie dirigée par les modèles (IDM, ou MDE : Model Driven Engineering) est un sujet de recherche important dans le domaine du génie logiciel. L'IDM c'est une approche de développement mettant à disposition un ensemble d'outils et de langages pour la manipulation des modèles. Ces derniers sont une manière intuitive et naturelle permettant de représenter l'état et le comportement d'un système, quel que soit son degré de complexité et à chaque étape de son cycle de vie. Les techniques d'ingénierie dirigée par les modèles appliquent une transition logique entre des modèles sémantiquement riches mais simplifiés et d'autres modèles incluant plus de détails techniques et de conception. Cette approche aurait été appliquée dans les différents domaines de génie logiciel. Elle a apporté plusieurs améliorations significatives dans le développement des systèmes logiciels complexes en fournissant des moyens permettant de passer d'un niveau d'abstraction à un autre ou d'un espace de modélisation à un autre.

Cependant, la gestion des modèles peut s'avérer lourde et coûteuse. Pour pouvoir mieux répondre aux attentes des utilisateurs, il est nécessaire de fournir des outils flexibles et fiables pour la gestion automatique des modèles ainsi que des langages dédiés pour leurs transformations.

Dans ce chapitre, nous proposons un tour d'horizon sur les aspects fondamentaux de l'IDM en mettant l'accent sur la transformation de modèles qui constitue le thème central de cette discipline.

2. Concepts fondamentaux de l'ingénierie dirigée par les modèles

2.1. Système

Un système est défini comme un ensemble organisé d'entités collaborant de manière unitaire, et en interaction permanente, pour assurer une ou plusieurs fonctions du système dans son environnement. Dans les systèmes, on s'intéresse surtout aux logiciels. Ces derniers doivent respecter certains critères de qualité tels que :

- ✓ **La convivialité** : Elle décrit la facilité d'apprentissage, d'utilisation, de préparation des données, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation.
- ✓ **La validité :** Un logiciel doit assurer la conformité de ses fonctionnalités avec celles décrites dans le cahier des charges.

- ✓ **La fiabilité :** Un logiciel doit être capable de gérer correctement ses propres erreurs de fonctionnement en cours d'exécution.
- ✓ **La réutilisabilité :** Un logiciel doit être apte à être réutilisé, partiellement ou dans son intégralité, pour le développement d'autres logiciels.
- ✓ **L'intégrité :** Un logiciel doit protéger son code et ses données et doit pouvoir gérer les autorisations d'accès.
- ✓ **La transparence :** Un logiciel doit avoir la faculté de masquer à l'utilisateur les détails inutiles à l'utilisation de ses fonctionnalités.
- ✓ **La portabilité :** Exprime la possibilité de compiler le code source et/ou d'exécuter le logiciel sur des plateformes différentes.
- ✓ **L'extensibilité :** Un logiciel doit se laisser maintenir, c'est-à-dire supporter des éventuelles modifications de ses fonctionnalités ou une extension vers des fonctions qui lui seront demandées sans compromettre son intégrité et sa fiabilité.
- ✓ **La vérifiabilité, L'autonomie, etc.**

2.2. Architecture

L'architecture désigne la structure générale inhérente à un système informatique. Elle permet la spécification des différentes parties du système ainsi que leurs relations (les règles d'interaction entre ces parties). La structure d'un système peut être représentée sous forme graphiques à l'aide des organigrammes tels que : les diagrammes entité-relation, les diagrammes UML, etc. Le diagramme peut concerner un logiciel, un réseau informatique, un sous-système **[10]**.

2.3. Plateforme

Une plate-forme en général est une base de travail à partir de laquelle on peut écrire, lire, développer et utiliser un ensemble de logiciels. Les plateformes peuvent supporter plusieurs technologies (tels que : J2EE, CORBA, etc.) et sont implémentées par plusieurs fournisseurs (tels que : WebLogic, WebSphere, etc.) **[10]**.

2.4. Une méthode orientée modèles

La séparation du modèle de l'architecture est l'un des premiers objectifs de l'IDM qui place le modèle au centre du développement] et assure que le développement d'un système se réalise par un processus fait de plusieurs transformations successives d'un modèle de base, jusqu'à obtention du code final. La figure 2.1 montre les grandes phases de la réalisation d'un système **[30]**.

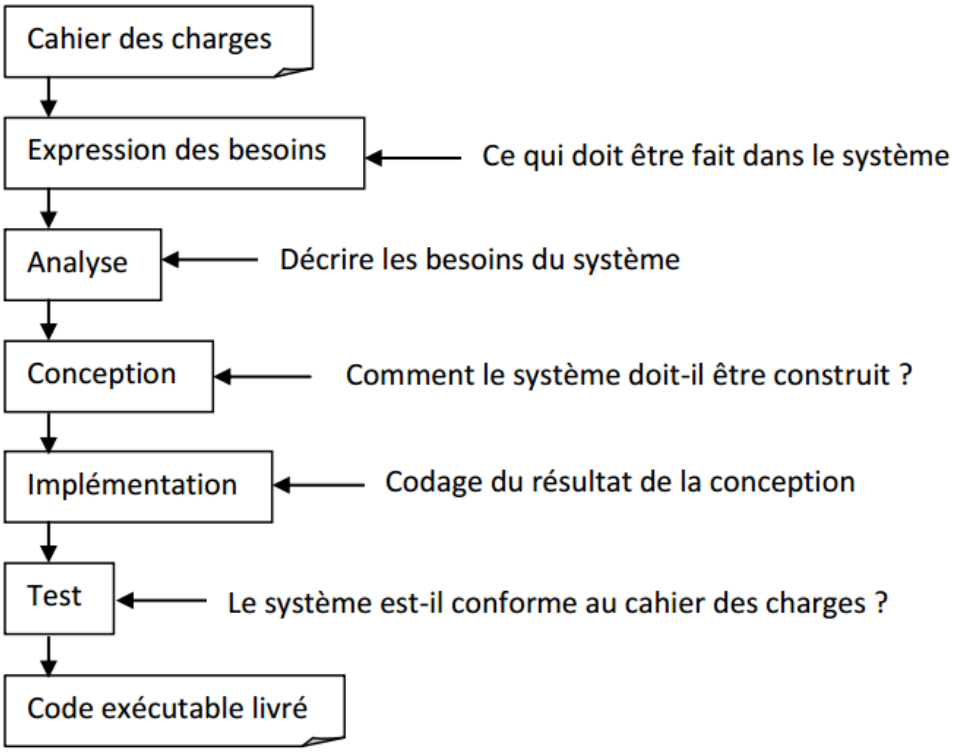

Figure 2.1 Phase de réalisation d'un système [30]

2.5. Modèle

Un modèle est une abstraction d'un système, modélisé sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé **[31]**.

Un bon modèle doit avoir les caractéristiques suivantes :

- ✓ **Abstrait :** Un modèle doit permettre de cacher certains détails inutiles à un moment donné de la conception.

- ✓ **Compréhensible :** Un modèle doit être facile à assimiler et à manipuler par tous les acteurs entrant dans l'élaboration du système.
- ✓ **Fidèle et précis :** Un modèle doit représenter fidèlement les propriétés et les caractéristiques du système.
- ✓ **Prédictif :** Un modèle doit fournir assez d'informations pour permettre de prédire les propriétés du système.
- ✓ **Économique :** Les coûts de construction et de tests sur le modèle doivent être limités et ne doivent pas dépasser les coûts de construction d'un prototype du système.

La création de modèles est faite en utilisant des langages bien définis. Ces derniers sont connus sous le nom de langages de modélisation ou formalisme de modélisation.

2.6. Formalisme de modélisation

L'IDM se base sur des formalismes ou des langages de modélisation. Le langage de modélisation est défini par : une syntaxe abstraite qui définit les concepts de base du langage, une syntaxe concrète qui définit le type de notation (graphique, textuelle ou mixte), des concepts abstraits, et une sémantique pour permettre l'interprétation des concepts par les acteurs et les surtout par machines.

La définition d'un langage de modélisation s'appuie sur le concept des modèles aussi, cette idée s'appelle La Méta-modélisation.

2.7. Méta-modèle

Selon Jean Marie Favre **[32]** « un méta-modèle est un modèle d'un langage de modèles ».

Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle, c.-à-d. le langage de modélisation. Il permet de définir précisément les concepts manipulés dans les modèles ainsi que les relations entre ces concepts **[33]**.

Le Méta-modèle à son tour est exprimé dans un langage de méta-modélisation spécifié par le Méta-Méta-modèle. Le langage utilisé au niveau du méta-méta-modèle doit être suffisamment

puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant (méta-circulaire).

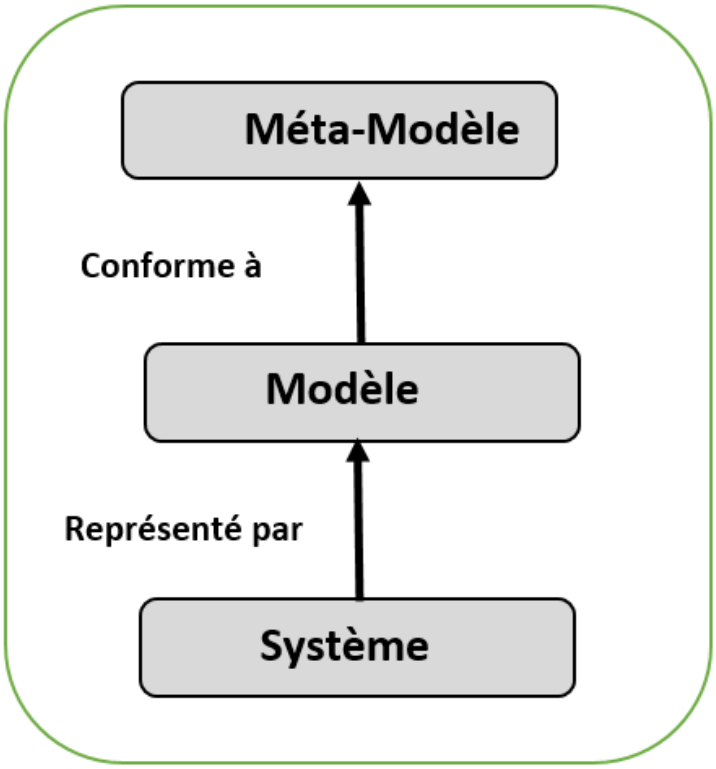

Figure 2.2 Relation entre système, modèle et méta-modèle.

L'IDM est un domaine basé sur les modèles et les technologies liées à leurs manipulations. Pour la pratique, il existe plusieurs manières (approches) d'utiliser les modèles dans le processus de développement d'un système. L'approche la plus développée et la plus utilisée est L'Architecture Dirigée par les Modèles (MDA : Model Driven Architecture).

3. Les approches de l'Ingénierie Dirigée par les Modèles (IDM)

3.1. Modèle de Calcul Intégré (MIC)

Cette approche a apporté une méthodologie de développement ainsi que des outils logiciels comme support. La méthodologie proposée est décomposée en deux phases. La première phase

consiste à analyser le domaine d'application afin de trouver les paradigmes de modélisation les plus appropriés et de définir avec précision le langage de modélisation qui sera utilisé. Un outil automatique peut alors utiliser ces informations pour générer automatiquement un environnement de modélisation dédié au domaine. Cet environnement est utilisé directement dans la seconde phase qui consiste à modéliser l'application désirée **[34]**.

3.2. Les usines logicielles (Software Factories)

Les usines logicielles "Software Factories" représentent la vision que Microsoft a proposée pour l'ingénierie dirigée par les modèles. Le terme d'usine logicielle fait référence à une industrialisation du développement logiciel similaire aux lignes d'assemblage dans l'industrie **[34]**.

3.3. L'Architecture Dirigée par les Modèles

L'architecture dirigée par les modèles ou Model Driven Architecture (MDA) est une démarche de réalisation de logiciel, proposée et soutenue par l'OMG. L'idée de base de cette approche est de définir des modèles indépendants de l'informatisation (Computation Independent Model, CIM), on transforme ces CIM pour obtenir des modèles totalement indépendants des plateformes techniques (Platform Independent Model, PIM), par la suite, on raffine ces PIM pour prendre en considération les problèmes liés à la technologie retenue et on obtient des modèles spécifique à la plate-forme cible (Platform Specific Model, PSM) **[34]**.

4. Architecture Dirigée par les Modèles

4.1. Définition

L'Architecture Dirigée par les Modèles (ADM) est une approche de développement proposée et soutenue par l'OMG (Object Management Group). Le principe clé et initial du MDA consiste à s'appuyer sur des standards tel que UML (Unified Modeling Language) et d'autre standards de l'OMG pour décrire séparément des modèles au cours des différentes phases du cycle de développement d'une application. Plus précisément, le MDA préconise l'élaboration de modèles :

- d'exigence (**CIM** : Computation Independent Model) dans lesquels aucune considération informatique n'apparait.
- d'analyse et de conception (**PIM** : Platform Independent Model).
- modèles de description de plateforme (**PDM** : Platform Description Model).
- de code (**PSM** : Platform Specific Model).

L'objectif majeur du MDA est l'élaboration de modèles pérennes (PIM), indépendants des détails techniques des plates-formes d'exécution (J2EE, .Net, PHP, etc.), afin de permettre la génération automatique de la totalité des modèles de code (PSM) et d'obtenir un gain significatif de productivité.

Parmi les standards fondamentaux de l'OMG liés à l'ADM nous pouvons citer :

- **UML (Unified Modeling Language)** : C'est un langage visuel semi-formel pour la modélisation des systèmes. Il permet de schématiser l'architecture, les solutions et les vues avec des diagrammes augmentés de texte **[35]**.
- **MOF (Meta-Object Facility) :** C'est un standard de méta-modélisation constitué d'un ensemble d'interfaces standards pour définir la syntaxe et la sémantique d'un langage de modélisation, créé principalement pour définir la notation UML [36].
- **OCL (Object Constraint Language) :** C'est un langage qui, intégré à UML, lui permet de formaliser l'expression des contraintes **[37]**.
- **XMI (XML Metadata Interchange) :** C'est un standard d'échange de métadonnées **[38]**.
- **CWM (Common Warehouse Metamodel) :** C'est une interface basée sur UML, MOF et XMI pour faciliter l'échange de métadonnées entre outils, plateformes et bibliothèques de métadonnées dans un environnement hétérogène **[39]**.
- **MOFM2T (MOF Model-to-Text language) :** C'est une spécification utilisée pour exprimer des transformations de modèles en texte **[40]**.

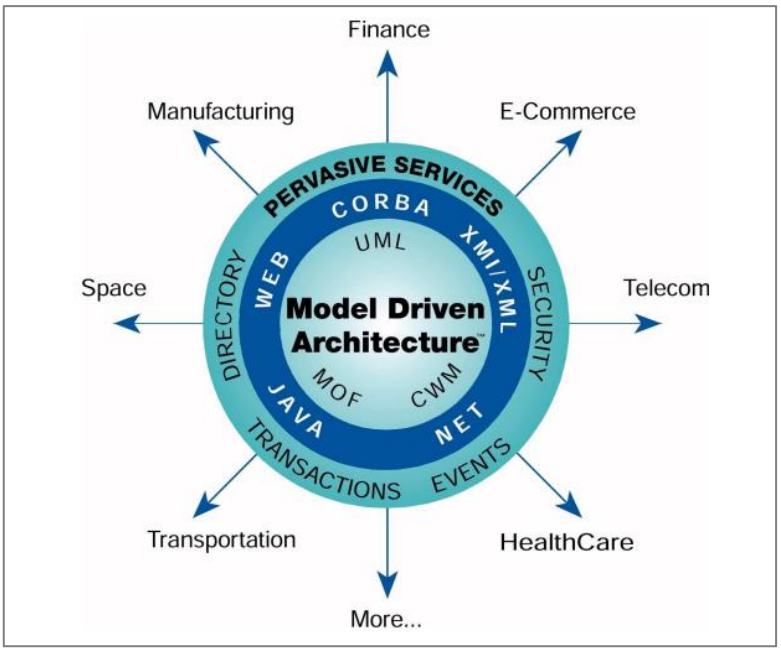

Figure 2.3 Les standards de l'ADM [41]

4.2. Une architecture à quatre niveaux

ADM est une architecture à 4 niveaux d'abstraction (figure 2.4) :

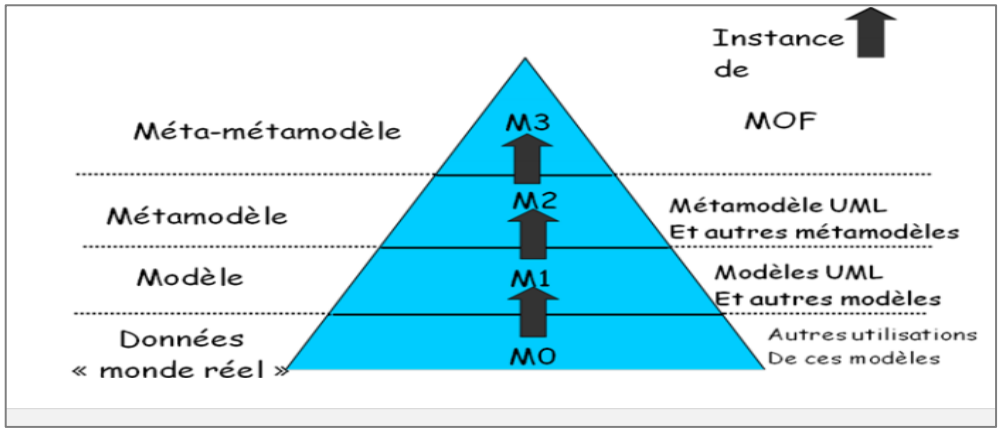

Figure 2.4 Les quatre niveaux d'abstraction pour l'ADM [42]

4.3. Les Modèles dans l'approche ADM

L'approche ADM permet la manipulation des modèles de natures diverses tels que : les modèles d'objets métiers, de processus, de service, de plateforme, etc. Nous pouvons distinguer quatre classes de modèles :

• Les modèles d'exigences (CIM : Computation Independent Model) : La réalisation de ce modèle est la première étape dans le développement d'un système puisqu'elle va permettre de modéliser toutes les exigences du client et définir les différentes interactions qui impliqueront le système dans ses environnements, interne ou externe. De ce fait, le CIM sera considéré comme référence pour vérifier la conformité du système avec les exigences du client. Le CIM ne donne aucun détail sur la façon de la réalisation ou le fonctionnement du système mais exprime clairement des liens de traçabilité avec les modèles futurs.

• Les modèles d'analyse et conception abstraite (PIM : Platform Independent Model) : Ces modèles doivent être indépendants de toute plateforme d'implémentation mais aussi contenir assez de détails pour permettre la génération automatique du code. Ils organisent le futur système en modules et sous-modules et relient le premier modèle d'exigence CIM avec le code.

• Les modèles de description de plateforme (PDM : Platform Description Platform) : Ces modèles fournissent l'ensemble de concepts techniques liés à la plateforme d'exécution et à ses services. Ils contiennent toutes les informations nécessaires à sa manipulation.

• Les modèles de code (PSM : Platform Specific Model) : Les modèles PSM facilitent la génération du code à partir de la combinaison des modèles PIM et PDM. Les PSM expriment, par exemple, les composants, les instructions, les conditions, etc.

4.4. Cycle de développement de l'approche ADM

Le cycle de développement de l'approche ADM est vu sous la forme d'un Y (voir la Figure 2.5) dont les branches représentent respectivement les spécifications fonctionnelles du système et les spécifications techniques de la plate-forme cible qui, une fois intégrées, mènent à l'implémentation. Par conséquent, le fossé entre le modèle et le système n'existe plus car le modèle est le système, ou au moins le système est censé être généré directement et automatiquement à partir du modèle.

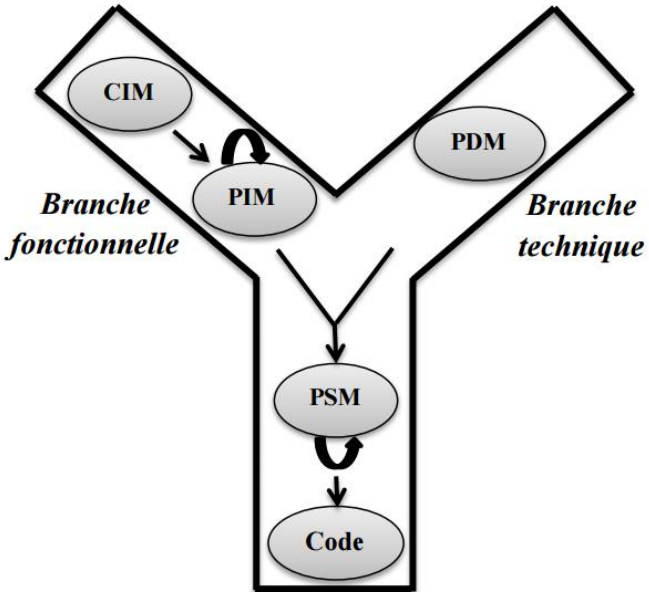

Figure 2.5 Le cycle de développement de l'ADM

4.5. Transformations de modèles en ADM

Elle est définie comme étant le processus de convertir un modèle d'un système à un autre modèle du même système **[43]**.

L'ADM considère le processus de développement du système comme une suite successive et stratégique de transformations de modèles. Ces transformations établissent de manière automatique des liens de traçabilité entre les trois types de modèles discutés précédemment, des transformations CIM vers PIM et PIM vers PSM dans cet ordre, ou inversement **[44].** La figure 2.6 montre ces transformations :

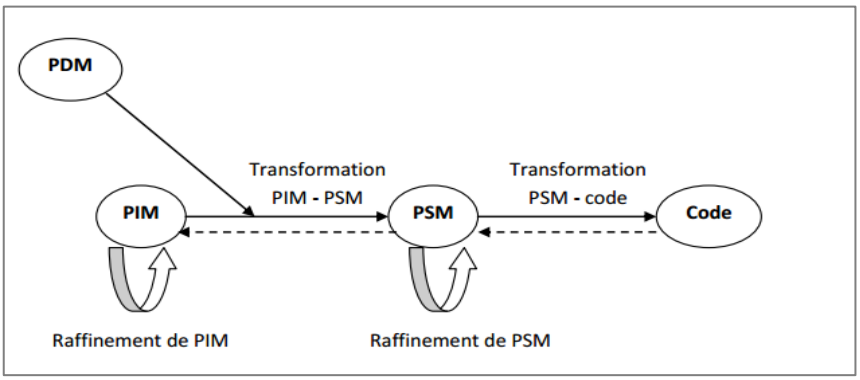

Figure 2.6 Transformations de modèles en ADM [30]

Une transformation a pour but de rendre les modèles opérationnels dans une approche IDM (tel que : ADM), ce qui augmente considérablement la productivité des applications.

La transformation de modèles est une initiative très importante dans toute approche orientée modèle. En effet, les transformations assurent les opérations de passage d'un ou plusieurs modèles d'un niveau d'abstraction donné vers un ou plusieurs autres modèles du même niveau (transformation horizontale) ou d'un niveau différent (transformation verticale). On peut citer comme exemple de transformation verticale, la transformation PIM vers PSM dans l'approche ADM. Le modèle transformé est appelé modèle source et le modèle résultant de la transformation est appelé modèle cible. Dans la section qui se suit nous introduisons les principes de la transformation de modèles, et les outils IDM utilisés dans la transformation de modèles.

5. La transformation de modèles

L'ingénierie dirigée par les modèles considère les opérations de transformations de modèles comme le moteur de développement, que ce soit pour l'analyse, l'optimisation ou la génération de code.

5.1. Définition

La transformation de modèles consiste à générer, à partir d'un ou de plusieurs modèles sources, un ou plusieurs modèles cibles du même système. Les modèles sont dans tous les cas conformes à leurs méta-modèles respectifs. Elle est gouvernée par un ensemble de règles utilisées par le moteur de transformation. Ce moteur prend en entrée le(s) modèle(s) source(s), exécute les règles de transformation et génère le(s) modèle(s) cible(s).

La figure 2.7 illustre les principaux concepts impliqués dans le processus de la transformation de modèles.

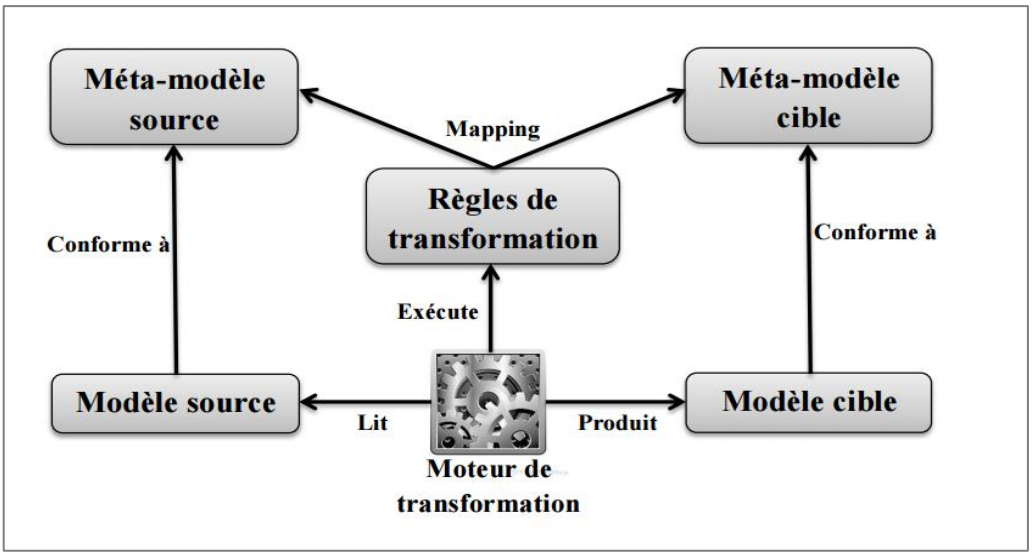

Figure 2.7 Concepts de base de la transformation de modèles [10]

5.2. Types de transformation

On peut catégoriser les types de transformation selon deux facteurs :

- Le niveau d'abstraction : Selon ce facteur, nous pouvons distinguer trois types de transformations :
  - ❖ **Transformations horizontales :** Ces transformations gardent le même niveau d'abstraction en modifiant les représentations du modèle source (ajout, modification, suppression ou restructuration d'informations).
  - ❖ **Transformations verticales :** La source et la cible d'une transformation verticale sont définies à différents niveaux d'abstraction. Un raffinement fait référence à une transformation qui baisse le niveau d'abstraction. Tandis qu'une abstraction désigne une transformation qui élève le niveau d'abstraction.
  - ❖ **Transformations obliques :** Ces transformations sont généralement utilisées par les compilateurs qui optimisent le code source avant la génération du code exécutable. Elles sont le résultat de la combinaison des deux premiers types de transformations.
- la nature des méta-modèles sources et cibles : Selon ce facteur nous distinguons deux types de transformations :

- **Transformations endogènes :** La transformation de modèles est qualifiée d'endogène si les modèles sources et cibles sont conformes au même métamodèle.
- **Transformations exogènes :** la transformation de modèles est dite exogène si elle se fait entre deux méta-modèles (source et cible) différents

5.3. Classification des approches de transformation de modèles

Les transformations de modèles peuvent être partagées en deux grandes classes : les transformations « Modèle vers Modèle » et les transformations « Modèle vers Code ».

5.3.1. Transformations de type Modèle vers code

Dans cette catégorie, on distingue entre les approches basées sur le principe du visiteur (Visitorbased approach) et celles basées sur le principe des templates (Template-based approach)

1) **Approche basée sur le visiteur (Visitor-based) :** approche de base pour la génération de code, elle consiste à fournir un mécanisme de visiteur pour traverser la représentation interne d'un modèle et créer le code. On peut citer comme exemple le framework Jamda qui fournit un ensemble de classes pour représenter les modèles UML, une API pour manipuler les modèles, et un mécanisme de visiteur pour générer le code.
2) **Approche basée sur les templates (Template-based) :** reposent sur l'utilisation des fragments de méta-code du code cible pour l'accès aux informations du modèle source. Ces approches sont actuellement très utilisées dans les outils MDA, tel que : AndroMDA (un générateur de code qui utilise la technologie ouverte Velocity pour l'écriture des patrons).

5.3.2. Transformations de type Modèle vers Modèle

On peut distinguer cinq Approches de transformation Modèle vers Modèle :

1) **Approches par manipulation directe :** Ces approches se basent sur une représentation interne des modèles source et cible, et sur un ensemble d'APIs pour les manipuler. Elles sont généralement implémentées comme des cadres structurants orientés objets qui fournissent un ensemble minimal de concepts – sous forme de classes abstraites par exemple. L'implémentation des règles et leur ordonnancement restent à la charge du développeur.

2) **Approches relationnelles :** Ces approches sont celles qui utilisent une logique déclarative reposant sur des relations d'ordre mathématique. L'idée de base est de spécifier les relations entre les éléments des modèles source et cible par le biais de contraintes. L'utilisation de la programmation logique est particulièrement adaptée à ce type d'approche. Généralement, les transformations produites sont bidirectionnelles.

3) **Approches basées sur les transformations de graphes :** Ces approches, qui exploitent les travaux réalisés sur les transformations de graphes. Elles sont similaires aux approches relationnelles dans le sens où elles permettent l'expression des transformations sous une forme déclarative. Néanmoins, les règles ne sont plus définies pour des éléments simples mais pour des fragments de modèles : on parle de filtrage de motif (pattern matching). Les motifs dans le modèle source, correspondant à certains critères, sont remplacés par d'autres motifs du modèle cible. Les motifs, ou fragments de modèles, sont exprimés soit dans les syntaxes concrètes respectives des modèles soit dans leur syntaxe abstraite.

4) **Approches dirigée par la structure :** Ces approches distinguent deux phases. La première consiste à créer la structure hiérarchique du modèle cible. La seconde consiste à ajuster les attributs et références dans le modèle cible.

5) **Approches hybrides :** Les approches hybrides sont une combinaison des différentes techniques. On peut notamment retrouver des approches utilisant à la fois des règles à logique déclarative et des règles à logique impérative. ATL et XDE sont deux exemples d'approches hybrides.

La figure 2.8 présente les deux classifications ainsi que les différentes Approches de transformation de modèles.

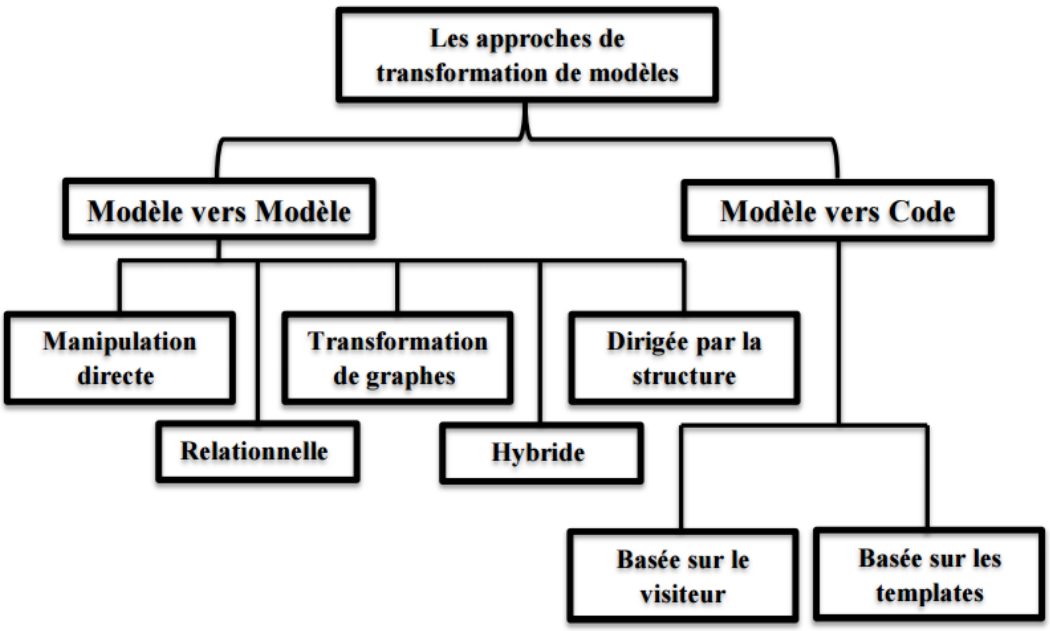

Figure 2.8 Les approches de transformation de modèles

5.4. Structure d'une transformation Modèle vers Modèle

Une transformation de modèle est principalement caractérisée par la combinaison des éléments suivants :

• **Spécification** : Certaines approches de transformation fournissent un mécanisme dédié de spécifications, tel que des pré-conditions et des post-conditions exprimées en OCL.

• **Règles de transformation :** Une règle de transformation est composée de deux parties : un côté gauche (LHS, Left Hand Side) qui accède au modèle source, et un côté droit (RHS, Right Hand Side) qui accède au modèle cible. Une règle comporte également une logique, qui exprime des contraintes ou calculs sur les éléments des modèles source et cible. Une logique peut avoir la forme déclarative ou impérative. Une logique déclarative consiste à spécifier des relations entre les éléments du modèle source et des éléments du modèle cible. Une logique impérative correspond le plus souvent, mais pas nécessairement, à l'utilisation de langages de programmation pour manipuler directement les éléments des modèles par le biais d'interfaces dédiées.

6. Discussion

La transformation du modèle est une zone relativement jeune. Bien qu'il soit lié aux domaines de transformation du programme et de méta-programmation, l'utilisation des langages de modélisation graphique et l'application du méta-modilisation orienté objet à la définition du langage définissent un nouveau contexte.

Au cours des dernières années le travail est focalisé sur les approches de la transformation, toutes les approches proposées sont engendrer par les deux approches majeures (modèle vers modèle, et modèle vers code), parfois en particularité dans l'étape de spécification et pour modéliser le comportement d'un système donné, une transformation qu'on appelle « ingénierie inverse » ou Reverse engineering est indispensable. Cette approche et contrairement à la deuxième approche citée précédemment (modèle vers code) sert à une transformation inverse du code (texte) vers un modèle cible (transformation code vers modèle). Ce type de transformation est Essentiel dans notre travail puisque le système à modéliser est basé sur les fichiers WSDL.

Bien qu'il existe des solutions satisfaisantes pour transformer des modèles en texte (comme les approches basées sur des modèles), ce n'est pas le cas pour transformer des modèles en modèles ou textes en modèles. De nombreuses nouvelles approches de la transformation dans ce contexte ont été proposées au cours des deux dernières années, mais peu d'expérience est disponible pour juger leur efficacité dans les applications pratiques. À cet égard, nous sommes encore à l'étape de l'exploration des possibilités et de l'élimination des exigences. Les outils de modélisation disponibles sur le marché commencent tout juste à offrir des capacités de transformation de modèle à modèle, mais elles sont encore très limitées et souvent improvisé, c'est-à-dire sans fondement théorique approprié.

7. Conclusion

Dans ce chapitre, nous avons introduit les concepts de base de l'ingénierie dirigée par les modèles c'est-à-dire l'approche ADM et la métamodilisation d'une part et la transformation de modèle d'autre part. Ces deux axes constituent les deux problématiques clé de l'IDM sur lesquelles la plupart des travaux de recherche se concentrent actuellement. Nous avons vu que la transformation de modèles est une notion clé et qu'elle joue un rôle décisif dans l'IDM et elle représente le noyau au de notre travail.

Une transformation sophistiquée nécessite le choix d'un bon langage de modélisation, et comme l'UML est souvent le plus utilisé dans ce domaine, nous avons fait le choix d'un profile UML (UML pour l'ingénierie des services ou UML-S) dédié à la composition des services web.

Le chapitre suivant sera consacré à la présentation d'UML-S et ses concepts principaux.

Chapitre 3

UML pour l'ingénierie des services

1. Introduction

Notre approche de développement, comme toutes les approches de type MDA, repose sur l'utilisation de modèles décrits dans un langage de modélisation clair et précis. Pour cet effet, l'OMG met en avant le langage UML qui est très populaire dans l'industrie.

UML est un langage graphique et textuel permettant de représenter les divers aspects d'un système. Il est destiné à comprendre et à définir des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.il possède treize diagrammes devisés en deux catégories statique et dynamique. Statique pour la représentation statique du système et dynamique pour représenter le comportement et la communication du système. La flexibilité d'UML est un facteur très important, car UML peut être étendu et personnalisé par l'utilisation du mécanisme des profils.

Dans ce chapitre nous commençons par une brève présentation d'UML, en suite nous présentons la notion des profils et nous terminons par la mise de l'accent sur le profile UML-S qui se présente comme la base de notre travail.

2. UML (Unified Modeling Language)

UML est un langage de modélisation qui permet de spécifier , visualiser, construire et documenter les artefacts des systèmes logiciels ainsi que pour la modélisation entreprise et les système non logiciels **[45]**.

Au niveau de '' Unified Modeling Language '' deux éléments importants sont à noter. Le terme '' Unified '' et le terme '' Langage ''. Le premier terme signifie Que les auteurs ont essayé de regrouper les éléments importants des concepts objets, alors que le deuxième montre qu'il s'agit d'un language de modélisation et non pas d'une méthode. UML est un language qui permet de modéliser non seulement des applications informatiques ou des structures de données, mais également les activités d'un domaine : mécanique, biologie, processus métier **[46]**.

UML comporte ainsi treize types de diagrammes représentant autant de *vues* distinctes pour représenter des concepts particuliers du système d'information. Ils se répartissent en deux grands groupes :

- **Diagrammes structurels ou diagrammes statiques (*UML Structure*)**

- ❖ diagramme de classes (*Class diagram*)
- ❖ diagramme d'objets (*Object diagram*)
- ❖ diagramme de composants (*Component diagram*)
- ❖ diagramme de déploiement (*Deployment diagram*)
- ❖ diagramme de paquetages (*Package diagram*)
- ❖ diagramme de structures composites (*Composite structure diagram*)

➢ **Diagrammes comportementaux ou diagrammes dynamiques (*UML Behavior*)**

- ❖ diagramme de cas d'utilisation (*Use case diagram*)
- ❖ diagramme d'activités (*Activity diagram*)
- ❖ diagramme d'états-transitions (*State machine diagram*)
- ❖ **Diagrammes d'interaction (*Interaction diagram*)**
  - diagramme de séquence (*Sequence diagram*)
  - diagramme de communication (*Communication diagram*)
  - diagramme global d'interaction (*Interaction overview diagram*)
  - diagramme de temps (*Timing diagram*)

Ces diagrammes, d'une utilité variable selon les cas, ne sont pas nécessairement tous produits à l'occasion d'une modélisation. Les plus utiles pour la maîtrise d'ouvrage sont les diagrammes d'activités, de cas d'utilisation, de classes, d'objets, de séquence et d'états-transitions. Les diagrammes de composants, de déploiement et de communication sont surtout utiles pour la maîtrise d'œuvre à qui ils permettent de formaliser les contraintes de la réalisation et la solution technique.

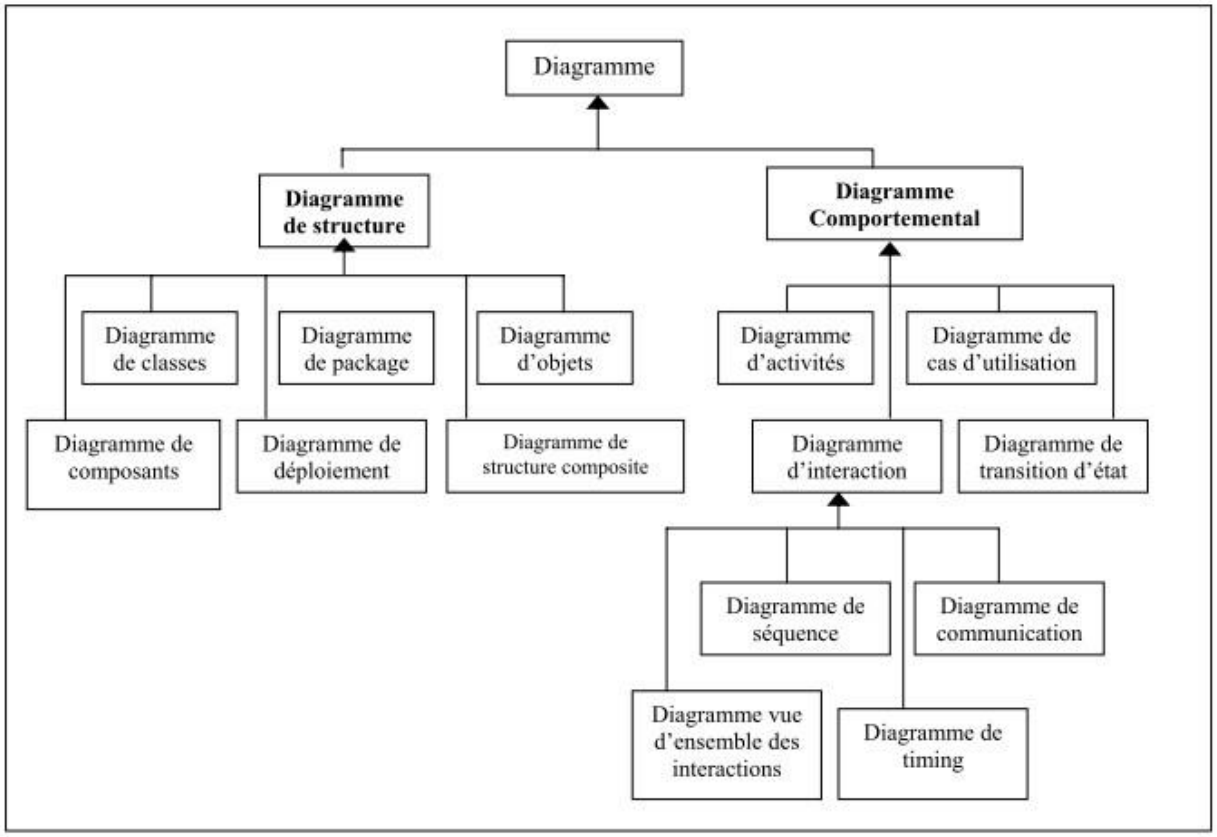

Figure 3.1 Les diagrammes UML [45]

Il existe des situations dans lesquelles un langage aussi générale et d'une portée aussi large peut ne pas convenir à la modélisation d'applications de certains domaines spécifiques. C'est le cas, par exemple, lorsque la syntaxe ou la sémantique des éléments UML ne peut pas exprimer des concepts spécifiques de systèmes particuliers, ou lorsque nous voulons restreindre ou personnaliser certains éléments UML généralement trop abondants et trop généraux. OMG définit une approche possible pour définir des langages spécifiques aux domaines. Une alternative à UML, en utilisant les mécanismes d'extension (stéréotypes, valeurs marquées et contraintes) pour spécialiser ses éléments, permettant des extensions personnalisées d'UML pour des domaines d'application particuliers. Ces personnalisations sont des ensembles d'extensions UML regroupés en profils UML.

3. Les Profils UML

Quand nous devons définir un nouveau langage pour modéliser un système qui restreint le nombre d'éléments UML ou ajoute certaines contraintes en respectant la sémantique originale, nous n'avons pas besoin de créer un nouveau langage à partir de zéro à l'aide du MOF. Au lieu de cela,

UML peut facilement être personnalisé en utilisant un ensemble de mécanismes d'extension qu'UML fournit lui-même. Plus précisément, le paquet Profiles inclus dans UML 2.0 définit un ensemble d'artefacts UML qui permet de spécifier un modèle MOF pour traiter les concepts spécifiques et la notation requise dans des domaines d'application particuliers (par exemple, en temps réel, en modélisation de processus métier, en finance, Etc.) ou des technologies de mise en œuvre (telles que .NET, J2EE ou CORBA) **[47]**.

Selon l'OMG, un profil UML fournit un mécanisme d'extension générique pour adapter les modèles UML à un domaine ou à une problématique spécifique. Les profils permettent aux spécialistes des différents domaines de décrire au mieux leurs systèmes à l'aide d'un ensemble d'extensions clairement définies. Un profil n'ajoute pas réellement de nouveaux concepts à UML mais se contente plutôt de spécialiser les concepts existants. Un profil peut également définir de nouvelles contraintes relatives aux concepts ou aux relations entre ceux-ci **[16]**.

Nous avons choisi un langage de modélisation UML-S qu'il utilise ce mécanisme de profil. Ce profile a été proposé par '' Christophe Dumez '' dans son travail intituler **« Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en œuvre de services Web composés ».**

4. Le profil UML-S

UML-S est ainsi défini comme un ensemble de personnalisations du métamodèle UML standard. Le processus de personnalisation est réalisé par l'ajout de stéréotypes, de valeurs étiquetées ou *tagged values* en anglais ainsi que de contraintes. Les stéréotypes sont représentés par des noms entre guillemets tel que *« WebService »* et placés au-dessus des noms des éléments tels que les classes. Ils permettent aux développeurs d'étendre le vocabulaire d'UML en ajoutant de nouveaux types d'éléments de modélisation, dérivés des éléments existants. À chaque stéréotype, il est possible d'assigner des propriétés nommées les valeurs étiquetées. La personne chargée de la modélisation pourra alors attribuer une valeur à chacune de ces propriétés au sein de son modèle. Un exemple de propriété dans le contexte des services Web serait WSDL_URL à laquelle le développeur attribuerait l'URL vers la description du service au format WSDL. Le dernier mécanisme d'extension réside dans l'ajout de contraintes. Ces contraintes permettent de raffiner la sémantique d'un élément du modèle en exprimant textuellement une condition ou une restriction à laquelle l'élément doit se conformer. Il est par exemple possible de définir une contrainte concernant la valeur d'une propriété d'un

élément. Pour éviter les ambiguïtés au niveau de ces contraintes, l'OMG a défini l'OCL qui est un langage standard pour l'expression de contraintes. Celles-ci sont généralement représentées graphiquement entre accolades à l'intérieur de rectangles dont le coin supérieur droit est replié. Nous utilisons tous ces mécanismes d'extension pour définir le profil UML-S. Mais comme notre travail sert à une transformation de modèle et plus particulièrement une ingénierie inverse pour extraire un modèle UML statique(c'est-à-dire les opérations et les types de données échangée entre les services composé) à partir d'un système de composition de services web, ou la sémantique jouent un rôle limité au cours de l'extraction ou plus précisément la transformation de modèle, nous n'avons pas focalisé sur le 3 éme mécanisme (les contraintes). Ce mécanisme est orienté vers la Verification formelle des modèles et leur validation ce qui n'est pas l'intérêt de notre travail.

Le profil UML-S est conçu pour répondre à plusieurs problématiques liées à la composition de services. Tout d'abord, UML-S utilise un ensemble restreint de diagrammes et d'éléments UML et tente également de réduire au maximum le nombre d'extensions apportées. Ceci permet d'obtenir des modèles simples et clairs, très proches du profil UML standard. Ceci simplifie également la tâche du développeur qui n'a pas à manipuler de nombreux types d'éléments différents. La deuxième problématique repose dans le niveau d'expressivité des modèles UML-S. Les extensions ajoutées doivent être pertinentes afin de permettre la transformation automatique des modèles en code d'exécution. L'objectif d'UML-S est de fournir des modèles UML-S suffisamment expressifs pour permettre de générer le code dans son intégralité tout en restant suffisamment génériques pour prendre en charge différentes technologies d'implémentation. UML-S permet ainsi de réaliser des modèles PIM indépendants de la plateforme. La dernière problématique se situe au niveau du workflows représentant un processus métier et dans notre contexte la composition de services. UML-S doit pouvoir modéliser de manière directe et compacte les structures de contrôle prises en charge par les langages de composition actuels et particulièrement BPEL qui est le plus utilisé actuellement. En effet, UML-S doit utiliser au mieux les fonctionnalités proposées par les technologies de composition existantes afin de permettre la modélisation claire sous la forme de workflows de services composés complexes.

5. Rôle d'UML-S dans le développement

Le langage de modélisation défini par le profil UML-S joue un rôle important au sein de notre approche de développement puisqu'il est utilisé pour la spécification des modèles décrivant la

composition de services. Ce langage permet ainsi au développeur de spécifier le comportement du service composé d'une manière abstraite, indépendante de la technologie d'implémentation.

La place d'UML-S dans le cycle de développement est illustrée dans la figure 3.2 Deux types de diagrammes UML-S existent afin de permettre la modélisation d'un service composé selon différents points de vue. Le diagramme de classes permet de modéliser l'aspect structurel du système, c'est à dire les interfaces des services et les types de données manipulés. Le diagramme d'activité modélise quant à lui l'aspect dynamique de la composition, c'est à dire les interactions entre les services. Le diagramme d'activité donne ainsi une représentation claire et précise du scénario de composition sous la forme d'un processus métier.

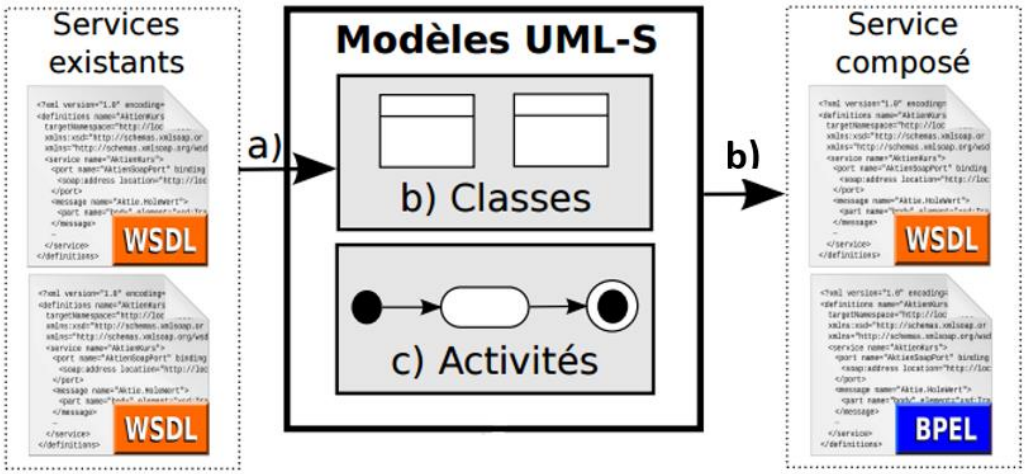

Figure 3.2 La place d'UML-S dans le cycle de développement

6. Diagramme de classes

Le diagramme de classes UML est un diagramme permettant de décrire la structure statique d'un système logiciel. Il modélise les différentes parties d'un système sous forme de classes et leurs relations sous forme d'héritage, d'associations ou d'agrégation. Chaque classe du diagramme est caractérisée par un nom, un ensemble d'attributs et un ensemble de méthodes ou opérations. Le diagramme de classes est généralement utilisé pour la modélisation orientée objet pour représenter les différentes classes d'objets et les liens entre ces classes, On obtient ainsi une vue statique de la structure du système logiciel.

7. Utilisation du diagramme de classes

À la différence de l'approche objet, les composants d'une architecture orientée services (SOA) ne sont plus des classes mais des services. Chaque service est un composant logiciel indépendant qui se suffit à lui-même et qui est caractérisé par une interface à laquelle les autres composants logiciels doivent se conformer afin d'en faire usage. Dans le contexte de SOA, il apparaît donc intuitif et logique d'utiliser le diagramme de classes UML afin de représenter les services et plus précisément leur interface. S'agissant d'un diagramme pour la modélisation de structures statiques, celui-ci ne sera en revanche pas utilisé pour décrire les interactions entre les services.

Nous nous contentons donc de représenter la partie statique de la composition, c'est à dire les services impliqués, leurs interfaces et les types de données manipulés. Dans le contexte de la composition de services, une classe peut donc être assimilée à un service propre aux architectures SOA ou à un type de donnée de manière similaire à l'approche objet. Une différence importante réside dans le fait qu'une classe de service comporte des opérations mais pas de propriétés alors qu'une classe représentant un type de donnée ne stipule que des attributs, appelés également propriétés. Afin d'accentuer visuellement la différence conceptuelle entre ces deux types de classe, le profil UML-S introduit le stéréotype « WebService ». Ce stéréotype apparaît sur le diagramme au-dessus du nom des classes représentant des services Web. UML-S définit également pour ce stéréotype une valeur étiquetée ou en anglais tagged value nommée WSDL_URL. Dans le cas où les services modélisés existent déjà, cette valeur contient l'URL vers leur description au format WSDL. Le développeur n'a pas besoin de fournir d'informations supplémentaires concernant ces services puisque toutes les propriétés importantes peuvent être récupérées à partir du WSDL.

8. Discussion

Étant donné qu'un document WSDL est conçu pour décrire l'interface et les types de données manipulées par un service Web, le diagramme de classe UML-S peut être considéré comme une représentation graphique du WSDL. Le document WSDL contient cependant plus d'informations que le diagramme UML-S, telles que le type de protocole de communication utilisé et le point d'accès du service. Ces informations sont utiles à l'implémentation mais ne présentent que peu d'intérêt pour le développeur à l'étape de spécification. C'est la raison pour laquelle UML-S ne cherche pas à représenter de manière exhaustive le contenu du document WSDL. Ceci permet d'obtenir une représentation visuelle plus compacte, plus facile à comprendre et donc à manipuler. Un document

WSDL peut être transformé en diagramme de classes UML-S de manière directe, en utilisant des règles de transformation.

9. Conclusion

Dans ce chapitre on a essayé de présenter une introduction au langage de modélisation UML, ainsi que nous avons présente la notion des profils UML et leur importance comme une extension pour adopter à l'exigence des défirent systèmes, Ensuite l'accent est mis sur le profile UML-S qu'on a choisi pour modéliser les particularités de système de composition des services web, nous avons vu aussi que le diagramme de class joue un rôle centrale dans notre système de composition. Un diagramme de classe est considérer comme une représentation graphique de notre système de composition des services web, Tel est l'objectif de notre travail.

Le chapitre suivant est consacré à l'implémentation d'un système de transformation de modèles (texte vers modèle) qui prend comme entrée les descriptions WSDL des services web a composé et il résulte un modèle UML Ce qui apparaît comme un diagramme de classe.

Chapitre 4

Implémentation

1. Introduction

Nous allons présenter dans ce chapitre un outil C-Sharp incluant les différentes fonctionnalités de modélisation d'une composition des services web, l'outil est considérer comme un moteur de transformation de modèles des fichiers WSDL vers un diagramme de class UML-S. Nous allons présenter aussi l'approche suivie lors de l'élaboration de notre logiciel

Notre approche suit le schéma suivant :

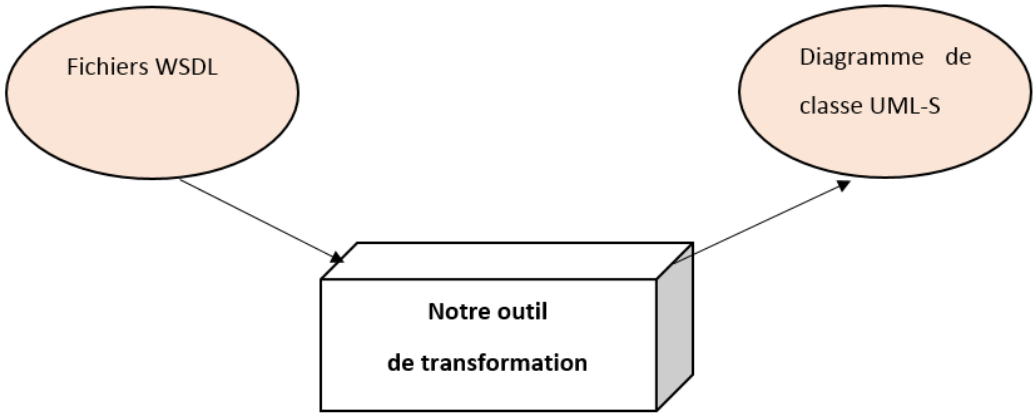

Figure 4.1 Schéma de notre approche

Le passage des fichiers WSDL vers leurs équivalents en diagramme de class a été réalisé par une transformation direct c'est-à-dire a travers une manipulation direct des modèles. La transformation considère une composition statique des services web composantes car le scénario de composition de ces services est déjà prédéfini, la composition est aussi une chirographie, elle résulte un seul service dit le service orchestrateur qu'il dirige les interactions entre les services composantes.

Cette transformation est réalisée sous l'environnement de programmation Microsoft visual studio 2013.

2. Approches existantes

Plusieurs approches basées sur la MDA Ont été proposés pour fournir une modélisation d'une composition des services web :

L'approche de **Gronmo, David Skogan, Ida Solheim et Jon Oldevik [48]** consistent à modéliser la composition, la découverte et la sélection. Dans la première étape (Découverte des services web), le développeur utilise un navigateur Web, un client de registre (généralement UDDI) pour rechercher et découvrir des services Web candidats qui peuvent être utilisés dans le service composite. La sortie de cette activité est une liste de descriptions de services Web, représentées sous forme de documents WSDL. Les descriptions de service Web sont ensuite converties en UML par une transformation d'ingénierie inverse (Import Web Services Descriptions). La sortie de cette étape est un ou plusieurs modèles UML des services Web découverts. Il sera plus facile pour le développeur de comprendre les modèles UML de l'interface du service Web que de comprendre les documents WSDL. Le développeur utilise ensuite un outil UML pour examiner et intégrer les modèles importés pour former un modèle de service Web composite (Model Composite Web Service). Cette activité comprend deux sous-activités, la modélisation d'interface et la modélisation du flux de travail (workflow). La modélisation d'interface spécifie l'interface du service et ses opérations, alors que la modélisation du flux de travail spécifie les processus internes du service. La sortie est un nouveau modèle UML représentant le nouveau service Web composite avec ses interfaces et son flux de travail. Ce modèle peut maintenant être utilisé pour générer la description WSDL du service composite (Export Web Service Descriptions) et pour générer sa description de workflow exécutable.

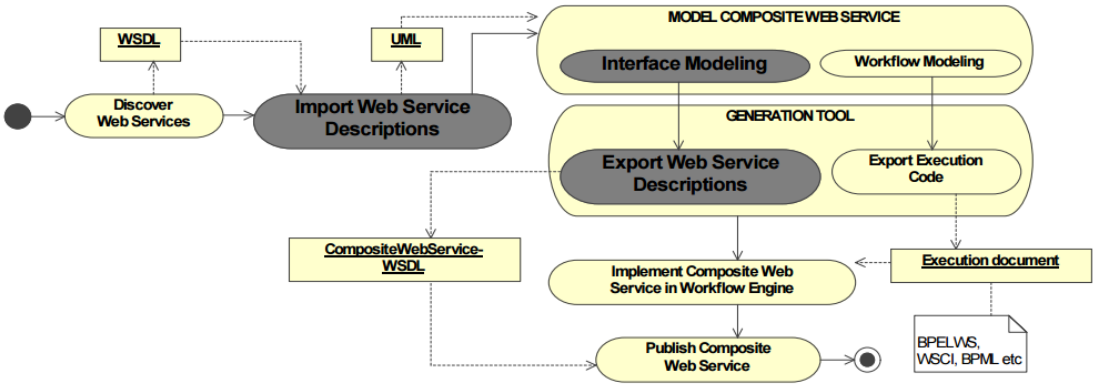

Figure 4.2 L'approche de Gronmo [48]

Bensaber et Malki [49] propose une méthodologie à trois phases : processus d'ingénierie inverse, processus d'annotation et processus de conversion. La conversion automatique du WSDL vers le profil UML est fournie par un ensemble de règles de conversion. L'approche utilise OCL pour représenter les effets, les conditions pré et post dans la modélisation UML.

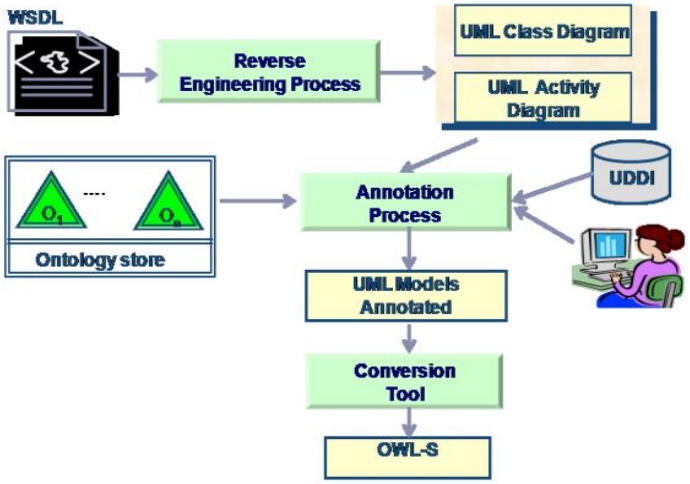

Figure 4.3 L'approche de Bensaber et Malki [49]

Weijun Sun, Shixian Li1, Defen Zhang et YuQing Yan [50] apportent une approche ou Le processus d'ingénierie inverse des services Web devient nécessaire afin de faciliter la réutilisation et la composition des services Web. C'est une méthodologie d'ingénierie inverse de service Web orienté modèle, où les descriptions de services Web (WSDL) sont transformées en modèles UML, puis les modèles UML générés sont intégrés dans des services Web composites, enfin les nouvelles descriptions de services Web (OWL) sont générées.

Christophe Dumez [16] dans son travail intituler **« Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en œuvre de services Web composés »** a proposé une approche conçue pour la spécification et la modélisation de services Web composés, tout en suivant les directives promulguées par l'OMG pour le ADM. S'agissant d'une approche ADM, les modèles sont au cœur de la méthodologie ainsi que leur transformation automatique. En effet, le développeur travaille à un niveau d'abstraction élevé en élaborant des modèles de type PIM pour décrire le comportement des services composés. Fidèles aux principes de l'ADM, il a proposé un

profil UML pour obtenir un langage de modélisation plus adapté au domaine des services Web. Ce profil nommé UML-S.

Pour modéliser les interfaces des services Web, UML-S fait l'analogie entre une classe et un service Web. En effet, les deux sont similaires dans la manière dont leur nom et leurs méthodes sont décrits. En outre, les méthodes de services Web peuvent gérer des objets complexes qui peuvent également être représentés à l'aide de classes UML. Pour distinguer un service Web d'une classe UML habituelle, UML-S ajoute un stéréotype WebService aux classes correspondant aux services Web. Les valeurs étiqueter sont des paires (nom-valeur) qui peuvent être associées à des classes, des opérations ou des attributs pour signifier un traitement spécial dans les transformations et / ou la génération de code.

Dans cette approche Le spécificateur importe les services Web qu'il veut composer à partir de l'URL de son WSDL (par exemple dans un registre UDDI). Une fois importé, il obtient un diagramme de classe présentant tous les services Web et les types de données complexes impliqués. Une fois cela fait, il devrait ajouter les classes correspondant au service Web composite qu'il veut créer. Si les méthodes de ce nouveau service renvoient des types de données complexes ou les prennent comme paramètres, le spécificateur devrait également définir des classes pour ceux-ci. Dans le cas où le service Web composite utilise le même type de données complexe qu'un autre service de service importé, il peut simplement lier le service à la classe du type de données déjà existant, en utilisant une association unidirectionnelle (du service à la classe).

3. Motivation

Toutes les approches citées dans la section précédente Partagent le même point de vue, c'est qu'un modèle UML des services web est très nécessaire dans le cycle de vie d'une composition. L'étape de modélisation est considérer comme une étape clé pour une composition correcte des services web.de ou là nous nous somme motivé de faire une implémentation pour garantir la génération d'un modèle diagramme de class UML correcte qu'il permet par la suite une composition impeccable des services web.

L'approche qui nous inspiré pour faire l'implémentation c'est celle de Christophe Dumez **[16]**, l'implémentation suit des règles de transformation claires et précises.

Les Outils technologiques utilisés pour réaliser l'outil de transformation sont cité dans la section suivante.

4. Outils technologiques utilisés

Dans cette section, nous présentons la plateforme choisie, le langage de programmation utilisé pour la réalisation, l'environnement de développement ainsi que l'outil permettant la visualisation graphique des web services manipulés par notre application et leur diagramme de classe équivalent.

4.1. la Platform Microsoft .NET

Microsoft.NET est le nom d'un ensemble de produits et de technologies de l'entreprise Microsoft dont la plupart dépendent du framework.NET, un composant du système d'exploitation Windows constituant un équivalent de machine virtuelle **[51]**.

Le Framework .Net a été conçu par Anders Hejlsberg, le père de Delphi. Celui-ci a développé entre autre le langage C#, qui devient le futur remplaçant de Delphi **[51]**.

4.2. Langage de programmation

Microsoft C# (prononcez C Sharp) est un langage de programmation qui a été conçu pour permettre la création d'une large gamme d'applications d'entreprise s'exécutant sur le .NET Framework. Évolution du Microsoft C et du Microsoft C++, C# est simple, moderne, à sécurité de type et orienté objet. Le code C# est compilé en tant que code managé, c'est-à-dire qu'il bénéficie des services du Common Language Runtime (CLR). Ces services incluent l'interopérabilité entre les langages, un garbage collection, une sécurité améliorée **[52]**.

4.3. Choix de l'environnement de travail

L'environnement de travail choisi est « Microsoft Visual Studio 2013 » de Microsoft, qui est un ensemble complet d'outils de développement permettant de générer des applications Web ASP.NET, des Services Web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C++, Visual C# et Visual J# utilisent tous le même environnement de développement intégré (IDE, Integrated Development Environment), qui leur permet de partager des outils et facilite

la création de solutions faisant appel à plusieurs langages. Par ailleurs, ces langages permettent de mieux tirer parti des fonctionnalités du Framework .NET, qui fournit un accès à des technologies clés simplifiant le développement d'applications Web ASP et de Services Web XML.

4.4. l'outil de visualisation graphique

Pour Créer des interfaces utilisateur interactives riches et convivial, nous avons choisi la technologie "Windows Forms " fourni par Microsoft. Et comme une grand part d'applications de nos jours doivent afficher des données à partir d'une base de données, d'un fichier XML, d'un service Web XML ou d'une autre source de données. Windows Forms fournit des contrôles flexibles qui facilite ces taches (ex : DataGridView).

Windows Forms est une technologie riche, elle fournit un ensemble de bibliothèques gérées qui simplifient les tâches d'application courantes telles que la lecture et l'écriture dans le système de fichiers. Lorsque vous utilisez un environnement de développement comme Visual Studio, vous pouvez créer des applications Windows Forms qui affichent des informations, demandent des entrées d'utilisateurs et communiquent avec des ordinateurs distants sur un réseau **[53]**.

Comme étant une technologie Microsoft, Windows-forms suit le principe de Conception des couches logique (Logical Layer Design) permettant une meilleure flexibilité a votre application par la définition d'une architecture multicouche (couche présentation, couche logique et la couche de données). Ce style architectural définit trois couches distinctes :

- Couche de présentation : permettant l'affichage de l'information aux utilisateurs.
- Couche logique : contient les fonctionnalités principales de manipulation de données.
- Couche de données : responsable de stockage des données persistante dans un magasin de données.

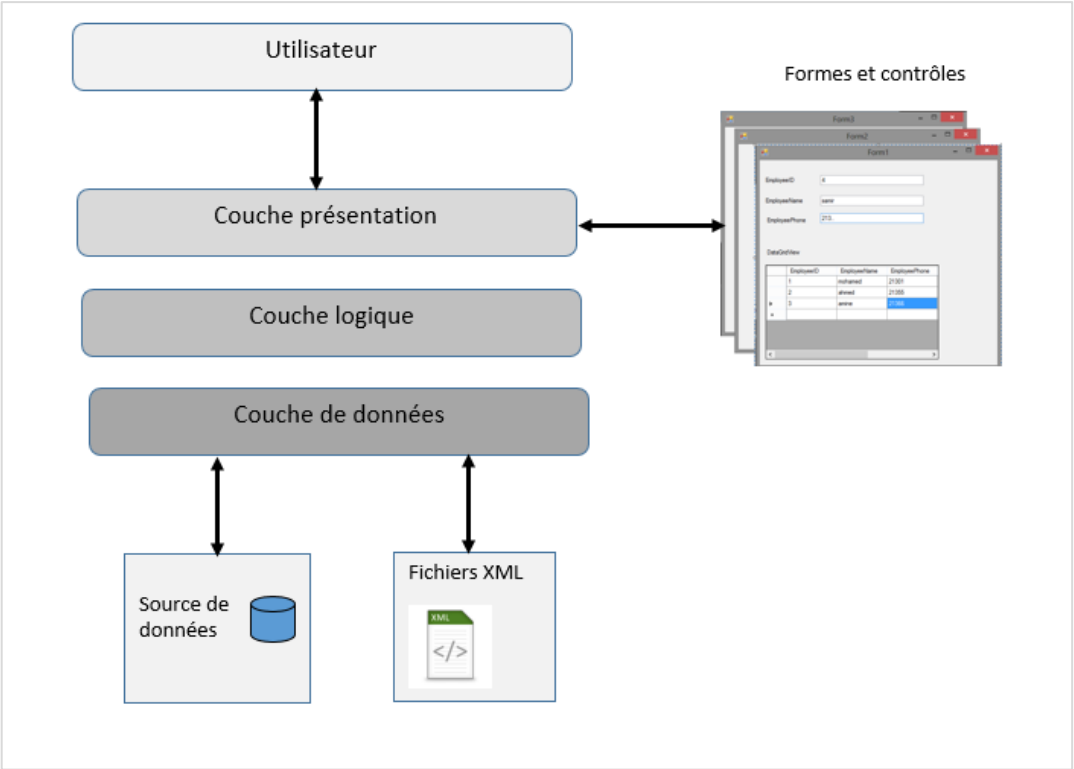

Figure 4.4 Architecture 3 couches d'une application windows-forms

Les Avantages :

- Facilite la maintenabilité par l'isolation des différentes couches des fonctionnalités de l'application.
- Minimisé le cout de développement (le temps).
- Vous souhaitez toujours minimiser l'impact de l'ajout de services à une application existante.

4.5. Bibliothèque .Net ServiceDescription

L'espace de noms **System.Web.Services.Description** se compose des classes qui vous permettent de décrire publiquement un service Web XML en utilisant le WSDL (Web Services Description Language). Chaque classe dans cet espace de noms correspond à un élément spécifique de la spécification WSDL et la hiérarchie des classes correspond à la structure XML d'un document WSDL valide [54].

La bibliothèque ServiceDiscription Représente une description complète et intégrale du service, y compris tous les informations de service et les spécifications pour leurs adresses, liens, contrats et

comportements respectifs **[55]**. Cette bibliothèque présente dans la couche de donnée comme un moyen de communication et l'interrogation des fichiers WSDL.

5. Implémentation

Comme notre outil est dédié à la transformation des fichiers wsdl vers un modèle UML-S diagramme de classe, une analyse sémantique des fichiers WSDL est très nécessaire pour comprendre la structure logique d'un fichier WSDL, ces éléments et les relations entre eux afin déterminer les règles de la grammaire de transformation de modèle. La grammaire de transformation de modèles est l'ensemble des règles appliqués sur un modèle source (ici c'est les fichiers WSDL) pour but d'obtenir un modèle cible plus au moins abstrait, dans cette section la première partie est consacrée à l'analyse de la structure interne des fichiers WSDL. La deuxième partie est consacrée à la présentation de notre grammaire de transformation illustrant les règles suivies pour faire le passage vers le diagramme de classe UML-S.

5.1. La Structure d'un fichier WSDL

Dans cette section on va expliquer comment lire un document WSDL en analysant la description des services Web. Au cours de l'article un diagramme d'arbre est développé à partir du contenu du document WSDL. L'arbre illustre la structure de WSDL et Permet une compréhension des éléments WSDL et leurs relations.

Un service web (service BLZ de banque) publiquement disponible sera utilisé comme un exemple dans cette analyse, Ce WSDL génère une requête et une réponse pour un service Web qui vous permet d'accéder à des codes de tri bancaires (appelés codes BLZ de banque). Dans la demande, vous devez passer le code BLZ d'une banque et revenir avec les détails de la banque.

Analyse :

L'élément racine d'un document WSDL est **<*definitions*>** Nous commençons donc l'arbre WSDL avec un nœud de définitions en tant que racine.

1) Service :

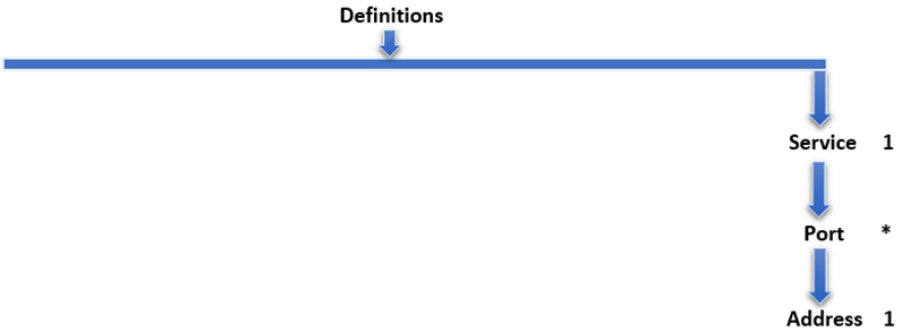

Figure 4.5 La partie <service> d'un fichier wsdl

Pour analyser un document WSDL, il est recommandé de le lire en bas vers le haut. Au bas du WSDL du service BLZ, nous trouvons un élément enfant de ***<definitions>*** appelées ***<service>*.**

Le nom du service est BLZService. Un service peut avoir plusieurs ports. Chaque port décrit un moyen d'accéder au service. Dans notre exemple BLZService, il existe trois ports. Un pour SOAP 1.1, un pour SOAP 1.2 et un pour la liaison HTTP.

Chaque port a un element fils <address>, L'élément d'adresse à un attribut nommé '' location '' indiquant l'adresse du service. Chaque adresse à un préfixe XML différent des autres éléments. Le préfix soap est pour la spécification de la liaison(ou <binding>) SOAP 1.1.

```
75  <wsdl:service name="BLZService">
76      <wsdl:port name="BLZServiceSOAP11port_http" binding="tns:BLZServiceSOAP11Binding">
77          <soap:address location="http://www.thomas-bayer.com/axis2/services/BLZService"
            ></soap:address>
78      </wsdl:port>
79      <wsdl:port name="BLZServiceSOAP12port_http" binding="tns:BLZServiceSOAP12Binding">
80          <soap12:address location="http://www.thomas-bayer.com/axis2/services/BLZService"
            ></soap12:address>
81      </wsdl:port>
82      <wsdl:port name="BLZServiceHttpport" binding="tns:BLZServiceHttpBinding">
83          <http:address location="http://www.thomas-bayer.com/axis2/services/BLZService"
            ></http:address>
84      </wsdl:port>
85  </wsdl:service>
```

Figure 4.6 Code WSDL de la partie <service>

2) Binding

La partie ***<binding>*** fournit des détails sur un transport spécifique. Le binding définit comment les messages sont transmis et les protocoles de transport utilisé pour chaque opération du service.

```
39    <wsdl:binding name="BLZServiceSOAP11Binding" type="tns:BLZServicePortType">
40        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"></soap:binding>
41        <wsdl:operation name="getBank">
42            <soap:operation style="document" soapAction=""></soap:operation>
43            <wsdl:input>
44                <soap:body use="literal"></soap:body>
45            </wsdl:input>
46            <wsdl:output>
47                <soap:body use="literal"></soap:body>
48            </wsdl:output>
49        </wsdl:operation>
50    </wsdl:binding>
```

Figure 4.7 Code WSDL de la partie <binding>

Chaque opération a un message d'entrée ***<Input>*** et peut avoir un message en sortie ***<output>.*** Il faut notée que Les messages manipulés peuvent avoir plusieurs formats.

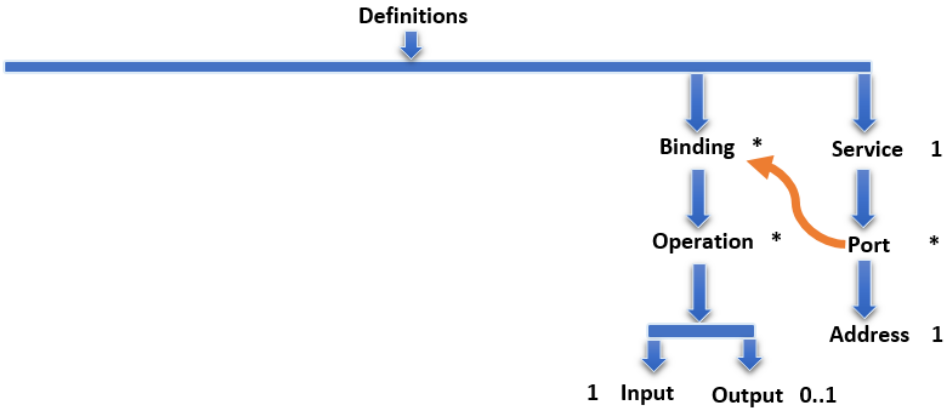

Figure 4.8 La partie <binding> d'un fichier wsdl

3) PortType

Dans la partie ***<portType>*** nous trouvons des éléments d'opération comme dans la Binding. Mais cette fois, l'Input et l'Output sortie décrivent la structure des messages, pas les options spécifiques au transport.

```
33    <wsdl:portType name="BLZServicePortType">
34        <wsdl:operation name="getBank">
35            <wsdl:input message="tns:getBank"></wsdl:input>
36            <wsdl:output message="tns:getBankResponse" wsaw:Action="
              http://thomas-bayer.com/blz/BLZService/getBankResponse"></wsdl:output>
37        </wsdl:operation>
38    </wsdl:portType>
```

Figure 4.9 Code WSDL de la partie <portType>

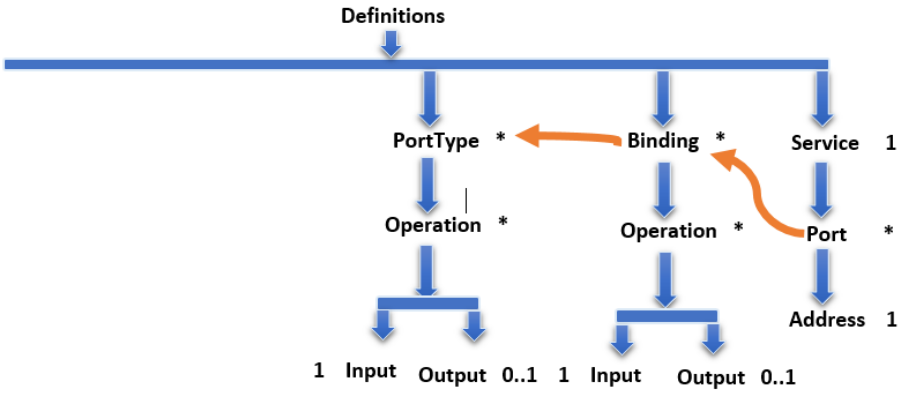

Figure 4.10 La partie <portType>d'un fichier wsdl

4) Message

Cette partie illustre tous les messages utilisés par le service pour communiquer avec d'autres services, L'élément ***<message>*** définit les éléments de données d'une opération. Chaque message peut consister en une ou plusieurs ***<part>***. Les éléments (***part)*** peuvent être comparés aux paramètres d'un appel de fonction dans un langage de programmation traditionnel.

```
27    <wsdl:message name="getBank">
28        <wsdl:part name="parameters" element="tns:getBank"></wsdl:part>
29    </wsdl:message>
30    <wsdl:message name="getBankResponse">
31        <wsdl:part name="parameters" element="tns:getBankResponse"></wsdl:part>
32    </wsdl:message>
```

Figure 4.11 Code WSDL de la partie <message>

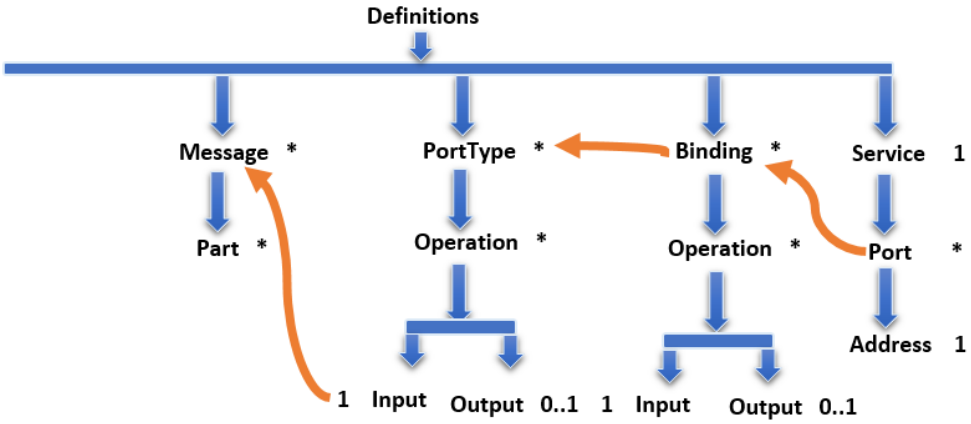

Figure 4.12 La partie <message>d'un fichier wsdl

5) Types

La partie <Types> peut avoir plusieurs éléments schémas XML, ces element contiennent des definitions des types de données simple, type de données complexes et des déclarations des éléments de la partie Message. Ces derniers peuvent fait l'appelle aux types de de données complexes ou élémentaires.

```
<wsdl:types>
    <xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="
    http://thomas-bayer.com/blz/">
        <xsd:element name="getBank" type="tns:getBankType"></xsd:element>
        <xsd:element name="getBankResponse" type="tns:getBankResponseType"></xsd:element>
        <xsd:complexType name="getBankType">
            <xsd:sequence>
                <xsd:element name="blz" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="getBankResponseType">
            <xsd:sequence>
                <xsd:element name="details" type="tns:detailsType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="detailsType">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="bezeichnung" type="xsd:string"></xsd:element>
                <xsd:element minOccurs="0" name="bic" type="xsd:string"></xsd:element>
                <xsd:element minOccurs="0" name="ort" type="xsd:string"></xsd:element>
                <xsd:element minOccurs="0" name="plz" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:schema>
</wsdl:types>
```

Figure 4.13 Code WSDL de la partie <types>

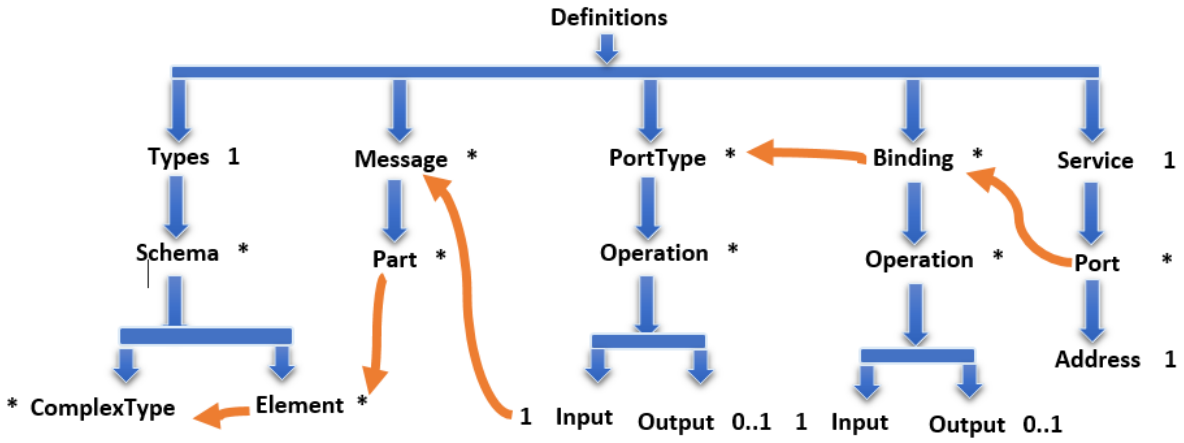

Figure 4.14 Diagramme final de la structure d'un fichier wsdl

5.2. Les règles de transformation

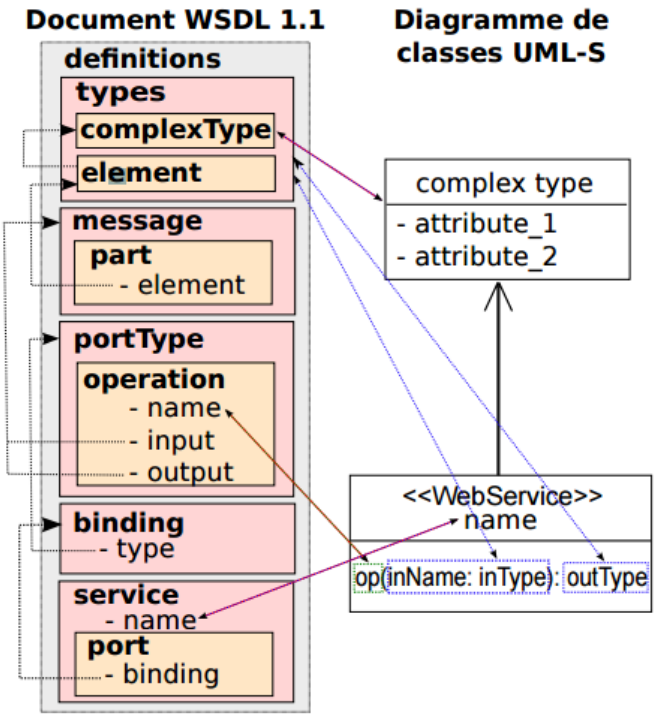

Figure 4.15 Transformation de WSDL 1.1 vers diagramme de classes UML-S [16]

Tout d'abord, l'analyseur WSDL devrait localiser la section service. Il indique le nom du service qui sera utilisé pour nommer la classe WebService, ainsi que le nom de l'interface (appelé portType dans WSDL 1.1). Une fois que le nom de l'interface est connu, l'analyseur doit le trouver dans le fichier WSDL. La section d'interface permet de remplir une partie des méthodes de la classe WebService. En effet, il mentionne les noms des méthodes fournies par le service Web à l'aide des balises d'opération. L'analyseur peut également récupérer des sections d'opération les noms des messages d'entrée / sortie des méthodes. Une fois que les noms des messages d'entrée / sortie sont récupérés, l'analyseur peut trouver leur définition dans la section types du WSDL. En effet, depuis WSDL 2.0, les messages sont définis en utilisant un schéma XML dans la section types, avec les autres types complexes, au lieu d'utiliser des sections de message distinctes. Les messages sont définis à l'aide d'une balise d'élément, à partir de laquelle, l'analyseur peut récupérer les noms des paramètres (ou produire s'il s'agit d'un message de sortie) ainsi que leur type (complexe ou non). À ce stade, il est possible de générer toute la classe WebService. Les seules choses manquantes sont les classes correspondant aux types complexes traités par le service Web et les liens d'association entre le service

Web et lesdites classes. Ils doivent être représentés comme des classes dans le diagramme UML-S et une association doit être ajoutée entre la classe WebService et la classe de type complexe. Les types complexes sont également définis dans la section types du WSDL, en utilisant un schéma XML. À partir de la définition XML du type complexe, il est possible de remplir le contenu de la classe UML-S, c'est-à-dire ses propriétés (noms et types). À ce stade, l'analyseur peut donc générer le diagramme de classe entier correspondant au WSDL du service Web importé. La conversion d'un document WSDL en diagramme de classe est simple compte tenu de la similitude des deux modèles. Cependant, le diagramme de classe UML-S est beaucoup plus convivial que WSDL et permet une meilleure compréhension du service Web.

La figure 4. explique la procédure de transformation d'un service web '' **Hopital** '' qui sert à un service de localisation de plus proche hôpital dans un emplacement donné :

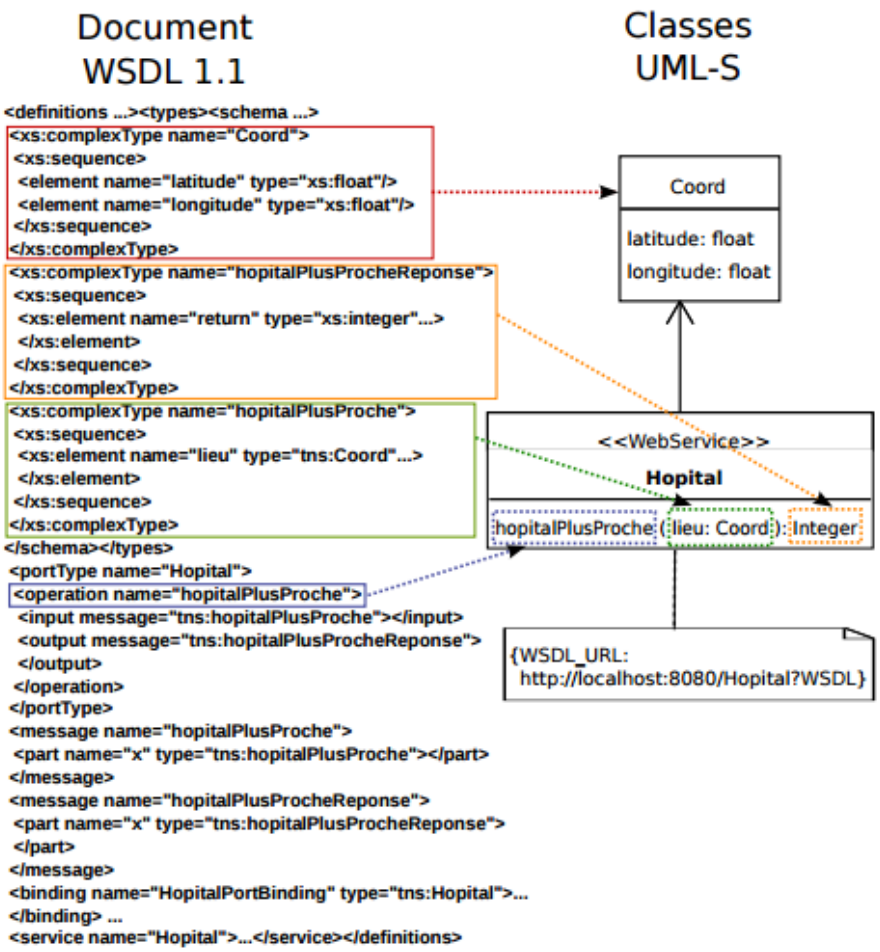

Figure 4.16 Transformation du WSDL en diagramme de classes [16]

6. Étude de cas

Les services web sont des composants conceptuellement limités à des fonctionnalités relativement simples qui sont modélisées par un ensemble d'opérations. Généralement, un seul service ne peut répondre aux besoins des clients qui sont de plus en plus complexes. Prenons l'exemple de la figure ci-dessous. Pour partir en vacance une personne doit trouver un service web pour chaque réservation qu'elle désire effectuer (réservation de train, d'avion, d'hôtel, etc.). Pour résoudre ce problème, des services Web peuvent être regroupés et interconnectés pour former un seul service du point de vue utilisateur. C'est ce qu'on appelle la composition de services web. Cette dernière permet de créer un service composite offrant une nouvelle fonctionnalité à partir des services web existants plus simples.

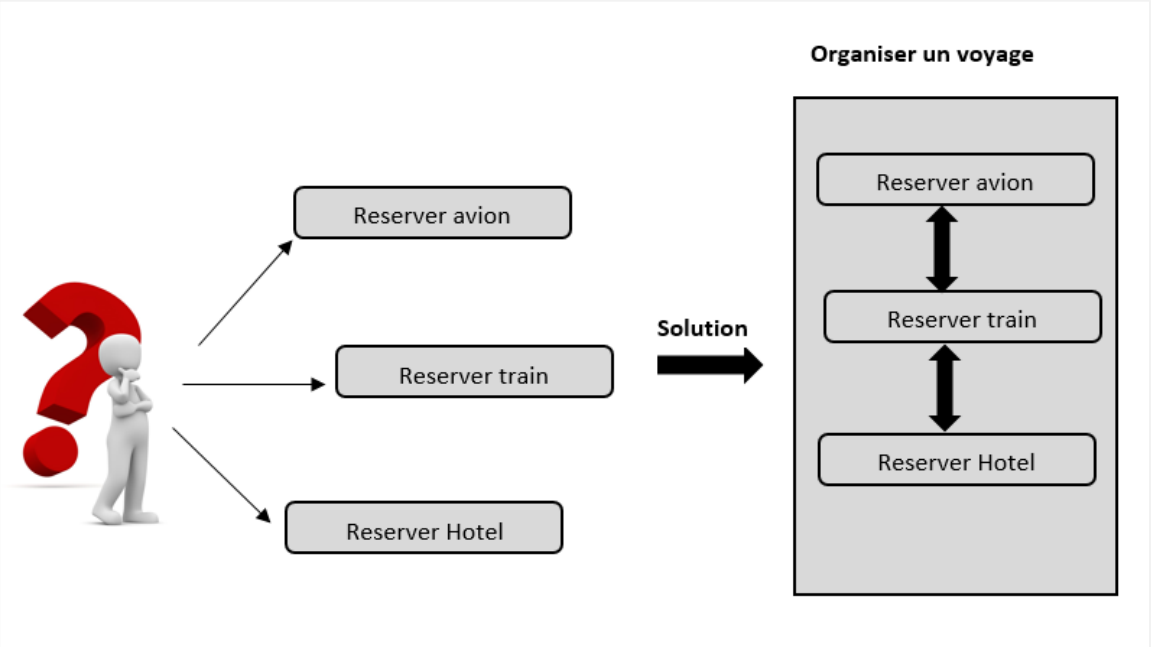

Figure 4.17 Exemple de planification d'un voyage

Nous allons modéliser la composition de trois services existants nommés **serviceHotel, serviceAvion et serviceTrain**. Le service serviceHotel fournit ici deux méthodes, **rechercherHotel ()** qui retourne les cordonnées des hôtels et **reserverHotel ()** qui retourne les détails de la réservation. Le service serviceAvion fournit aussi 2 méthodes **rechercher ()** pour la recherche des vols et **reserver**

() pour faire la réservation, la deuxième méthode retourne les détails de réservation. Le troisième service nommé serviceTrain fournit une seule méthode **reserver ()** de réservation de train.

En utilisant ces trois services, nous allons élaborer un scénario de planification d'un voyage. Nous désirons ici créer un service composé de planification de voyage que nous allons appeler **planifierVouyage** Ce service permettra aux utilisateurs de planifié un voyage en utilisant un seul service qui regroupe tous les fonctionnalités des trois autres services.

La première étape de modélisation consiste à sélectionner des services existants puis à les importer dans notre environnement de développement :

1) Sélection des services :

L'ajout des services web à l'environnement de modélisation est fait à travers leur URI, une fois l'URI de service est fourni, le fichier WSDL correspondant va être téléchargé et mis en disposition de développeur.

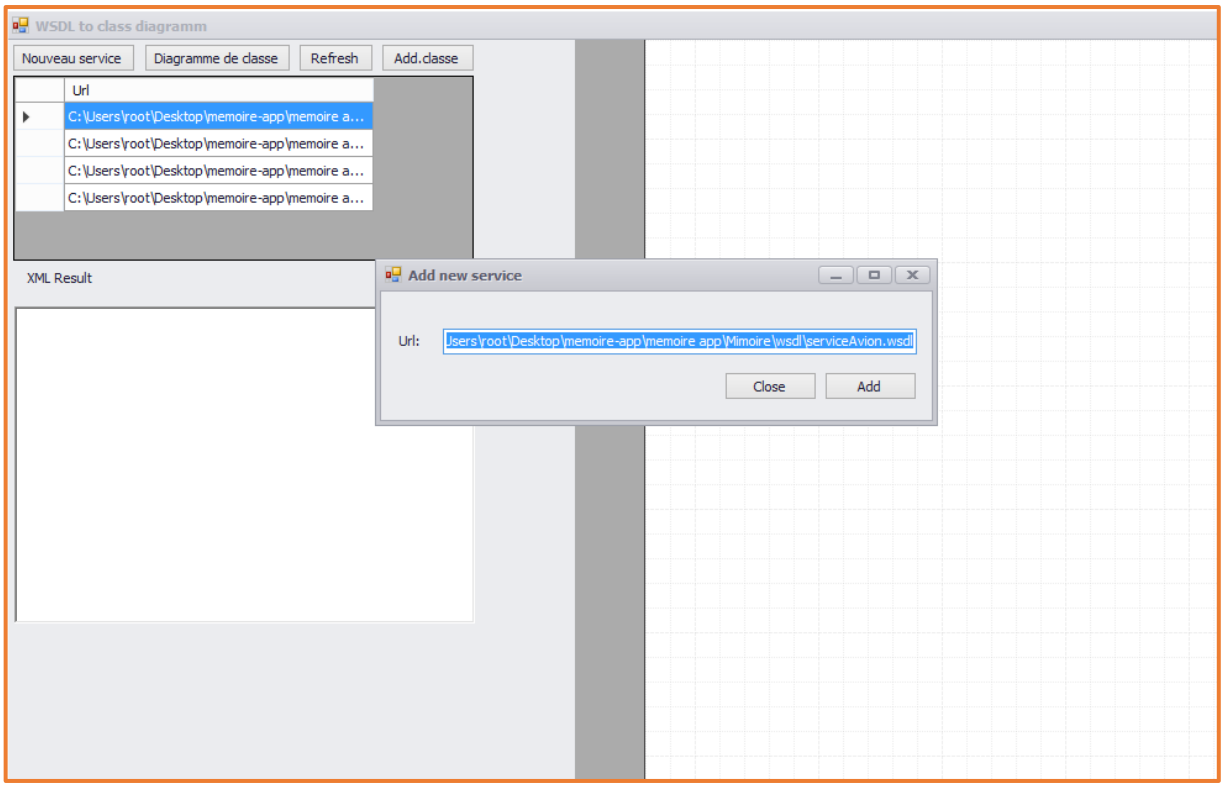

Figure 4.18 La sélection des services web

Une fois tous les services sont importer, le diagramme de classes correspondant peut être généré.

2) Génération des classes

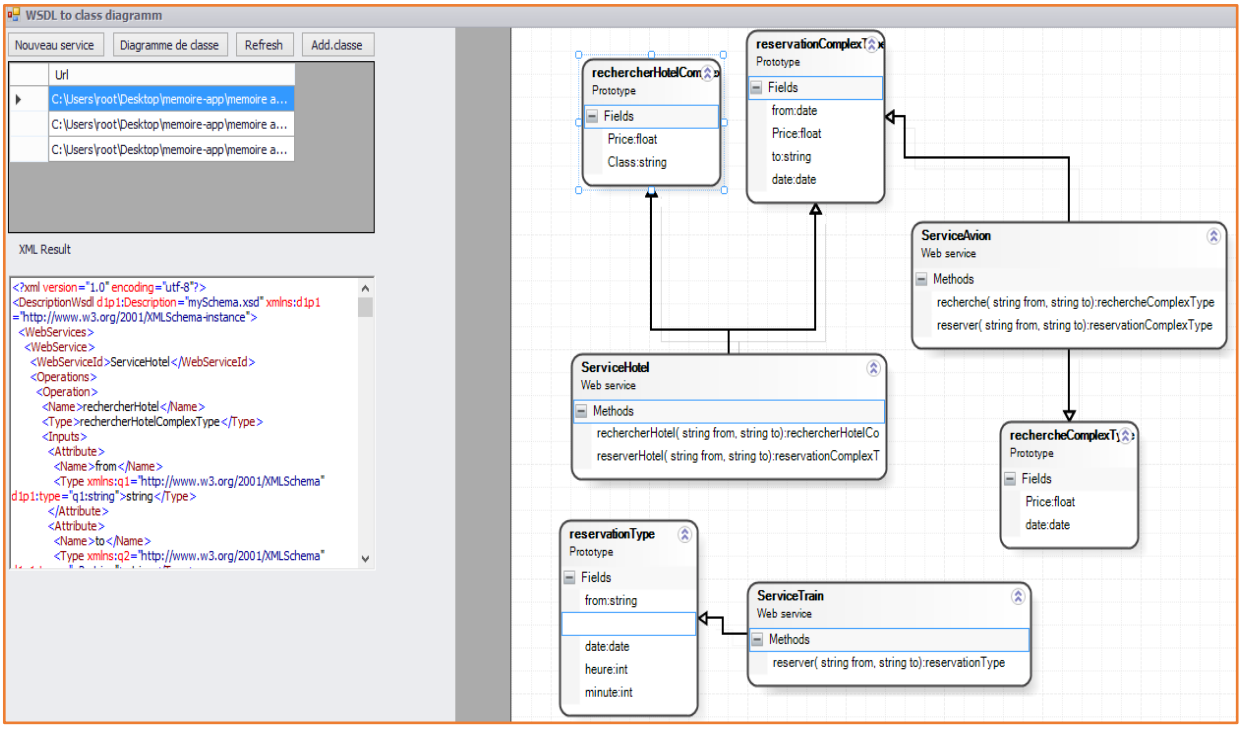

Figure 4.19 Génération de diagramme de classes

Chaque classe généré avec un stéréotype ‘’Web service’’ correspond à un service web, les classes avec le stéréotype ‘’ prototype ‘’ sont celles qui correspondent à des type de donnés complexes manipulé par les services web. Lors de l’import des services si plusieurs d’entre eux utilisent le même prototype, le développeur n’a donc pas besoin d’ajouter une nouvelle classe pour ce prototype et peut simplement ajouter une association entre le prototype existant et le nouveau service importé , dans notre cas le type de données **reservstionComplexType** est un prototype commun entre deux services (serviceAvion et serviceHotel).

3) Service composé

Une fois les services existants importés et le diagramme de classes généré, le développeur est alors en mesure de définir l’interface du nouveau service composé qu’il désire créer. Dans l’exemple considéré, ce nouveau service se nomme **planifierVouyage** et fournit une seule opération appelée **planifierVouyage ()** qui prend une destination et un prix en paramètre et retourne des détails de réservation de type **reservstionComplexType.**

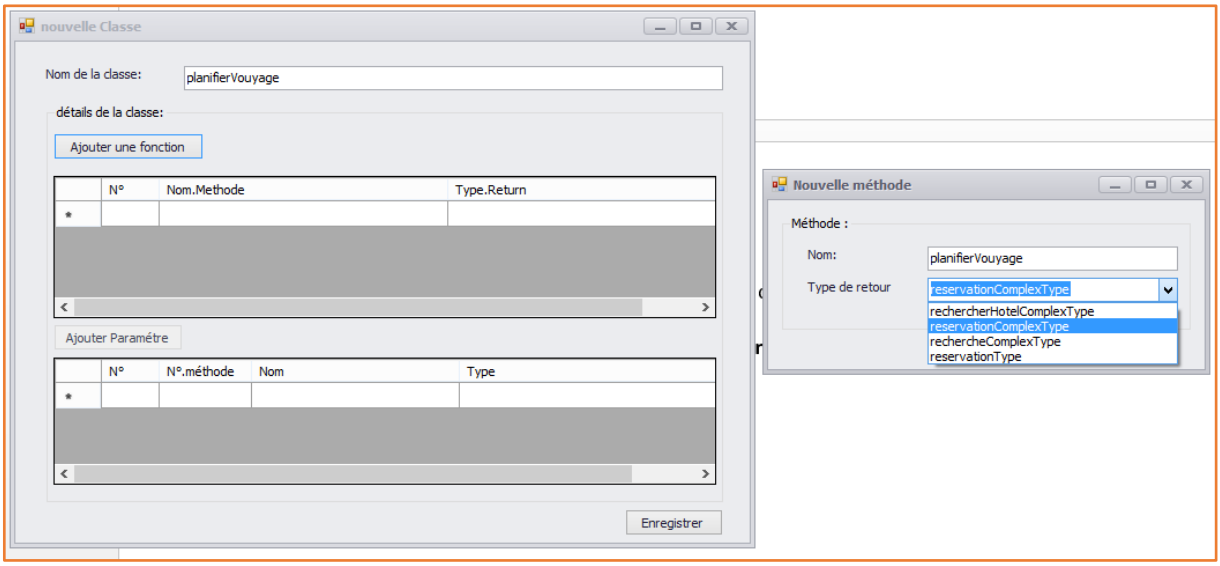

Figure 4.20 L'ajout de la classe de service composé

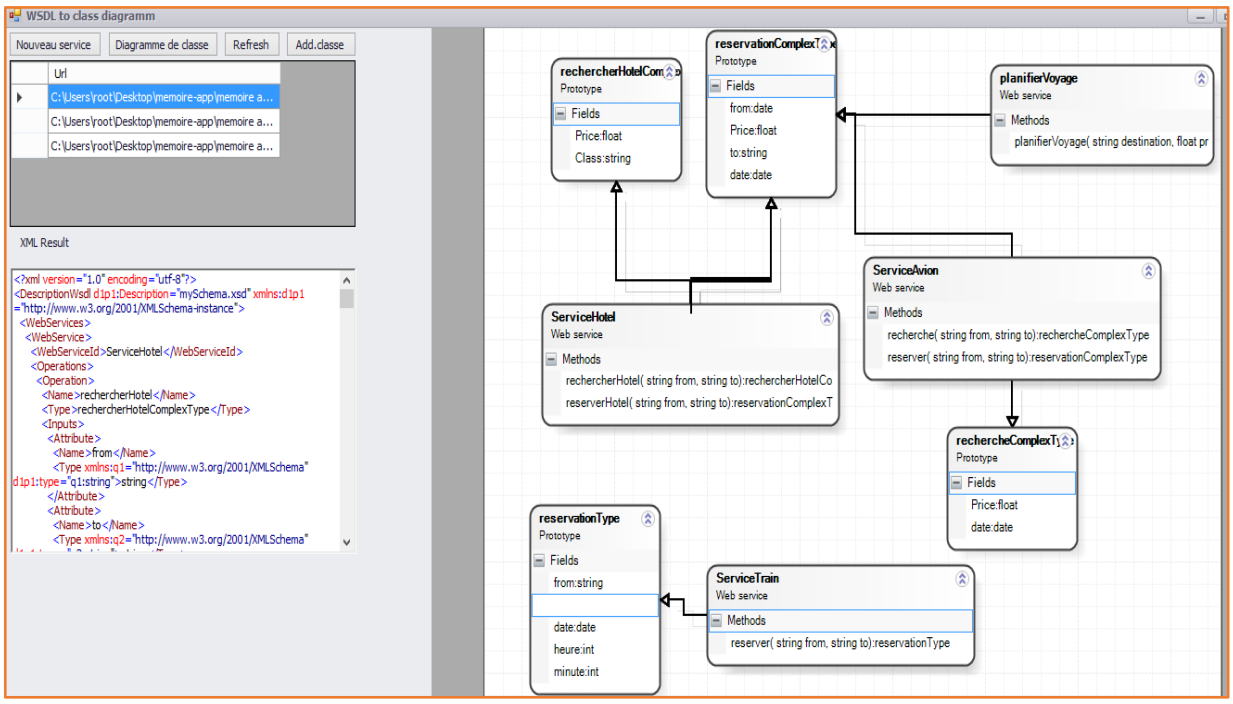

Figure 4.21 Diagramme de classes final

Comme il est possible de le constater sur la figure 4.21, le diagramme de classes UML-S fournit une représentation graphique très lisible des interfaces des services Web et des types de données qu'ils manipulent.

7. Discussion

La transformation des spécifications WDSL vers un modèle UML fonctionne pour toutes sortes de services. La nature des règles de transformation nous permet de transformer n'importe quel document WSDL. Cependant, il ne fournit pas toujours de modèles sémantiquement utiles, en raison du faible contenu sémantique de certains documents WSDL.

8. Conclusion

Dans ce chapitre, nous avons d'abord présenté les approches existantes dans la littérature, ensuit nous avons présenté les outils technologique utilisés et l'environnement de développement qui a été conçu pour l'implémentation. Nous avons également décrit la structure logique d'un fichier WSDL et les règles de passage vers le diagramme de classes équivalent. Après, nous avons traité un exemple inspiré de la vie réel pour démontrer l'efficacité et faisabilité de l'approche suivie.

Ce logiciel peut être utilisé pour importer des services Web existants et les transcrire sous la forme d'un diagramme de classes UML-S. Il permet également au développeur de modifier et de créer des modèles UML-S de manière simple et efficace.

Conclusion générale

Le travail présenté avait pour objectif le traitement du problème de la modélisation et de la composition des services web. L'approche suivie s'agit d'une approche dirigée par les modèles fidèle aux principes de MDA. Les modèles réalisés dans le cadre d'une approche MDA doivent être décrits dans un langage clair et précis afin de pouvoir aider à la compréhension du système modélisé. Nous proposons d'utiliser le langage UML 2.0. Plus précisément, nous utilisons une extension à UML 2.0 nommée UML-S grâce au mécanisme des profils. Ce profil est spécifique au contexte des services Web et de leur composition et permet ainsi d'adapter UML à ce domaine, tout en restant conforme avec son métamodèle standard. Nous utilisons le diagramme de classes afin de modéliser les aspects statiques ou structurels des services web. Le diagramme de classes permet ainsi de modéliser les interfaces des services Web, c'est à dire les opérations qu'il les mette à disposition et les types de données manipulés.

Après la sélection des descriptions WSDL intervenants dans la composition, une grammaire de manipulation directe des modèles est utilisée pour l'extraction des données des services web et donc la génération d'un diagramme de classes clair et précis qui modélise la structure statique de la composition.

Suite à ce travail, les perspectives qui peuvent être envisagées sont les suivantes :

La première perspective réside dans l'étape de la sélection. Dans ce travail, nous considérons uniquement la composition statique de services. Cela signifie que les services sont sélectionnés dès l'étape de spécification et que le scénario de composition est également prédéterminé. Une sélection dynamique(ou en temps réel) des services web peut être une amélioration considérable, car elle fournit un moyen de flexibilité. Dans une composition dite dynamique, seul le scénario de composition est prédéfini au moment de la spécification, les services sont quant à eux sélectionnés dynamiquement au moment de l'exécution à travers des mécanismes de découverte.

Une autre perspective réside au niveau de la méthode choisie pour composer les services Web. Dans notre travail nous avons modélisé une composition de type orchestration. Cela signifie qu'un nouveau service dit composé résulte de cette composition. Ce nouveau service va agir tel un "chef d'orchestre" pour diriger et contrôler la composition des services sélectionnés. Il serait intéressant d'étendre notre approche afin de permettre la composition de services en réalisant un comportement de chorégraphie.

Bibliographie

Bibliographie

[1] "SOA design principales for dummies ". Par Claus T.Jensen, publié par john Wiley, 2013.

[2] M. DODANI, 'From objects to services: A journey in search of component reuse nirvana', Journal of Object Technology, Vol. 3, No. 8, pp. 49-54, 2004.

[3] BACHTARZI Fayçal, '' une approche de composition des services web basé sur la transformation des graphes '', université costantine 2 abdelahamid mehri, 2014.

[4] DAHA Hanane et Melle LIFA Siham, '' Une approche formelle pour la composition des services web '', UNIVERSITÉ ECHAHID HAMMA LAKHDAR - EL-OUED, 2015.

[5] article par Swati Takale ,15 avril 2016, publié par buzzle.com,

[Online] http://www.buzzle.com/articles/advantages-and-disadvantages-of-service-oriented-architecture-soa.html

[6] M. DRISS, "Approche multi-perspective centrée exigences de composition de services Web", Université Rennes1, Dec 2011.

[7] IBM site officiel, [online] https://www.ibm.com/developerworks/webservices/standards/

[8] W3C site official,

https://www.w3.org/TR/ws-arch/

[9] Webopedia.com

http://www.webopedia.com/TERM/W/Web_Services.html

[10] CHEMMA Sofiane. Une approche de composition de services Web à l'aide des Réseaux de Petri orientés objet, UNIVERSITÉ ABDELHAMID MEHRI, CONSTANTINE 2, 2014.

[11] Mohamed GHARZOULI. '' Composition des Web Services Sémantiques dans les systèmes Peer-to-Peer '', article, Université de Bejaïa, 2011.

[12] T. MELLITI, " Interopérabilité des services Web complexes, Application aux systèmes multi-agents (in french)", PhD thesis, Department of Computer Science, University of Paris IX Dauphine, 2004.

[13] D. C. Fallside and P. Walmsley, "XML Schema Part0: Primer Second Edition", W3C Recommendation, 28 Oct 2004, [Online]. Available: http://www.w3c.org/TR/xmlschema-0/ .

[14] M. Kay, "XSL Transformations (XSLT)", version 2.0, W3C Recommendation, 23 Jan 2007, [Online]. Available: http://www.w3.org/TR/xslt20/ .

[15] J. Robie, D. Chamberlin, M. Dyck and J. Snelson, "XML Path Language (XPath)", version 3.0, W3C Recommendation, 08 Apr 2014, [Online]. Available : http://www.w3.org/TR/xpath-30/ .

[16] C. Dumez, Approche dirigée par les modèles pour la spécification, la Vérification formelle et la mise en œuvre de services Web composés, thèse de doctorat, Université de Technologie de Belfort-Montbéliard, 2010.

[17] Mohamed GHARZOULI, ''Composition des Web Services Sémantiques dans les systèmes Peer-to-Peer'', Thése de doctorat, Université de Bejaïa, 2011.

[18] Uddi.org "Introduction to UDDI: Important Features and Functional Concept" http://www.uddi.org/pubs/uddi-tech-wp.pdf

[19] S. Dustdar and W. Schreiner, "A survey on web services composition", International Journal of Web and Grid Services, vol. 1, number. One, Aug 2005.

[20] Mike Rosen, ''Orchestration or Choreography? '' Avril 2008.

http://www.bptrends.com/publicationfiles/04-08-COL-BPMandSOA-OrchestrationorChoreography-%200804-Rosen%20v01%20_MR_final.doc.pdf

[21] Hajar Omrana. '' Vers une Composition Dynamique des Services Web : une approche de Compossibilité Offline ''. Web. UNIVERSITE MOHAMMED V AGDAL – ECOLE MOHAMMADIA ´D'INGENIEURS, 2014.

[22] Razika DRIOUCHE, '' Proposition d'une architecture d'intégration des applications d'entreprise basée sur l'interopérabilité sémantique de l'EbXML et la mobilité des agents '', thèse doctorat, UNIVERSITE MENTOURI DE CONSTANTINE, 2007.

[23] Nerea Arenaza, '' Composition semi-automatique de Services Web '', Projet de Master, école polytechnique fédérale de Lausanne Février 2006.

[24] Wael SELLAMI, '' Une approche formelle pour la vérification des propriétés non fonctionnelles d'orchestration des services web '', Mémoire de Master, Université de Sfax, 2010-2011.

[25] Douglas K. Barry with David Dick "Web Services, Service-Oriented Architectures, and Cloud Computing the Savvy Manager's Guide Second Edition", 2013.

[26] OASIS Web Services Reliable Messaging (WSRM) TC, [Online] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrm

[27] https://fr.wikipedia.org/wiki/UML_(informatique)

[28] CHEMAME Chaouki, '' Implémentation d'une infrastructure collaborative asynchrone basée sur les services web '', Mémoire de Master, 2010.

[29] Wafaa AIT-CHEIK-BIHI, '' Approche orientée modèles pour la vérification et l'évaluation des performances de l'interopérabilité et l'interaction des services '', Soutenue le 21 Juin 201 L'UNIVERSITE DE TECHNOLOGIE DE BELFORT-MONTBELIARD 2

[30] Mouna BOUARIOUA, '' Une approche basée transformation de graphes pour la génération de Modèles de réseaux de Petri analysables à partir de diagrammes UML '', UNIVERSITÉ CONSTANTINE 2 ,2013 .

[31] BENOÎT COMBEMALE, '' Approche de méta-modélisation pour la simulation et la vérification de modèle Application à l'ingénierie des procédés '', le 11 juillet 2008 à l'ENSEEIHT, Toulouse.

[32] J. M. Favre,"Towards a Basic Theory to Model Driven Engineering", UML 2004 – Workshop in Software Model Engineering (WISME 2004), 2004.

[33] BEKERROU Makhlouf, '' Vers la génération de modèles de simulation '', université de Bretagne occidentale, 2010.

[34] Hachichi Hiba, ‘’ support de cour IDM ‘’, université Djilali bounaama ,2016.

[35] Object Management Group OMG. “Unified modeling language 2.2 specification”, [Online]. Available: http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/ .

[36] OMG, “MetaObject Facility (MOF)”, version 2.0, [Online]. Available : http://www.omg.org/mof/ .

[37] OMG, “Object Constraint Language (OCL)”, version 2.0, [Online]. Available : http://www.omg.org/ocl/ .

[38] OMG, “XML Metadata Interchange (XMI)”, version 2.0, [Online]. Available: http://www.omg.org/xml/ .

[39] OMG, “Common Warehouse Metamodel (CWM)”, version 1.1. [Online]. Available: http://www.omg. org/technology/documents/formal/cwm.html

[40] OMG, “MOFM2T Model-to-Text language”, [Online]. Available: http://www.omg.org/spec/MOFM2T/1.0/ .

[41] OMG, ‘’ MDA - The Architecture of Choice for A Changing World", [Online]. Available : http://www.omg.org/mda/ .

[42] Jean Bézivin. ‘’ La transformation de modèles. Cours 6 ‘’, INRIA-ATLAS & Université de Nantes, 2003.

[43] OMG, ‘’ MDA Guide ‘’, Version 1.0, 2003, ”, [Online]. Available : http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf .

[44] Krzysztof Czarnecki and Simon Helsen, "Feature-based survey of model transformation approaches", IBM Syst. J., 45(3):621–645, ISSN 0018-8670, 2006.

[45] OMG, ‘’ MDA Guide ‘’, Version 1.0.1, 2003, ”, [Online]. Available :

http://www.omg.org/news/meetings/workshops/UML_2003_Manual/002MDA_Guide_v1.0.1.pdf

[46] Laurent AUDIBERT, "UML 2 - de l'apprentissage à la pratique", janvier 2009, [Online]. Available :

http://laurent-audibert.developpez.com/Cours-UML/?page=introduction-modelisation-objet#L1-4

[47] Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno, “An Introduction to UML Profiles”, the European Journal for the Informatics Professional, Vol. V, No. 2, April 2004 [Online]. Available:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.120.5306&rep=rep1&type=pdf

[48] Roy Gronmo, ‘’ Model-driven Web Service Development ‘’, International Journal of Web Services Research, 1(4), Oct-Dec 2004.

[49] DJAMEL AMAR BENSABER, ‘’ Bringing Semantics to Web Services: Model driven approach ‘’, Université de Sidi Bel Abbes, 2008.

[50] Weijun Sun, ‘’ A Model-driven Reverse Engineering Approach for Semantic Web Services Composition ‘’, e, SUN YAT-SEN University Guangzhou, 510275, China, 2009.

[51] MELLAHI Mustapha et AMRANI Halim, ‘’ Conception et réalisation d’une application web pour la gestion et la planification des séances de télémédecine ‘’ , Université SAAD DAHLEB, Blida, 2008/2009.

[52] Site officiel de Microsoft, “ Langage Visual C# ”,2003

https://msdn.microsoft.com/fr-fr/library/aa287558(v=vs.71).aspx

[53] Site officiel de Microsoft, “Windows Forms Overview”, 2017,

https://msdn.microsoft.com/en-us/library/8bxxy49h(v=vs.110).aspx

[54] Site officiel de Microsoft, ‘’System.Web.Services.Description Namespace’’,

https://msdn.microsoft.com/en-us/library/system.web.services.description(v=vs.110).aspx

[55] Site officiel de Microsoft, ‘’ServiceDescription Class’’,

http://msdn.microsoft.com/en-us/library/system.servicemodel.description.servicedescription(v=vs.110).aspx