الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université de 8 Mai 1945 – Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Ingénierie des Medias

Thème :

Migration d'une application orientée objet vers une architecture orientée service

---

Encadré Par :

Mr Seridi Ali

Présenté par :

Benzeltout Adam

Yahyaui Walid

Juin 2012

Remerciements

Tout d'abord, nous tenons à exprimer nos plus vifs remerciements et nos gratitude à notre directeurs de thèse, M.Seridi Ali, pour son encadrements continus, pour les remarques constructives qu'il nous 'a fournies ainsi que pour leurs précieux conseils durant toute la période de notre travail.

Nous les remercie également pour la confiance qu'il nous a accordée et pour la grande liberté d'idées et de travail qu'il nous 'a donnée

En dehors de son apports scientifiques, on n'oubliera pas aussi de le remercier pour sa qualités humaines, sa hospitalité et son soutien qui nous a permis de mener à bien notre thèse de Master.

Notre reconnaissance va à nos ami Behnous et Brekhta pour leur disponibilité et leurs connaissances dans de nombreux domaines. ils ont ainsi largement contribué au bon déroulement de nos travaux, et ont ensuite été présents au quotidien pour nous 'aider à surmonter les diverses difficultés, nous 'encourager, et nous prodiguer de bons conseils.

nous remercions aussi nos parents, pour leurs patience et leurs encouragement, leurs patience et leurs amour

nous remercions nos collègues du département d'informatique qui ont su rendre notre travail agréable, par leur simple présence et l'ambiance qu'ils ont su créer.

Merci à tous ceux qui y ont cru à nous et m'ont soutenu.

Benzeltout Adam

Yahyaui walid

Dédicace

A la mémoire de mes grands-parents.

A mes très chers parents qui ont toujours été là pour moi, et qui m'ont donné un magnifique modèle de labeur et de persévérance. J'espère qu'ils trouveront dans ce travail toute ma reconnaissance et tout mon amour,

C'est grâce à leur soutien, leur patience et leur amour que je suis là aujourd'hui. Je leur suis très reconnaissant pour les sacrifices qu'ils ont dû faire pendants mes longues années d'études et d'absence

A ma chère sœur : Nadia

A mes tantes et à mes oncles.

A chaque cousins et cousines.

A mes meilleurs amis : Fouad, Ahmed, Walid, Brahim , Amira et Maria

A mes collegues :Houssem ,Rachide,Amine , Majda et Meriem

Je dédie ce mémoire

Aux deux personnes que j'ai tant aimé qu'elles assistent à ma soutenance :

Behnous Atman et Brakhta zringo.

Benzeltout Adam

Dédicace

A Dieu Le Tout Miséricordieux, ton amour, ta miséricorde et Tes grâces à mon endroit m'ont fortifiée dans la persévérance et l'ardeur au travail.

Je dédie ce modeste travail et ma profonde gratitude à celle qu a attendu avec patience les fruits de sa bonne éducation...

A ma Mère.

A celui qui m'a indique la bonne voie en me rappelant que la volonté fait toujours les grands hommes...

A mon Père

pour l'éducation qu'ils m'ont prodigué

Avec tous les moyens

Et au prix de tous les sacrifices qu'ils ont consentis à mon égard,

Pour le sens du devoir qu'ils mon enseigné depuis mon enfance.

A mes chères sœurs : Sabrina ,Nihad,Warda

A mon chère frère : Haroun

A mes tantes et à mes oncles.

A chaque cousins et cousines.

A mes meilleurs amis : Adam, Athmen, Riad(Zringou), Anouar, Imad,

A mes collègues :Houssem ,Rachide,Amine , Azzedine ,et tout la section

Merci pour tout votre amour et votre confiance, pour m'avoir aidé à ranger mon éternel désordre et pour votre énorme support pendant la rédaction de mon projet (oui c'est toi CH)

Yahiaoui Walid

Résumé

La variabilité importante des demandes entreprises avec l'utilisation des systèmes très anciens a poussé vers la naissance d'un domaine de recherche très vaste qui consiste à migrer d'un système patrimonial vers un nouveau système a base de service.

Cette technique de migration englobe le redéveloppement et l'ingénieuring telque le wraping et l'intégration

Dans ce contexte, l'objectif de ce mémoire est de proposer une approche d'évolution de puis Légacy système vers une architecture orienté service.

L'architecture orienté service est considéré comme une opportunité pour un grand gain dans le domaine économique, avec les nouveaux concepts, la communication entre les services porte sur une grande agilité, vu la grande utilité de l'architecture orienté service la naissance de la technologie web service à faciliter la communication.

Notre étude s'appuie sur la collaboration de plusieurs classes orientées objets dont le but de les rendre un ensemble des services, en se basons sur deux technique déjà proposé qui sont le redéveloppement et le wrapping.

Alors nous proposons un outil de transformation (boite noire) qui prend comme entré une application orienté objet (oo) avec plusieurs classes qui communique entre eux, en reposant sur les deux technique vus précédemment nous aurons comme sortie ensembles de service

Finalement par la réalisation de cet outil nous avons essayé de faire un travail modeste pour satisfaire les besoins.

MOT CLE

Architecture orienté service, Web Services, legacy systém , migration, Service, Redéveloppement , wrapping , ingénieuring , intégration

Listes des figures

Introduction Générale

La dernière décennie a été marquée par le développement rapide des systèmes d'information distribués, et tout particulièrement par la disponibilité de l'accès à l'Internet. Cette évolution du monde informatique a entrainé le développement des nouvelles méthodologies d'interaction entre les applications distantes, avec la considération du problème d'hétérogénéité entre ces applications, la multitude des systèmes d'exploitation et la multitude des langages de programmation.

La tendance est vers l'utilisation des architectures orientées service SOA (Service Oriented Architecture). SOA est une approche architecturale permettant la création des systèmes basés sur une collection de services qui sont des nouvelles technologies permettent l'interaction entre les applications hétérogènes; ils permettent à des applications de dialoguer à distance via internet indépendamment des plates-formes et des langages sur lesquelles elles reposent. Ils peuvent interagir de manière intelligente tout en étant capable de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes (adaptées la notion de coordination et collaboration).

La possibilité d'intégrer ou de composer des services existants en nouveaux services est la plus importante fonctionnalité assurée par les architectures orientées services. La composition de services doit permettre de créer, d'exécuter, de maintenir des services qui reposent sur d'autres services et ceci de façon efficace.

L'architecture orientée service (AOS) a connu ces dernières années un grand engouement des entreprises de tout secteur et de toute taille en raison de ses avantages économiques et technologiques. Pour exploiter ces avantages, plusieurs organisations ont décidé de faire évoluer leurs systèmes patrimoniaux (SP) existants vers une telle architecture. La Migration vers la AOS est devenue l'une des techniques importantes de modernisation des SP. Elle aide les organisations d'une part à réutiliser leurs anciens systèmes existants en leurs donnant une nouvelle vie, et d'autre part à profiter des avantages des systèmes à base de service.**[07]**

Parmi les systèmes qui sont concernés pars la modernisation vers une architecture orienté service, c'est les systèmes orientée objet.

C'est dans ce cadre qu'intervient notre travail. Il s'agit de réaliser un outil qui permet de transformer des d'applications orientées objets en d'autres applications orientées services.

L'outil prend en entrée une application orienté objet constitué d'un ensemble de classes Java et génère en sortie une nouvelle application orientée service constitué d'un ensemble de services Web interconnecté entre eux.

A travers ce travail, nous visons à atteindre les objectifs principaux suivants :

- ✓ Étudier les différents concepts nécessaires à la mise en œuvre d'une application orientée services.
- ✓ Proposer une approche de migration d'une application orientée objet vers une nouvelle orientée service.

Notre mémoire s'article sur **05** chapitres :

- **Le 1er chapitre :** Introduit le domaine étudié (SOA) et présente les concepts de base relatifs à ce domaine. Il tente de donner un aperçu global et précis sur tout ce qui concerne les caractéristiques, les objectifs, l'architecture, ainsi que les différents standards utilisés pour la SOA.

- **Le 2éme chapitre :** a pour objectif de présenter le concept de migration et identification des services.

- **Le 3éme chapitre :** Présente les services web et les différents standards qui lui sont reliés. Ce dernier représente la technologie la plus mature qui implémente la SOA

- **Le 4éme chapitre** présente la conception générale puis détaillée de l'application. Il décrit l'étude conceptuelle de notre outil ainsi que le processus de création de services web. Nous commencerons par présenter l'architecture de l'outil. Ensuite, nous détaillerons chaque module dans cette architecture.

- **Le 5éme chapitre** présente les outils utilisés pour le développement et une étude de cas.

Nous terminerons ce mémoire par une conclusion générale et quelques perspectives.

I. Architecture SOA

1. Introduction

L'architecture SOA (Services Oriented Architecture) est de plus en plus utilisée dans les entreprises. Cette Architecture Orientée Service apporte beaucoup de nouveautés au monde des systèmes d'information et à l'informatique en général. D'ici fin 2012, 70 % des entreprises opéreront leurs applications métiers par le biais d'une architecture SOA et le marché mondial des Web Services (une implémentation de SOA) est passé de 950 millions de dollars en 2004 à 6,2 milliards en 2008 **[01]**.

Depuis des années, de multiples études pour adapter la SOA ont été menées dans toutes les entreprises. En effet, cette architecture permet une refonte complète tout en gardant des briques existantes mais peut très bien être instaurée de manière incrémentale. Les entreprises, dont les banques, qui n'ont, bien sûr, pas envie de repartir de zéro avec leur systèmes d'information, peuvent ainsi progressivement utiliser une architecture SOA de plus en plus complète.

Depuis le début, le terme SOA est évoqué mais la traduction de cet acronyme « Service-Oriented Architecture » par Architecture Orientée Service ne permet pas de comprendre exactement ce qu'il signifie. Une définition simple pourrait être la notion d'intégrer et de manipuler les différents composants d'un système informatique en tant qu'ensembles fonctionnels appelés services **[02]**.

Cette architecture à la mode répond aux problèmes de réutilisation d'outils (ou produits) des entreprises. Pour mieux comprendre sa définition, il faut voir cette architecture comme une philosophie. C'est une approche permettant de réutiliser et d'organiser des ressources existantes, dans une solution autorisant une interopérabilité entre plateformes et environnements, une évolutivité des modules applicatifs et une flexibilité autorisant l'utilisation dynamique d'applications.

Cette solution permet donc d'intégrer divers systèmes : chaque ressource peut être accessible en tant que service possédant une interface. L'implémentation du fournisseur de service est donc libre de changer sans qu'il y ait un impact sur son utilisation. On peut voir ce service comme une boîte noire : on sait qu'elle va rendre le service voulu sans savoir comment est faite la boite noire. On peut choisir de la remplacer par un autre service implémenté différemment mais répondant aussi à la même fonctionnalité.

2. Définition de la SOA

Il existe plusieurs façons de percevoir et de définir une Architecture Orientée Services. La plupart de ces définitions se concentrent sur l'aspect technique de la SOA, quoique d'autres présentent des caractéristiques métiers. Voici une liste (non exhaustive) de définitions proposées et issues de plusieurs sources. Ces définitions sont intéressantes puisqu'elles illustrent plusieurs points de vue sur la SOA. Nous allons commencer par une définition très courte qui est celle du W3C :

"SOA is a set of components which can be invoked, and whose interface descriptions can be published and discovered." **[03].**

Traduction en français :

"SOA est un ensemble de composants qui peuvent être invoqués, et dont l'interface descriptions peuvent être publié et a découvert "

Cette définition du W3C présente avec une manière très simpliste ce qui peut être fait avec un service ou avec sa description. Elle ne s'est pas occupée de la notion d'architecture ni de la manière avec laquelle le service peut être conçu.

Une définition technique a été présentée dans **[04]**

"A Service-Oriented Architecture (SOA) is a software architecture that is based on the key concepts of service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation."

Traduction en français :

"Une Architecture Orientée Services (SOA) est une architecture logicielle qui est basé sur les concepts clés de service, référentiel de services, et le bus de service. Un service se compose d'un contrat, une ou plusieurs interfaces, et une mise en œuvre."

Les auteurs mettent le point sur l'aspect technique de la SOA qui consiste en une application frontale qui utilise un ou plusieurs services. Ces services sont publiés dans des registres de services et la communication entre ces services est assurée par un bus de service.

D'un point de vue métier (Marks, 2006) présente la SOA comme suit **[05]:**

"SOA is a conceptual business architecture where business functionality, or application logic, is made available to SOA users, or consumers, as shared, reusable services on an IT network."Services" in an SOA are modules of business or application functionality with exposed interfaces, and are invoked by messages."

Traduction en français :

"SOA est une architecture d'entreprise où la logique conceptuelle des fonctionnalités métier, ou l'application, est mis à la disposition des utilisateurs SOA, ou les consommateurs, comme partag

services réutilisables sur un IT réseau. "Services" dans une architecture SOA sont des modules d'affaires ou de fonctionnalité de l'application avec des interfaces exposées, et sont invoquées par les messages. "

Cette définition prête une grande attention à l'orientation métier dans la SOA de manière que nous pouvons avoir l'impression que la SOA est un concept purement métier et que le niveau technique n'existe que pour le maintien des réseaux et la communication entre les services.

3. L'architecture Orienté Service s'est Quoi et Pourquoi

L'architecture orientée services (Service Oriented Architecture ou SOA) est un modèle d'interaction applicative qui met en œuvre des services : avec une forte cohérence interne (par l'utilisation d'un format d'échange pivot, le plus souvent XML), et des couplages externes « lâches » (par l'utilisation d'une couche d'interface interopérable, le plus souvent un service web WS).

Le service est une action exécutée par un « fournisseur » (ou « producteur ») à l'attention d'un « client » (ou « consommateur »), cependant l'interaction entre consommateur et producteur est faite par le biais d'un médiateur (qui peut être un bus) responsable de la mise en relation des composants logiciels implémentant les services. Le service étant à grandes mailles, il englobe et propose les fonctionnalités des composants du système. Ces systèmes peuvent aussi être définis comme des couches applicatives.

L'architecture orientée services est une réponse très efficace aux problématiques que rencontrent les entreprises en termes de réutilisabilité, d'interopérabilité et de réduction de couplage entre les différents systèmes qui implémentent leurs systèmes d'information. Les architectures SOA ont été popularisées avec l'apparition de standards comme les Services Web dans l'e-commerce (commerce électronique) (B2B, inter-entreprise, ou B2C, d'entreprise à consommateur), basés sur des plates-formes comme J2EE ou .NET et la déclinaison libre Mono de cette dernière. Elles mettent en pratique une partie des principes d'urbanisation.

Au sein de l'architecture orientée services, on distingue les notions d'annuaire, de bus, de contrat et de service, ce dernier étant le noyau et le point central d'une architecture orientée services. La déclinaison ou plus précisément l'implémentation de la SOA avec des WebService est la WSOA (WebService Oriented Architecture).

L'architecture orientée services est une technologie qui doit permettre aux applications la mettant en œuvre un découpage modulaire de leur architecture. Les fonctionnalités nécessaires à ce concept architectural sont apportées par une plateforme sur laquelle sont exécutés les logiciels qui les

utilisent. L'emploi d'une architecture orientée services permet la réutilisation des briques logicielles et leur partage entre les diverses applications présentes sur la plate-forme. La technologie SOA présente l'avantage de permettre un découpage du travail entre les différents programmeurs d'une application. Elle facilite aussi son évolution et sa maintenance après sa mise en production.

4. Objectif de la SOA

Une architecture orientée services (notée SOA pour Services Oriented Architecture) est une architecture logicielle s'appuyant sur un ensemble de services simples.

L'objectif d'une architecture orientée services est donc de décomposer une fonctionnalité en un ensemble de fonctions basiques, appelées services, fournies par des composants et de décrire finement le schéma d'interaction entre ces services.

L'idée sous-jacente est de cesser de construire la vie de l'entreprise autour d'applications pour faire en sorte de construire une architecture logicielle globale décomposées en services correspondant aux processus métiers de l'entreprise.

Lorsque l'architecture SOA s'appuie sur des web services, on parle alors de WSOA, pour Web Services Oriented Architecture).

5. Principes généraux d'une architecture orientée service

Il n'existe pas à proprement parlé de spécifications officielles d'une architecture SOA, néanmoins les principales notions fédératrices que l'on retrouve dans une telle architecture sont les suivantes :

La notion de service, c'est-à-dire une fonction encapsulée dans un composant que l'on peut interroger à l'aide d'une requête composée d'un ou plusieurs paramètres et fournissant une ou plusicurs réponses. Idéalement chaque service doit être indépendant des autres afin de garantir sa réutilisabilité et son interopérabilité.

La description du service, consistant à décrire les paramètres d'entrée du service et le format et le type des données retournées.Le principal format de description de services est WSDL (Web Services Description Language), normalisé par le W3C.

La publication (en anglais advertising) et la découverte (discovery) des services.La publication consiste à publier dans un registre (en anglais registry ou repository) les services disponibles aux utilisateurs, tandis que la notionde découverte recouvre la possibilité de rechercher un service parmi

ceux qui ont été publiés. Le principal standard utilisé est UDDI(Universal Description Discovery and Integration), normalisé par l'OASIS.

L'invocation, représentant la connexion et l'interaction du client avec le service. Le principal protocole utilisé pour l'invocation de services est SOAP(Simple Object Access Protocol).

6 . Avantages d'une architecture orientée service

Une architecture orientée services permet d'obtenir tous les avantages d'une architecture client-serveur et notamment :

- Une modularité permettant de remplacer facilement un composant (service) par un autre
- Une réutilisabilité possible des composants (par opposition à une système tout-en-un fait sur mesure pour une organisation).
- De meilleures possibilités d'évolution (il suffit de faire évoluer un service ou d'ajouter un nouveau service)
- Une plus grande tolérance aux pannes
- Une maintenance facilitée
- offrir la possibilité de rendre les services consommables à travers différents canaux
- La capacité à progressivement changer le système : Commutation des prestataires de services, l'extension des services, en modifiant les fournisseurs de services et les consommateurs Tous ces éléments peuvent être faits en toute sécurité, grâce au couplement bien contrôlé.

SOA donne aux clients le pouvoir à utiliser les meilleurs composants qui ne les enferment dans une solution fournisseur unique. SOA assure le support convenable pour le travail en offrant le choix pour

- Base de données (Microsoft SQL, Oracle, Informix, DB2)
- Système d'exploitation (Windows, Linux, AIX, Solaris)
- Platform de développement (.NET and Java

7. Les Composant De SOA

L'architecture orientée services est une méthode de construction d'applications qui utilisent des services communs aux fonctions de support des entreprises.

SOA est un concept architectural, cette architecture SOA est caractérisée par un ensemble de composants à la fois obligatoires et optionnels. Une solution SOA se compose des trois principaux composants logiques:

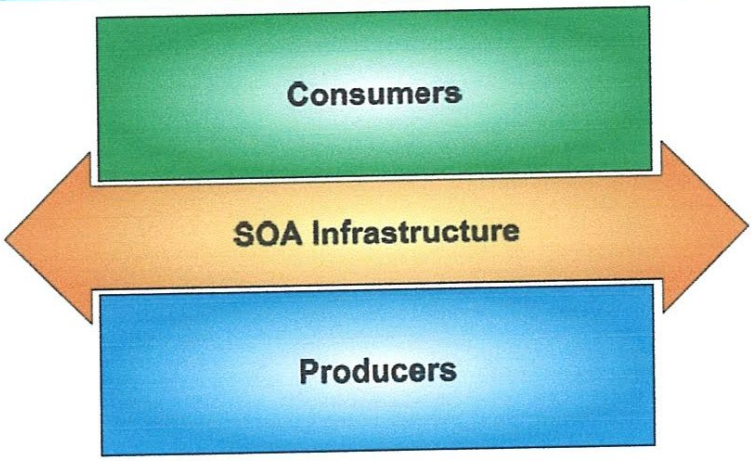

Figure 1.1: les trois composantes de SOA

La couche de l'infrastructure SOA est divisée en trois sous composants :

- Application
- Service
- Support de service

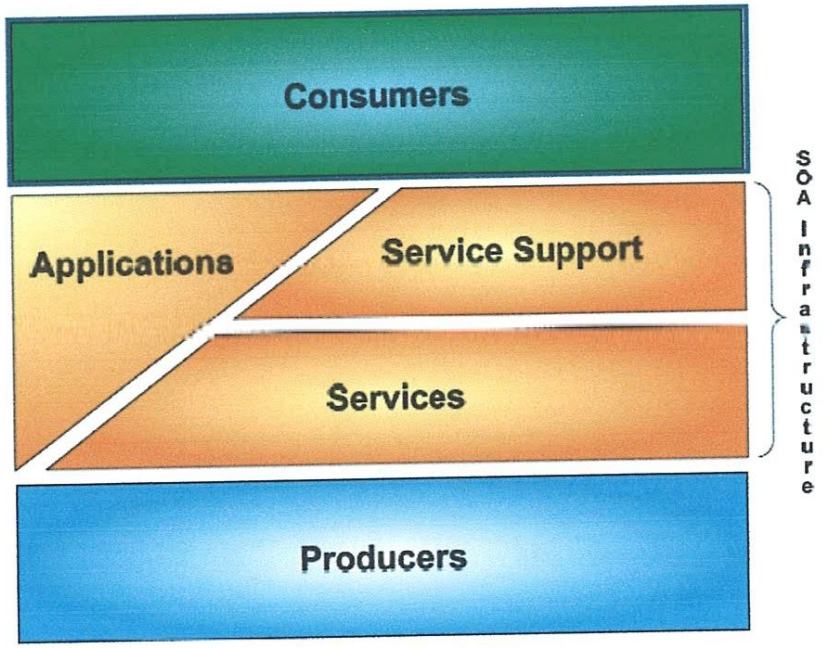

Figure 1.2 : SOA infrastructure

Les 5 composants principaux de SOA sont définit comme suit :

-Les consommateurs : Une entité qui utilise le service offert par le producteur

-Applications : Fournir une interface graphique et des degrés divers du logique métier étroitement couplé pour que les consommateurs exécutent leurs tâches

-Services : une entité qui effectue une tâche spécifique lorsqu'il est invoqué

-Support de service : une entité qui fournit les fonctions de support de fond pour SOA

-Les producteurs : une entité qui offre un service spécifique ou fonctionnalité

8. Fonctionnement de SOA

Le service est le composant clé de l'architecture, il fournit une fonctionnalité bien définie. C'est aussi un composant autonome qui ne dépend d'aucun contexte ou service externe. Les services sont enregistrés par des composants dits fournisseurs dans un annuaire fourni par l'architecture. Les services référencés sont alors mis à la disposition de consommateurs ou clients. La figure ci-dessous montre qu'un composant souhaitant mettre à disposition un service (étiquette 1), l'enregistre dans l'annuaire (étiquette 2). Puis, les composants clients souhaitant utiliser ce service (étiquette 3), émettent une requête pour rechercher le service désiré dans l'annuaire (étiquette 4). Si la réponse est positive, une liaison (appelée aussi binding) est mise en place pour l'utilisation du service (étiquette 5).

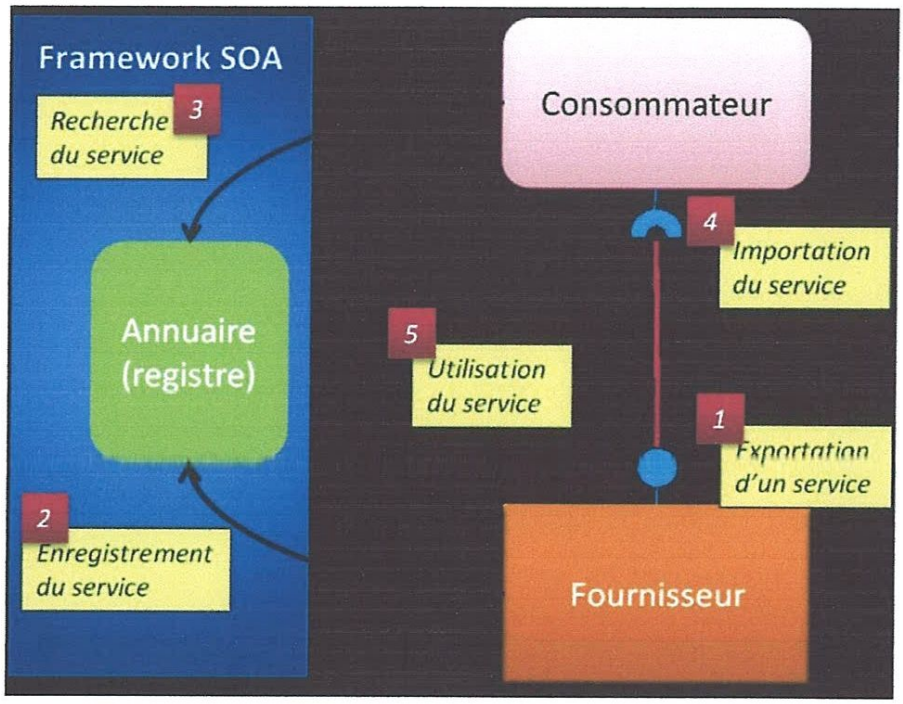

Figure 1.3 : Fonctionnement SOA

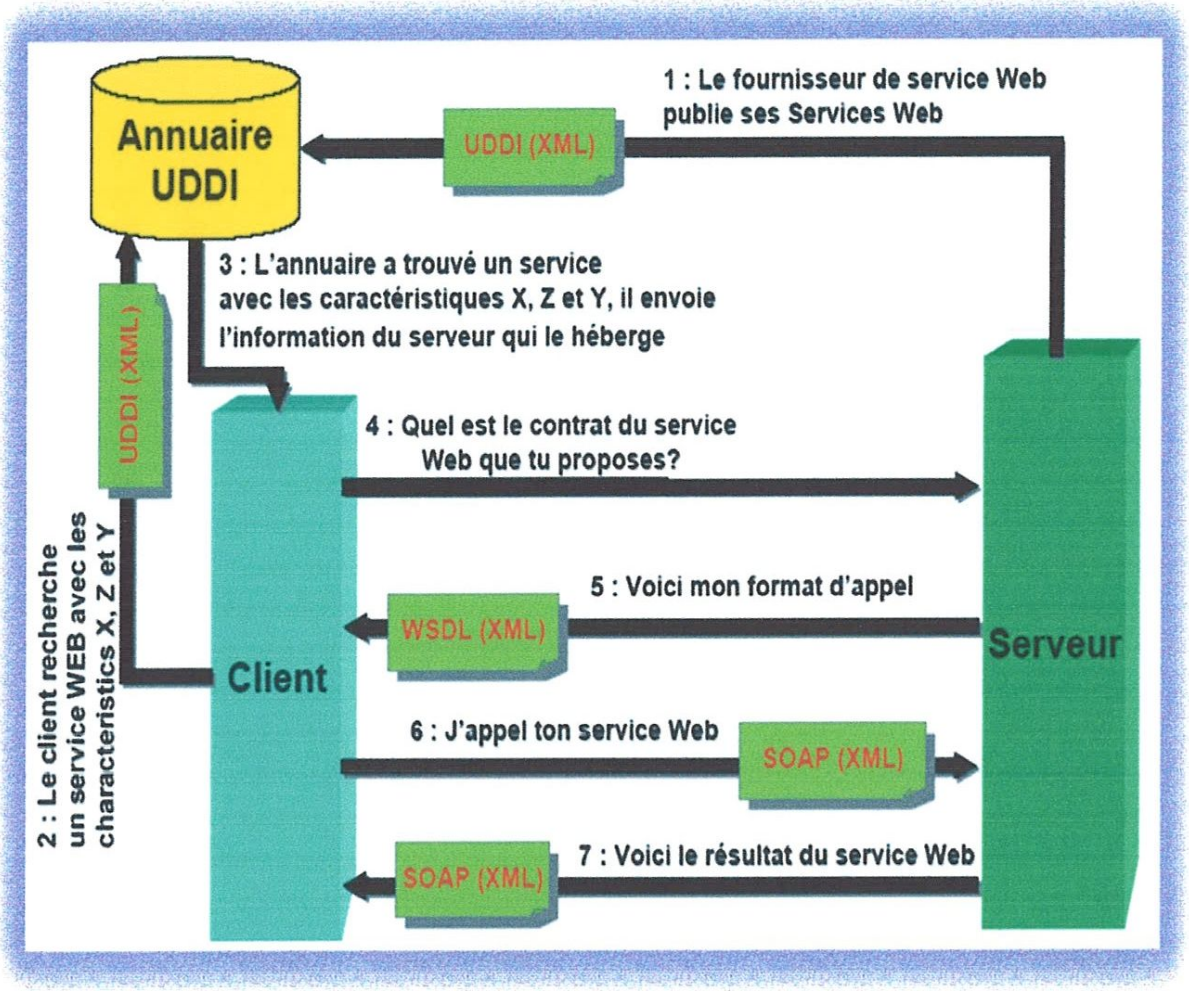

Figure 1.4 SOA et web services fonctionnement

Description de la procédure :

- Le fournisseur de service crée le service Web, puis publie son interface ainsi que les informations d'accès au service, dans un annuaire de services Web.
- L'annuaire de service rend disponible l'interface du service ainsi que ses informations d'accès, pour n'importe quel demandeur potentiel de service.
- Le demandeur de service accède à l'annuaire de service pour effectuer une recherche afin de trouver les services désirés.

Ensuite, il se lie au fournisseur pour invoquer le service.

9. Les Concepts

9.1. Le service

Une architecture orientée services consiste essentiellement en une collection de services qui interagissent et communiquent entre eux. Cette communication peut consister en un simple retour de données ou en une activité (coordination de plusieurs services).

Le service est l'unité atomique d'une architecture SOA, une application est un ensemble de services qui dialoguent entre eux par des messages.

Le couplage entre services est un couplage lâche et les communications peuvent être synchrones ou asynchrones.

Le service peut : - être codé dans n'importe quel langage ;

- s'exécuter sur n'importe quelle plate-forme (matérielle et logicielle).

Le service doit : - Offrir un ensemble d'opérations dont les interfaces sont publiées ;

- Etre autonome (disposer de toutes les informations nécessaires à son exécution : pas de notion d'état) ;

- Respecter un ensemble de contrats (règles de fonctionnement

Un service est une entité de traitement qui respecte les caractéristiques suivantes :

- **Large Granularité** (coarse-grained) : Les opérations proposées par un service encapsulent plusieurs fonctions et opèrent sur un périmètre de données large au contraire de la notion de composant technique.

- **Interface** : Un service peut implémenter plusieurs interfaces, et aussi plusieurs services peuvent implémenter une interface commune.

- **Localisable :** Avant d'appeler (bind, invoke) un service, il faudra le trouver (find).

- **Instance unique :** A la différence des composants qui sont instanciés à la demande et peuvent avoir plusieurs instances en même temps, un service est unique. Il correspond au design pattern Singleton.

- **Couplage faible (loosely-coupled) :** Les services sont connectés aux clients et autres services via des standards. Ces standards assurent le découplage, c'est-à-dire la réduction des dépendances. Ces standards sont des documents XML comme dans les web services.

- **Encapsulation :** Le service présente une stricte séparation entre l'interface et l'implémentation

- **Isolation des responsabilités :** Un service est responsable sur une tache précise ou gère une ressource spécifique, cela fournit une seule place où chaque fonction peut être accomplie

- **Autonomie :** Cela lui permet d'être déployé, modifié et maintenu indépendamment des autres (disposer de toutes les informations nécessaires à son exécution : pas de notion d'état), donc plus de souplesse dans la gestion des services, le self-healing (auto guérison et recovery) est aussi souhaitée

- **Réutilisation :** Les premières caractéristiques ensemble permettent au service d'être combiné dans multiple processus métier et d'être partagé et utilisé comme un bloque de construction d'autres services composé

- **Découverte et liaison dynamique :** Un service est un programme disponible a travers le réseau il doit être possible de le découvrir sur le réseau et découvrir ce qu'il fait, c'est une nécessité pour les clients mobile et les consommateurs

- **Stateless :** ne dépend d'aucun contexte ou service externe, n'a pas a garder la trace de la dernière exécution

- **Self-describing et contrat :** Le contrat de service fournit une description complète de l'interface du service, des opérations, des paramètres d'E/S, les pré et post conditions et des contraintes sur les opérations.

- **Composable :** Peut être composé d'autre service et peut composer avec d'autres un nouveau service

- **Gouverné par une politique :** La relation entre service est gouvernée par des politiques et SLAs qui doivent être respectées.

- **Interopérabilité :** être codé dans n'importe quel langage ; s'exécuter sur n'importe quelle plate-forme (matérielle et logicielle).

- **Communication par message :** les messages sont généralement échangés de façon asynchrone, RPC est rarement utilisé.

- **Interface standard :** offrir un ensemble d'opérations dont les interfaces sont publiées

Une déclinaison du service est par exemple le service Web qui utilise WSDL (un meta langage XML) comme langage de description, un annuaire UDDI pour en permettre la localisation et un protocole de transport comme http dans l'architecture REST et SOAP pour l'architecture SOA.

9.2 Les Services Métier

Un service métier est la représentation d'une activité métier élémentaire ou complexe. Par exemple l'annulation d'une commande est un ordre simple de suppression. Mais le processus de recrutement d'un nouvel employé dans une entreprise est représenté par un service plus complexe.

Un service métier vu par un processus peut combiner plusieurs services de granularité plus fine.

9.3 Les Services Web

Un Service Web est un programme informatique permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur Internet ou sur un Intranet, par et pour des applications ou machines, sans intervention humaine, et en temps réel.

Le service web se caractérise en effet par une standardisation des implémentations, par une localisation à distance des services et par une récupération de l'interface d'accès permettant l'exécution du traitement correspondant. Pour cette raison, comme le stipule d'ailleurs le Gartner (initiateur de SOA), le service web est à ce jour - et sera probablement demain - au cœur des SOA, même si toute technologie objet assez avancée peut servir de base à la mise en place d'architectures orientées service.

10. Conclusion

Les architectures orientées services sont rapidement acceptée par le monde de l'informatique comme un son, modularité approche pour construire et déployer des services à travers l'entreprise étendue. Toutefois, la mise en œuvre pratique de ces architectures nécessite une planification minutieuse. Les entreprises intéressées doivent d'abord s'assurer qu'ils sont adaptés pour mettre en œuvre et de les aider dans le long terme.

En développant et en suivant une feuille de route mise en œuvre, les entreprises peuvent aborder de manière proactive un éventail de défis auxquels ils vous rencontrerez le long du chemin. Chaque entreprise sera confrontée à un ensemble unique de défis, les approches correspondantes pour résoudre ces défis varient, aussi bien. L'impact des défis, à la fois pendant et après la mise en œuvre-dépend aussi de le contexte de l'entreprise donnée.

Le prochain chapitre sert à présenter les techniques utilisées et détailler clairement le mot 'Service web'.

II. Migration vers une architecture orienté service

1. Introduction

Au cour des quarante dernières années, les systèmes informatique se sont développés de manière exponentielle, sous forme d'architecture logicielle de plus en plus complexe et difficile à gérer pour les entreprises.

Les architectures traditionnelles ont atteint leur limite en terme de capacité, alors que les besoins traditionnels des organisation informatique demeurent.

Les départements informatiques doivent toujours répondre rapidement aux nouvelles exigences des entreprises, tout en réduisant constamment le coup informatique de l'entreprise et en absorbant et on intégrant de manière transparente de nouveaux partenaires et clients commerciaux.

Le secteur d'activité des logiciels a connu de nombreuse architectures conçu pour permettre un traitement entièrement distribué, des langages de programmation destiné a n'importe quelle plate-forme et une myriade de produit de connectivité conçu pour permettre une intégration plus rapide et plus efficace des applications et pour réduire considérablement les temps de mise en œuvre

L'architecture orientée service (AOS) a connu ces un grand engouement des entreprises de tout secteur et de toute taille en raison de ses avantages économiques et technologiques. La Migration vers la AOS est devenue l'une des techniques importantes de modernisation, Elle aide les organisations d'une part à réutiliser leurs anciens systèmes existants en leurs donnant une nouvelle vie, et d'autre part à profiter des avantages des systèmes à base de service.

2. Evolution vers une Architecture AOS

Aujourd'hui, la faiblesse des progiciels réside principalement dans la dynamique des entreprises. Si certaines applications restent pérennes, d'autres doivent être rapidement améliorées ou remplacées. Certaines sont stratégiquement en "première ligne" alors que les performances limitées de certaines autres ne constituent pas un frein à la bonne marche de l'ensemble.
Il fallait donc imaginer une architecture modulaire qui saurait incorporer l'existant tout en permettant le développement de nouveaux services performants et mutualisables

L'architecture orienté service a évolué selon un échèle de trois degré on premier temps on Trouve :

2.1. Architectures Orientées Objets

Les architectures orientées objets nécessitent de communiquer avec l'instance particulière d'un objet qui se caractérise par son état, son comportement et son identité. Les communications sont par nature avec état [stateful] puisqu'il faut communiquer avec une instance d'objet crée au préalable. L'information d'état est donc gérée sur le serveur. Toute communication avec l'objet impose un aller-retour avec le serveur. Les protocoles utilisés habituellement (IIOP, DCOM ou RMI) ne sont pas conformes aux standards de l'Internet. C'est la faillite totale de l'extensibilité [scalability] de ce style d'architecture dans un environnement largement distribué qui explique le passage aux styles d'architectures orientées ressources ou services.

Au début des années 1990, la technologie objet était considérée comme la technologie la plus prometteuse pour écrire du logiciel. Force est de constater que quinze ans plus tard, cette technologie n'a pas apporté ce que l'on en attendait. Ce qui fonctionnait très bien localement dans une seule machine n'a pas supporté le "passage à l'échelle". La communication inter programmes avec des objets distribués comme RMI, CORBA ou DCOM a été un échec sérieux **[06].**

Bien que les ressources ressemblent à des objets, en particulier pour les propriétés, il y a beaucoup de différences. La plus grosse différence est liée au nombre de méthodes. La programmation objet a tendance à multiplier les méthodes ce qui augmente la complexité des interfaces.

Pour cela l'évolution pour une architecture orienté service a donné beaucoup de benefices

Si on reviens a une architecture orienté service on trouve :

2.2 Architectures orientées services

Un service est une fonction logicielle autonome et sans état qui accepte des requêtes et qui renvoie des réponses au travers d'un interface standard bien défini. Un service est donc une unité de traitement qui fournit un résultat à un consommateur. Fournisseurs et consommateurs sont habituellement des agents logiciels qui agissent par délégation de leurs propriétaires. Les services ne doivent pas dépendre de l'état d'autres fonctions ou d'autres traitements externes. Les technologies employées pour réaliser un service comme le langage de programmation ne font pas partie de la définition d'un service.

Dans une architecture orientée services, tous les messages ou toutes les requêtes pour un service spécifique sont envoyés à l'adresse unique du service.

2.3 Définition du Style d'Architecture Orientée Service (SOA)

Le style d'architecture orientée services (SOA) est un style d'architecture qui se définit par :

Un couplage lâche entre les composants d'un système pour ne pas dépendre de l'état d'autres services et pour faciliter la réutilisation,

Des services sans état pour faciliter l'extensibilité et l'éventuelle orchestration,

Des services fortement interopérables ce qui implique l'utilisation de vocabulaires de données très bien définis.

Comme on peut le constater, il n'y a rien de nouveau sous le soleil. Les concepts de couplage lâche, sans état ou d'interopérabilité des données existent depuis pratiquement les débuts de l'informatique. Ce qui a changé, c'est la portée de ces concepts autrefois limitée à des systèmes autonomes. Avec l'URI comme mode d'adressage de services, le couplage lâche est devenu minimal. Il n'y a plus besoin d'aucune information sur la localisation du service, le type de système, la méthode de programmation, le mode d'accès ou le nommage pour s'adresser à un service. Juste une chaîne de caractères sur un bout de papier suffit.

De même, l'interopérabilité des données a fait d'énormes progrès. Dans le passé, il fallait choisir le jeu de caractères utilisé. On pouvait tout représenter (russe, chinois, grec, latin,...) mais pas mélanger

3. L es bénéfices attendu d'une architecture orienté service

Les architectures orientées services (SOA) sont reconnues comme une approche qui peut entraîner un meilleur alignement des TI avec les processus d'affaires et ainsi rendre une entreprise plus compétitive. Il y a aussi d'autres bénéfices d'affaires qui devraient être pris en considération dans la définition du "business case" dont voici une brève description

3.1 . Bénéfices de la réutilisation

L'élimination des redondances par le partage des services procure à long terme une réduction des coûts de développement et d'assurance qualité quand vient le temps de procéder à des changements ainsi que des économies importantes au niveau des frais d'entretien pour un service partagé.

Effectuer des modifications ou introduire de nouvelles fonctions sur un seul service implique aussi un cycle de développement plus rapide, donc une capacité à répondre plus efficacement aux besoins d'affaires.

Être plus compétitif en réagissant plus rapidement aux impératifs du marché est parfois difficile à calculer en termes financiers, c'est pourquoi il faut insister sur la réduction des frais de développement, de test et d'entretien qui sont plus faciles à estimer.

3.2. Bénéfices du développement d'applications composites (multi-services):

L'assemblage de services multiples pour l'introduction de nouvelles applications permet des économies substantielles en termes d'intégration.

Ici encore l'impact financier provient autant d'une réduction des frais de développement informatique que d'une augmentation des revenus de l'entreprise entraînée par une meilleure réponse aux besoins d'affaires.

3.3 Bénéfices du couplage lâche ("loosely coupled"):

Le couplage lâche procure une flexibilité qui rend l'entreprise plus agile en isolant les interventions informatiques nécessaires et en permettant un déploiement itératif.

De plus, le coupage lâche permet des économies de budget parce que les ressources qui interviennent n'ont pas besoin de connaître les technologies de chaque service impliqué.

Finalement, le couplage lâche facilite la connectivité interne et externe, donc un plus grand potentiel d'automatisation et d'alliances commerciales.

En résumé, un modèle financier de justification d'un projet SOA devrait aborder les trois éléments suivants :

3.4. Efficacité d'affaires

- Plus grande agilité et meilleure réponse à la dynamique de marché
- Plus grande efficacité des processus
- Meilleur déploiement des ressources

3.5. Réduction des coûts

- Réduction des frais d'entretien
- Réduction des efforts nécessaires pour supporter les changements organisationnels
- Choix technologiques plus flexibles étant donné le couplage lâche des applications

3.6. Réduction des risques

- Niveau plus élevé de qualité de services des TI
- Déploiement itératif
- Développement plus rapide qui assure un retour sur investissement plus rapide.

4. Identification du service

L'identification du service c'est définir un ensemble de services au sein du contexte de l'entreprise qui prend en charge l'architecture d'entreprise .

De ma aux exigences d'affaires et de la logique informatique existante nière générale, c'est le processus de recherche et d'extraction des services par rapport dans les systèmes. La position, le sens et les activités de service d'identification sont différents dans la conception SOA en vertu de diverses environnements.

4.1 Question clé et méthodologie de recherche

Supposons un architecte de logiciel prévoit de concevoir un système SOA et on voudra trouver les services potentiels. Ces questions sera probablement demandé:

Q1: Quel est l'identification des services? Où puis-je commencer à d'identification?

Q2: Quand dois-je procéder à l'identification des services pendant le processus de architecturer un système SOA?

Q3: Comment puis-je effectuer une identification de service? Etes-Y at-il des règles universelles pour l'identification du service ou de meilleures toute pratiques?

L'identification du service est essentielle dans l'analyse et la conception d'une SOA. La conception SOA avec des objectifs différents de la position, les activités de d'identification peuvent différer. Ainsi, nous étudions la relation entre l'identification du service de l'ingénierie existante traiter et essayer de trouver ce que l'identification des services devrait faire les différents cas et quand il doit être effectué.

A. Service d'Ingénierie

Une variété de cadres de conception SOA et les méthodologies ont été introduites par les chercheurs Le processus de conception et architecture de système axé sur le service est nommé Service du génie. Malgré la différence dans la terminologie, le cycle de vie de service, y compris l'identification, analyse, conception, mise en œuvre et déploiement. Du point de

«service», ces phases sont convient à la fois de réaliser les exigences opérationnelles et la la migration vers SOA Systèmes SOA sont basées sur les produits logiciels, ce qui signifie l'identification des services devrait également être en mesure de s'adapter à base en génie logiciel ou la réingénierie du logiciel.

B .Génie logiciel orientée services

À partir des besoins d'affaires, la conception et la la mise en œuvre d'un système axé sur le service pour remplir l'exigences.

Il s'agit d'une pratique typique du logiciel d'ingénierie. Un générale processus de génie logiciel comprend: exigence, analyse, la conception et le développement..

5. Les approches de SOA

Avant qu'un développeur écrit une seule ligne de code, il est essentiel d'identifier a la fois spécifique des conducteurs de l'entreprise et SOA les dépendances entre les entreprises et les technologies sous-jacentes. Négliger le contexte d'affaires peut aboutir a un projet en infrastructure SOA qui est poursuivi pour ses propres fins, ou lorsque les investissements sont réalisés qui ne s'alignent pas bien avec les besoins et les priorités de l'entreprise.

Deux approches sont généralement recherchés pour la mise en œuvre SOA: top-down et bottom-up. Ces deux approches ont des pièges possibles qui peuvent empêcher la réussite. La plupart des organisations qui ont tenté de déployer une infrastructure SOA grâce a l'approche top-down ont découvert que lorsque l'infrastructure est enfin livré, il est hors de la synchronisation avec les besoins de l'entreprise.

De même, une approche bottom-up peut échouer, car il peut conduire a une mise en œuvre chaotique des services créés sans égard aux objectifs organisationnels. Le 'middle-out' est un succès de l'approche hybride des deux autres approches. Business pilotes et d'une vision stratégique sont d'abord employés pour définir clairement les orientations et les priorités. Sur cette base, l'organisation prend plusieurs étapes successives de construire des tranches de bout en bout des capacités,a chaque itération de la livraison d'une nouvelle application dynamique de retour a l'entreprise qui est utilisé pour créer des aller-retour.

Microsoft a depuis longtemps préconisé ce "monde réel" pour tirer les architectures orientées services: l'approche est axée sur la rapidité du temps de la valeur, et il offre des résultats a travers itérative, progressive des mesures qui facilitent l'alignement étroit des ressources a l'évolution des entreprises conditions.

Par contre L'approche « top-down » qui est littéralement « l'approche du haut vers le bas » a prédominé la gestion des projets de développement dans les années 70 et 80. Ceux-ci ont été conçus par des experts puis amenés et implantés dans les communautés sans toutefois avoir eu les opinions des bénéficiaires ; ils ont échoué en grand nombre. A titre d'exemple, les résultats de plusieurs projets réalisés pendant 10 ans au Togo ont été partagés entre les acteurs ; les projets se sont révélés non viables et non durables (Blanchet, 2001). La pertinence de cette approche a été remise en cause. Aujourd'hui son utilisation est contestée par de nombreux spécialistes et les organisations de développement.

6. Comparaison entre les deux approche bottom up et top down

La question de la bonne approche a donné naissance à bien des écrits et des théories chacune sous tendant des positions, selon nous, trop dogmatiques.

Le point de vue de Xebia dans ce domaine est simple :

Bien qu'une approche Top-Down soit préférable dans le cadre d'une démarche SOA, elle n'est cependant que rarement possible car elle signifie une refonte de tout ou partie du S.I, jugée bien souvent trop coûteuse et trop risquée. Il est donc bien souvent obligatoire de passer, au moins temporairement (àl'échelle d'un schéma directeur cela signifie quelques années), par une phase de conception ascendante (Bottom-up).

C'est dans les approches Bottom-up que les technologies de types EAI et ESB prennent tout leur sens puisqu'elles permettent dans un premier temps de faire communiquer, entre elles, les différentes briques applicatives tout en garantissant leur totale indépendance et étanchéité les unes vis-à-vis des autres.

Cependant, l'approche Bottom-Up seule ne permet pas de mettre en place une SOA. Il s'agira tout au mieux d'une « mise en mode services ».

Ces services seront vus localement et ne s'inscriront pas dans une Architecture Orientée Services à l'échelle dc l'cntreprise.

La bonne solution est donc de mixer les deux démarches, Top-Down pour une portion du S.I. qui est en cours de refonte (ou à refondre) et Bottom-Up pour le reste, ce qui concourt au bon fonctionnement de l'ensemble au quotidien.

7. Les différente approchent de migration vers SOA

Le remplacement : qui consiste à retirer complètement l'application et la remplacer carrément par une nouvelle récupérée sur étagère. Cela est faisable lorsque les fonctionnalités et les règles métiers sont bien compréhensibles et le SP est obsolète ou difficilement maintenable.

Le redéveloppement : Réfère à l'application de l'approche de réingénierie et de retro-ingénierie afin d'ajouter des fonctionnalités orientées service aux systèmes patrimoniaux. Selon Chikofsky La retro-ingénierie est le processus d'analyse d'un système sujet pour créer une représentation du système dans un niveau plus haut d'abstraction", la réingénierie est elle définie comme "L'examen et la modification d'un système pour le reconstituer sous une nouvelle forme".

La réingénierie peut inclure des activités telles que la retro-ingénierie, la restructuration, la re-conception, et la ré-implémentation. Il y'a trois principaux enjeux dans la réingénierie orientée service: l'identification des services, le packaging des services et le déploiement des services.**[07]**

Le Wrapping : fournit une nouvelle interface orientée service aux composants du SP existant, les rendant facilement accessible par les autres composants logiciels. Il s'agit d'une technique de modernisation à boîte-noire puisqu'elle se concentre sur l'interface de l'ancien système tout en cachant la complexité de son fonctionnement interne.

Cette stratégie est utilisée lorsque la réécriture du code existant est trop coûteuse, et lorsqu'une solution rapide et économique est nécessaire. Elle peut constituer une bonne option si l'ancien système a une valeur commerciale élevée et le code est de bonne qualité. Le principal problème est que cette stratégie ne change pas les caractéristiques fondamentales des applications existantes et ne va pas résoudre les problèmes déjà présents, comme les problèmes de maintenance et de mise à niveau.

La migration: Selon l'auteur, la migration comporte le redéveloppement et le wrapping. Il n'est toujours pas évident de pouvoir distinguer la migration du redéveloppement et du wrapping. Le terme migration est utilisé en référence de toute approche qui déplace le SP entier dans un nouvel environnement.

En plus des quatre catégories citées,a dressé un ensemble de critères qui permettent de comparer les différentes approches d'évolution vers des architectures orientées service, qui sont :

- La stratégie de modernisation adoptée : (migration, remplacement, redéveloppement ou wrapping) ;

- Le type du système patrimonial à faire évoluer: Procédural, orienté objet, C, code binaire exécutable, etc.

- Le degré de complexité de l'approche d'évolution: temps, coût, complexité de la méthode

- La profondeur de l'analyse du SP : superficielle, profonde.

- Adaptabilité du processus d'évolution: le processus s'adapte t'il bien au SP pour minimiser l'étendue des modifications nécessaires.

- Support d'outils: A quelle dégrée le processus est automatisé, et est-ce que l'outil est développé ou proposé.

- Le degré de convergence: Est-ce que l'approche présente une stratégie complète pour évoluer vers une AOS, ou juste des spécifications de modernisation.

- Validation et Maturité: Est-ce que l'approche proposée a été appliquée et validée avec succès sur un nombre suffisant de cas ou est ce qu'elle reste dans le cadre d'idées théoriques ? Est-ce une technique commerciale prouvée, etc.

8. conclusion

Après l'achèvement de l'ère du l'orienté objet La migration vers une architecture orienté service est considéré comme une solution très efficace vu ses avantages économique et technique, la SOA est considéré comme l'aube de la programmation moderne.

III. Technologie SOA et Services Web

1. Introduction

Le service web est une nouvelle technologie née à la fin des années quatre-vingt-dix, sous l'impulsion des géants de l'informatique comme MICROSOFT, IBM, SUN et encore SAP. Il a une grande partie basée sur les technologies XML(eXtencible Marcup Langage)qui fournit le point fort de service web ; XML permet a ce dernier de gérer l'interopérabilité entre les applications hétérogènes. Aujourd'hui il a fait ses preuves et se présente comme une solution de production performante, soutenue et standardisée par la W3C (World Wide Web Consortium) **[08]**.

Ce chapitre traite les concepts de base des services web, les standards utilisés et l'architecture des applications orientés services

2. Definitions des services web

Selon la société W3C :

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards"**[08].**

Traduction en français

"Un service Web est un système logiciel conçu pour permettre l'interopérabilité machine-to-machine à l'interaction sur un réseau. Il dispose d'une interface décrite dans un format machine processus de mesure (spécifiquement WSDL). D'autres systèmes d'interagir avec le service Web de la manière prescrite par sa description en utilisant des messages SOAP, typiquement transmise via HTTP avec une sérialisation XML en conjonction avec d'autres liées au Web des normes »

Selon la définition on peut extraire les caractéristiques des services web suivantes :

- Un service web est un système logiciel conçu pour supporter les interactions entre les machines à travers le réseau ;

- Il possède une interface décrite en WSDL(web Service Description Language) ;

- Il peut interagir avec d'autres Services Web par sa description en utilisant des messages SOAP(

Simple Object Access Protocol)

- Les messages SOAP transmis en utilisant le protocole http(Hyper txt Transfert Protocol) avec une sérialisation XML. La définition des services web ne serait pas complète si l'on n'évoquait pas ses principaux standards : SOAP, WSDL et UDDI que nous détaillerons dans la section suivante.

3. Standards des services web

L'infrastructure des services web s'est concrétisée autour de trois principales technologies considérées comme des standards : SOAP, WSDL et UDDI(Universal Description,Discovery and Integration) qui ont été définies par les consortiums W3C **[03]**. Ils ont été développés pour assurer les besoins requis pour la mise en œuvre des services web.

Toutes ces technologies sont basées sur le langage XML.

3.1. XML (Extensible Markup Language)

XML est un langage de balisage. Il est fourni par W3C en février 1998 pour encoder tout type de données transportable sur les protocoles de l'internet comme HTTP. XML représente la technologie de base des services web. En effet, grâce à XML les services web sont indépendants des plateformes et langages de développement, ce qui permet leur interopérat

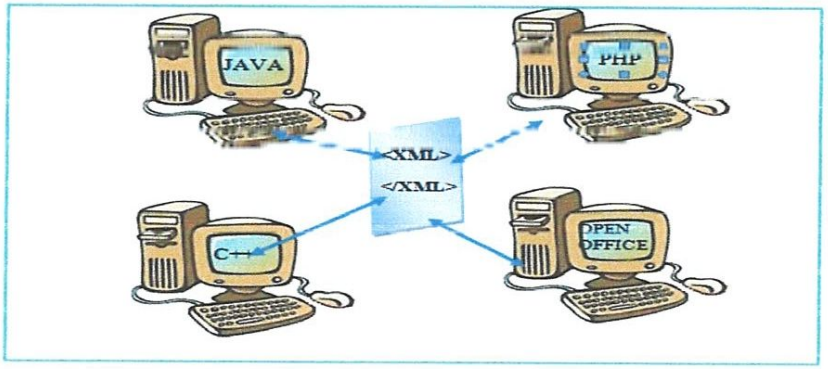

Figure 3.1. : Interaction entre les applications hétérogènes à travers XML

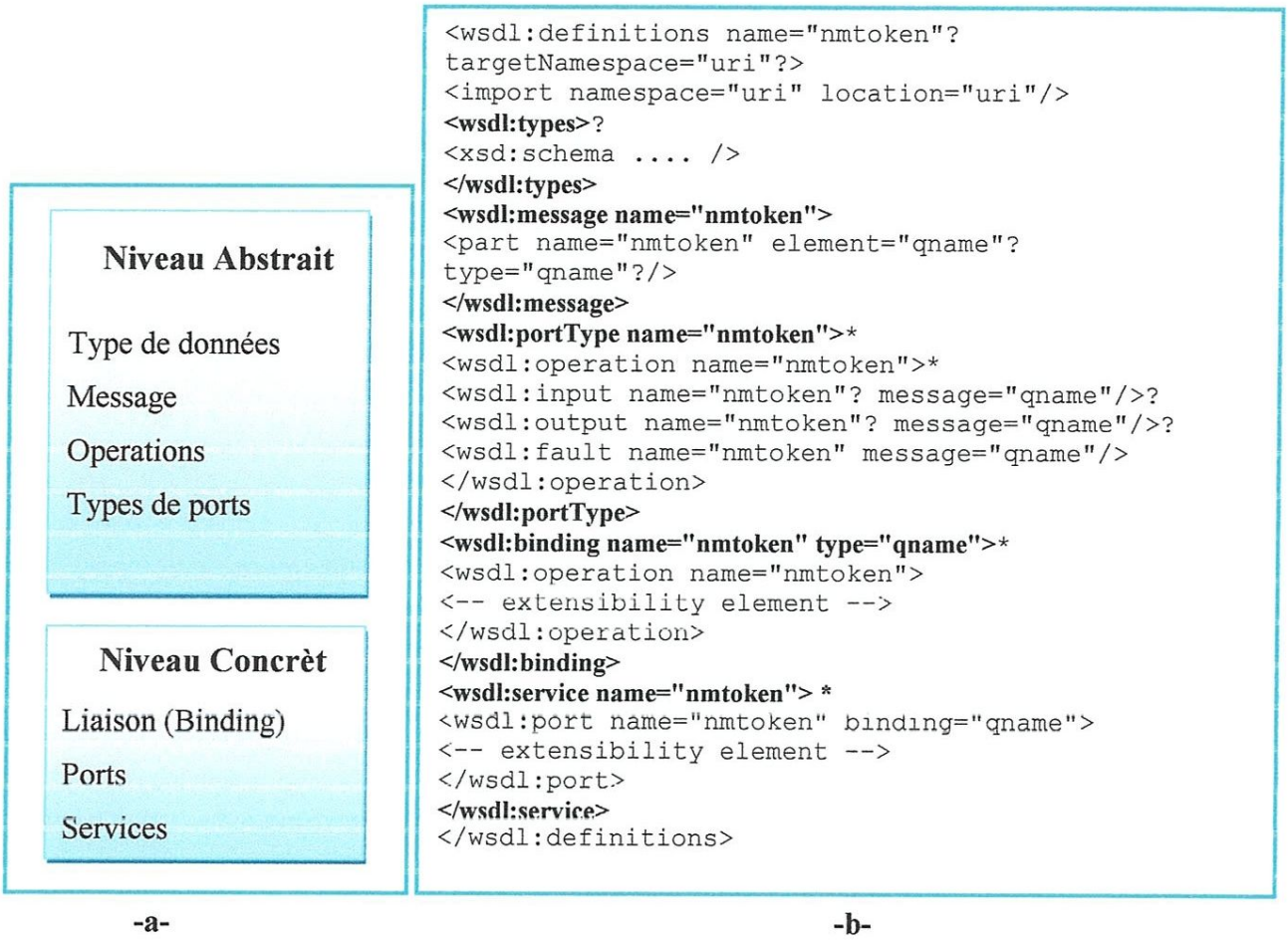

Figure 3.4: Structure d'un document WSDL

a) La partie abstraite est composée de quatre éléments suivants :

- **Type de données** : c'est l'élément qui définit les types de données utilisées dans les messages échangés par le service web.

- **Message :** cet élément représente une définition abstraite de toutes les données transmises. C'est-à-dire il décrit les paramètres d'appel et de retour de chaque opération.
- **Operations :** c'est l'élément qui décrit d'une manière abstraite les actions supportées par le service.
- **Types de ports:** c'est l'élément qui représente un ensemble d'opérations correspondant chacune à un message entrant ou sortant.

b) La partie concrète:

- **Liaison (Binding):**cet élément définit les protocoles de communication utilisés lors des invocations du service web. Ainsi que le format des messages.
- **Port :** est une adresse d'accès au service.
- **Service :** c'est l'élément qui regroupe une collection de ports .

3.3.2. Exemple d'un document WSDL

```
<?xml version="1.0" encoding="UTF-8" ?>
-<wsdl:definitionstargetNamespace="http://example"xmlns:apachesoap="http://xml.apache.org/xml-
soap"xmlns:impl="http://example"xmlns:intf="http://example"xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"xml
ns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   +<!--WSDL created by Apache Axis version: 1.4
   Built on Apr 22, 2006 (06:55:48 PDT)
 -->
+<wsdl:types>
+<wsdl:message name="addRequest">
+<wsdl:message name="addResponse">
+<wsdl:portType name="Math">
+<wsdl:binding name="MathSoapBinding" type="impl:Math">
+<wsdl:service name="MathService">
 </wsdl:definitions>
```

Figure 3.5: Structure d'un document WSDL

Le WSDL est un langage qui permet de décrire un service Web, il est indispensable à leurs déploiement; WSDL est utilisé via le standard UDDI pour faire la publication de ce service .

3.4. UDDI (Universal Description, Discovery and Integration)

Dans un environnement ouvert comme Internet, le modèle des services web n'est d'aucune utilité s'il n'existe pas un moyen de localiser les services .Un troisième standard a été conçu pour résoudre ce problème de localisation de ses services.

UDDI a été défini initialement par ARIBA, IBM et Microsoft. C'est un protocole permet de publier et de faire une recherche des informations qui concernent une application et ses services web. La publication des informations concernant les fournisseurs des services et les services doivent être spécifié en XML afin que la recherche et l'utilisation soient faites de manière dynamique et automatique. UDDI constitue un méta-service (annuaire) qui contient

des informations sur les fournisseurs et les services web avec des fonctions de publication et de recherche, ces information ont une structure bien définit .

La structure de données d'UDDI

UDDI regroupe l'ensemble des informations divisé en trois parties qui doivent être décrites en XML. Chacune d'elles peut être utilisée pour faire une rechercher via UDDI. Ces parties sont les suivantes : les pages blanches, les pages jaunes et les pages vertes .

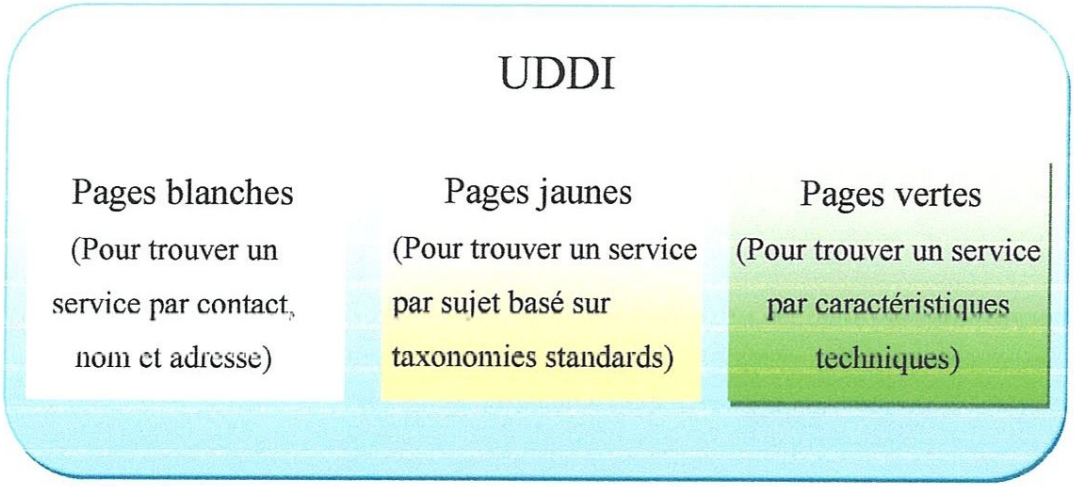

Figure 3.6 : Structure de données de l'annuaire UDDI.

- les pages blanches (white papers) : permet de connaitre les informations à propos de l'application qui fournit le service. Les pages blanches regroupent les informations pour identifier l'application comme le nom et l'adresse physique.
- les pages jaunes (yellow papers) : permet de détailler la description d'application qui est faite dans les pages blanches telles que la catégorie d'application, les services offerts par cette application et la qualité des services, etc. Cette description permet de faire la classification des applications.
- les pages vertes (green papers) : elles contiennent les informations techniques sur les services web proposés .

Le standard UDDI offre deux fonctionnalités de base : la publication des différents types d'information sur les services et leurs fournisseurs, et la consultation du contenu des registres .

Après l'identification des trois standards principaux des services web, on passe à la présentation de l'architecture des services web qui est basé sur ces standards.

1. Architecture des services web

L'architecture des services web fournit un prototype nécessaire à la compréhension des services web et des relations entre les standards utilisés. Cette architecture se base sur l'interaction entre les trois composants suivants :

- Le **fournisseur de service (service provider) :** il montre deux aspects ; un aspect correspond au propriétaire du service et un aspect technique qu'il représente la plateforme qui héberge le service. Il permet de créer le service Web, puis publie son interface ainsi que les informations d'accès au service, dans un annuaire de services Web.
- Le **client de service ou le demandeur (service requester) :** il s'agit d'une application qui cherche et invoque un service web ; Il accède à l'annuaire de service pour effectuer une recherche afin de trouver les services désirés. Ensuite, il se lie au fournisseur pour invoquer le service. Techniquement, le client peut être une simple application Windows ou web, comme il peut être un autre service web.
- L'**annuaire de service (service Broker) :** L'annuaire est un intermédiaire entre le fournisseur de service et le client de service. C'est un registre de description qui offre aux fournisseurs le moyen pour publier et indexer leurs services web sur le réseau. Il permet aussi aux clients de rechercher, localiser les services désirés et obtenir des informations qui correspondent à leur demande **[01]**.

5. Fonctionnement des applications orientées service web :

L'interaction entre les composants de l'architecture de service web suit une séquence d'opérations tel que : la publication, la recherche, la connections et l'invocation de services, ces opérations doivent être exécuté dans l'ordre qui est déjà montré dans la figure 1.4 page 10 .

Mais en fait rappel pour le fonctionnement des application orientées service web :

1) D'abord, le fournisseur de service crée le service Web et définit sa description puis le publie dans un annuaire de services Web sous la forme de fichier WSDL.

2) Ainsi, le client accède à l'annuaire de service pour effectuer une recherche afin de trouver les services désirés.

3) Lorsque le client trouve la description de service Web désirés dans l'UDDI, ce dernier envoie la référence de ce service au client pour sélectionner.

4) Le client télécharge le fichier WSDL de service sélectionné depuis le fournisseur.

5) Ensuite, le client utilise la description du service sélectionné pour récupérer les informations nécessaire qui lui permettre de se connecter au fournisseur du service et d'interagir avec lui ; c'est-à-dire le client invoque le service Web et lui demander d'exécuter certaines de ses fonctionnalités.

6) Enfin, le fournisseur envoie le résultat de l'exécution des fonctionnalités du service web, ce qui termine le cycle de vie de service web.

6. Bénéfices d'utilisation des services web

Les services web permettent de **[10]** :

✓ Exposer des fonctionnalités à travers le web; les traitements des opérations des services web peuvent être invoqués via une requête HTTP, ce qui peut permettre à plusieurs applications de consommer ces services web.

✓ Réduire le temps de mise en marché des services offerts par les diverses entreprises.

✓ Offrir une meilleure interopérabilité entre les applications et les systèmes hétérogènes ; les services web permettent à des programmes écrits en des langages différents et sur des plates formes différentes de communiquer entre eux par le biais de norme XML.

✓ fonctionner sans voire des problèmes de sécurité; grâce au protocole HTTP qui est supporté par tous les navigateurs Internet et les serveurs.

✓ Fournir un couplage faible entre les fonctionnalités exposées et les applications qui les utilisent, à tel point que les consommateurs et les fournisseurs de services peuvent être écrits en des plateformes ou des langages différents (Java, PHP, ...).

7. Méthodologie de développement du service web

Il existe deux approches pour la création des services web: on a l'approche **Code first** et l'approche **WSDL first.**

7.1 Code First :

C'est la technique de création de services Web la plus courante, elle consiste à inférer une interface de service Web à partir du code source orienté service.

Cette technique est souvent appelée « **Code First** » ou « **Implementation First** » **c'est-à-dire** la priorité au code du service web ; car l'interface du Service Web, décrite **d'une façon** formelle dans un document **WSDL (Web Service Description Language)**, qui est dérivée de code de service web.

Figure 3.7 : Développement d'un service Web par la méthode « Code First »

La technique de développement de services Web appelée « **Code First** » consiste à écrire d'abord le code du service Web (voir étape n° 1 de la figure ci-dessus). Après compilation, l'infrastructure des services Web utilise ce code pour générer de façon dynamique un fichier WSDL (étape n° 2). Puis en lance le déploiement de service (étape n° 3). Lorsque les clients demandent la définition du service Web, ils récupèrent le fichier WSDL généré et créent le client à partir de cette définition.

7.2.WSDL first :

« **WSDL first** » (priorité au WSDL) : Cette technique qui consiste a creer d'abord le fichier WSDL est aussi appelée parfois « **Schema First** » (priorité au schéma) ou « **Contract First** ».

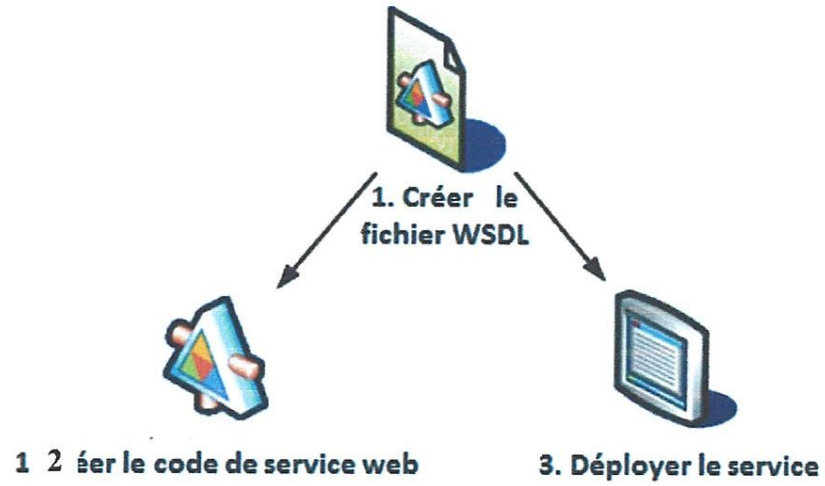

Figure 3.8 : Développement d'un service Web par la méthode « WSDL First »

- La technique de développement de services Web **WSDL first** consiste à travailler comme suit :

Comme l'illustre la figure ci-dessus, la création de services Web à partir de la méthode « WSDL First » comporte trois étapes majeures :

1. Créer le fichier WSDL.
2. Créer le code du Service Web.
3. Déployer le service Web.

- Vous pouvez effectuer les étapes 2 et 3 dans l'ordre de votre choix .

8. Les services Web java

Il existe plusieurs APIs standards pour la mise en œuvre et l'utilisation des services web en Java. Les principales que nous allons discuter ici sont les APIs Java pour XML : JAX-RPC (Appels de procédure distante) et SAAJ (SOAP with Attachments API for Java). Ces APIs sont destinées à vous aider à développer des services Web et des clients service Web.

8.1. SAAJ (SOAP with Attachment API for Java) : permet l'envoi et la réception de messages respectant les normes SOAP 1.1 et SOAP with attachments : cette API propose un niveau d'abstraction assez élevé permettant de simplifier l'usage de SOAP. Les classes de cette API sont regroupées dans le package javax.xml.soap.

Initialement, cette API était incluse dans JAXM. Depuis la version 1.1, elles ont été séparées. SAAJ propose des classes qui encapsulent les différents éléments d'un message SOAP: SOAPMessage, SOAPPart, SOAPEnvelope, SOAPHeader et SOAPBody) .

Tous les échanges de messages avec SOAP utilisent une connexion encapsulée dans la classe SOAPConnection. Cette classe permet la connexion directe entre l'émetteur et le receveur du ou des messages.

8.2 JAX-WS : JAX-WS (Java API for XML based Web Services) :

C'est une nouvelle API qui est fortement recommandé pour les nouveaux développements. Elle propose un modèle de programmation pour produire (côté serveur) ou consommer (côté client) des services web qui communiquent via des messages XML de type SOAP.

Elle a pour but de faciliter et simplifier le développement des services web notamment grâce à l'utilisation des annotations. JAX-WS fournit les spécifications pour le cœur du support des services web pour la plate-forme Java SE et Java EE.

Cette API repose sur plusieurs autres JSR :

- JSR 181 (Web Services MetaData for the Java Platform) : propose un ensemble d'annotations qui permettent de définir les services web.
- JSR 109 et JSR 921 (Implementing Enterprise Web Services) : décrit comment déployer, gérer et accéder aux services web via un serveur d'applications.
- JSR 183 (Web Services Message Security APIs) : décrit la sécurisation des messages SOAP.

9. Conclusion

On conclut que les services web sont des technologies récentes. Ils permettent d'apercevoir une nouvelle façon de concevoir et déployer les applications à travers le web. La grande force des services web est d'utiliser des standards (SOAP, WSDL, UDDI) ouverts et reconnus qui sont basés sur XML. L'utilisation de ces standards permet d'écrire des services web dans plusieurs langages et de les utiliser sur des systèmes d'exploitation différents.

Les services web sont des applications qui réalisent chacune une tâche spécifique et pour fournir une solution à une tâche complexe, on peut regrouper des services web pour n'en former qu'un seul; on parle alors de composition de services web. Nous allons détailler dans le chapitre suivant les concepts de base de la composition des services web

IV. Conception

1. Introduction

Depuis une décennie, l'architecture orientée service (AOS) a connu un grand engouement des entreprises de tout secteur et de toute taille en raison de ses avantages économiques et technologiques. Pour exploiter ces avantages, plusieurs organisations ont décidé de faire évoluer leurs systèmes patrimoniaux (SP) existants vers une telle architecture. La Migration vers la AOS est devenue l'une des techniques importantes de modernisation des SP. Elle aide les organisations d'une part à réutiliser leurs anciens systèmes existants en leurs donnant une nouvelle vie, et d'autre part à profiter des avantages des systèmes à base de service. Plusieurs approches de modernisation existent dans la littérature.**[11]**

Une remarque qui peut être faite est qu'un grand nombre de logiciels qui sont en cours d'exploitation sont devenues dépassées technologiquement, ce qui nécessite de les moderniser. [Sneed 2006] affirme que n'importe quel système qui date de plus de cinq ans, est concerné par une nécessité de modernisation. **[12]**

Parmi les systèmes qui sont largement répondu et qui sont concernée par la migration vers une architecture orienté service c'est les systèmes à base d'objet.

A travers notre projet, nous essayons de contribuer dans ce domaine, en proposons une solution qui permet de faire évoluer une application orientée objet vers une architecture orientée service.

A travers ce chapitre, nous allons décrire les différents composants utilisés pour implémenter notre outil.

Nous allons définir premièrement les objectifs, les étapes à suivre, puis nous allons présenter la conception globale, et la conception détaillée du système.

2. Objectif:

L'objectif principal de ce travail est de réaliser un système de transformation d'application orientée objet vers une autre application orientée service. L'outil développé prend en entrée un ensemble de classes en interaction entre eux et donne en sortie un ensemble des services.

Notre outil repose sur l'utilisation des applications orientées objet Java.

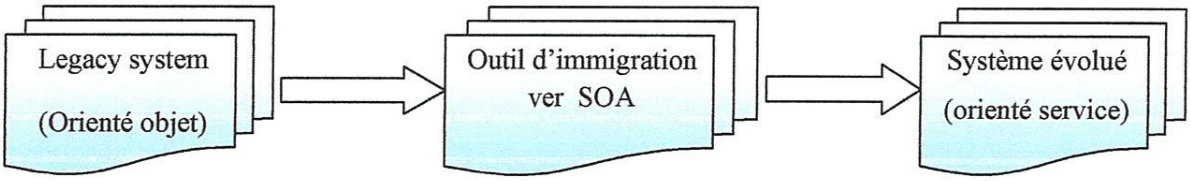

Figure 4.1 : vue générale de l'application

3. Fonctionnement générale de l'application :

Le fonctionnement de l'application est schématisé sur la figure 4 .2 :

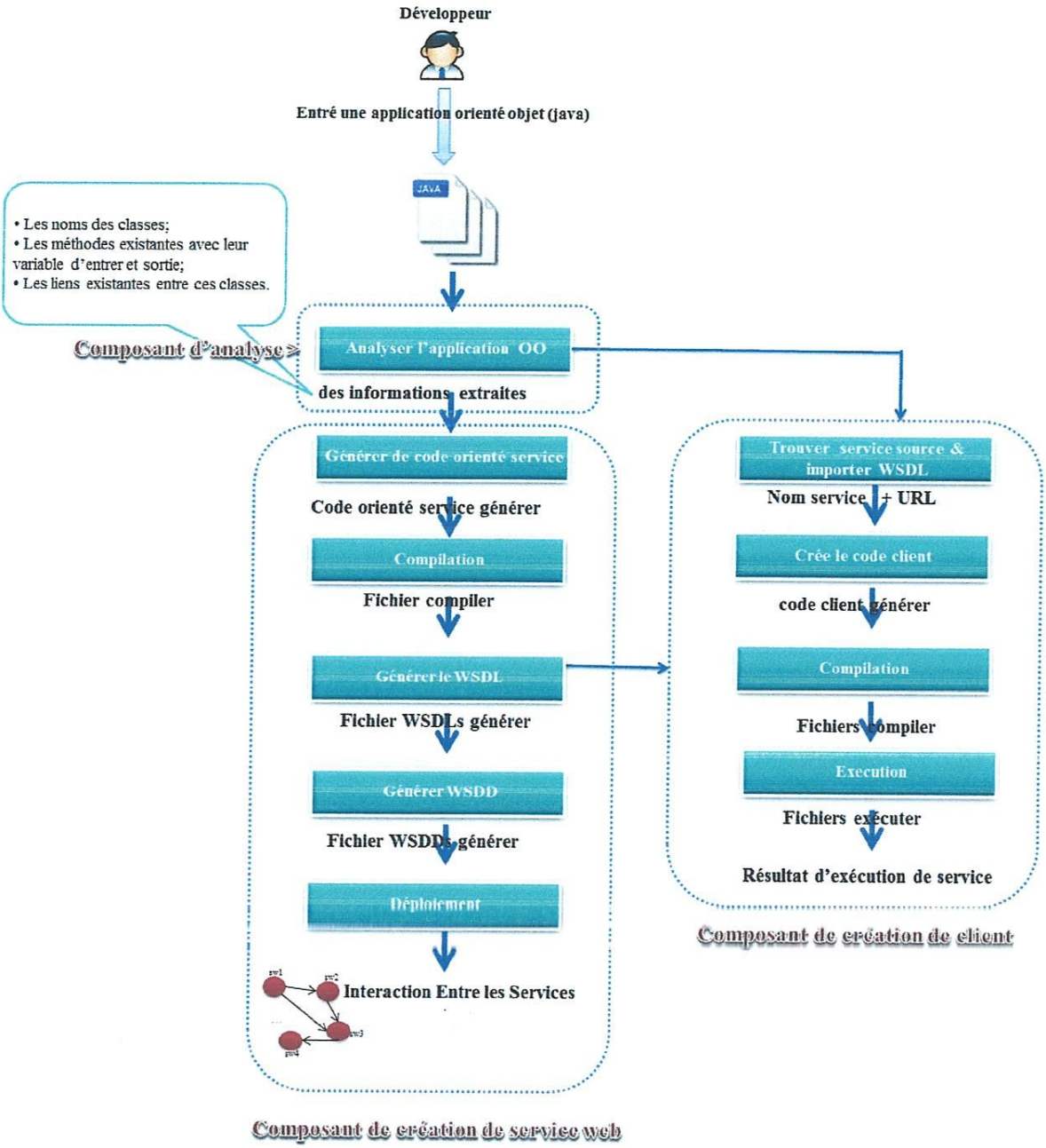

Figure 4.2 : Fonctionnement générale de l'application

- Sélectionner le projet qui contient le code source orienté objet comme entrée de notre outil,
- Lancer l'analyseur pour construire l'arbre syntaxique abstraite de chaque fichier (code source orienté objet) qui va nous aider à extraire les informations désirés.
- Générer le code de service web et cela par la modification de code source initial.
- Compiler le code obtenu.
- Générer les fichiers WSDL pour chaque service web en utilisant les fichiers compilés « .class »
- Générer les fichiers de configuration de déploiement WSDD.

- Déployer les services web.
- Générer un client pour le service web composite.
- Compiler le client généré.
- Tester la chorégraphie.

4. Processus de migration de l'application OO vers Une application SOA

Le processus de la génération d'une application orientée service à partir du code orienté objet passe par les étapes suivantes (voir Figure 4.3) :

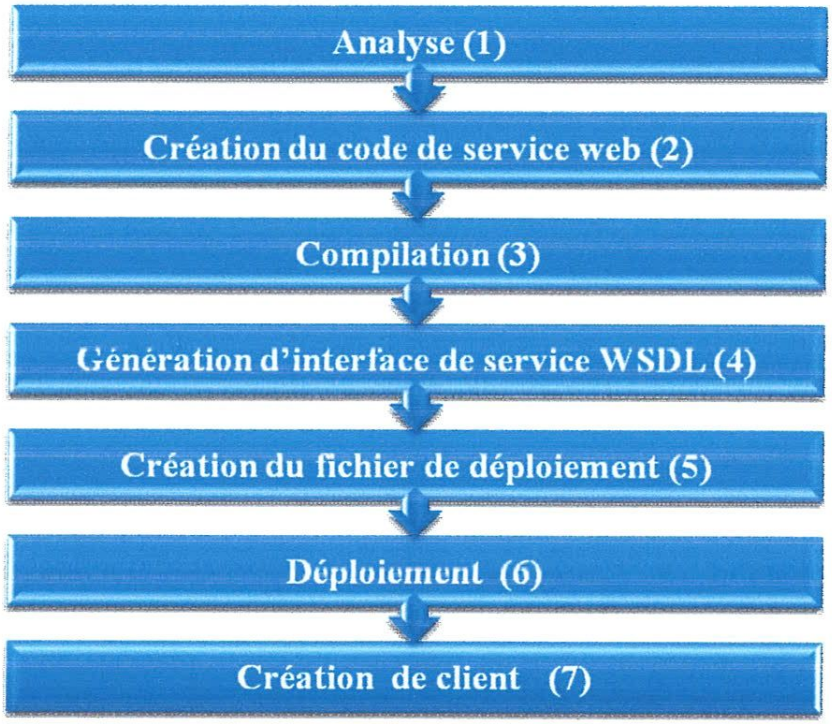

Figure 4.3 : Etapes de génération de l'application orientée service

1) **Étape d'analyse** : Cette étape consiste à analyser le code source orienté objet (les classes les méthodes et les instructions…) et extraire les informations concernant ce code pour les utiliser dans l'étape de création de service et dans l'étape de création du client

2) **Étape de création du code de service web:** cette étape permet de créer le code de service web (le nouveau code) à partir du code source orienté objet.

3) **Étape de compilation :** cette étape permet de compiler le nouveau code obtenu.

4) **Étape de génération de l'interface WSDL :** elle consiste à créer les interfaces qui contiennent la description des services Web (génération des fichiers WSDL).

5) Étape de création du fichier de déploiement : ce fichier est utilisé pour le déploiement. Il a une extension XML.

Le déploiement est effectué de façon automatique sur un serveur web (tel que Axis 1.4).

6) Étape de déploiement : c'est la publication de l'interface WSDL dans un serveur web. nous utilisons le résultat de l'étape de compilation et le résultat de l'étape de création du fichier de déploiement.

7) Étape création d'un client : C'est l'étape qui consiste à construire un client qui invoque les fonctionnalités de service web composite et l'exécuter pour obtenir le résultat.

Dans ce qui suit, nous allons présenter la conception de notre outil logiciel qui sera modélisé à l'aide du langage UML, en utilise le diagramme de composants, de classes et de séquences.

Donc, nous avons choisi UML pour ses points forts suivants : il permet :

- un gain de précision, de stabilité et encourage l'utilisation d'outils (langage formel et normalisé).
- l'analyse et facilite la compréhension de représentations abstraites complexes. De plus, son caractère polyvalent et sa souplesse en font un langage universel (support de communication performant).
- d'obtenir une modélisation de très haut niveau indépendante des langages et des environnements.
- de faire collaborer des participants de tous horizons autour d'un même document de synthèse.
- d'exprimer dans un seul modèle tous les aspects statiques, dynamiques, juridiques, spécifications,... etc.
- de documenter un projet.

5. Architecture globale

Comme nous avons présenté dans La figure 4.3 qui montre l'architecture globale de notre outil. Elle est constituée de trois principaux composants : le composant d'analyse, le composant de création de service web et le composant de création de client. Le composant

d'analyse est conçu indépendamment et permet d'interagir avec les deux autres composants (la création de service et la création de client).

La collaboration de ces composants permet d'atteindre l'objectif global de notre système qui est la création d'une application orientée service. Voici donc la figure 4.4 qui illustre l'architecture globale du système et les interactions entre les différents composants.

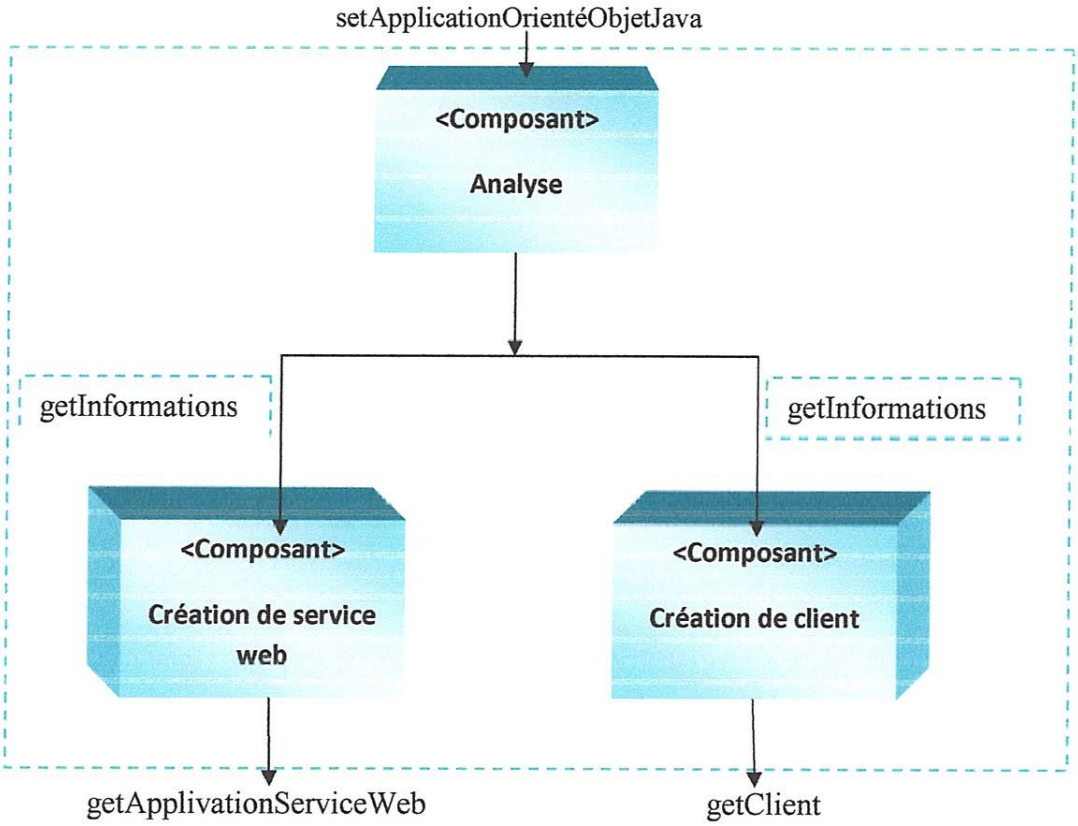

Figure 4.4 : Diagramme de composant de l'application.

- **Entrée:** l'application orienté objet en java.
- **Sortie :** l'application orientée service
- **Composant Analyse :** Analyse le code en entrée pour extraire des informations tel que les noms des classes, les noms des méthodes et les instructions… etc.
- **Composant Création de service web:** Ce composant contient toutes les fonctions nécessaires pour la création d'une application orientée service qui sont :
  - la création du code de service web ;
  - la compilation ;

- la génération de l'interface WSDL ;
- la génération du fichier de déploiement ;
- le déploiement.

➢ **Composant Création de client:** c'est la création du code client qui permet de tester le service web .

Nous allons détailler dans la partie qui suit les rôles des différents composants de l'outil dans le processus de création une application orientée service.

6. Conception détaillée

6.1. Composant Analyse

6.1.1. Description du composant analyse :

Ce composant a pour objectif d'analyser en entrée une application orientée objet (Java dans notre cas).

On va analyser les classes de cette application par l'utilisation d'un analyseur pour extraire des informations concernant ce code tel que : le nom de la classe, les méthodes, les paramètres d'entré et les méthodes appelées d'autre classes (les interactions entre les classes) ...etc. Cet analyseur permet de construire à partir d'un code source un arbre syntaxique abstrait qui a le but d'assurer l'analyse de toutes les instructions de la classe ; c'est à dire visiter tous les nœuds de code initial et les analyser un par un. Pour la création de cet arbre on a utilisé un outil nommé ASTParser.

- On va expliquer les étapes d'analyse par le diagramme des classes suivant :

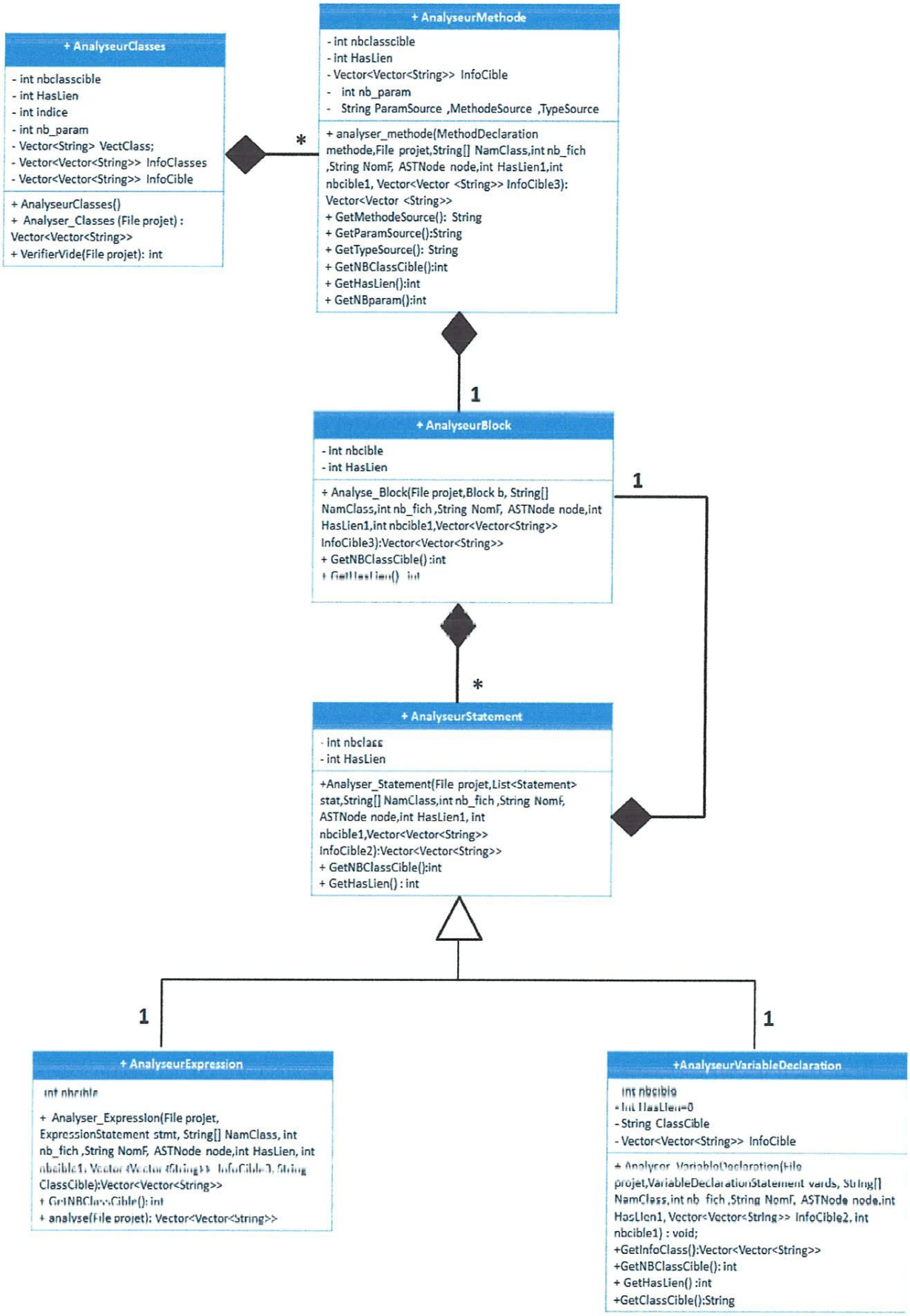

Figure 4.5 : Diagramme de classe pour package d'analyse

- **AnalyseurClasses :** Sert à analyser le projet classe par classe.
- **AnalyseurMéthode :** Permet d'analyser les méthodes d'une classe définie
- **AnalyseurBlock :** Permet d'analyser le block d'une méthode ou instruction bien défini.
- **AnalyseurStatement :** Permet d'analyser une instruction d'un block.
- **AnalyseurVariableDeclaration :** Permet d'analyser une instruction de type variable déclaration.
- **AnalyseurExperssion :** Permet d'analyser une instruction de type expression.

6.1.2.Fonctionnement du composant analyse:

Le diagramme de séquence qui décrit le scénario nominal pour l'analyse du projet orienté objet java est comme suite :

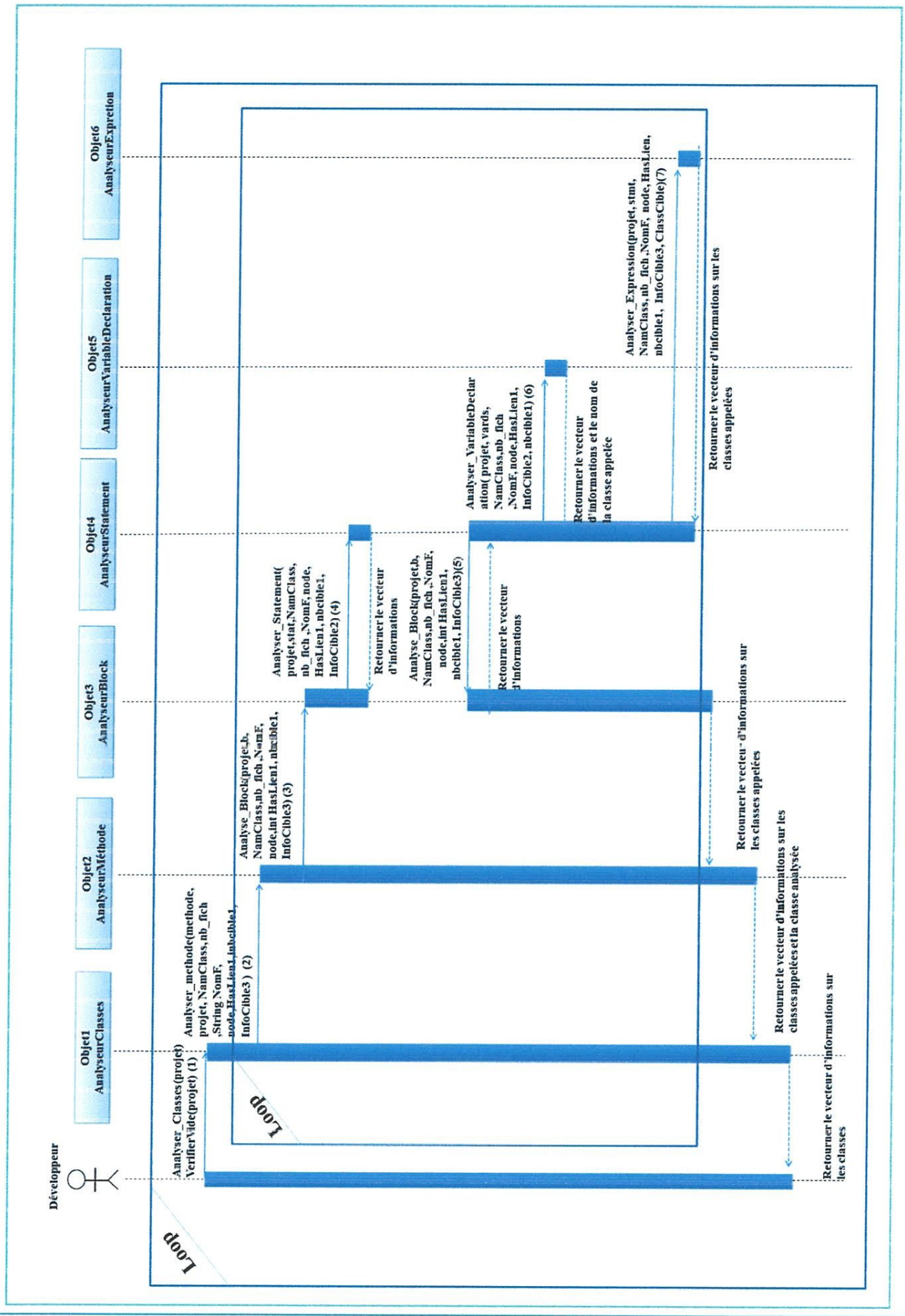

Figure 4.6 : Diagramme de séquence pour un scénario d'analyse

- On lance l'analyse de projet orienté objet java importé par le développeur (1) par l'appel de l'analyseur de classes en lui passant comme argument le nom de projet.
- Analyser les méthodes de chaque classe du projet (2) par l'appel de l'analyseur des méthodes en lui passant les arguments nécessaires , puis analyser le block de chaque méthode (3) par l'appel de l'analyseur de bloque en le passant les arguments nécessaires, ensuite analyser les instructions(4) par l'appel de l'analyseur des instructions en le passant les arguments nécessaires. L'instruction peut être une variable déclarée ou une expression ou une autre instruction qui peut contenir un block (5) par exemple for, while, do et try.
- Analyser la variable déclarée (6) par l'appel de l'analyseur de variable en lui passant les arguments nécessaires. Cet appel retourne comme résultat un vecteur d'informations et le nom de classes appelées si elle existe.
- Analyser l'expression (7) par l'appel de l'analyseur de l'expression en lui passant les arguments nécessaires. Cet appel retourne comme résultat un vecteur d'informations qui contient les informations sur la classe appelée si elle existe.
- Chaque analyseur à la fin retourne à son appelant un vecteur d'informations et d'autres informations qui sont nécessaires pour le bon fonctionnement de l'analyseur.

-Finalement, on stocke ces informations pour les utiliser ultérieurement.

- L'opération d'analyse se répète jusqu'à analyser tous les fichiers de projet (loop).

6.2. Composant Création d'un service web

6.2.1. Description du composant création d'un service web :

Ce composant conçu pour créer un service au niveau du code (le code orienté objet java), on réalise un outil qui fait la transformation d'un ensemble de classes orienté objet interconnecté entre eux par les appels des méthodes à un ensemble des services web qui sont en interactions entre eux par les invocations de leurs opérations. Donc on génère une interaction entre les services web au niveau du code.

Le mécanisme de création d'un service web est réalisé dans le diagramme des classes suivant :

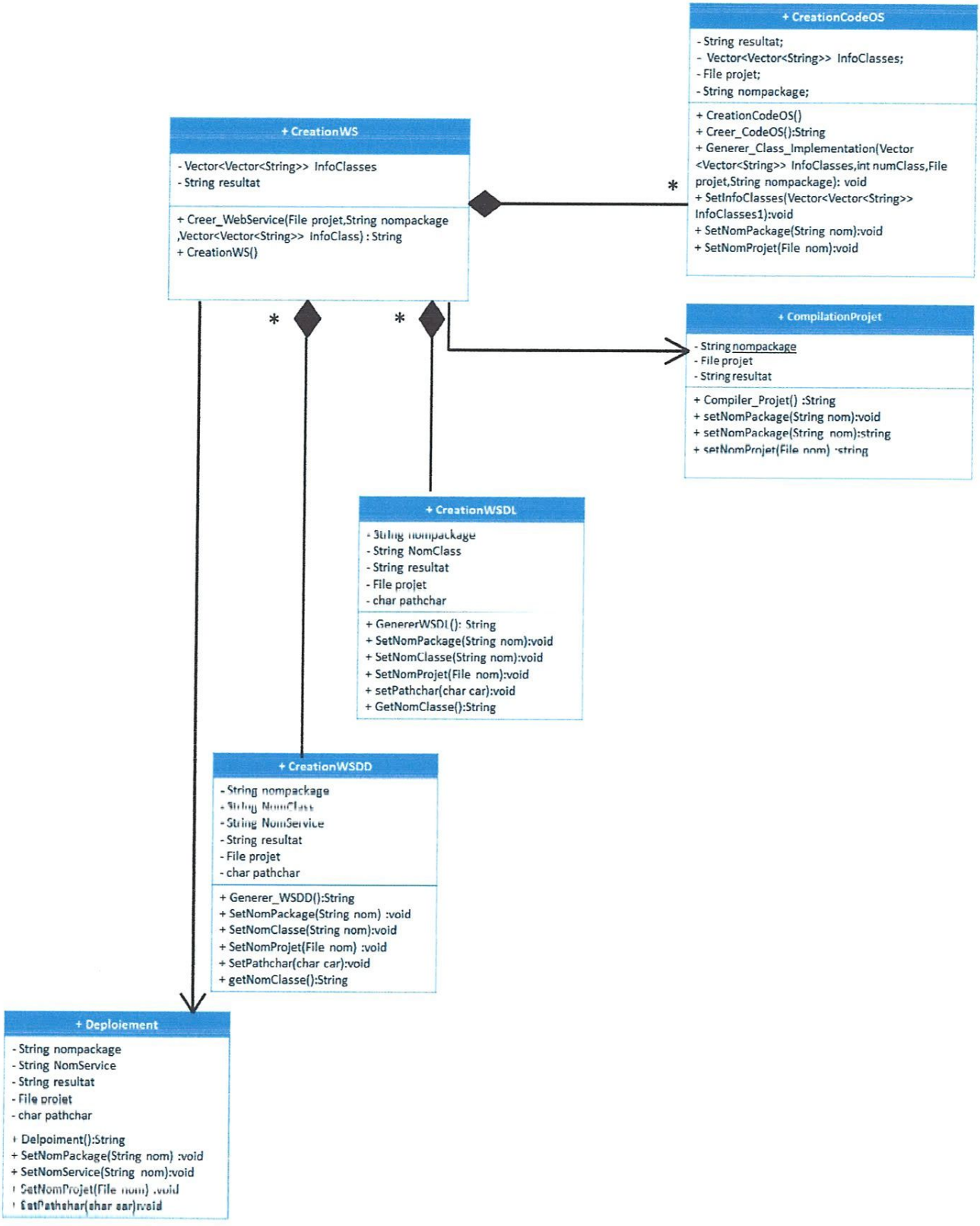

Figure 4.7 : Diagramme des classes de Composant création service web

A. Création du code du service web :

Dans cette étape, nous effectuons des modifications sur le code orienté objet analysé de la manière suivante :

- Chaque classe devient un service ;
- Chaque méthode devient une opération de service ;
- Chaque appel d'une méthode devient une invocation à une opération d'un service.

B. Compilation :

Dans cette étape on compile le code du service web, et pour effectuer cette opération, on fait un appel au compilateur java (API de JDK) à travers l'instruction ''Javac'' qui prend comme entrée un code java et produit en sortie un fichier dont l'extension est « .class ». Ce fichier est nécessaire pour la génération de WSDL et WSDD

C. Génération de l'interface WSDL:

C'est une étape qui consiste à générer le fichier WSDL qui décrit une interface publique d'accès au Service Web, ce fichier est constitué d'une suite d'éléments décrivant un service web sous forme d'un ensemble d'opérations. Notre génération de WSDL repose sur l'utilisation de l'outil java2wsdl.

D. Création du fichier de déploiement (WSDD):

Il y a plusieurs modes de déploiement, parmi ces modes on utilise le fichier WSDD. C'est un fichier XML qui porte l'extension « wsdd ». Ce fichier est appelé descripteur de déploiement. Il permet de faire un déploiement automatique (mode de déploiement avancé). Il permet de spécifier au serveur web (Axis) le nom du service, la classe associée et un certain nombre d'autres paramètres de configuration tels que la définition des droits d'accès aux méthodes du service.

E. Déploiement :

Dans cette étape, nous disposons désormais deux fichiers nécessaires pour effectuer le déploiement : le fichier « .class » et le fichier de déploiement « .wsdd ».

F. Fonctionnement du composant création d'un service web :

Dans cette section, on réalise un diagramme de séquence qui décrit le scénario nominal de la création d'un service web .

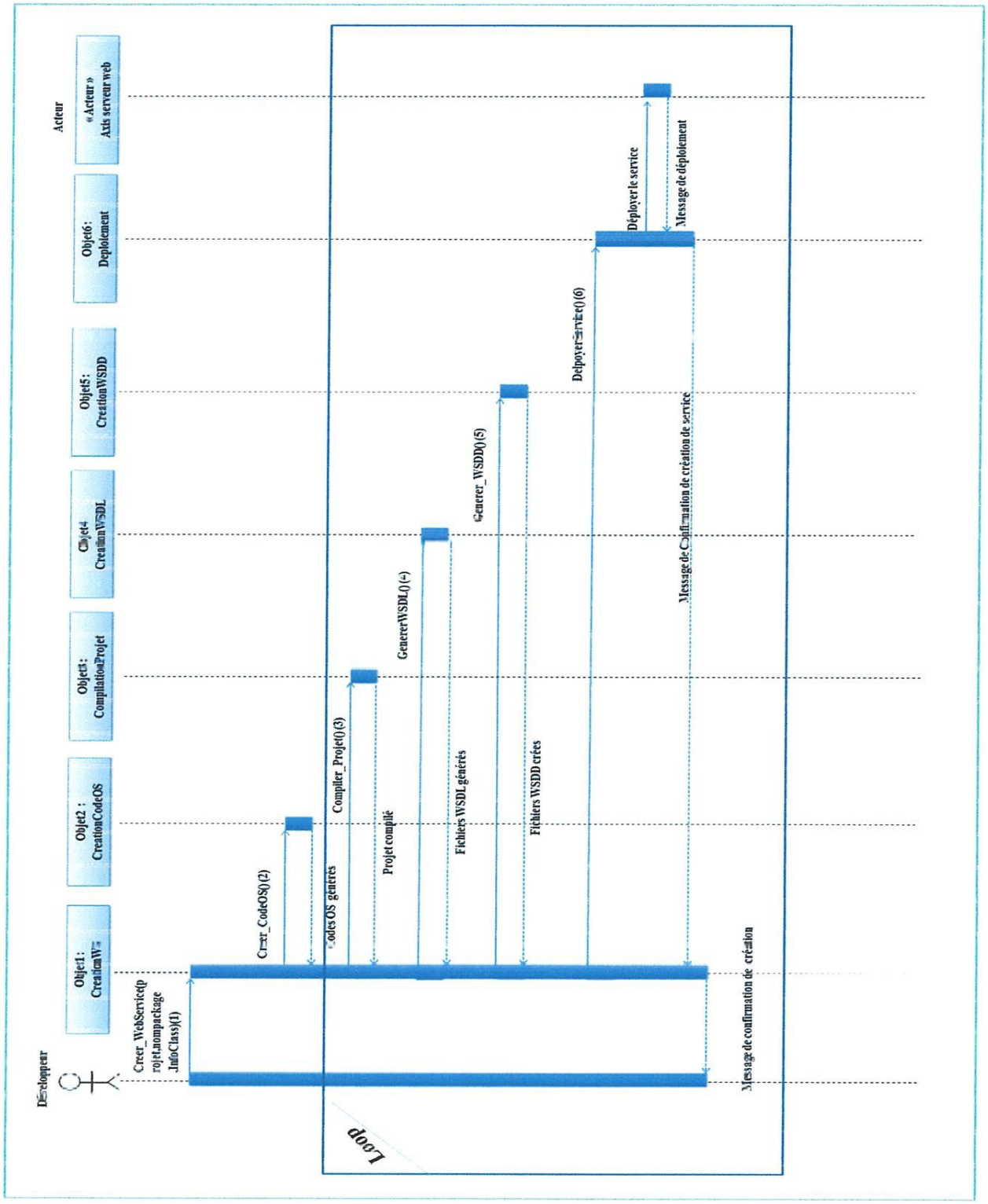

Figure 4.8 diagramme d'activité de la création d'un service web

- Dès que l'analyse de projet est terminée, le développeur peut lancer la création de code orienté service pour chaque classe (1). La création de ce nouveau code nécessite les informations récupérées par l'analyseur (2).

- Après la création, Il faut compiler le nouveau code (3), puis générer le fichier WSDL (4) et créer le fichier WSDD (5) pour chaque classe.

- Et enfin, déployer le service (6) au niveau de serveur web Axis et retourner un message de confirmation de déploiement au développeur.

6.3. Composant création de client :

6.3.1. Description du composant création de client :

Ce composant est conçu dans le but d'invoquer les services généré par notre outil.

Il permet de lancer toutes les fonctionnalités de l'application transformé sous forme de services

On va expliquer les étapes de création d'un client par le diagramme des classes suivant :

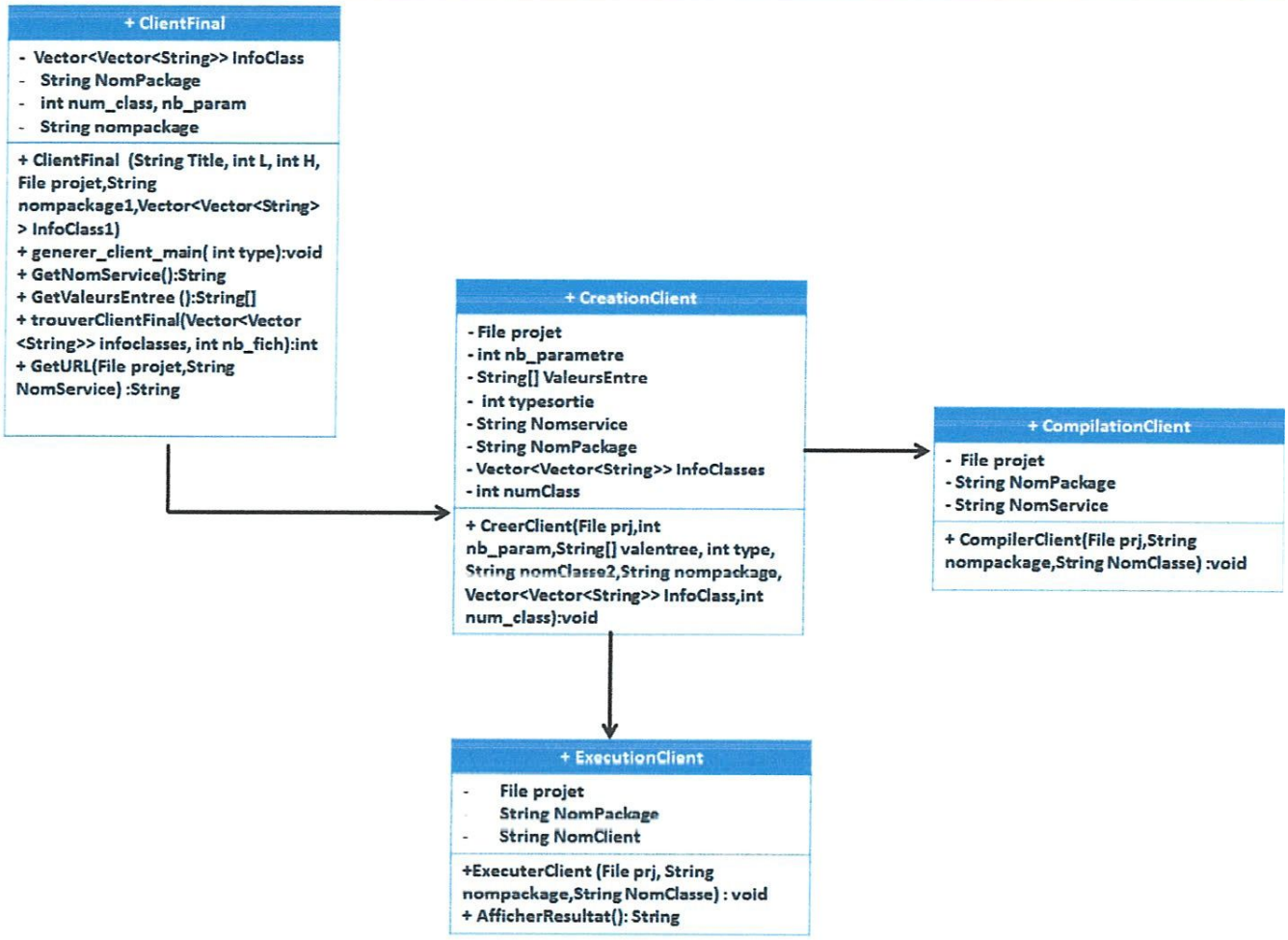

Figure 4.9 : Diagramme des classes de Composant « création client ».

- **ClientFinal :** Permet de trouver le service pour laquelle on va créer le client (service), d'acquérir les informations nécessaires pour la création de client final (par exemple les paramètres d'entrées), importer le fichier WSDL de classe composite et de chercher à l'intérieur de ce fichier l'URL de service demandé et d'attendre le résultat d'exécution de client après la création.
- **CreationClient :** Permet de créer le code source de client et de le stocker dans un répertoire.
- **CompilationClient :** Permet de compiler le code client final et de générer le fichier « class » associé.
- **ExecutionClient :** Permet de lancer l'exécution de client final pour communiqué avec le service web .

6.3.2. Fonctionnement du composant création de client :

Le diagramme de séquence décrit le scénario nominal pour la création du client est comme suite :

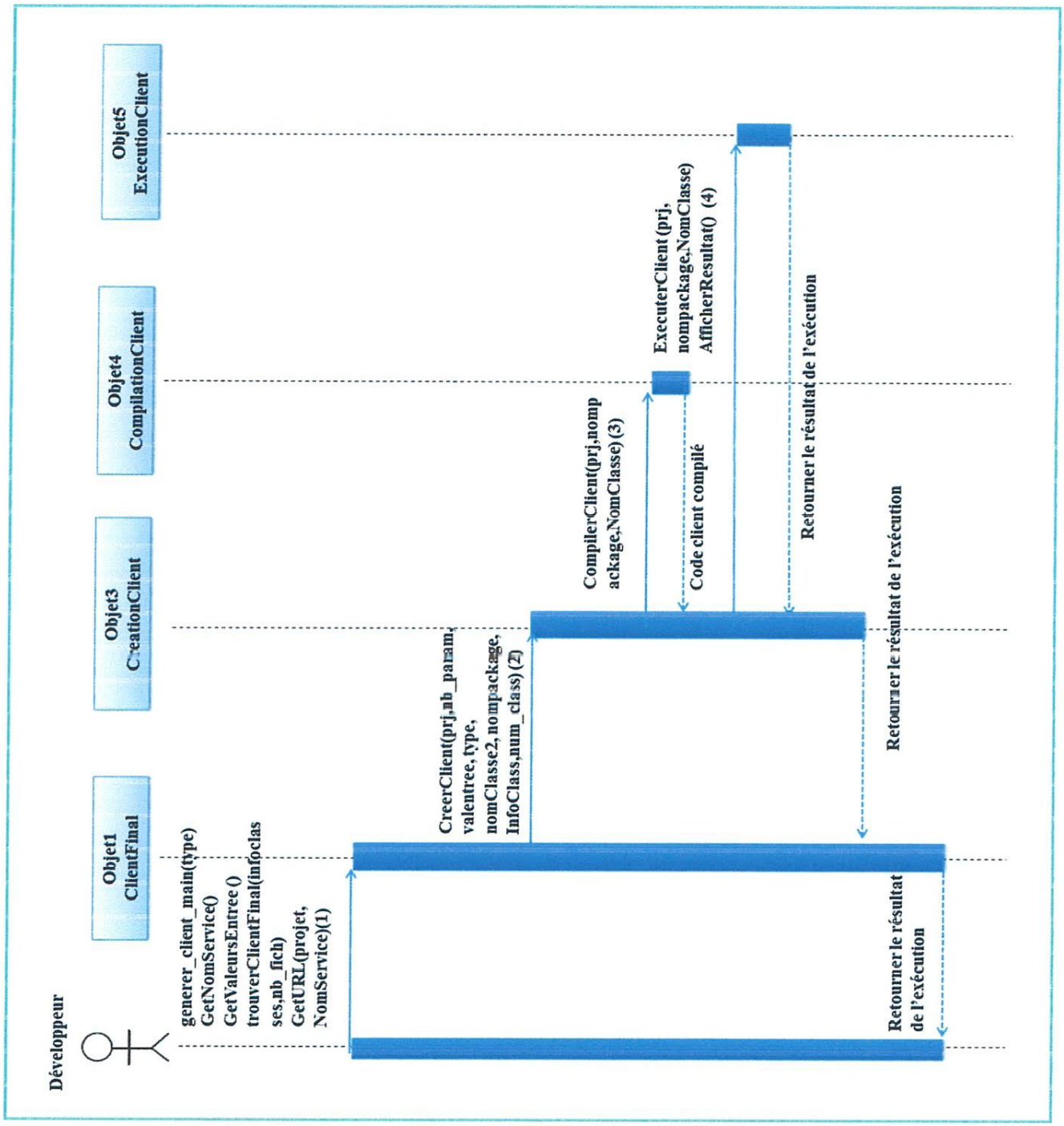

Figure 4.10 : Diagramme de séquence pour un scénario de création d'un client.

- Premièrement, le développeur lance la création de client de service web (1) en cherchant le nom de service racine, puis importe le fichier WSDL associé pour extraire l'URL de service.

- Ensuite, le développeur entre les valeurs des paramètres de service désigné et lance la création du code client (2), la compilation (3) et l'exécution (4).
- En fin, un résultat de l'exécution de client est retourné au développeur.

7. Conclusion

Nous avons présenté dans ce chapitre l'étude conceptuelle du notre outil ainsi que son architecture générale qui est composé du composant analyse, composant création de service et composant client. On outre, nous avons détaillé chaque composant en utilisant le diagramme de classe et le diagramme de séquence.

Le prochain chapitre sert à présenter les techniques utilisées pour implémenter l'application conçue dans ce chapitre. Ainsi que, une étude de cas sur un exemple concret.

V. Implémentation

1. Introduction

Après la présentation de l'architecture de notre outil dans le chapitre précédent, nous passons à la phase de réalisation, qui sera l'objectif de ce chapitre dans lequel on va décrire les différents aspects techniques liés à l'implémentation de notre projet. Nous commencerons par la présentation de l'environnement logiciel, sur lequel l'application a été réalisée. Ensuite, nous présenterons un exemple concret montrant le fonctionnement et l'interface graphique de notre outil.

2. Outils de développement

Notre application a été réalisée avec les outils suivants :

2.1. Le langage utilisé

Nous avons utilisé le langage Java pour implémenter notre outil. C'est un langage orienté objet simple conçue par James Gosling en 1994 chez Sun. Il est portable ; ne dépend pas d'une plateforme donnée. Il peut être utilisé sous Windows, sur Macintosh et sur d'autres plates-formes sans aucune modification. Java est donc un langage multiplateforme, ce qui permet aux développeurs d'écrire un code qu'ils peuvent exécuter dans tous les environnements. Le langage Java possède une riche bibliothèque de classes comprenant des fonctions diverses telles que les fonctions standards, le système de gestion de fichiers, les fonctions multimédia et beaucoup d'autres fonctionnalités.

2.2. La plateforme Eclipse

Nous avons utilisé la plateforme Eclipse pour réaliser notre outil. C'est un environnement de développement intégré (Integrated Development Environment-IDE) qui a le but de fournir une plate-forme modulaire pour permettre de réaliser des développements informatiques. Il est développé par IBM. Il a depuis été rendu open source et son évolution est maintenant gérée par la fondation Eclipse. Il est distribué depuis Septembre 2005 sous licence CPL (Common Public Licence).

Sa conception modulaire est basée sur un moteur de chargement de plugins et de différents plugins, ce qui fait d'Eclipse une boite à outils facilement améliorable ou modifiable. La licence d'Eclipse permet de fournir différents plugins : open source, gratuits ou payants.

2.3. Tomcat

Tomcat est un serveur web qui gère les servlets J2EE et les JSP. Il est souvent employé en combinaison avec un serveur web Apache : Apache s'occupe de toutes les pages web traditionnelles, et Tomcat uniquement des pages d'une application web Java. Tomcat implémente les spécifications des servlets et des JSP de Sun Microsystems.

Il inclut des outils pour la configuration et la gestion, mais peut également être configuré en éditant des fichiers de configuration XML. Comme Tomcat inclut un serveur HTTP interne, il est aussi considéré comme un serveur HTTP, il est aussi une plateforme pour le développement et le déploiement d'applications web et des web services.

TOMcat a été écrit en langage Java, il peut donc s'exécuter via la JVM (machine virtuelle java) sur n'importe quel système d'exploitation.

2.4. Apache Axis 1.4

Axis (Apache eXtensible Interaction System) est un moteur de Web services qui implémente le protocole SOAP. Il est un projet open-source du groupe Apache. Son but est de proposer un ensemble d'outils pour faciliter le développement, le déploiement et l'utilisation des services web écrits en java. Axis propose de simplifier au maximum les taches pour la création et l'utilisation des services web. Il permet notamment de générer automatiquement les fichiers WSDL à partir d'une classe java et le code nécessaire à l'appel du service web .

Le **JDT** (Java Development Tools) de Eclipse fournit des API[1] pour manipuler le code source Java, de détecter les erreurs, d'effectuer des compilations, d'effectues des changements dans le code source. Tout cela peut être réalisé par l'utilisation de ASTParser.

ASTParser

est un analyseur qui permet de construire à partir d'un code source java un arbre syntaxique abstrait (abstract syntax tree, ou AST, e, anglais). L'arbre syntaxique abstrait est la façon dont Eclipse regarde votre code source : tous les fichiers source java sont entièrement représentés sous forme d'arborescence de nœuds AST. Ces nœuds sont tous les sous-classes de la classe ASTNode. Chaque sous-classe est spécialisée dans un élément du langage de programmation Java. Par exemple, il y a des nœuds pour les déclarations de méthodes (MethodDeclaration), déclaration de variables (VariableDeclarationFragment)...etc .

2.5. JDK

On utilise l'API de JDK qui permet la compilation d'un code source Java. Pour compiler un programme en Java, il faut utiliser l'instruction javac suivi du nom du fichier qui produit un pseudo-code (byte code) dont l'extension « .class ». Cette API nous aide aussi à compiler le nouveau code orienté service généré par notre outil.

2.6. JAVA2WSDL:

C'est une API qui crée un fichier WSDL à partir d'une classe Java. Elle permet d'aider les développeurs pour créer facilement et rapidement les fichiers WSDLs.

3. Etude de cas et test

Pour assurer et expliquer le fonctionnement de notre outil, on va exploiter l'outil sur un exemple d'application orienté objets.

Cet exemple va nous permettre d'expérimenter le passage d'une application orientée objet vers une application orientée service et le déploiement de cet application sur un serveur web (Apache Axis). Usuellement, Axis est déployé sur le serveur (conteneur de servlets)

---

comme Apache Tomcat, mais peut tout aussi bien l'être sur un autre serveur d'applications. A la fin, nous allons montrer comment développer et exécuter un client Java pour consommer le web service

Nous allons tout d'abord installer sur la machine le serveur Tomcat 5.5 et l'application Axis 1.4.

4. Architecture de l'application

L'architecture de notre d'application est constituée de 5 packages :

1 le package qui contient les classes de l'analyseur

2) le package qui contient les classes principales

3) le package qui contient les classes de création des web service

4) le package qui contient les classes de génération du client final

4. 1. Le package Analyseur

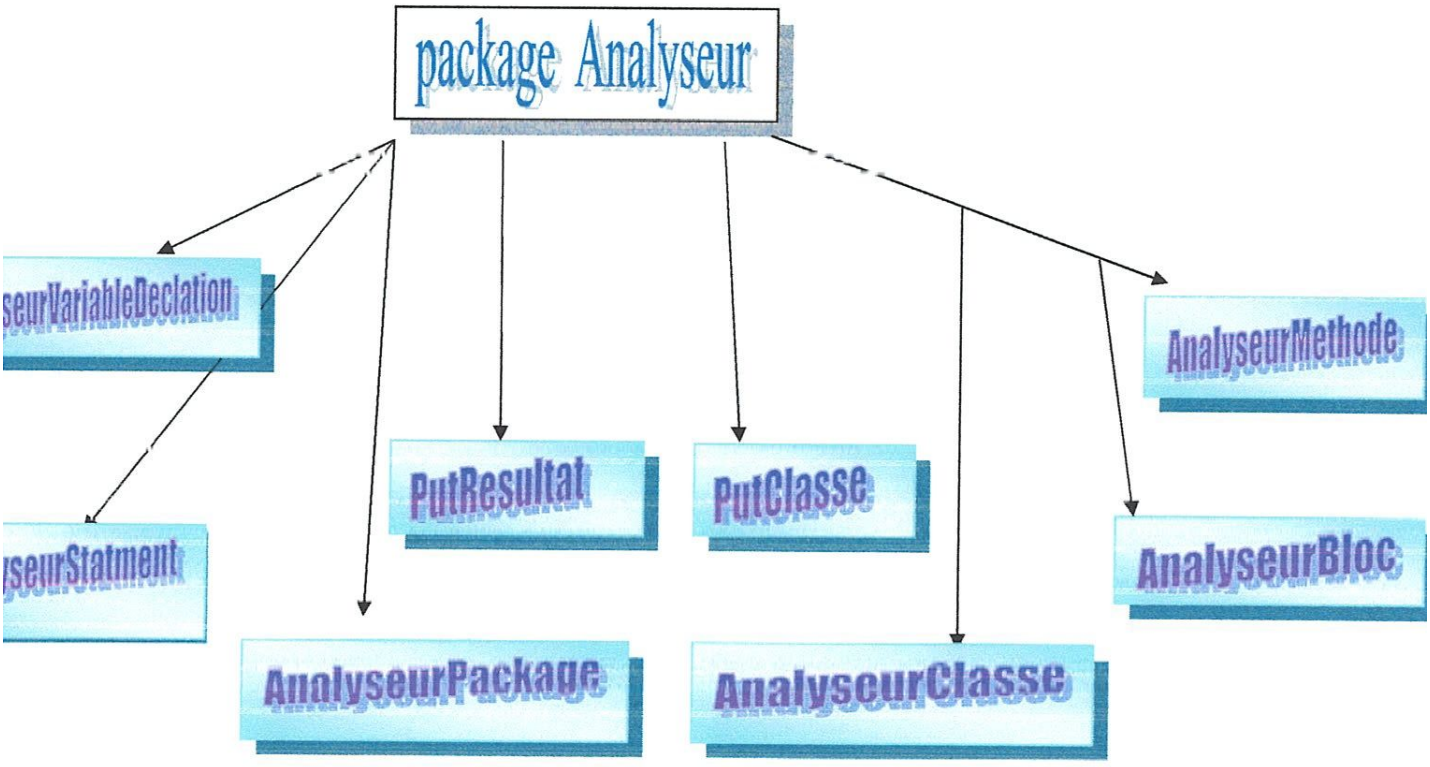

Figure 5.1 : Architecture générale du package Analyseur

Ce package contient 5 classe qui communiquent entre eux par des méthodes différente.

a. analyseur du package elle se charge de l'extraction du nom du package par la fonction GETNAME PACKAGE qui prend comme paramètre le fichier à analyser. Cela est considèré comme la première étape de l'analyse en utilisant l'outil **STP Parser** qui transforme les classes en un arbre syntaxique.

La deuxième classe est la classe :

b. Analyseur Classe.

Cette classe se charge de l'analyse des différentes classes qui constituent l'application. Dans cette classe nous avons traité les informations dans des vecteur pour extraire les méthodes source, les classes sources, les méthodes source et les type source.

Cette classe fait appel à la classe Analyseur Méthode qui se charge par le traitement des différente méthodes, à l'aide de l'outil Parseur nous pouvons distinguer si la classe a un lien avec une autre classe cible.

c. Analyseur Méthode

Elle a un lien avec la classe **Analyseur Classe** en réalité c'est elle qui retourne le type source par la fonction GET TYPE SOURCE et aussi les paramètres source ainsi que les classes cibles et les liens. Elle fait appel à la classe Analyseur Bloc .l'outil ASTP Parser se charge de la détection des méthodes et leur paramètres

D. Analyseur Bloc

C'est la troisième classe dans le package Analyseur l'une des plus importante classe dans ce package elle fait son travail puis stock le résultat dans un vecteur. Cela est pris en charge par la classe **AnalyseurStatment .**

Pour la classe **AnalyseurStatment** on trouve l'analyse des expressions. Cette dernière est reliée avec la classe **AnalyseurVariableDeclaration.**

e. AnalyseurVariableDeclaration se charge du traitement des variables qui sont représentées par des nœuds dans l'arbre syntaxique.

Les deux dernières classes se chargent des résultats

f. Put Classe : elle retourne les informations sur la classe (le nom de la classe, la méthode, le type ainsi que les paramètres).

g. Put Résultat : met les résultats finaux dans des vecteurs pour les classer dans la table par la suite.

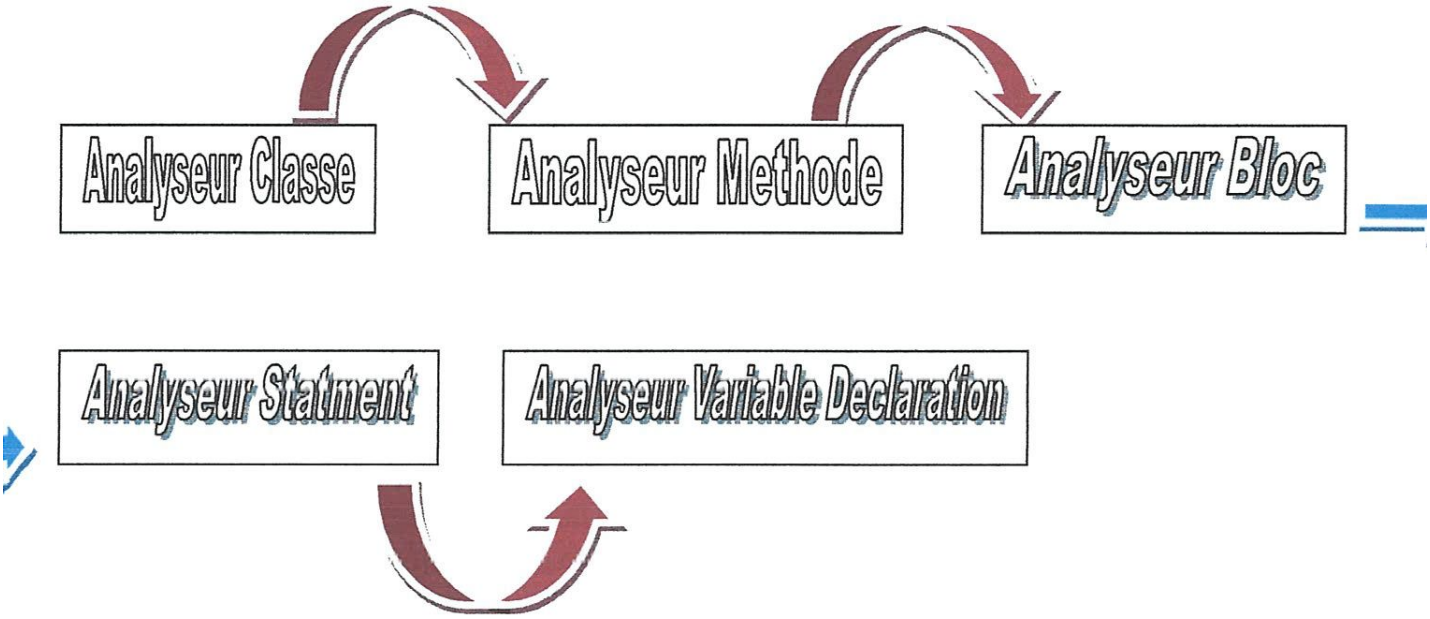

Figure 5.2 : représentation des appels entre les classes

4.2. le package Classes principale

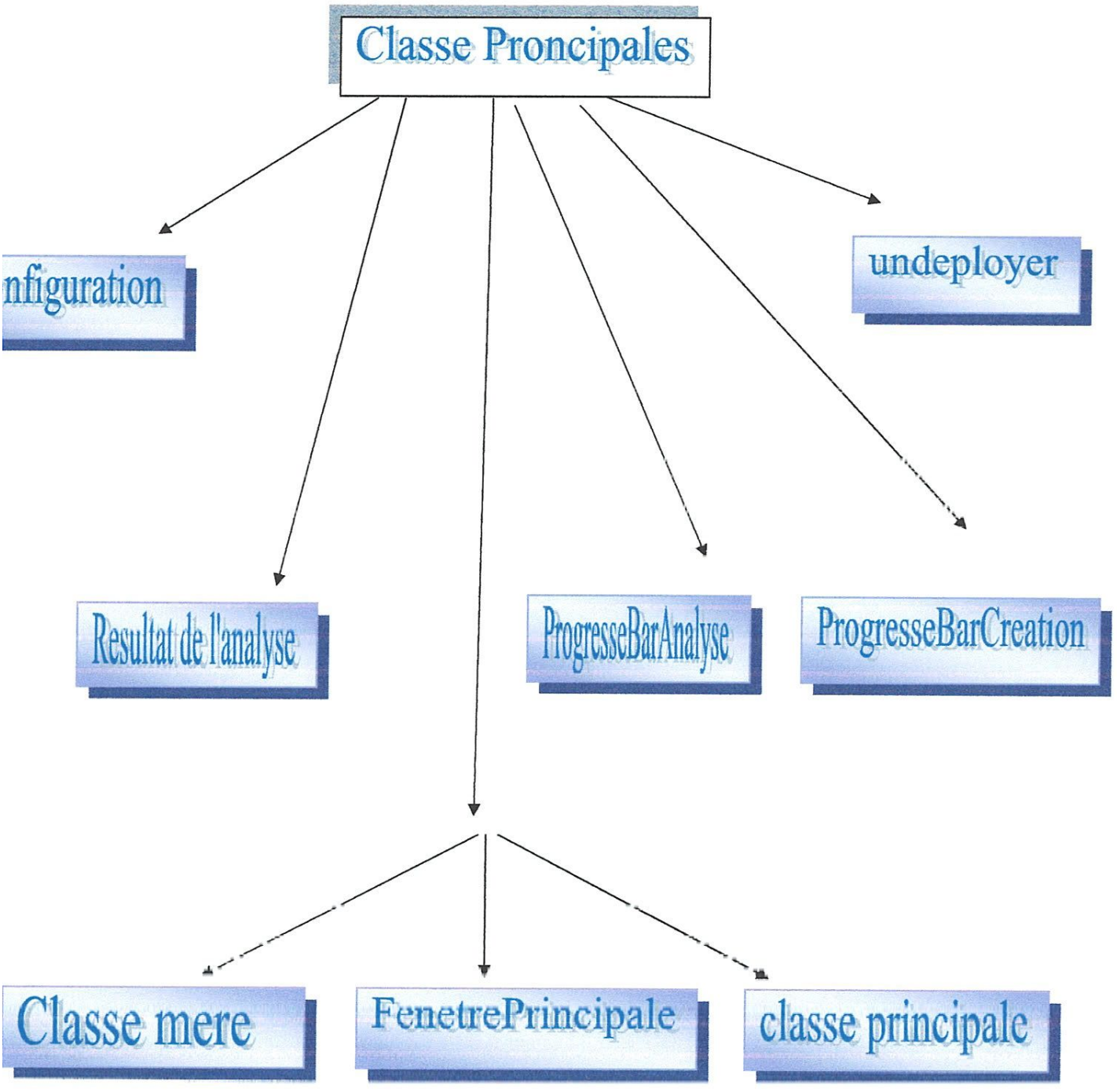

Figure 5.3: Le package : classe principales

Ce package contient 8 classes la première classe est :

a. La classe principale

Cette classe affecte les actions aux boutons concernés

b. La classe configuration

Cette classe présente l'interface de configuration pour ce qui concerne la vérification du chemin d'axis ainsi que le bouton Vérifier avec sa fonction, pour que l'axis fonctionne bien le serveur TOMCAT doit être lancé .

c. La classe Affichage Résultat d'analyse

Dans cette phase on a utilisé la notion Jtable: JTable un composant Swing complexe pour présenter et éditer des données sous forme de table, donc les résultats de l'analyseur vont être représentés dans une table de donné, dans cette table on trouve les classes sources avec les méthodes source, les type sources ainsi que les paramètres sources et les classes cibles.

d. La classe fenêtre principale

Cette classe présente l'interface pour l'application par d'autre façon c'est la première vue de l'application.

e. La classe progressBarAnalyse

Cette classe se charge de faire la construction de la barre d'analyse pour visualiser la progression d'analyse du projet.

f. La classe ProgressBarCreationSW

Cette classe se charge de faire la construction de la barre de création du web service pour visualiser la progression de création des services du projet.

g. La classe TestEtConsultation

A travers cette classe on peut afficher les services déployés dans le serveur axis pour le fonctionnement on a utilisé la classe desktop avec sa méthode GET desktop qui peut lancer le navigateur par défaut avec une URL donnée.

h. La classe Undeployer

Cette classe permet de désactiver le déploiement des services dans le serveur axis. Au cours du développement de cette classe nous avons utilisé les deux classes Input Stream et Output Stream pour gérer les entrées sorties avec l'exécution des deux fichier :

C:\\VrfAxis.dat et C:\\Résultat.txt"," pour la désactivation du déploiement.

4.3. le package Création SW

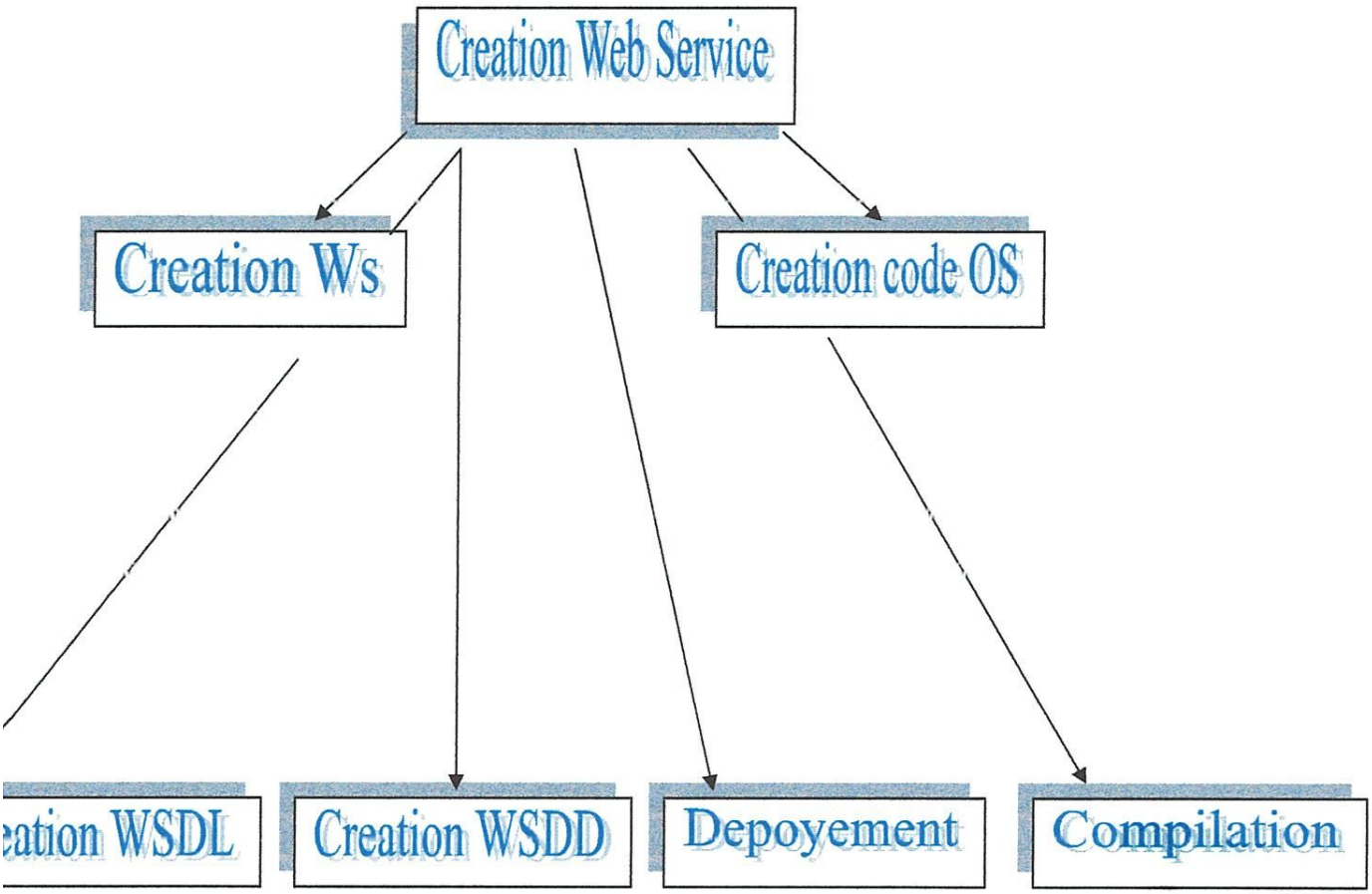

Figure 5.4: Le package : Création web service

Le package Création SW contient 6 classes.

A. La classe Compilation conçu pour faire la compilation du projet, dans cette phase nous avons utilisé les fichiers **exp4.bat** et **Résultat.txt** le premier utilisé comme sortie et exécute les instructions suivantes :

```
String prj= projet+"\\impl\\*.java";
String s11="cd\\";

String s22=pathchar +":";  // C ou D....

String s33="cd "+pathchar+":\\program files\\Apache Software Foundation\\Tomcat
5.5\\webapps\\axis";

String s44="javac -d build "+ prj;
```

Figure 5.5: Le fichier EXP4.bat

L'exécution du fichier se fait par la classe **RUNTIME** en exécutant la méthode **getRuntime().exec,** puis le résultat de l'exécution est mis dans le deuxième fichier.

b. La classe CreationCodeOS

Dans cette classe nous avons implémenté la méthode **Créer_CodeOS()** qui fait le test. Si la classe a un lien avec une classe cible en utilisant le vecteur infoclasses qui contient les informations sur la classe Si la classe n'a aucun lien on fait la copie des classes sans aucun ajout par la méthode **copie_class,** sinon s'il existe un lien avec d'autre classes cibles nous implémentons la methode **Generer_Class_Implementation** et on ajoute les biblio suivantes :

```
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceException
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
```

Figure 5.6: Bibliothèque pour la méthode générer classe implémentation

c. La Classe création WS

Cette classe se charge de la création des services web. D'abord nous allons créer le code orienté service en faisant l'appel à la méthode **Créer_CodeOS(),** puis nous lançons la compilation du projet par l'appel de la méthode **compile()** ensuite la génération du fichier wsdl par l'appel de la méthode **creerwsdl()**, enfin nous générons le fichier WSDD et puis nous passons à la phase de déploiement du service. le fichier wsdd est généré par l'appel de la méthode **creewsdd()** et le déploiement par la methode **dep().**

d.La classe CreationWSDD

Dans cette classe nous avons créé une fonction Generer_WSDD() pour créer un fichier déployer .wsdd, le fichier wsdd est un descripteur de déploiement et nous aurons comme sortie les messages suivants:

```
<deployment
xmlns="+c+"http://xml.apache.org/axis/wsdd/"+c>.
<xmlns:java="+c+"http://xml.apache.org/axis/wsdd/providers/
java"+c+">"<service name="+c+NomService+c+"
style="+c+"RPC"+c+">"Parametername="+c+"className"+c+"v
alue="+c+nompackage+".impl."+NomService+c+"/>;<paramete
r name="+c+"allowedMethods"+c+"
value="+c+"*"+c+"/>"</deployment>
```

Figure 5.7 Le fichier WSDD

e.La classe CreationWSDL

Pour cette classe nous avons implémenté la méthode GenererWSDL() pour créer le fichier exmp4.bat qui contient les instructions suivantes pour générer le fichier wsdl :

```
char  c=""'String s1="cd\\";

String s2=pathchar+":";

String s3="cd "+pathchar+":\\program files\\Apache Software
Foundation\\Tomcat 5.5\\webapps\\axis"

String s5="java org.apache.axis.wsdl.Java2WSDL -o
"+NomClass+".wsdl -
l"+c+"http://localhost:8080/axis/services/"+NomClass+c+" "
```

Figure 5.8 :Exmp4.bat pour le fichier wsdl

f. La classe compilation

Pour cette classe nous avons implémenté la method Delpoiment() pour créer le fichier emp4.bat qui contient les instructions suivantes pour faire le deploiement :

```
String s2=pathchar+":";

String s3="cd "+pathchar+":\\program files\\Apache Software
Foundation\\Tomcat
5.5\\webapps\\axis\\build\\"+nompackage+"\\impl\\";

 String s5="java
org.apache.axis.client.AdminClientdeploy"+NomService+".wsdd";
```

Figure 5.9 : méthode de compilation

4.4 Le package CreationClientFinal

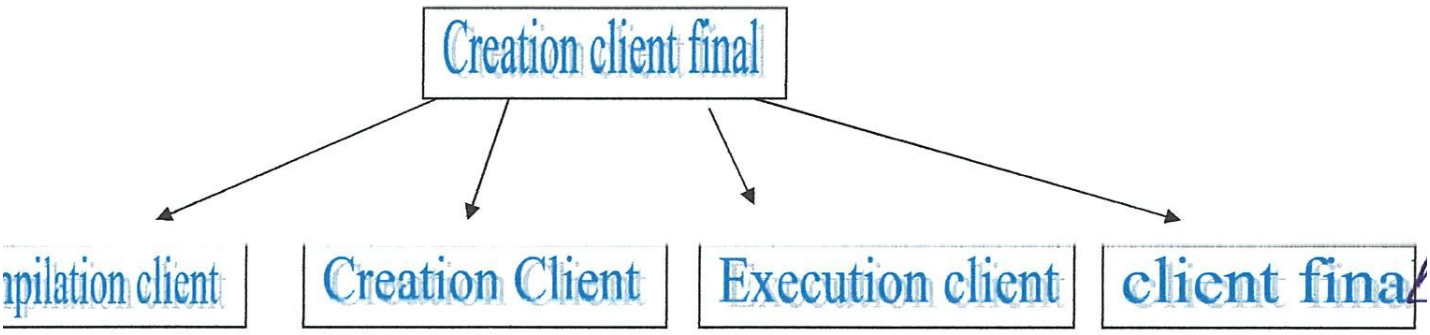

Figure 5.10 : Création client final

a. La classe création client finale

Pour cette classe nous avons créé le fichier **FileOutputStream** dans lequel nous aurons comme flux de sortie les bibliothèques qui concerne le web service web :

```
(import java.net.MalformedURLException;
("import java.net.URL;");
("import java.rmi.RemoteException;");
("import javax.xml.namespace.QName;");
("import javax.xml.rpc.ServiceException;");
("import org.apache.axis.client.Call;");
("import org.apache.axis.client.Service;");
```

Figure 5.11 : bibliothèque pour le client final

Nous aurons aussi l'instanciation du service web et l'appel des services par les instructions suivantes :

```
Service service = new service() ;

Call appel =(call)service.creatCall

appel.setTargetEnpointAddress(new URL(url))
```

Figure 5.12 : Instanciation du service

Puis nous lançons l'appel aux classe Exécution et compilation, pour la compilation et l'exécution du client.

b.La classe Executionclient

Pour l'exécution nous avons utilisé un fichier outputstream D://expl.bat que nous executons à partir de la classe runtime en utilisant la méthode get.

c.La classe compilation client

La même chose pour la compilation pour ce qui concerne le fichier D://expl.bat mais juste le flux de donné du fichier a changé. Nous avons utilisé les instructions suivantes pour la compilation

```
String s1="cd\\";
String s2="c:";
String s3="cd "+s7;
Strings5="javac -d build
build\\"+NomPackage+"\\impl\\Client_"+NomService+".
```

Figure 5.13: instruction de compilation

5. Présentation de notre exemple

Notre exemple est un projet java. Il est composé de cinq classes dont l'objectif principal est de réparer un objet matériel déclaré par une simple demande de réparation par l'employée. Ce dernier indique dans sa demande la description de l'objet matériel. Il envoie sa demande est le fait passer au service concerné par la réparation des objets (service maintenance).

Alors ce service essaye d'appeler un réparateur de sa part. S'il a réglé la panne alors c'est bon sinon le service fait l'achat du composant qui n'est pas réparable de puis un établissement de ventes. Ce dernier vérifie son stock s'il contient ce composant sinon il fait une commande vers un fournisseur pour l'acheter.

5.1 Le diagramme des classes

Les classes cet exemple sont présentées dans le diagramme suivant :

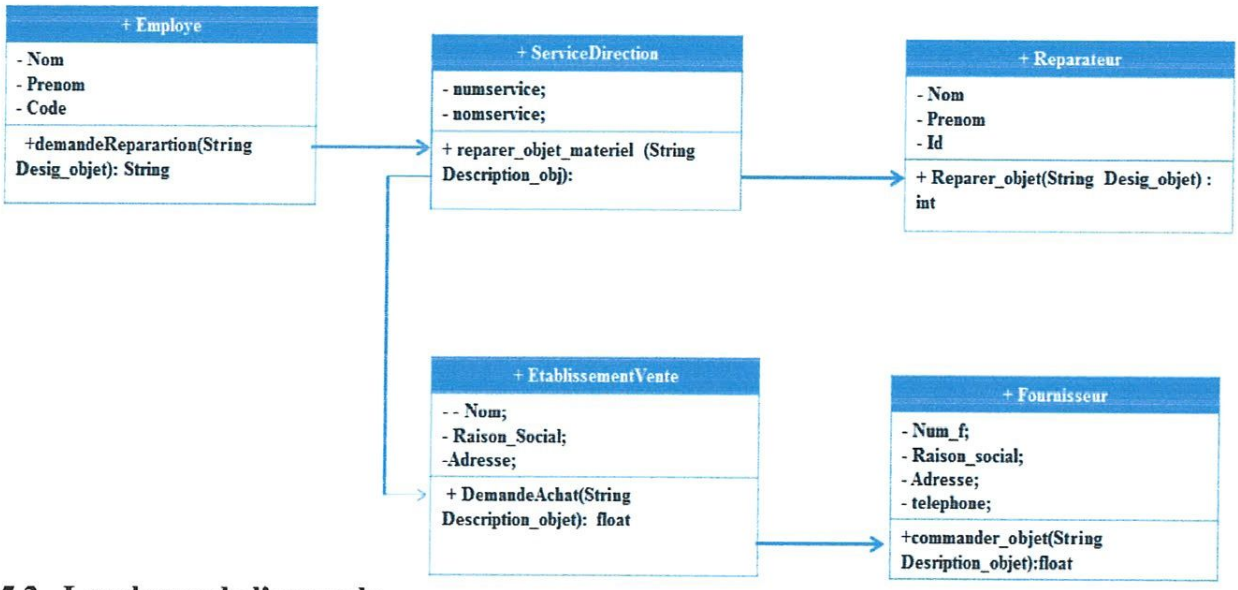

Figure 5.14: Le diagramme des classes de notre exemple

5.2. Les classes de l'exemple

5.2.1. Classe 1 :Employe

Cette classe possède la méthode suivante :

Methode :*demandeReparartion*

L'employé fait une demande de réparation pour réparer un objet matériel nécessaire pour accomplir son travail. Cette demande est déposée au niveau d'un service spécial qui est par exemple le service maintenance.

```
public String demandeReparartion(String Desig_objet)
{       String resultat=null;
        ServiceDirectionserv = new ServiceDirection();
         resultat = serv.reparer_objet_materiel(Desig_objet);
    return resultat;
}
```

Figure 5.15: Le code source de la méthode demandeReparartion

5.2.2. Classe 2 : ServiceDirection

Cette classe possède la méthode suivante :

Méthode : *reparer_objet_materiel*

Le service fait appel à un réparateur pour essayer de détecter la panne et de réparer cet objet s'il est possible, sinon il fait appel à un établissement de vente pour lui fournir la pièce nécessaire.

```
public String reparer_objet_materiel(String Description_obj){
        int type_panne=0;
        float prix=0;
        String resultat=null;
        Reparateur reparateur = new Reparateur();
        type_panne = reparateur.Reparer_objet(Description_obj);
        if(type_panne==0)  resultat= "l'objet est reparer (solution logiciel, rinstalation)";
        if(type_panne==1)
        {  EtablissementVente etabvente = new EtablissementVente();
        prix= etabvente.DemandeAchat(Description_obj);
        if(prix==0) resultat= "l'objet n'est pas repare ( reste en attente ) )";
        else resultat= "l'objet est repare (solution Matriel, prix = "+prix+" )";     }
```

Figure 5.16: Le code source de la méthode reparer_objet_materiel

5.2.3 .Classe 3 :Reparateur

Cette classe possède la méthode suivante :

Methode :Reparer objet

Si la panne est de type 0 ça veut dire une logiciel, le réparateur doit faire son travail par exemple la réinstallation du système d'exploitation sinon si la panne est de type 1 il doit retourner un message qui indique que la solution est matériel et qu'il faut acheter une la pièce.

```
{ int type_panne;
      String resultat=null;
      Random rand = new Random();
      type_panne= rand.nextInt(2);
      if(type_panne==0)  resultat= "l'objet
est repare (solution logiciel, rinstalation
XP)";
      else  resultat= "l'objet n'est pas
repare (solution Materiel, Acheter piece)";
            return type_panne;
```

Figure 5.17 Le code source de la méthode reparer_objet

1. démarrer : permet d'ouvrir la deuxième fenêtre.
2. Quitter : pour fermer l'application
3. Aide : pour afficher l'aide sur l'outil.

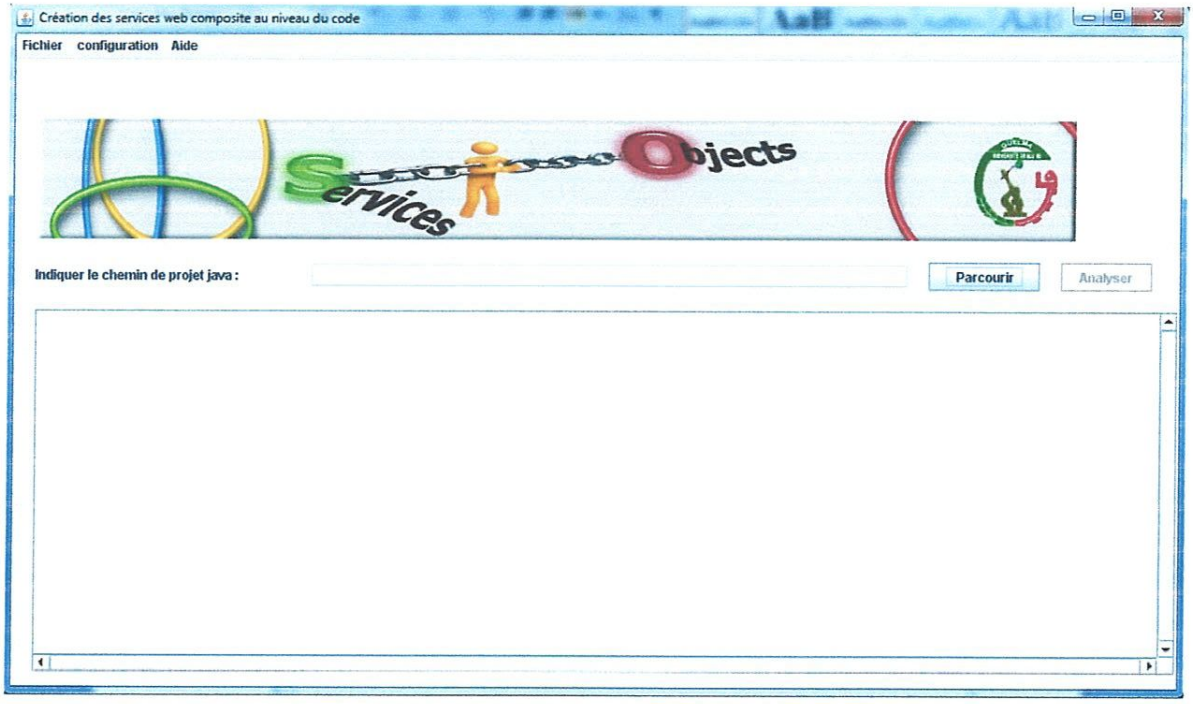

Figure 5.21 : Importation du projet a analysé

Cette fenêtre contient 08 boutons qui sont :

1. parcourir : permet d'ouvrir une boite de dialogue qui vous permet de sélectionner un projet.
2. Analyser : permet de lancer l'analyseur de projet sélectionné.
3. Afficher résultat Analyseur : permet d'afficher le résultat de l'analyseur dans une table.
4. Créer Service web : permet de lancer la création de service web pour le projet sélectionné et analysé dans un nouveau répertoire qui s'appelle « impl ».
5. Consultation et test : permet d'afficher les services web déployés au niveau de l'axis.
6. Créer client final : permet d'afficher une autre fenêtre qui vous permet de créer et exécuter le client final de service web composite.
7. Undeployer WS : permet de désactiver le déploiement de service web crée et déployé.

8. Quitter : permet de fermer l'application.

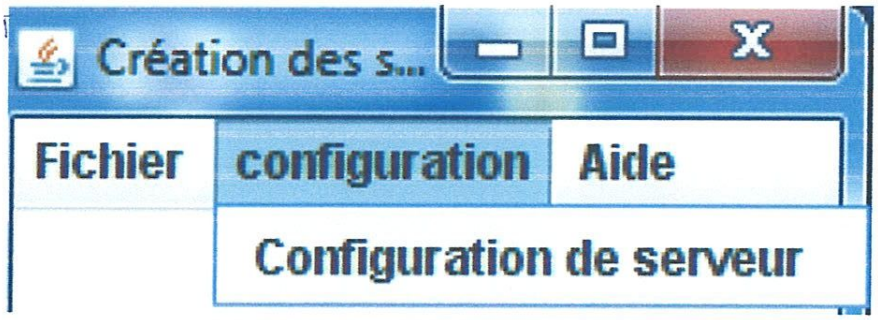

Figure 5.22 : configuration

Puis une autre fenêtre optionnelle s'affiche et vous demande d'entrer le chemin correcte de l'axis.

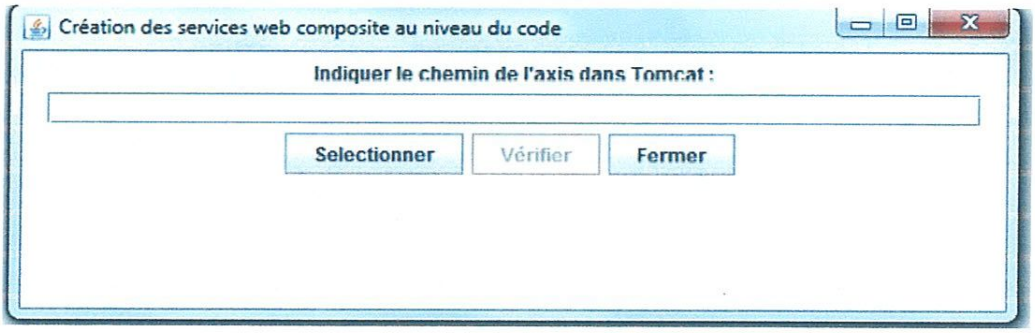

Figure 5.23 : vérification du chemin de l axis

Elle contient 03 boutons :

1. Sélectionner : permet de sélectionner de chemin de l'axis de votre système.
2. Vérifier : permet d'afficher un message si le chemin de l'axis entrée est correcte ou non ;

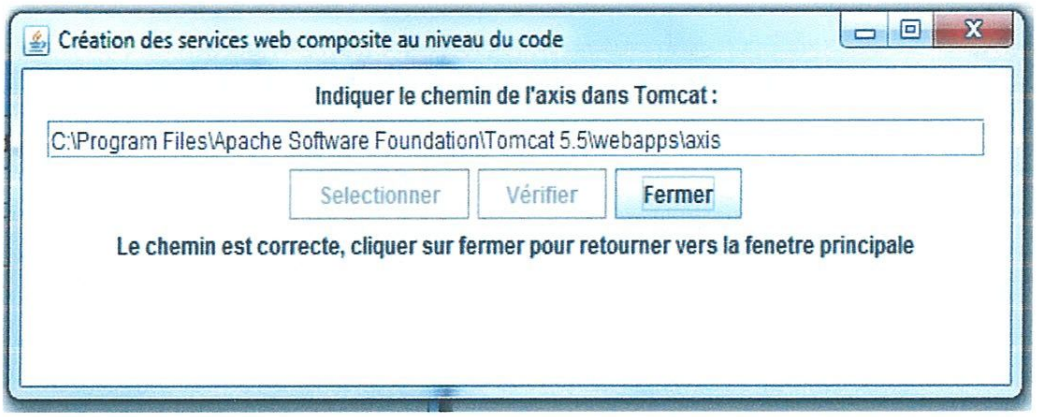

Figure 5.24 : vérification du chemin de l'AXIS

3. Fermer : permet de fermer cette fenêtre et de retourner vers la fenêtre principale.

La fenêtre qui représente la phase d'analyse de projet :

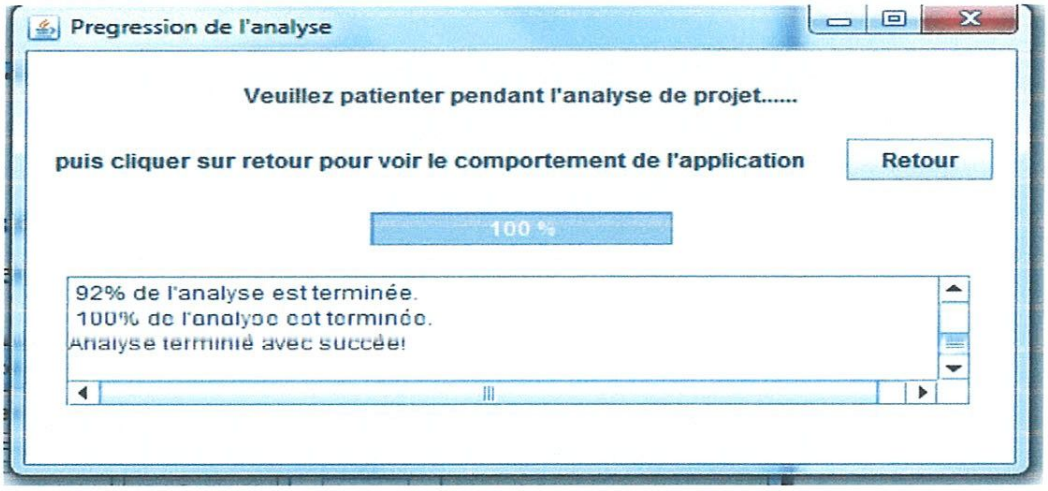

Figure 5.25 : Analyse du projet

Elle conteint un bouton qui permet à la fin de l'analyse de retourner vers la fenêtre principale pour afficher le résultat de l'analyse.

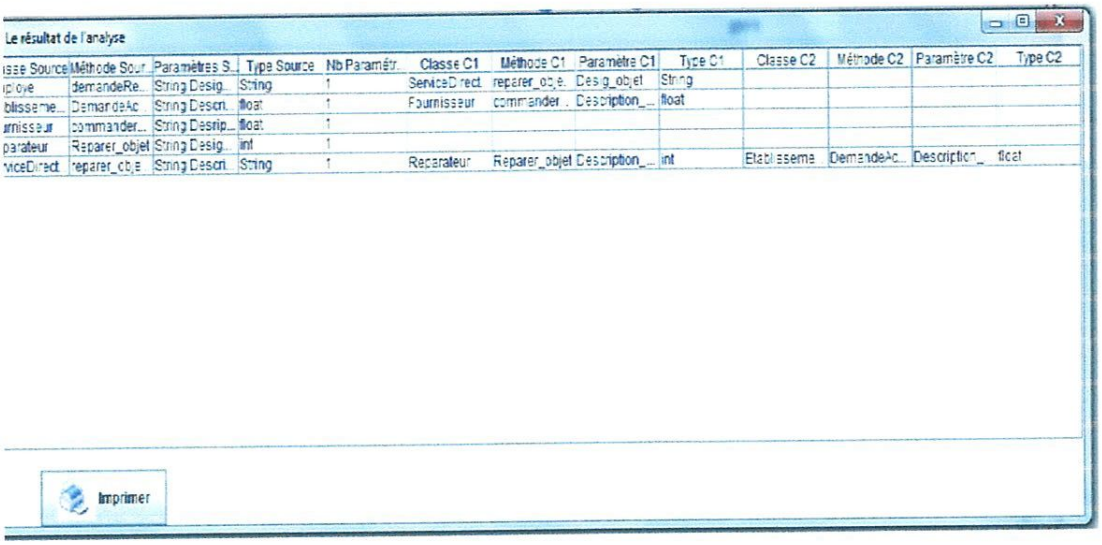

Figure 5.26 : Résultat de l'analyse dans une table

Cette fenêtre affiche le résultat de l'analyse de projet dans une table. Elle contient un bouton qui permet d'imprimer la table.

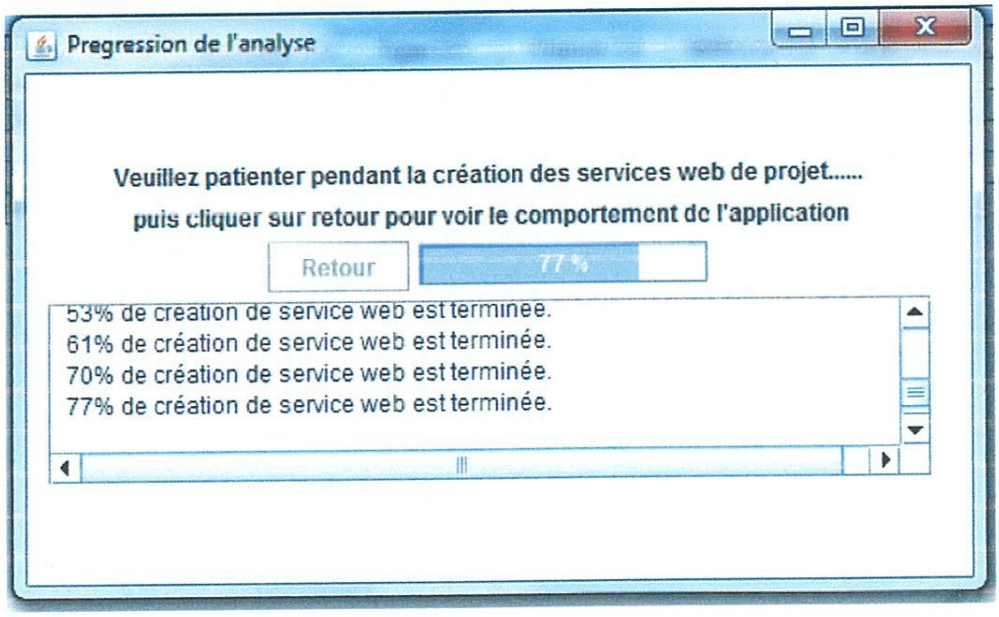

Figure 5.27 : progression de création des services web

Cette fenêtre représente la phase de création de code orienté service.

Figure 5.28 : les services déployés au niveau de l'axis

Cette fenêtre s'affiche après le clic sur le bouton Consultation et test. Elle permet d'afficher les services déployés au niveau de l'axis.

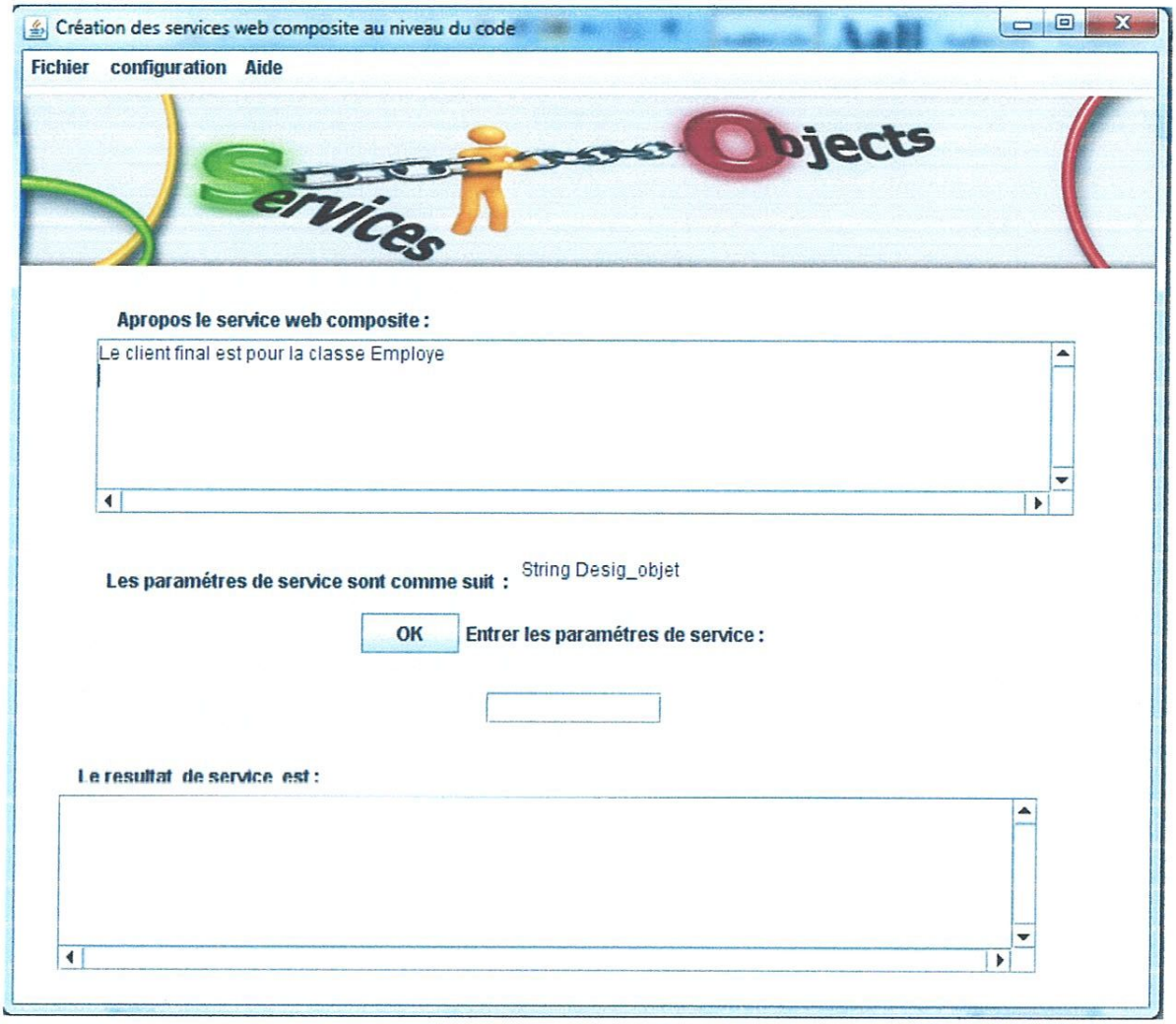

Figure 5.29 : Création du client

Cette fenêtre s'affiche après le clic sur le bouton client final. Elle permet d'afficher le nom et les paramètres de service résultat.

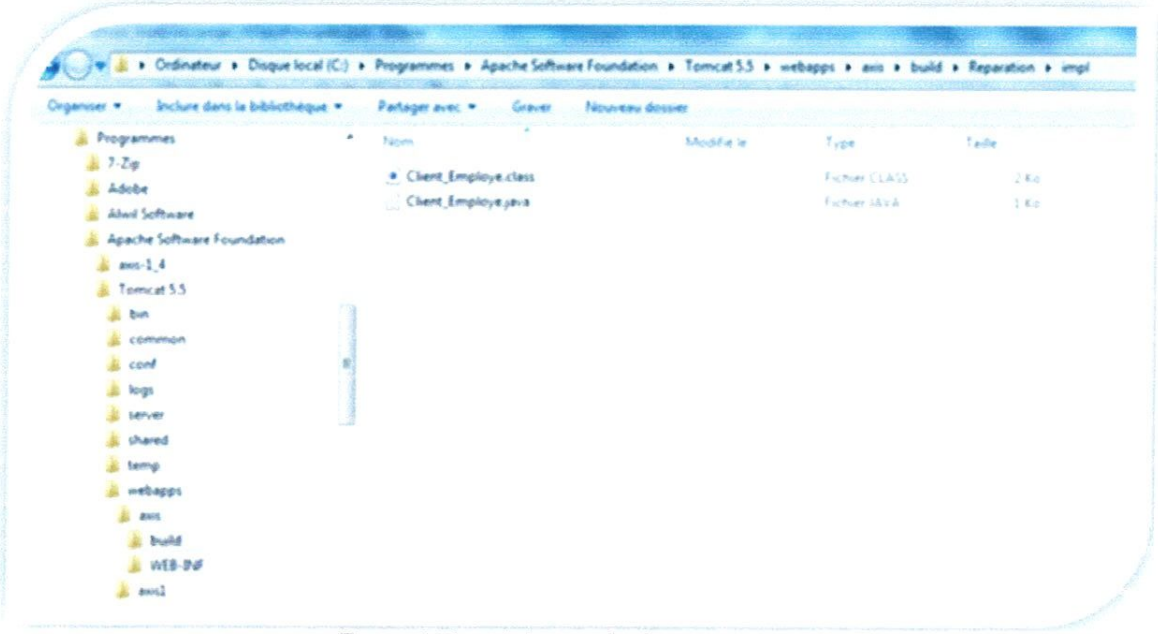

Figure 5.30 : génération du client

Le client généré pour invoquer le service

Le résultat de l'exécution de client sera affiché comme suit :

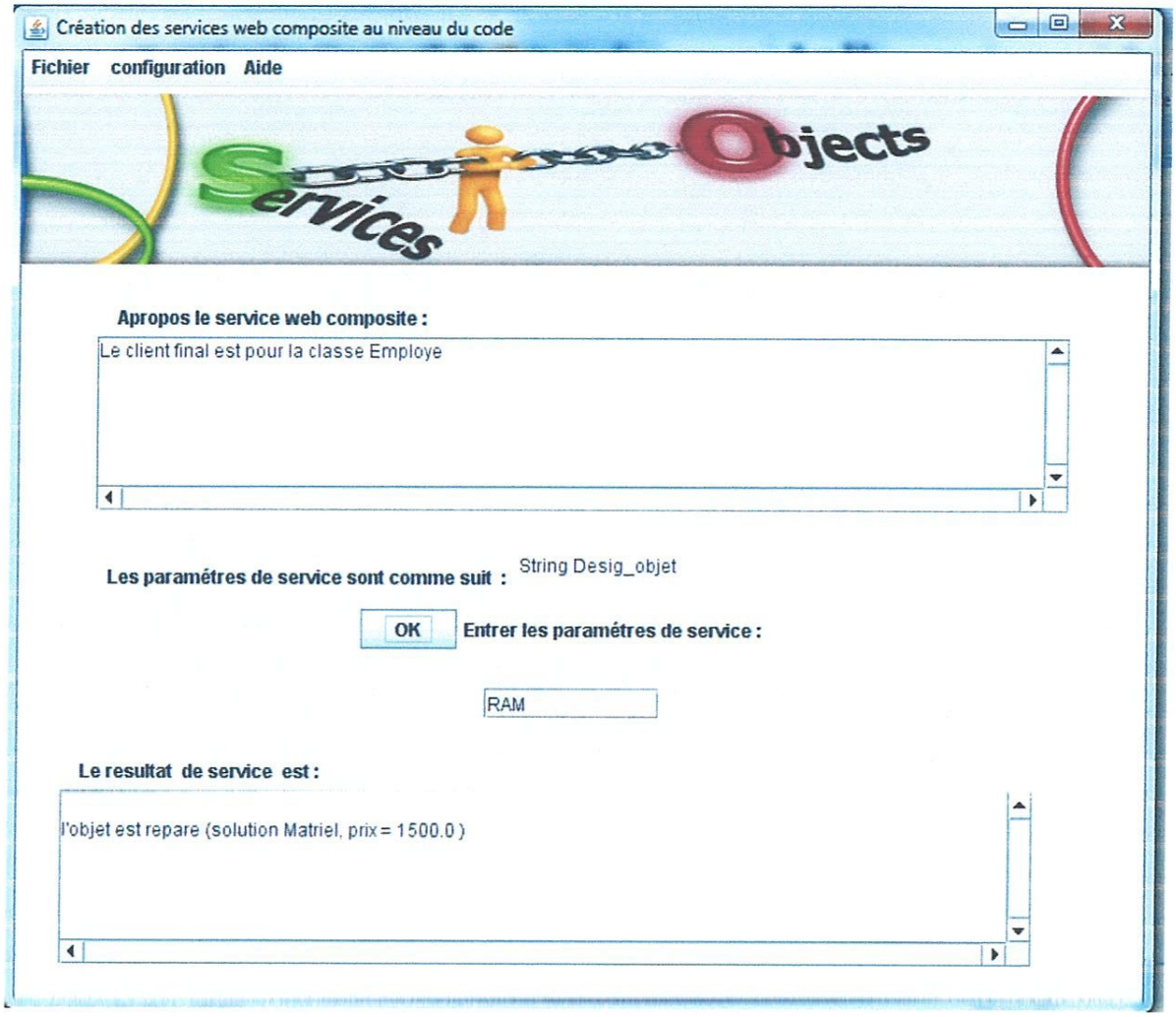

Figure 5.31 : Résultat du service

Le résultat de création et exécution de client de service Employé.

7. Conclusion

Dans ce chapitre nous avons présenté les outils utilisé pour développer notre outil de migration, a savoir l'éclipse, javaEE, TomCat , axis et autres.

Les détails d'implémentation ont été aussi présentés

Pour valider notre application nous avons du aussi utiliser un exemple lequel nous avons illustré les différentes étapes de migration d'une application orienté objet vers un model orienté service.

VI. Conclusion Générale et Perspective

L'architecture orienté service représente un nouveau paradigme prometteur pour le développement, le déploiement et l'intégration d'applications Internet. Ce paradigme concrétisé autour la technologie des services web et ces trois principales standards : WSDL qui assure la partie description des services web, SOAP qui fournit un moyen de communication entre les services et UDDI qui permet de publier ses services web. Toutes ces technologies sont basées sur le langage XML.

La tendance est d'adopter cette nouvelle architecture soit en redéveloppant le patrimoine logiciel existant, ce qui revient très cher ; soit par l'évolution de ce dernier vers la nouvelle architecture SOA.

Dans ce contexte, notre travail consiste à développer un outil qui permet la migration d'une application orientée objet vers une nouvelle orientée service.

Pour créer la nouvelle version en services web, nous nous somme basé sur le code source orienté objet. Pour cela, on a réalisé un outil qui permet d'analyser, d'extraire un ensemble d'informations pour enfin les utiliser dans la génération du code source orienté service. Une étape suivante consiste a décrire ses interfaces par la génération des fichiers WSDLs et les déployé par la création et l'utilisation des fichiers WSDDs. Un client est générer à la fin pour invoquer le service web.

Les principales difficultés rencontrées lors de la réalisation de notre outil étaient :

- La configuration des variables d'environnements pour qu'on puisse compiler et déployer et exécuter les services web.
- Le développement d'un analyseur qui permet de traiter les nœuds d'un code source orienté objet.
  La recherche et le temps consacré pour la présélection du moteur de génération (axis)
- La compatibilité de la plate forme nécessaire pour bien démarrer le service avec notre système (compatibilité avec win7 64 bit)

Comme perspective nous proposons d'améliorer notre outil par :

- Ajouter une étape préalable d'identification des services candidats
- l'ajout d'un module de publication de service web dans l'annuaire UDDI ;
- le travail sur des applications graphiques orientées objet.
- la transformation des applications orientées objet implémentées en C++ ou C#.

[11] Livre Business Knowledge for It in Global Retail Banking Par Corporation Essvale

[Ser 2011]

[12] [Sneed 2006]