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: Informatique Académique

Thème :

IDENTIFICATION ET EXTRACTION DE SERVICES A PARTIR D'UNE APPLICATION ORIENTEE OBJET

Encadré Par :

- Mr. SERIDI Ali

Présenté par :

- DOGHMANE Med Nadhir

-DAOUD Ahlem

Juin 2013

Remerciement

Nous tenons avant tout à remercier le bon DIEU qui nous a donné la volonté et le courage pour la réalisation de ce modeste travail.

Nous remercions vivement Mr.SERIDI Ali notre promoteur pour sa précieuse assistance, sa disponibilité et son soutien qu'il nous a accordé tout au long de ce projet.

Que les membres de jury trouvent ici l'expression de nos sincères remerciements pour l'honneur qu'ils nous ont fait en acceptant de juger notre travail

Nous exprimons également notre gratitude à tous les professeurs et enseignants qui ont contribué à notre formation depuis notre premier cycle d'étude jusqu'à la fin de notre cycle universitaire.

Sans omettre bien sur de remercier profondément tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Nadhir et Ahlem.

Dédicaces

Je rends grâce à DIEU de m'avoir donné le courage et la vie ainsi que la santé d'avoir pu terminer mes études.

Je dédie ce modeste travail :

A ma chère mère qui m'a assuré un soutien inconditionnel sans lequel je n'aurais jamais pu terminer mes années d'étude pour ses conseils et son encouragement.

A mon père, pour m'avoir donné la possibilité de faire ce que je voulais et pour son soutien sans faille tout au long de mes études et pour son affection, sa patience et sa compréhension.

A mes très chers frères

A mes très chères sœurs

A Mon marie

A tous mes très chers oncles, tantes et mes cousins.

A tous les professeurs et enseignants qui ont participé à ma formation depuis mon premier cycle d'étude.

A tous mes chères amies

A tous mes amis d'études

A tous ceux qui m'ont aidé de loin ou de près durant mes études.

Pour finir je dédie ce travail à tous ceux qui m'aiment.

Doghmaine. N

Dédicaces

Je rends grâce à DIEU de m'avoir donné le courage et la vie ainsi que la santé d'avoir pu terminer mes études.

Je dédie ce modeste travail :

A ma chère mère qui m'a assuré un soutien inconditionnel sans lequel je n'aurais jamais pu terminer mes années d'étude pour ses conseils et son encouragement.

A mon père, pour m'avoir donné la possibilité de faire ce que je voulais et pour son soutien sans faille tout au long de mes études et pour son affection, sa patience et sa compréhension.

A mes très chers frères

A mes très chères sœurs

A Mon marie

A tous mes très chers oncles, tantes et mes cousins.

A tous les professeurs et enseignants qui ont participé à ma formation depuis mon premier cycle d'étude.

A tous mes chères amies

A tous mes amis d'études

A tous ceux qui m'ont aidé de loin ou de près durant mes études.

Pour finir je dédie ce travail à tous ceux qui m'aiment.

Daoud .A

Résumé

Aujourd'hui, plusieurs entreprises se basent sur des systèmes d'information complexes, monolithiques, et inflexibles, parfois non conformes aux changements rapides du marché.
L'Architecture Orientée Services (SOA) apparaît comme la meilleure approche pour répondre aux besoins des entreprises.SOA est une architecture d'entreprise et la logique conceptuelle des fonctionnalités métier, ou de l'application, est mise à la disposition des utilisateurs SOA, où les consommateurs partagent des services réutilisables sur un réseau IT (Information technology). Les "Services" dans une architecture SOA sont des modules d'affaires ou de fonctionnalités de l'application avec des interfaces exposées, et sont invoquées par les messages.

A travers notre projet, nous proposons un outil de migration d'un système patrimonial qui prend comme entrée, une application orienté objet (OO) et comme sortie un système évolué(SOA).
Le processus de conversion passe par deux étapes

- Identification des services potentiels à travers l'analyse du code orienté objet et regroupement des classes fortement couplées sous forme de service.
- Procéder à la migration de ces classes vers des services, après validation des services identifiés.

Mots clés :

Architecture Orientée Services, Système patrimonial, Application orienté objet, Migration, Réingénierie.

Sommaire

Introduction générale……...1

I. SOA et service web

1. Introduction…….... 4
2. Historique ……5
3. Définition de service ……5
4. Principes généraux d'une architecture orientée service ……8
5. Définition de la SOA…… 9

5.1. Définition technique ……..9

5.2. Définition métier …… 10

6. Bénéfices de la SOA ……...10
7. Les caractéristiques de la SOA……..…… 11
8. Architecture SOA ……...11
9. Les acteurs d'une architecture SOA……14
10. L'objectif d'une architecture orientée service……..14
11. les avantages et les inconvénients d'une architecture orientée service……..15

11.1. Les avantages d'une architecture orientée service…….…… 15

11.2. Inconvénient d'une architecture orientée service……..…… 16

12. Conclusion ……16

II. Technologie d'implémentation de la SOA

1. Introduction……18
2. Standards des services web…… 18

2.1. XML (eXtensible Markup Language) …… 19

2.2.1. SOAP (Simple Object Access Protocol) ……...20

2.2.2. Structure d'un message SOAP……...21

2.3. UDDI (Universal Description Discovery and Integration) ……..……...21

2.3.1. Définition de l'UDDI (Universal Description Discovery and Integration) ..……...21

2.3.2. Utilisation de l'annuaire UDDI ……22

2.4. WSDL (Web Services Description Language) …..…...……...……...23

2.4.1. Définition de WSDL (Web Services Description Language) …….....……...23

2.4.2. Structure des documents WSDL…….... 23

2.4.2.1. Définitions abstraites ……24

2.4.2.2. Descriptions concrètes….. ……24
3. Pourquoi les services web? ……24
4. Fonctionnement des Web services ……..……25
5. Architecture des services web…… 25
7. Méthodologie de développement du service web…….. 26
7.1 Code First……..……26
7.2. WSDL first……27
8. Les services Web java……..28
8.1. SAAJ (SOAP with Attachment API for Java) …… 28
8.2. JAX-WS: JAX-WS (Java API for XML based Web Services) ……28
9. Conclusion……...…29

III. Migration ver SOA

1. Introduction……..……31
2. Quelle différence entre une architecture orientée objet et une SOA ?.....31
3. La migration vers SOA……32
3.1. Bénéfices de la migration……32
3.2. Difficultés de la migration……32
3.2.1. Logiciels patrimoniaux……..33
4. Les bénéfices attendu d'une architecture orienté service ……...33
4.1. Bénéfices de la réutilisation ……33
4.2. Bénéfices du développement d'applications composites (multi-services)……...34
4.3 Bénéfices du couplage lâche ("loosely coupled")…….....34
4.4. Efficacité d'affaires ……..…34
4.5. Réduction des coûts ……34
4.6. Réduction des risques……..… 34
5. Stratégies d'évolution vers une architecture orientée service ……35
5.1. Approches invasive……...35
5.2. Approches non invasive……35
6. Les approches de SOA ……39
6.1. Top Down…… 39
6.2. Bottom Up ……..40
6.3. Outside In (Meet In The Middle) ……40
6.4. Middle Out ……...41

7. Conclusion……...42

IV. Conception

1. Introduction ……44

2. L'objectif…… 44

3. Fonctionnement générale de l'application……45

4. Processus de migration de l'application OO vers Une application SOA……... 46

5. conclusion……52

V. Implémentation

1. Introduction……54

2. Outils de développement……54

2. 1. Le langage utilisé……54

2.2. La plateforme Eclipse……..54

2.3. JBoss Application Server……54

2.4JAX-WS (Java API(application programming interface) for XML Web):……55

2.5. SoapUI……55

3. Configuration des outils……56

3.1. La configuration de serveur JBoss……56

4. Architecture de l'application……...58

4.1Package analyseur ……58

4.2 Pacage couplage ……58

4.3. Package principale……..…… 60

4.4. Package CreationDynamicProjet ……61

4.5. Package WebService ……63

5. La présentation de l'application……...64

6. Conclusion……...68

Conclusion générale et perspective…… 69

Bibliographier …… 70

Liste figure

Figure 1.1 : *Historique de développement de la SOA……5*

Figure 1.2 : *Les proprietés de service……6*

Figure 1.3 : *Contrat standardisé de service……6*

Figure 1.4 : *Autonomie de service……7*

Figure 1.5 : *Communication par messages de service……7*

Figure1.6 : *Politiques de fonctionnement de service ……8*

Figure1.7 : *Frontières explicites de service……8*

Figure 1.8 : *Basic SOA architecture……9*

Figure 1.9 : *Basic logiciel SOA components……12*

Figure 1.10 *: SOA infrastructure ……12*

Figure1.11 *: SOA sous-détail……13*

Figure1.12 : *Les acteurs d'une architecture SOA……14*

Figure2.1 : *exemple XML……20*

Figure 2.2 : *structure d'un message SOAP……20*

Figure 2.3 : *Données du registre UDDI……23*

Figure 2.4: *Architecture WSOA (Web Services Oriented Architecture)……26*

Figure 2.5 : *Développement d'un service Web par la méthode « Code First »……26*

Figure 2.6 *: Développement d'un service Web par la méthode « WSDL First »……27*

Figure 2.7 *: JAX-WS……29*

Figure 3.1 *: Les différentes approches de la migration……39*

Figure 4.1 : *vue générale de l'application*……..45

Figure 4.2 : *Fonctionnement générale de l'application*……45

Figure 4.3 : *Etapes de génération de l'application orientée service*……46

Figure 4.4 : *Des exemples de couplage*……48

Figure 4.5 *: fonction de la Partie sélection et validation de services*……50

Figure 4.6 : *la structure interne de Composant de création de servie web*……51

Figure 4.7 : *Etape Composant de création de servie web*……52

Figure 5.1 *: La configuration de serveur JBoss*……57

Figure 5.2 : *Package analyseur*……58

Figure 5.3 : *Couplage entre les classes*……59

Figure 5.4 : *Couplage entre les package*……59

Figure 5.5 *: Package CreationDynamicProjet*……60

Figure5.6: *Package CreationDynamicProjet*……61

Figure 5.7 *: Package WebService*……63

Figure 5.8 *: fenêtre principale*……64

Figure 5.9*: Parcourir et transférer ver projet dynamic*……65

Figure 5.10 *: Afficher* ……66

Figure 5.11 *:* Démarrer le stan……66

Figure 5.12*:* Le couplage entre les packages……..67

Figure 5.13*:* Le couplage entre les classes……..67

1. Introduction générale :

Les Systèmes d'Information sont en constante évolution et les obstacles rencontrés sont largement débattus depuis des années, mais ces questions acquièrent aujourd'hui une acuité particulière. Les applications construites et structurées pour répondre à des besoins particuliers dans un contexte donné ne sont plus adaptées aux réalités présentes. Les changements technologiques ajoutent encore des contraintes supplémentaires. Aux delà d'un certain stade, les coûts induits par les modifications deviennent prohibitifs, et les délais incompatibles avec les demandes métiers. Le système devient trop rigide, prisonnier de son architecture antérieure.
Parmi les solutions proposées pour résoudre certains de ces problèmes c'est la migration vers des architectures plus agiles et plus souples tel que la SOA.

L'architecture orientée services (Service Oriented Architecture - SOA) est une approche architecturale de dernière génération qui permet de passer d'une vision « application » à une vision « services ». Elle réunit en un seul nom un grand nombre de concepts tels que, la composition, l'abstraction, l'autonomie, le couplage faible, la modélisation et le monitoring des processus métier, etc **[44].**

Ils existent des acteurs les plus présents dans le domaine de SOA qui se sont positionnés parmi les premiers autour des services web, et qui proposent des solutions matures à ce niveau. En effet, il est possible d'envisager la mise en place d'une architecture orientée services sans utiliser les services web, on peut constater que ces derniers sont particulièrement adaptés à ce type d'architecture, et qu'ils ont fortement contribué à son avènement.

Parmi les systèmes qui sont concernés par la modernisation vers une architecture orientée 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 applications orientées objets en d'autres applications orientées services.

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

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

- Identification des services potentiels à travers l'analyse du code orienté objet et regroupement des classes fortement couplées sous forme de service.
- Procéder à la migration de ces classes vers des services, après validation des services identifiés.

Notre mémoire s'article sur 05 chapitres :

Chapitre 01 : Dans ce chapitre nous allons commencer par quelques lignes de l'historique et la définition de la SOA ensuite nous allons parler des bénéfices, des caractéristiques, et de l'architecture SOA, l'objectif, principes généraux et enfin les avantages et les inconvénients.

Chapitre 02 : Dans ce chapitre nous présenterons les fonctionnements et l'architecture de service web, les standards de service web ensuite nous décrirons la méthodologie de développement des services web**.**

Chapitre 03 : a pour objectif de présenter le concept de migration et d'identification des services.

Chapitre 04 : Nous allons définir premièrement les objectifs, le fonctionnement général de l'application, ainsi que le processus de migration de l'application orientée objet vers une application orientée service.

Chapitre 05 : présente les outils utilisés pour le développement et les détails d'implémentation de notre projet.

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

Chapitre 01

SOA et Service Web

1. Introduction :

La réalité dans les entreprises IT (Information technology) est que l'infrastructure est hétérogène à travers la diversité des systèmes d'exploitation, des applications, des logiciels On utilise certaines applications pour exécuter des processus actuels, alors partir de zéro pour construire de nouvelles infrastructures n'est pas la meilleure option. Les entreprises devraient réagir rapidement aux changements des affaires avec agilité; tirer profit des investissements existants dans les applications et l'infrastructure d'applications pour répondre aux besoins récents, soutenir de nouveaux canaux d'interactions avec les clients **[03].**

Les objets peuvent s'envoyer des messages, grâce aux appels de méthodes exposées par l'objet avec lequel ils souhaitent communiquer sans pour autant savoir comment ledit objet implémente le traitement qu'on lui demande d'exécuter. Malgré le fait que des technologies comme DCOM, RMI ou .Net Remoting permettent de transporter les objets et donc de dépasser les frontières de la machine grâce au réseau, on s'est souvent heurté à des problèmes de compatibilité entre plateformes, d'où le besoin d'une standardisation et la mise en commun des protocoles (SOAP, XML, …). De là est née la notion d'architecture orientée services (SOA) est un acronyme qui s'est répondu partout depuis l'année 2002. On le retrouve comme mot clé dans un grand nombre d'articles de journaux scientifiques, comme slogan utilisé par les fournisseurs de solutions logicielles, comme titre pour des dizaines de livres **[11]**.

L'architecture orientée services (SOA) est une approche architecturale de dernière génération qui permet de passer d'une vision « application » à une vision « services ». Elle réunit en un seul nom un grand nombre de concepts tels que, la composition, l'abstraction, l'autonomie, le couplage faible, la modélisation et le monitoring des processus métier, etc **[10]**.

Dans ce chapitre nous allons commencer par quelques lignes de l'historique et la définition de la SOA ensuite nous allons parler des bénéfices, des caractéristiques, et l'architecture de la SOA, l'objectif, principes généraux et enfin les avantages et les inconvénients.

2. Historique :

Les concepts de la programmation et des architectures logicielles n'ont cessé d'évoluer ces dernières années.

Après que les techniques conventionnelles de programmation ont été remplacées par le développement de modèle à objets, la programmation par composants est apparue et a permis un développement plus modulaire des applications. Basé sur cette technique de programmation, un nouveau concept architectural est apparu : l'architecture orientée services ou SOA (Service Oriented Architecture).

SOA est apparu en 1996 dans une note de recherche du Gartner Group.

« L'architecture orientée service constitue un style d'architecture basée sur le principe de séparation de l'activité métier en une série de services. »

« Ces services peuvent être assemblés et liés entre eux selon le principe de couplage lâche pour exécuter l'application désirée. »

« Ces services sont définis à un niveau supérieur de la traditionnelle approche composants » **[3].**

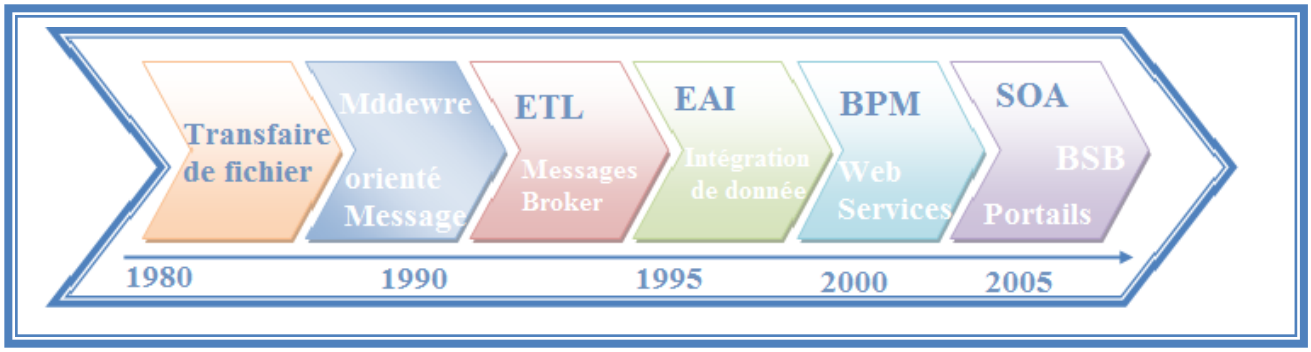


Figure 1.1 : *Historique de développement de la SOA*

3. Définition de service :

« Un Service est un composant logiciel distribué, exposant les fonctionnalités à forte valeur ajoutée d'un domaine métier » **[12].**

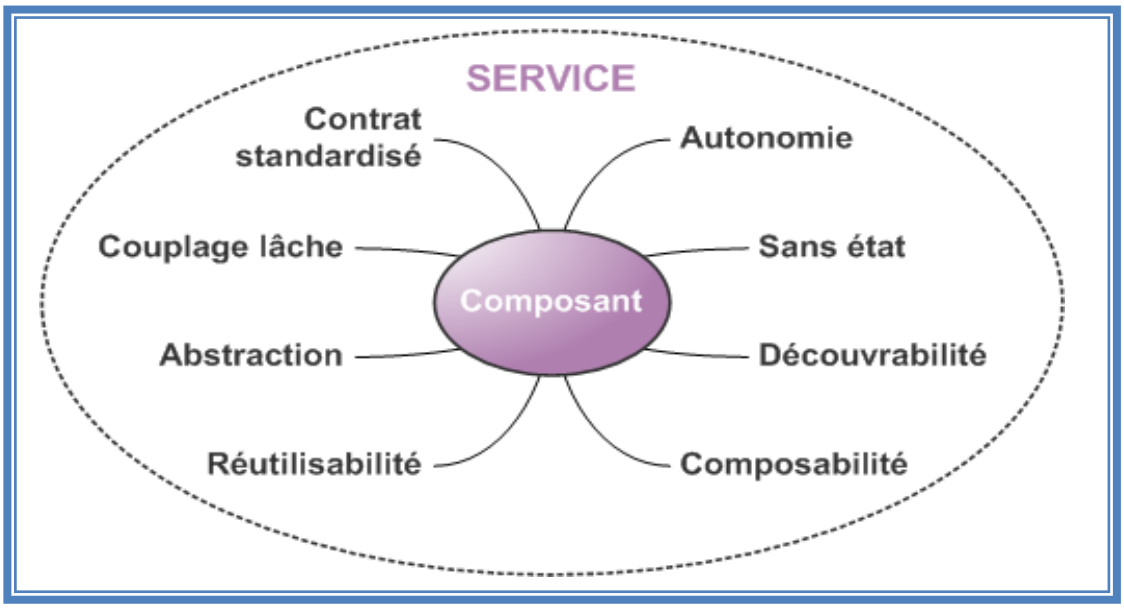


Figure 1.2 : *Les proprietés de service*

Plusieurs notions associées aux services constituent les piliers d'une architecture orientée service **[13] :**

- **Contrat standardisé :** L'ensemble des services d'un même Système Technique sont exposés au travers de contrats respectant les mêmes règles de standardisation.

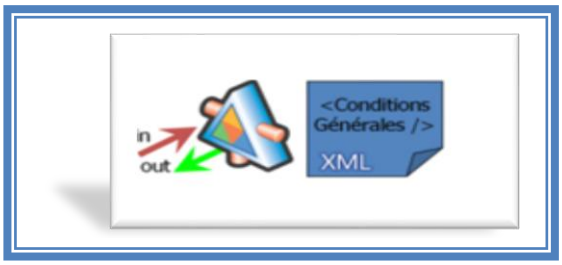


Figure 1.3 : *Contrat standardisé de service*

- **Couplage lâche :** Le contrat d'un service doit imposer un couplage lâche de ses clients.
- **Abstraction :** Le contrat d'un service ne doit contenir que les informations essentielles à son invocation. Seules ces informations doivent être publiées.
- **Réutilisabilité :** Un service exprime une logique agnostique et peut ainsi être positionné comme une ressource réutilisable.

- **Autonomie :** Un service doit exercer un contrôle fort sur son environnement d'exécution sous-jacent. Plus ce contrôle est fort, plus l'exécution d'un service est prédictible.

Figure 1.4 : *Autonomie de service*

- **Stateless *(sans état)* :** Un service doit minimiser la consommation de ressources en déléguant la gestion des informations d'état quand cela est nécessaire.
- **Découvrabilité :** Un service est complété par un ensemble de métas données de communication au travers des quelles il peut être découvert et interprété de façon effective.
- **Composabilité :** Un service doit être conçu de façon à participer à des compositions de services **[12].**
- **Communication par messages**

Les services communiquent par messages : ils s'échangent des messages respectant le contrat avec un format commun (par exemple XML).

Dans le cas de transactions complexes, un message peut transiter par de multiples services avant de revenir à l'initiateur. Chaque service intervient alors sur le message pour le compléter ou le modifier.

Figure 1.5 : *Communication par messages de service*

- **Politiques de fonctionnement**

Les services négocient en utilisant des « politiques » : chaque service doit adhérer à la politique du tiers avec lequel il désire interagir pour coopérer. Les politiques précisent le cadre d'exécution du contrat : par exemple les règles de sécurité, le niveau transactionnel, la fiabilité des messages, l'asynchronisme, la qualité de service, etc.

Figure1.6 : *Politiques de fonctionnement de service*

- **Frontières explicites**

Les frontières entre services sont explicites : les pré-requis et les restrictions pour traverser la frontière et accéder aux services sont clairement définis et connus : identité, protocoles...

Figure1.7 : *Frontières explicites de service*

4. Principes généraux d'une architecture orientée service

Il n'existe pas à proprement parler 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 plusieurs 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 , 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 notion de 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*). **[16]**

5. Définition de la SOA :

5.1. Définition technique :

En présente dans ce titre quelque définition de la SOA (*architecture orientée services*) :

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

L'architecture orientée services (SOA) est une évolution de l'informatique distribuée basée sur le paradigme de conception de requête / réponse pour les applications synchrones et asynchrones.

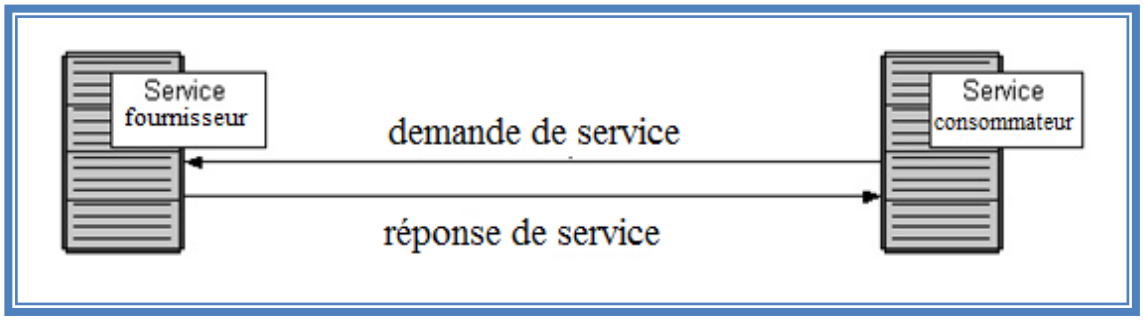


Figure 1.8 : *Architecture de base de la SOA*

L'aspect le plus important de l'architecture orientée services est qu'elle sépare l'implémentation de service de l'interface. En d'autres termes, elle sépare le «Quoi» du

«comment». Les consommateurs de services considèrent un service comme un point final qui supporte un format de requête. Les consommateurs de services ne sont pas concernés par la façon dont le service va assurer l'exécution de leurs demandes, ils s'attendent seulement au résultat **[3].**

Le modèle d'architecture orientée services (SOA) s'appuie sur la notion de collaboration entre services selon un principe de « couplage lâche », proposant ainsi une approche moins intrusive et plus flexible **[13].**

5.2. Définition métier:

"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."**[26].**

Traduction en français :

"SOA est une architecture conceptuelle d'entreprise, où les fonctionnalités métier, et la logique de l'application, est mise à la disposition des utilisateurs SOA, ou les consommateurs, comme des services partageable, et réutilisables réseau dans un IT. les "Services" dans une architecture SOA sont des modules d'entreprise ou des fonctionnalités d'application avec des interfaces exposées, qui sont invoquées par les messages. "

La SOA est un modèle architectural qui a pour principe d'augmenter l'efficacité, l'agilité et la productivité d'une entreprise en plaçant la notion de services comme le moyen principal pour atteindre les objectifs stratégiques **[27].**

6. Bénéfices de la SOA :

SOA avec sa nature faiblement couplée permet aux entreprises de :

- Brancher de nouveaux services
- Améliorer les services existants de façon granulaire pour répondre aux nouveaux besoins.
- Offrir la possibilité de rendre les services consommables à travers différents canaux.
- Exposer les applications existantes de l'entreprise comme des services, tout en préservant les investissements d'infrastructure informatique existante **[3].**

7. Les caractéristiques de la SOA :

- Réutilisation et composition : Ceci est particulièrement puissant pour créer des nouveaux processus d'affaires rapide et fiable.
- Recomposition : La capacité de modifier les processus d'affaires existants ou d'autres applications basées sur l'agrégation des services.
- 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.
- La capacité à construire progressivement le système. Cela est particulièrement vrai pour toute intégration basée sur SOA
- Web service ne signifie pas SOA cela veut dire même si on expose tous des web services, tous les concepts SOA seront respectés
- SOA ne résout pas les problèmes existant dans les implémentations
- SOA nécessite un langage métier commun (Contrat, grammaire XML) **[3]**.

8. Architecture 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 [3]:

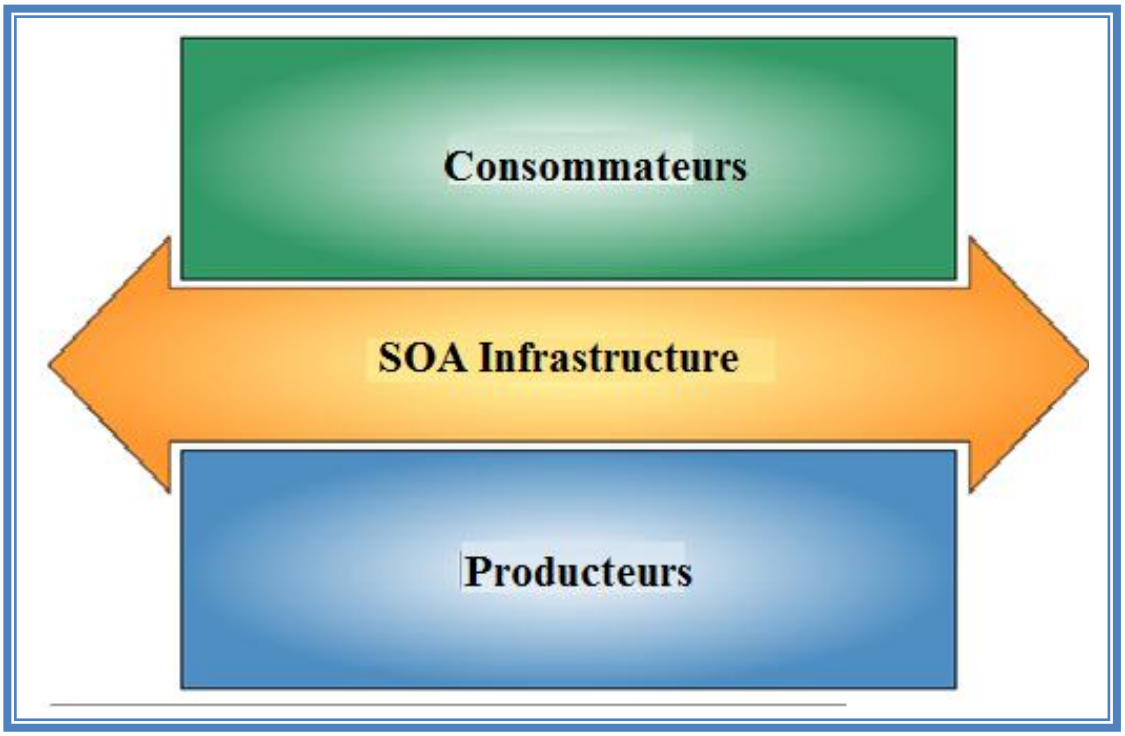


Figure 1.9 : *Composants de base de la SOA*

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

- Application
- Service
- Support de service

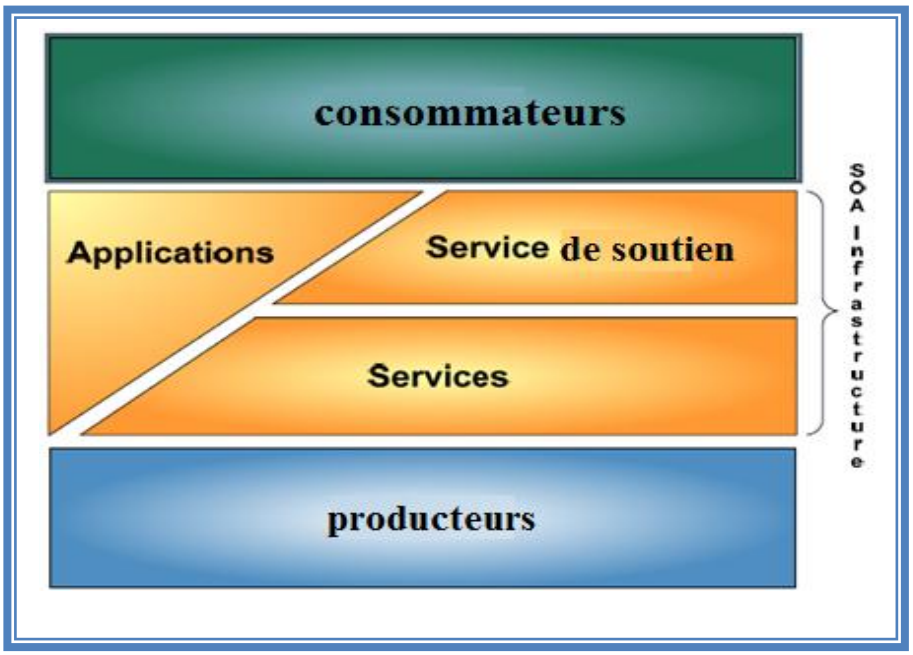


Figure 1.10 *: infrastructure de la SOA*

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

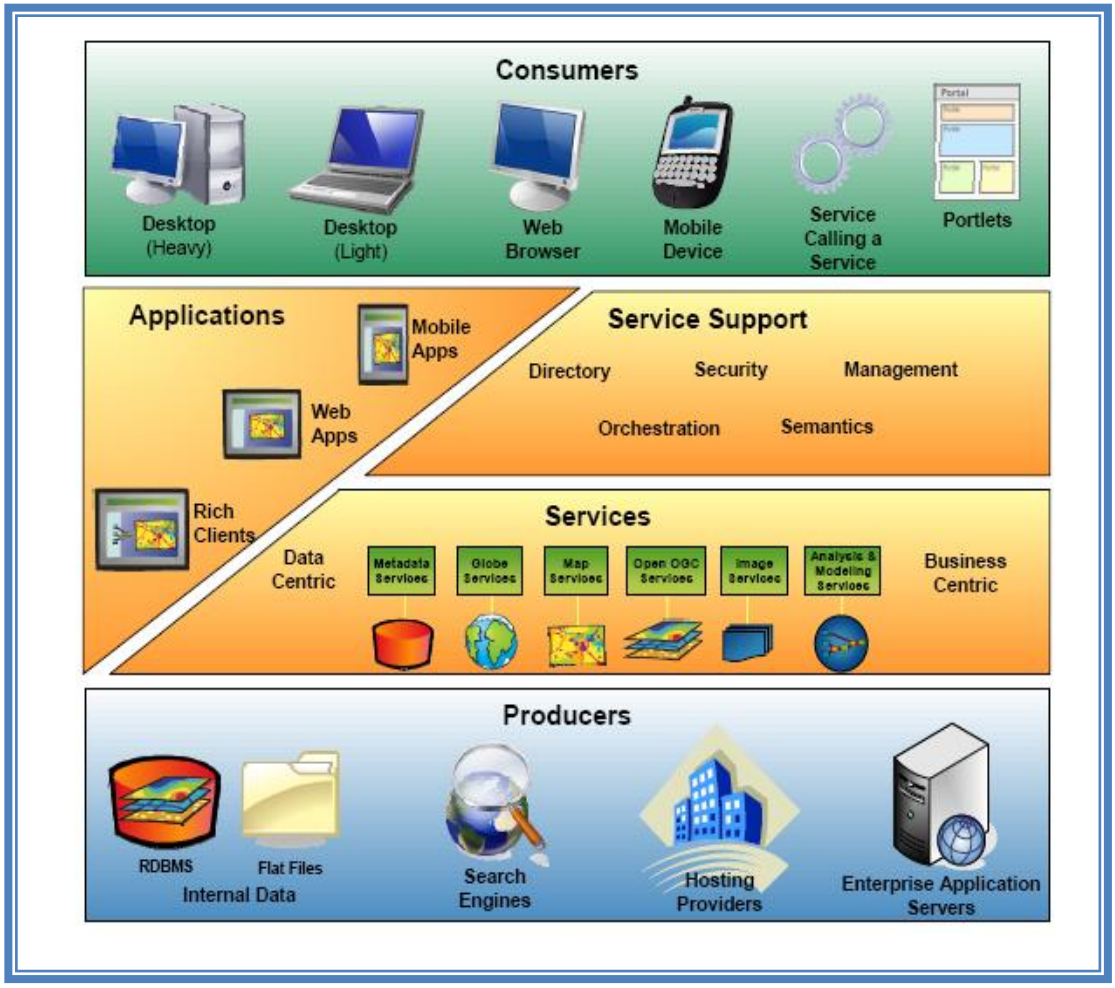


***Figure1.11** : SOA détaillée*

9. Les acteurs d'une architecture SOA

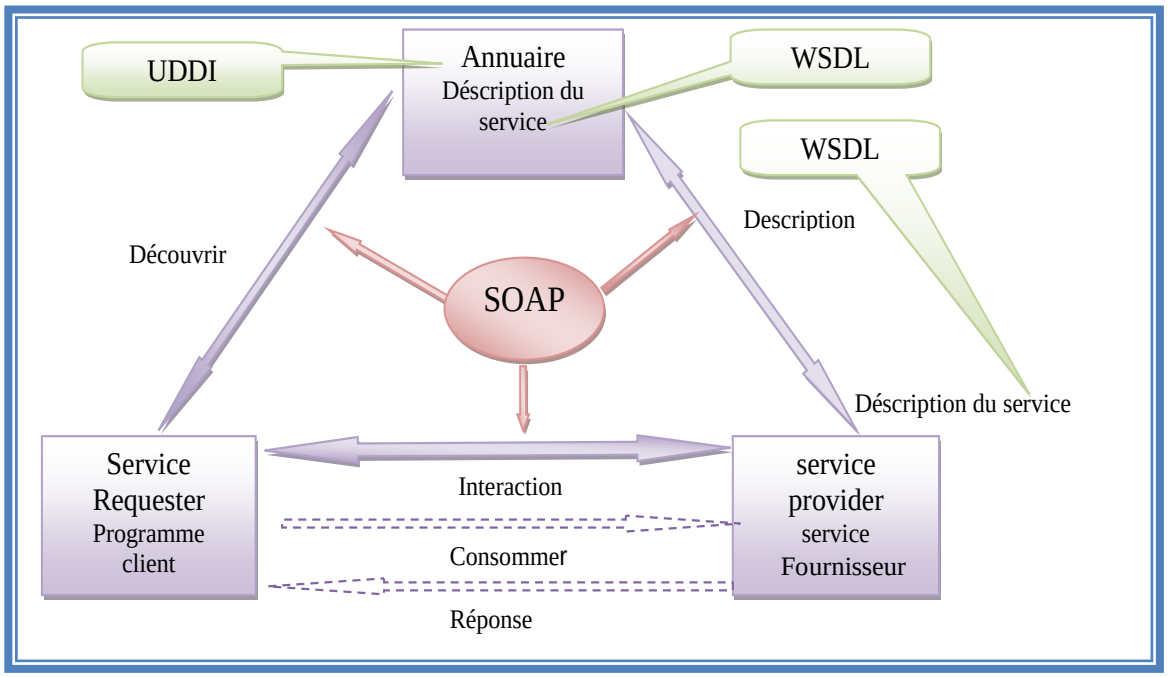


Figure1.12 : *Les acteurs d'une architecture SOA*

10. L'objectif d'une architecture orientée service

L'objectif des architectures orientées services (SOA) est de permettre le développement des applications logicielles complexes et distribuées tout en appuyant les concepts de modularité, et donc de réutilisation de composants logiciels. En outre, l'explosion du nombre d'applications mobiles connectées au web crée une demande croissante d'applications accessibles via Internet, ce qui demande une restructuration de l'architecture des applications existantes. Une fois le paradigme "orienté service" adopté, il sera possible de combiner ces services, afin de fournir de nouveaux services plus élaborés **[14].**

La SOA a comme objectif de décomposer une fonctionnalité en un ensemble des services, fournis par des composants et de décrire finement le schéma d'interaction entre ces fonctions basiques. L'idée cachée est de cesser de construire la vie de l'entreprise autour d'applications, mais de construire une architecture logicielle globale décomposée en services correspondant aux processus métiers de l'entreprise **[15]**.

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*) **[16].**

11. les avantages et les inconvénients d'une architecture orientée service :

Voici un résumé des avantages et inconvénients SOA

11.1. Les avantages d'une architecture orientée service :

Une architecture orientée services permet d'obtenir tous les avantages d'une architecture client-serveur et notamment **[25] :**

- Une modularité permettant de remplacer facilement un composant (service) par un autre.
- Une réutilisabilité possible des composants (par opposition à un 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.
- Obligation d'avoir une modélisation poussée.
- Possibilité de découpler les accès aux traitements.
- Localisation et interfaçage transparents (ouverture accrue).
- Possibilité de mise en place facilitée à partir d'une application objet existante.
- Réduction des coûts en phase de maintenance et d'évolution.
- Facilité d'amélioration des performances pour des applications importantes (répartition des traitements facilitée) **[3]**.

11.2. Inconvénient d'une architecture orientée service :

- Coûts de conception et de développement initiaux plus conséquents
- Nécessité d'appréhender de nouvelles technologies
- Existant non SOA dans les entreprises
- Performances réduites pour des traitements simples (couche supplémentaire) **[3].**

12. Conclusion :

En a vu que L'architecture orientée service apporte de nombreux avantages (agilité, partage des ressources applicatives, réutilisation, facilité d'intégration ...) mais la SOA crée aussi de nouvelles problématiques que les applications traditionnelles ne connaissaient pas.

Technologie D'implémentation de la SOA

1. Introduction :

En termes de technologies, les SOA ne sont pas étroitement liées à un éditeur ou une mouvance particulière. Il est ainsi possible d'envisager l'implémentation de projets respectant les caractéristiques des SOA, aussi bien sous une technologie Microsoft qu'avec des solutions Open source, J2EE ou non.

Les acteurs les plus présents dans le domaine des SOA sont ceux qui se sont positionnés parmi les premiers autour des services web, et qui proposent des solutions matures à ce niveau. En effet, même s'il est possible d'envisager la mise en place d'une architecture orientée services sans utiliser les services web, force est de constater que ces derniers sont particulièrement adaptés à ce type d'architecture, et qu'ils ont fortement contribué à son avènement.

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, 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 **[5].**

Dans ce chapitre nous présentons les fonctionnements et l'architecture de service web, les standards de service web ensuite nous décrirons la méthodologie de développement du service web**.**

2. Standards des services web :

Les services web sont décrits par des documents WSDL (Web Service Description Langage), qui précisent les méthodes pouvant être invoquées, leurs signatures et les points d'accès du service (URL, port). Les services Web sont accessibles via SOAP, la requête et les réponses sont des messages XML transportés sur http **[1].**

- Les Web Services proposent aux utilisateurs du Web des fonctionnalités pratiques grâce à un protocole Web standard (dans la plupart des cas, le protocole utilisé est SOAP)
- Les Web Services offrent un moyen de décrire leurs interfaces suffisamment en détail pour permettre à un utilisateur de créer une application cliente capable de converser avec eux cette description est généralement fournie dans un document XML nommé WSDL (Web Services Description Language)
- Les Web Services sont inscrits afin que les utilisateurs potentiels puissent les trouver facilement. Ceci est possible grâce à UDDI (Universal Discovery Description and Integration).

2.1. XML (eXtensible Markup Language) :

SOA, XML = Echange de données entres applications permettant, par exemple, d'éviter de ressaisir une facture fournisseur.

Le **XML**, acronyme de eXtensible Markup Language (qui signifie: langage de balisage extensible), est un langage informatique qui sert à enregistrer des données textuelles. Ce langage a été standardisé par le W3C en février 1998 et est maintenant très populaire. Ce langage , grosso-modo similaire à l'HTML de par son système de balisage, permet de faciliter l'échange d'information sur l'internet. Contrairement à l'HTML qui présente un nombre finit de balises, le XML donne la possibilité de créer de nouvelles balises à volonté.[6]

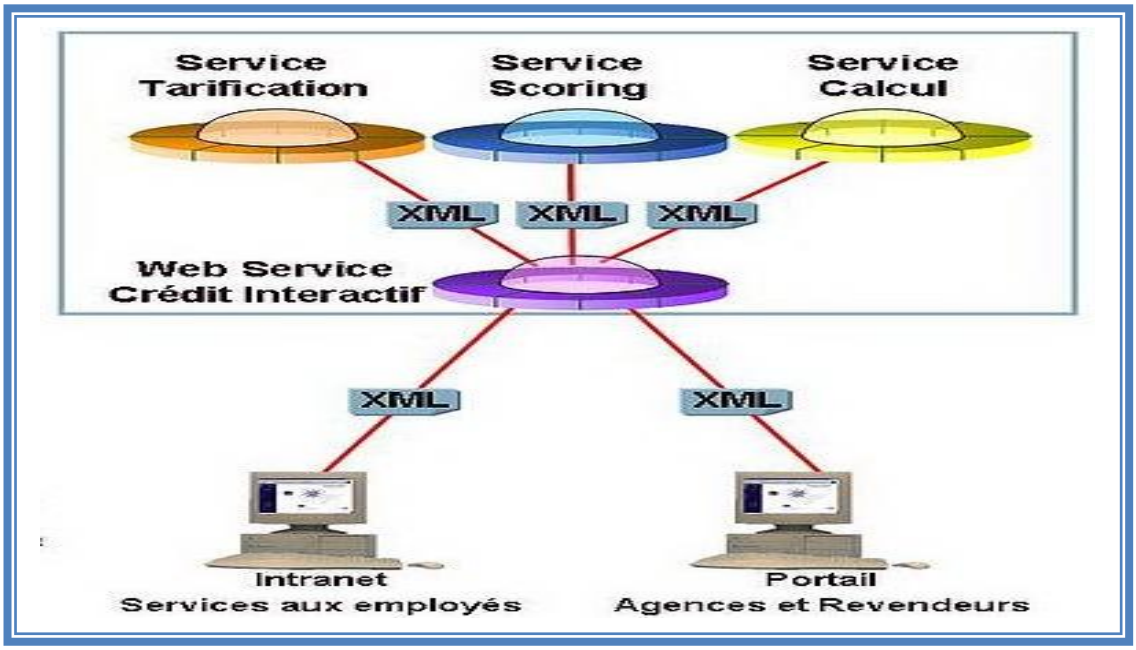


Figure2.1 : *exemple XML*

2.2.1. SOAP (Simple Object Access Protocol)

SOAP (Simple Object Access Protocol) est un protocole d'invocation de méthodes sur des services distants. SOAP utilise les protocoles HTTP et XML et Basé sur XML **[7].**

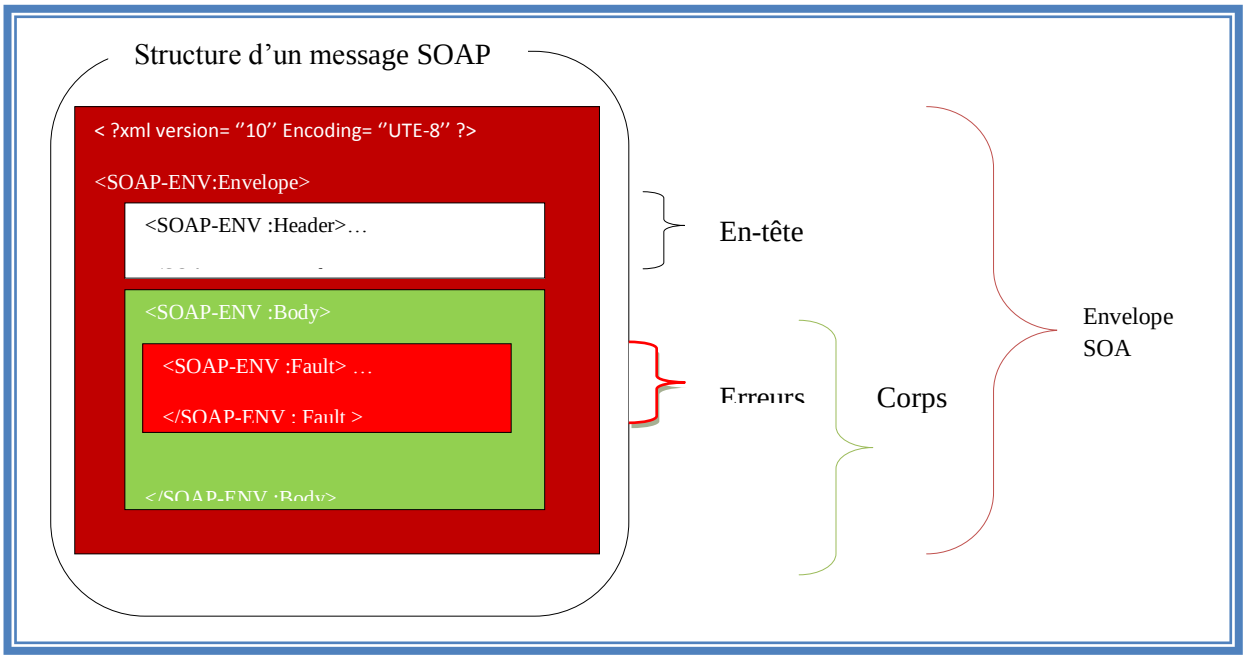


Figure 2.2 : *structure d'un message SOAP*

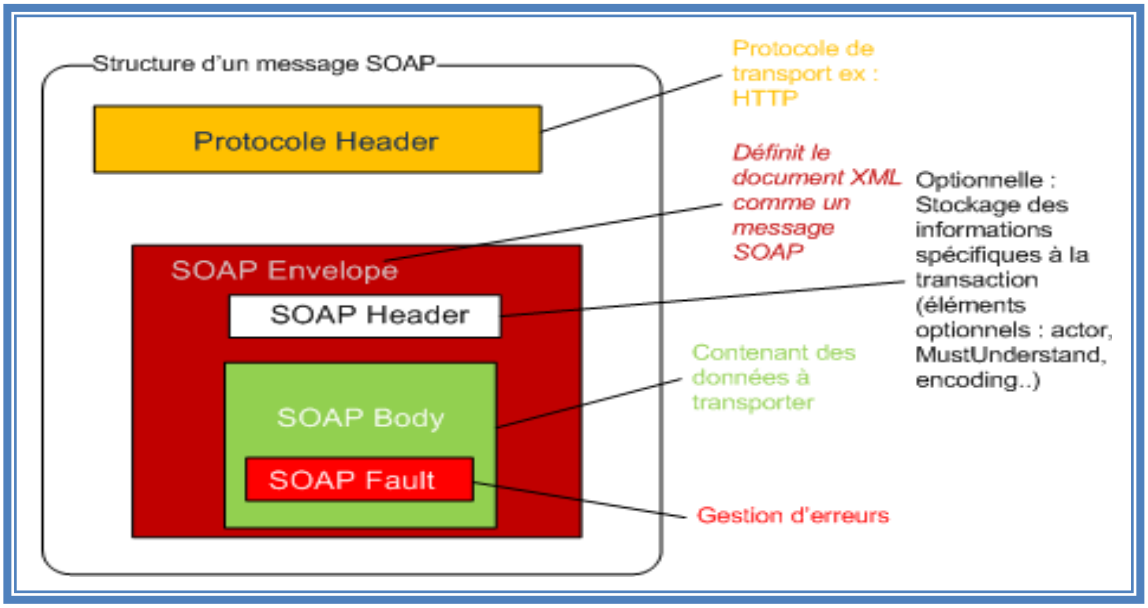


Figure 2.2` : *structure d'un message SOAP*

2.2.2. Structure d'un message SOAP

la structure type d'un message SOAP qui est composée des élément suivants :

- une enveloppe (Envlope)qui définit le document XML comme un message SOAP
- un en-tête optionnel(Header) pour stocker les informations métiers spécifiques a la transaction(authentification, jeton d'autorisation, état de la transaction,…)
- un corps(Body)contenant les données à transporter
- Une gestion d'erreur (fault) qui dentifié la condition d'erreur (si application)
- et enfin des attachement optionnels **[8].**

2.3. UDDI (Universal Description Discovery and Integration)

2.3.1. Définition de l'UDDI (Universal Description Discovery and Integration)

L'annuaire UDDI permet de publier et de découvrir des informations sur une entreprise et ses services web.

Il contient des informations sur les entreprises et les services qu'ils publient. L'inscription d'une société à cet annuaire lui permet de se présenter et d'accélérer les échanges B2B. L'enregistrement d'un service web se fait auprès d'un opérateur. L'annuaire UDDI facilite la localisation d'un service web.

UDDI repose sur le protocole SOAP. Les requêtes et les réponses sont des objets UDDI envoyés sous forme de messages SOAP.

L'annuaire UDDI est consultable sous plusieurs facettes :

- **Pages blanches :** informations générales sur les entreprises.
- **Pages jaunes :** description au format WSDL des services web déployés.
- **Pages vertes :** fournissent des informations techniques détaillées sur les services fournis.

Un UDDI Registry est comparable au service DNS (Domain Name Service) pour les applications métiers **[9].**

2.3.2. Utilisation de l'annuaire UDDI :

La publication des services web inclut la production et la publication des descriptions de service.

Il existe des outils pour générer des parties de WSDL et de créer des entrées dans l'annuaire UDDI à partir de métadonnées.

Une description de service peut-être publiée (=enregistrée dans UDDI Registry) en utilisant des mécanismes variés.

Il existe plusieurs types de nœuds UDDI utilisables par le fournisseur de service :

- Nœud UDDI pour une application interne : se trouvant dans le firewall
- Nœud portail UDDI : publier des services web pour les partenaires externes, fonctionnent à l'extérieur des firewall.
- Nœud catalogue du partenaire avec UDDI : derrière un firewall, le partenaire est choisi avec une autorisation d'accès spécifiée.
- Nœud de place de marché UDDI : relations inter entreprises.

Tous les sites opérateurs doivent reposer sur le protocole SOAP via http pour le transport des messages UDDI **[9].**

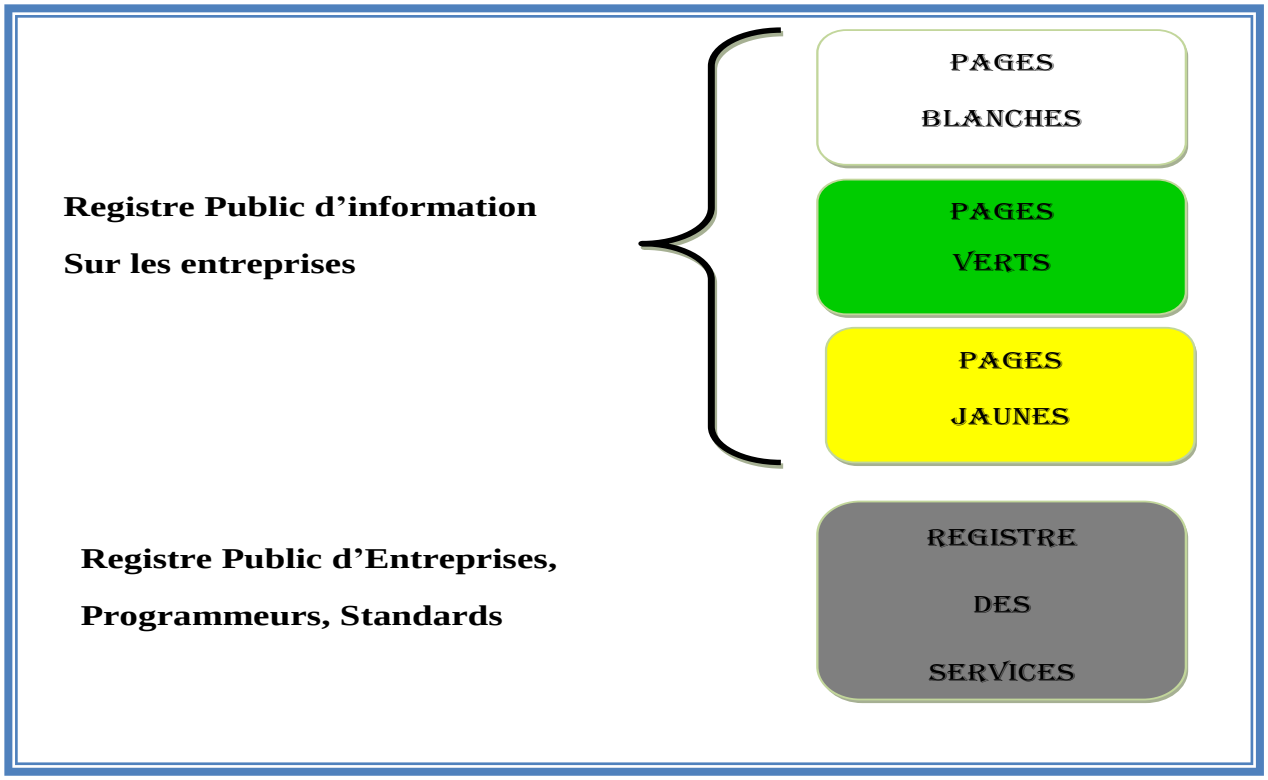


Figure 2.3: *Données du registre UDDI*

2.4. WSDL (Web Services Description Language) :

2.4.1. Définition WSDL (Web Services Description Language)

WSDL (Web Services Description Language) est un format de représentation des interfaces de services Web en XML. Une analogie avec CORBA peut être faite, en effet WSDL est la représentation XML du langage IDL (description d'interfaces).

WSDL a deux rôles prépondérants:

- Il sert de référence à la génération de Proxie.
- Il assure le couplage entre le client et le serveur par le biais des interfaces **[10].**

2.4.2. Structure des documents WSDL :

Le document WSDL peut être divisé en deux groupes de sections. Le groupe du haut est constitué des définitions abstraites, tandis que le groupe du bas contient les descriptions concrètes. Les sections abstraites définissent les messages SOAP de façon totalement indépendante de la plate-forme et de la langue ; elles ne contiennent aucun élément spécifique à la langue ou à la machine. Cela facilite la définition d'un ensemble de services pouvant être

implémenté par différents sites Web. Les questions spécifiques au site, telles que la sérialisation, sont reléguées dans les sections du bas, qui contiennent les descriptions concrètes **[10].**

2.4.2.1. Définitions abstraites :

- **Types**

  Définitions de type indépendantes de la langue et de la machine.
- **Messages**

  Contient les paramètres de fonction (entrées séparées des sorties) ou les descriptions de document.
- **PortTypes**

  Réfère aux définitions de message de la section Messages pour décrire les signatures de fonction (nom d'opération, paramètres d'entrée, paramètres de saisie).

2.4.2.2. Descriptions concrètes :

- **Bindings**

  Indique la (les) liaison(s) de chaque opération dans la section PortTypes.
- **Services**

  Indique les adresses de port de chaque liaison.

3. Pourquoi les services web?

Les services web ont été mises en places afin de répondre à un certains nombre de besoins :

- Remplacer les protocoles actuels (RPC, DCOM, RMI) par une approche les entièrement ouverte et interopérable, basée sur la généralisation des serveurs Web avec scripts CGI
- Faire interagir des composants hétérogènes, distants, et indépendants avec un protocole standard (SOAP)
- Simplifier la communication entre ces composants

- Ne pas créer de nouvelles technologies, mais se baser sur celles qui existent déjà (XML, HTTP)

Les services web sont largement utilises par les entreprises, ce qui leur permet d'exposer un certain nombre de services et d'échanger les informations entre elles **[3].**

4. Fonctionnement des Web services :

Pour expliquer le fonctionnement des web services, il convient de distinguer plusieurs étapes :

1. Recherche dans un annuaire UDDI : le client cherche un service particulier, il s'adresse à un annuaire qui va lui fournir la liste des prestataires habilités à satisfaire sa demande. L'annuaire UDDI peut être comparé à un moteur de recherche sauf que les documents sont remplacés par des services.
2. Recherche de l'interface du composant à contacter : une fois la réponse reçue (en XML) de l'annuaire, le client va chercher à communiquer via une interface. Cette interface décrite en langage WSDL fournit l'ensemble des services disponibles. Elle offre la possibilité de vérifier que le contrat correspond bien aux besoins demandés.
3. Invocation du service : le client doit maintenant passer les paramètres attendus par le service et assurer la communication avec le serveur. L'outil utilisé pour cela est le Proxy, c'est l'objet qui permet la communication entre le client et le serveur. Il est généré par le client en utilisant l'interface WSDL **[4].**

5. 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.

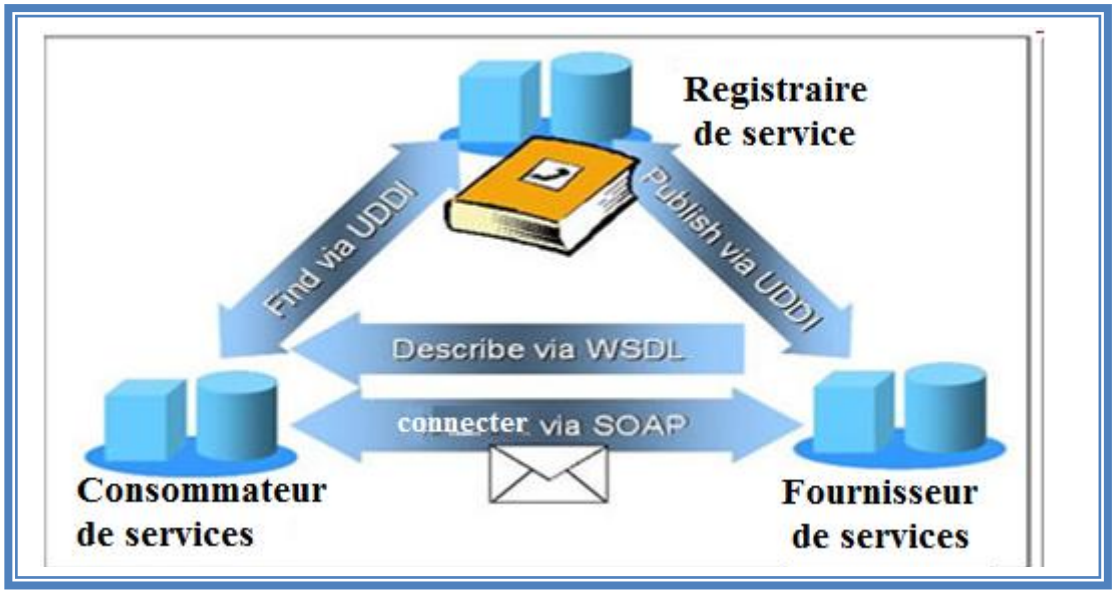


Figure 2.4: *Architecture WSOA (Web Services Oriented Architecture)*

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 « **Implémentation 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.

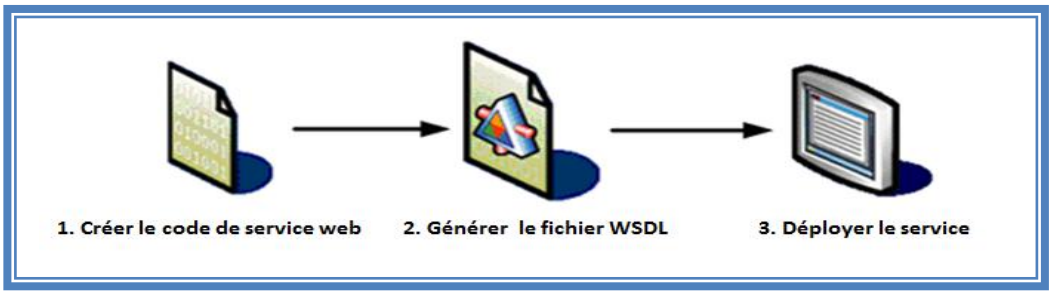


Figure 2.5 : *Développement d'un service Web par la méthode « Code First »*

La technique de développement de services Web appelée « **Code First** » consiste à écrire d'abord le code du service Web (voir étape nº 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 nº 2). Puis en lance le déploiement de service (étape nº 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 à créer d'abord le fichier WSDL est aussi appelée parfois « **Schema First** » (priorité au schéma) ou « **Contract First** ».

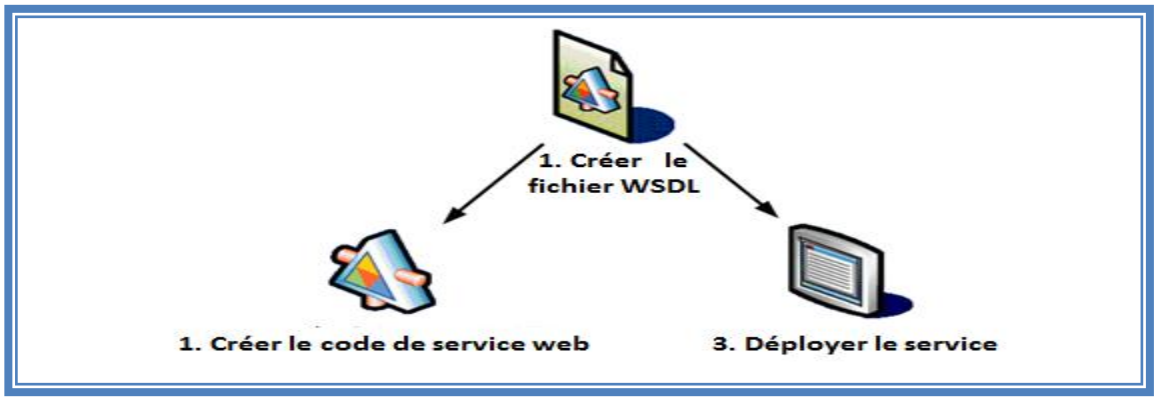


Figure 2.6 *: Développement d'un service Web par la méthode « WSDL First »*

- 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.

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 attachements : 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: SOAP Message, SOAP Part, SOAP Enveloppe, SOAP Header et SOAP Body) .

Tous les échanges de messages avec SOAP utilisent une connexion encapsulée dans la classe SOAP Connections. 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

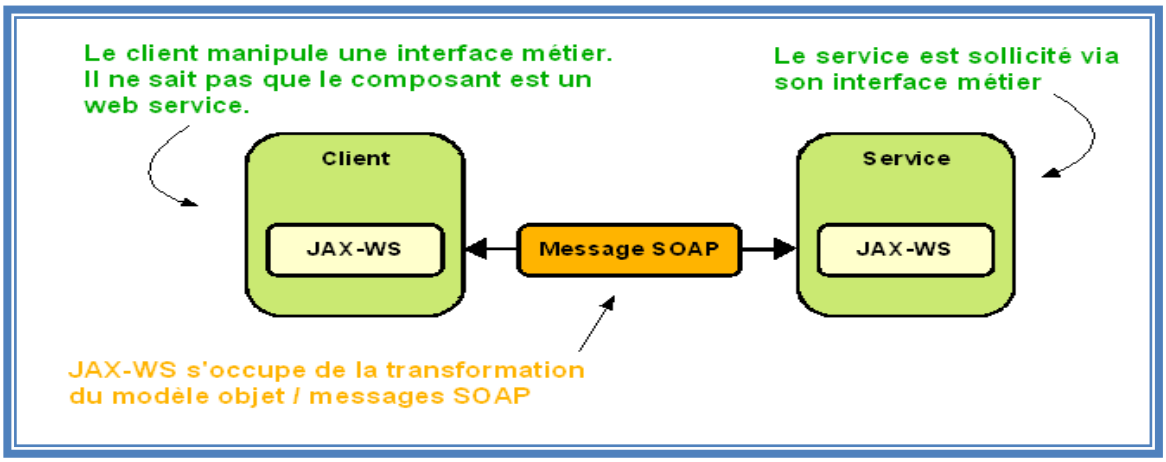


***Figure 2.7** : JAX-WS*

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

Les Web Services possèdent une simplicité de mise en œuvre : Ils rendent en effet accessibles Depuis Internet des fonctionnalités d'une application existante sans modifier en profondeur le système d'information de l'entreprise. Il s'agit de l'ajout de briques supplémentaires. Pour cela, les Web Services exploitent les standards d'échange Internet.

Les services Web avec ses protocoles et ses standards avance vers toujours plus de normalisation.

Déjà, le protocole d'échange de messages SOAP pour standardiser la couche transport et le langage WSDL pour la description de l'interface. Les Web Services reposent sur des bases solides (SOAP et WSDL) qui ont prouvé leurs efficacités et leur maturité même si une normalisation complète n'existe pas encore.

Chapitre 03

Migration ver SOA

1. Introduction :

Les Systèmes d'Information sont en constante évolution et les obstacles rencontrés sont largement débattus depuis des années, mais ces questions acquièrent aujourd'hui une acuité particulière. Les organisations sont confrontées à des demandes de changement toujours plus étendues et plus fréquentes. Ces changements sont liés à la fois aux réorganisations (fusion, acquisitions), à l'ouverture des processus et à la multiplication des évolutions des marchés: On l'observe par exemple dans le secteur des services de télécom, dans les banques assurances, le transport et l'énergie, ou dans le commerce en ligne. Les applications construites et structurées pour répondre à des besoins particuliers dans un contexte donné ne sont plus adaptées aux réalités présentes. Les changements technologiques ajoutent encore des Contraintes supplémentaires. Aux delà d'un certain stade, les coûts induits par les modifications deviennent prohibitifs, et les délais incompatibles avec les demandes métiers. Le système devient trop rigide, prisonnier de son architecture antérieure. Parmi les solutions proposée pour palier les pbs cité c'est la migration vers des architecture plus agile et souple tel que la SOA. En effet la SOA a connu un grand engouement des entreprises de tout secteur et de toute taille en raison de ses avantages économiques et technologiques. La Migration vers l'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 **[03].**

Les promesses de la SOA ont globalement été tenues : agilité et reconfiguration de l'intégration, compatibilité des protocoles entres vendeurs, performances transactionnelles, fiabilité de l'architecture et ouverture vers les nouveaux standards. Mais la migration n'est pas toujours aisée [18].

2. Quelle différence entre une architecture orientée objet et une SOA ?

Au sein d'une architecture orientée objets, les données manipulées sont directement associées au mode de traitement qui leurs est appliquées.

Ce n'est pas le cas au sein d'une architecture de SOA dans laquelle ces deux éléments sont dissociés. Le terme de "service" vient d'ailleurs de cette caractéristique. Par définition, un service a en effet pour but de proposer un résultat particulier, en fonction d'informations qui lui sont envoyées par un tiers. Ce dernier pouvant d'ailleurs très bien décider de transmettre parallèlement ces données à un service complémentaire ou même concurrent qui les prendrait en charge de la même façon **[20]**.

3. La migration vers SOA :

La migration ou la modernisation implique des changements plus étendus qu'une simple maintenance, mais une partie significative de système est conservée. Ces changements incluent souvent la restructuration du système, l'amélioration des fonctionnalités ou la modification des attributs du logiciel **[24]**.

3.1. Bénéfices de la migration :

La migration offre plusieurs bénéfices aux entreprises et à leurs services, parmi ces bénéfices nous citons **[23]:**

- L'adaptation aux nouveaux besoins,
- L'amélioration des services clients,
- Une intégration plus étroite avec les partenaires et les fournisseurs,
- La réduction du coût d'usage des systèmes d'information,
- L'amélioration de la qualité des données,
- L'amélioration du contrôle et de la gestion de sécurité,
- L'augmentation de la flexibilité et la réactivité des systèmes d'information,
- L'élimination de la dépendance forte à l'ensemble des anciennes compétences.

3.2. Difficultés de la migration :

Les efforts de la modernisation de systèmes d'information d'entreprises ont échoué dans la plupart du temps à cause de plusieurs facteurs, nous citons les suivant **[23]:**

- La complexité des systèmes patrimoniaux : la complexité est considérée comme le plus grand limiteur du processus de migration, elle est produite à cause de la taille énorme de système, de l'incompréhensibilité des systèmes à migrer et des phases successives de maintenance.
- Les risques de migration : les risques de migration ne sont pas majeurs, il est possible d'accepter un certain risque si nous devons accomplir une tâche de migration. Malheureusement, beaucoup d'entreprises sont incapables ou ne veulent pas contrôler le risque correctement. Ceci également prévenir de l'insuffisance de compréhension de gestion des risques et des techniques de réduction du risque.

3.2.1. Logiciels patrimoniaux

Les logiciels patrimoniaux (Legacy Softwares), sont des programmes qui ont été Développés avec des technologies qui sont devenues avec le temps périmées. Brodie dans définit les systèmes d'information patrimoniaux comme « n'importe quel système qui résiste significativement à la modification et à l'évolution ». Les SP incluent désormais non seulement les langages des années 1970 - Fortran, COBOL, PLI, C -, mais aussi les langages de quatrième génération des années 1980

CSP, les frames ORACLE, etc et même les premiers langages Orientés objet des années 1990s tels que C++, Smalltalk. De sa part considère que n'importe quel organisme qui a utilisé une technologie d'information pour plus de cinq ans concerné par un problème de logiciel patrimonial. Cela est dû au cycle de rénovation des technologies qui est passé à moins de cinq ans **[22].**

4. Les bénéfices attendu d'une architecture orienté service :

Les architectures orientées services (SOA) sont reconnues comme une approche qui peut entraîner un meilleur alignement des IT 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 **[30]**.

4.1. 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. **[30].**

4.2. 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 **[30].**

4.3 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 **[30]** :

4.4. Efficacité d'affaires :

- Plus grande agilité et meilleure réponse à la dynamique de marché
- Plus grande efficacité des processus
- Meilleur déploiement des ressources **[30].**

4.5. Réduction des coûts :

- Réduction des frais d'entretien
- Réduction des efforts nécessaires pour supporter les changements organisationnels
- Choix technologiques plus flexibles étant donnés le couplage lâche des applications **[30]**.

4.6. Réduction des risques :

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

[30].

5. Stratégies d'évolution vers une architecture orientée service :

Plusieurs travaux ont été proposés pour faire évoluer les systèmes patrimoniaux, la majorité tourne autour de la réingénierie **[32]**. Parmi les premières propositions nous trouvons celle de Brodie & Stonebraker **[31]** qui se sont concentré sur la migration des SP suivant une approche graduelle et complète **[32].** Il y'a même quelques travaux qui ont essayé de synthétiser les différentes stratégies ou approches et les classer selon différents critères :

Bisbal **[33]** a classé les approches d'évolution des systèmes patrimoniaux (SP) en trois catégories :

- Redéveloppement: qui réécrit les applications existantes;
- Wrapping: qui emballe (wraps) un composant existant dans un nouveau plus accessible ;
- Migration: qui transforme un SP en un système plus flexible tout en gardant les données et les fonctionnalités originales

Erradi [34] de sa part, propose une autre classification des différentes approches pour l'évolution des systèmes patrimoniaux vers AOS. Ainsi deux classes principales ont été identifiées **[22] :**

5.1. Approches invasive:

Elles se basent sur la transformation (migration) des SP à travers une réingénierie envahissante qui exige une analyse profonde et détaillée du code existant, la compréhension des fonctionnalités du système et de l'architecture des données, ce qui implique la rationalisation des données et des règles métiers (business rules), suivi d'un processus itératif qui implique le refactoring, la consolidation, et la re-conception des activités afin de rendre le code plus modulaire et faciliter la migration incrémentale vers une architecture flexible **[22].**

5.2. Approches non invasive

Elles se basent sur l'intégration des SP avec le reste des applications SOA, sans modifier le code source mais en ajoutant une couche (wrapping) qui cache les complexités internes et expose de nouvelles interfaces sous forme de services **[22]**.

Umar [8] propose de voir l'évolution des SP vers une architecture orientée service suivant deux Stratégies principales :

1. Stratégie de migration : qui consiste à modifier et restructurer un SP pour avoir à la fin un nouveau système dans notre cas AOS, et qui elle-même peut être effectuée de deux façons :

   b) **Graduelle** : des parties du système sont converties l'une après l'autre ;

   c) Complète (soudaine) : stratégie big bang, tout est remplacé en une fois

2. Stratégie d'intégration : qui consiste à garder la structure interne d'un SP et de l'interconnecter avec l'extérieur en utilisant les services web et ESB. La technique la plus utilisée est le wrapping **[32]** essaye plutôt de proposer un modèle décisionnelle qui permet d'exposer les avantages et les inconvénients de chaque stratégie, puis il offre un support pour les entreprises pour décider et opter pour une des stratégies selon leur cas précis.

Marchetto **[35]** met l'accent sur la migration et classifie les travaux effectués sur la migration en quatre catégories :

- **Approches Orientées Métier (Business-oriented**): Approches qui utilisent les informations du processus métier réalisé par l'application cible comme point de départ pour réaliser la migration;
- **Approches Basées fonctionnalités (Functionality-based):** Approches qui se concentrent principalement sur l'identification des fonctionnalités implémentées par le système cible, puis les localiser dans le code en utilisant une des techniques tel que: le slicing ou feature location.
- **Approches Basées modèle *(Model-based):*** Approches qui construisent un nouveau système orienté service en partant du modèle du système cible, ce modèle est généralement extrait par l'application des techniques de retro-ingénierie en commençant par l'analyse des artefacts du système cible tels que le code, les traces d'exécution et la documentation.
- **Approches Basées interaction *(Interaction-based) :*** Approches qui utilisent comme point de départ pour la migration, les interactions entre l'utilisateur et le système cible ou entre les composants systèmes (tel que : les bases de données et la logique métier);

He Yuan Huang **[36]** propose une classification en deux grandes catégories qui sont l'intégration et la migration**.** Il défend l'approche d'intégration qui selon ce papier, la façon la plus normale pour l'évolution des SP vers une Architecture Orientée Service, invoquant les risques que comporte l'approche de migration qui consomme beaucoup de temps, de coûts et aussi peut être source d'erreurs lors de la compréhension des fonctionnalités du système.

Pour la migration deux classes d'approches peuvent être identifiées:

a) Ceux qui se basent sur le code source et la documentation et qui utilisent l'analyse statique pour abstraire la conception du haut niveau du système cible ;
b) Ceux qui sont une variété des méthodes de profiling, de test et de traçage pour l'observation du comportement du système, mais qui nécessitent un travail manuel intensif [**36].**

Beaucoup d'autres travaux se mettent d'accord pour la classification des approches d'évolutions des systèmes patrimoniaux vers une architecture orientée service en deux grande catégories qui sont l'intégration et la migration avec plus au moins quelques sous classes.

Almonaies dans **[37]** propose la même classification que **[33]** (redéveloppement, wrapping et migration), puis ajoute une quatrième catégorie qui est le remplacement. Donc il propose une classification en quatre catégories qui sont :

a) 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. **[22]**

b) 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 [38]"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. **[22]**

c) 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 **[22].**

d) 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 **[22]**.

En plus des quatre catégories citées **[38]**, 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, OO, OC, 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 [22].

6. Les approches de SOA :

Le choix entre ces approches dépend de certains critères décisifs, tel que :

- L'état des systèmes informatiques existants (nature, degré de complexité, capacités fonctionnelles et techniques).
- Les objectifs attendus de la migration.
- Les capacités des groupes de travail...etc [21].

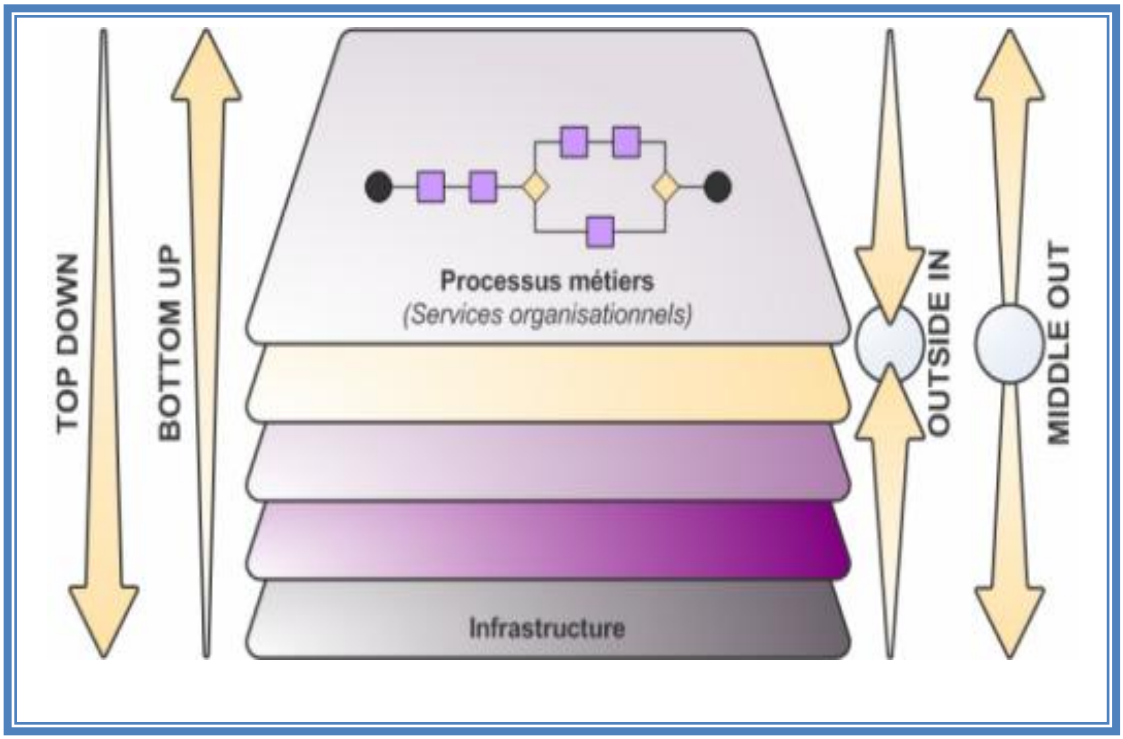


***Figure 3.1** : Les différentes approches de la migration*

6.1. Top Down :

Le point de départ de cette approche est le suivant : puisque l'objectif de la mise en œuvre d'une SOA est d'aligner le SI sur le métier de l'entreprise, c'est en toute logique qu'elle doit partir de la définition (ou de la formalisation) des processus métiers pour descendre ensuite au travers des différentes strates du SI afin de définir les services nécessaires à la réalisation de ces processus (en commençant par la définition des services de plus au niveau, c'est-à-dire offrant le plus de valeur ajoutée métier).

Cette approche représente la voie royale **:**

- Elle permet de piloter la SOA par les besoins métiers.
- Elle minimise la redondance de services.

Elle n'est cependant que rarement possible car :

- Elle signifie une refonte de tout ou partie du SI, jugée bien souvent trop coûteuse et trop risquée.
- Elle rend difficile l'adhésion à la démarche des équipes MOE (Maîtrise d'œuvre) **[21]**.

6.2. Bottom Up :

A l'inverse, cette approche prône une phase de conception ascendante. Une analyse de l'existant (**cartographie applicative**) permet de déterminer les fonctions existantes du SI. Ainsi cartographiées, il est possible d'identifier les fonctions du SI qui sont éligibles au rang de service. Une fois ces fonctions mises en mode service, elles sont exploitables au sein de services à forte valeur ajoutée et / ou de processus métiers.

Elle présente cependant des travers rédhibitoires :

- Elle bloque le pilotage de la SOA par les besoins métiers.
- Elle ne favorise pas la mise en place d'un effort transverse : elle ne permet pas de sortir de la « culture projet ». En effet, les services émergeants de cette approche restent très fortement couplés à leur application d'origine.
- Elle rend très difficile la justification de l'investissement auprès du métier, qui n'en verra les bénéfices qu'une fois les services auront été rendus exploitables au sein de processus métiers.

Enfin, cette approche se limite à une mise en mode service de fonctions existantes sur le SI **[21].**

6.3. Outside In (Meet In The Middle):

Cette approche préconise de mener en parallèle :

- Un chantier Top Down pour définir les processus métiers et les services de plus haut niveau nécessaires à leur réalisation.

- Un chantier Bottom Up afin de cartographier l'existant applicatif dont dispose l'entreprise pour supporter les services métiers à forte valeur ajoutée.

De part sa nature, cette approche réunit les bénéfices des approches Top Down et Bottom Up**.** Elle permet de piloter la SOA par les besoins métiers tout en facilitant la réutilisation de services et la capitalisation sur l'existant **[21]**.

6.4. Middle Out

Cette approche peut être introduite par la phrase « BPM et SOA sont les deux faces de la même pièce ». Assertion qui fait référence à un autre débat stérile ayant fait couler beaucoup d'encre et qui rappelle qu'il est impossible de dissocier la définition des processus métiers de l'établissement du catalogue de services**.**

Par opposition à l'approche Outside In, cette méthode propose de commencer « In the middle », c'est-à-dire là où le métier et les IT parlent le même langage (ou en tout cas presque). Elle s'attaque donc d'emblée à ce qui reste un des principaux freins à l'adoption des SOA: La compréhension du métier de l'entreprise par les maîtrises d'œuvre et inversement, la compréhension des contraintes IT par les maîtrises d'ouvrage**.**

Une fois les différentes parties d'accord sur un premier socle de services « métiers » nécessaires :

- La maîtrise d'ouvrage engage un chantier Middle Up pour spécifier les processus métiers.
- La maîtrise d'œuvre engage un chantier Middle Down pour spécifier le socle de services de plus bas niveau permettant la réalisation des services métiers.

Cette approche fait donc la part belle au dialogue entre métier et IT et pose la compréhension mutuelle comme pré-requis à la mise en œuvre d'une SOA. Elle limite par contre le pilotage de la SOA par les besoins métiers, puisque le point de départ est l'identification des services métiers nécessaires et non la définition des processus réalisant le métier de l'entreprise **[21]**.

7. Conclusion :

La migration des systèmes d'information d'entreprise est considérée comme la meilleure solution permettant l'intégration des applications dans le périmètre de l'entreprise ainsi qu'à l'extérieur de son périmètre. Elle permet de garder et réutiliser les parties intéressantes qui supportent les besoins métiers de système intégré et en même temps d'accompagner l'évolution technologique. Dans ce chapitre, nous avons présenté plusieurs travaux permettant la migration des systèmes d'information d'entreprises vers une architecture orientée services. Comme nous avons vu, il existe principalement quatre grandes classes d'approches de migration : *l'approche Top Down, l'approche Bottom Up, l'approche Outside In et l'approche Middle Out*. Ces approches se distinguent principalement par la stratégie d'évolution et de développement adopté dans un environnement SOA.

- La migration vers une architecture SOA ne peut être que graduelle.
- Elle implique des modifications fondamentales dans l'organisation de l'entreprise en général …

Chapitre 04

Conception

1. Introduction :

Depuis une décennie, l'architecture orientée service (SOA) 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 SOA est devenue l'une des techniques importantes de modernisation des SP. Elle adèles 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 **[22]**.

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 **[43]** affirme que n'importe quel système qui date de plus de cinq ans, est concerné par une nécessité de modernisation. 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 simple qui permet de migrer d'un système orienté objet vers un nouveau orienté service. A travers ce chapitre, nous allons décrire les différentes étapes utilisées pour atteindre cet objectif.
Nous allons définir premièrement les objectifs, le fonctionnement général de l'application, puis le processus de migration de l'application OO vers Une application SOA.

2. L'objectif :

L'objectif principal de ce travail est de réaliser la migration d'un système OO vers un nouveau système orienté service pour atteindre ce but nous détaillons les points suivant à suivre :

- Identification des services potentiels à partir de l'application OO initiale à travers l'analyse des interactions des classes puis le regroupement de ces dernières pour la formation de services.
- Validation des services résultants de la première étape par le concepteur.
- Procéder à la migration de l'application OO vers une nouvelles orientée service.

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 :

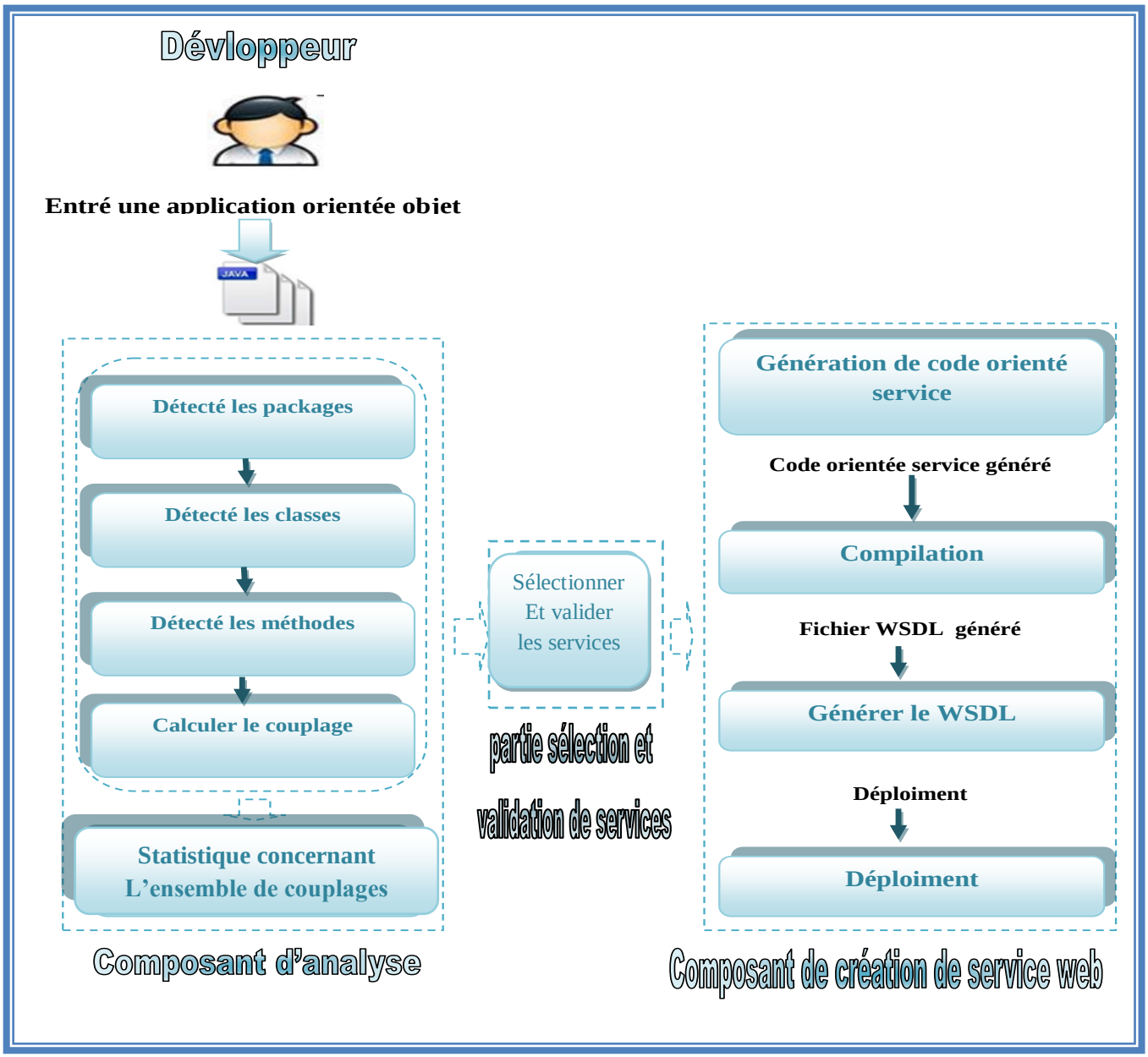


Figure 4.2 : *Fonctionnement générale de l'application*

- Sélectionner le projet qui contient le code source orienté objet.
- Analyser le projet et générer les informations existantes.
- Calcul de couplage entre classes.
- Proposer les services candidats
- Sélection et validation des services
- Conversion du projet OO en dynamique web projet.
- Compiler les classes de ce projet.
- Générer le code de service web jax-wset cela par la modification du code source initial.
- Compiler le code obtenu.
- Générer les fichiers WSDL pour chaque service web en utilisant les fichiers compilés« .class »
- Validation des services web.
- Tester les services
- Déploiement des services générés.

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 (Figure 4.3) :

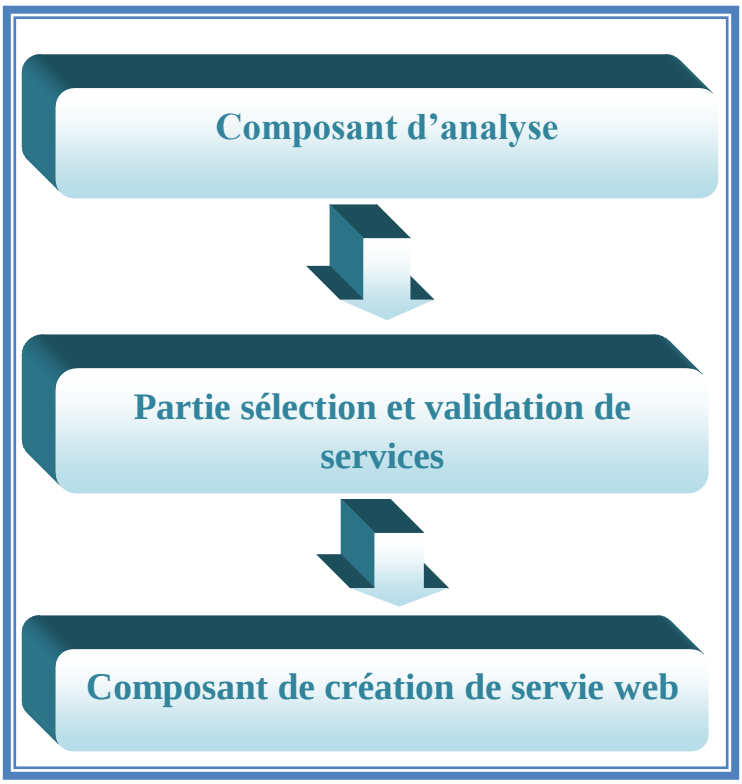


Figure 4.3 : *Etapes de génération de l'application orientée service*

Étape 01 : Composant d'analyse

Cette étape consiste à analyser le code source orienté objet (les classes, les méthodes et les instructions…) afin d'extraire l'interaction entre les différents éléments des classe, tel que les appels entre méthodes, l'utilisation des instances de classes, etc. Ces informations vont permettre de calculer le couplage entre classes en étudiant les dépendances vers et à partir des classes, des packages.

Étape 02 : Partie sélection et validation de services

Cette étape nous permet de calculer la dépendance entre classes à partir de l'analyse des résultats de l'étape précédente et à partir des calculs de couplage nous pouvons déduire les classes qui sont fortement ou faiblement couplé. A la fin des calculs, ce composant va nous proposer les groupes de classes candidates pour former des services. Le concepteur aura le choix de valider la proposition automatique ou de faire une sélection manuelle des regroupements de classe pour former les services.

Métriques de couplage

Le couplage a été introduit, d'abord, pour les systèmes procéduraux. Stevens et al. **[39]** définissent le couplage comme "la mesure de la force de l'association établie par une connexion d'un module avec un autre". D'après Pressman**[41]**, "le couplage est une mesure d'interconnexion parmi les modules formant la structure du logiciel". Un module présentant un fort couplage est un module complexe. Cette complexité se traduit par la difficulté à comprendre le module, à détecter les erreurs, à les corriger et à le changer. L'interaction d'un module avec plusieurs modules du système le rend plus difficile à maintenir puisque chaque changement subi par le module peut affecter les modules auxquels ce dernier est associé.

Dans notre cas nous avant utilisé cette métrique pour regrouper les classes liées dans un seul service, les classes n'ayant aucune relation avec les autres peuvent être converties en des services indépendants. Dans ce qui suit nous donnons quelques cas de figure. (la figure 4.4).

Cas 01 :

Dans ce cas la classe 4 fait des appels vers les classes 1,2 et 3 mais ces dernières ne sollicitent aucune classe externe alors nous proposons les classes 1, 2 et 3 comme des services fournisseur.

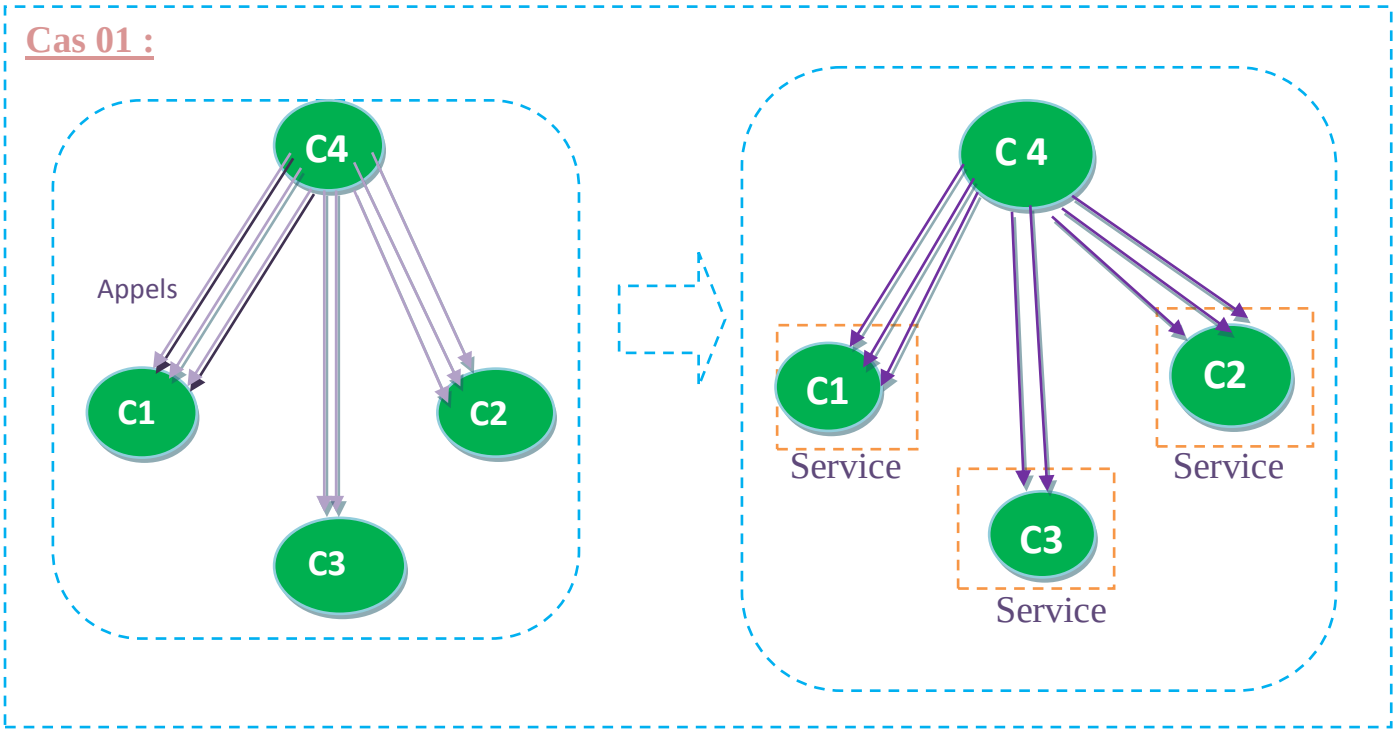


Figure 4.4 : *Des exemples de couplage.*

Cas 02 :

Le deuxième cas la classe 3 reçoit des appels externes, mais n'émet aucun appel vers l'extérieur donc nous choisissons cette classe comme un service fournisseur.

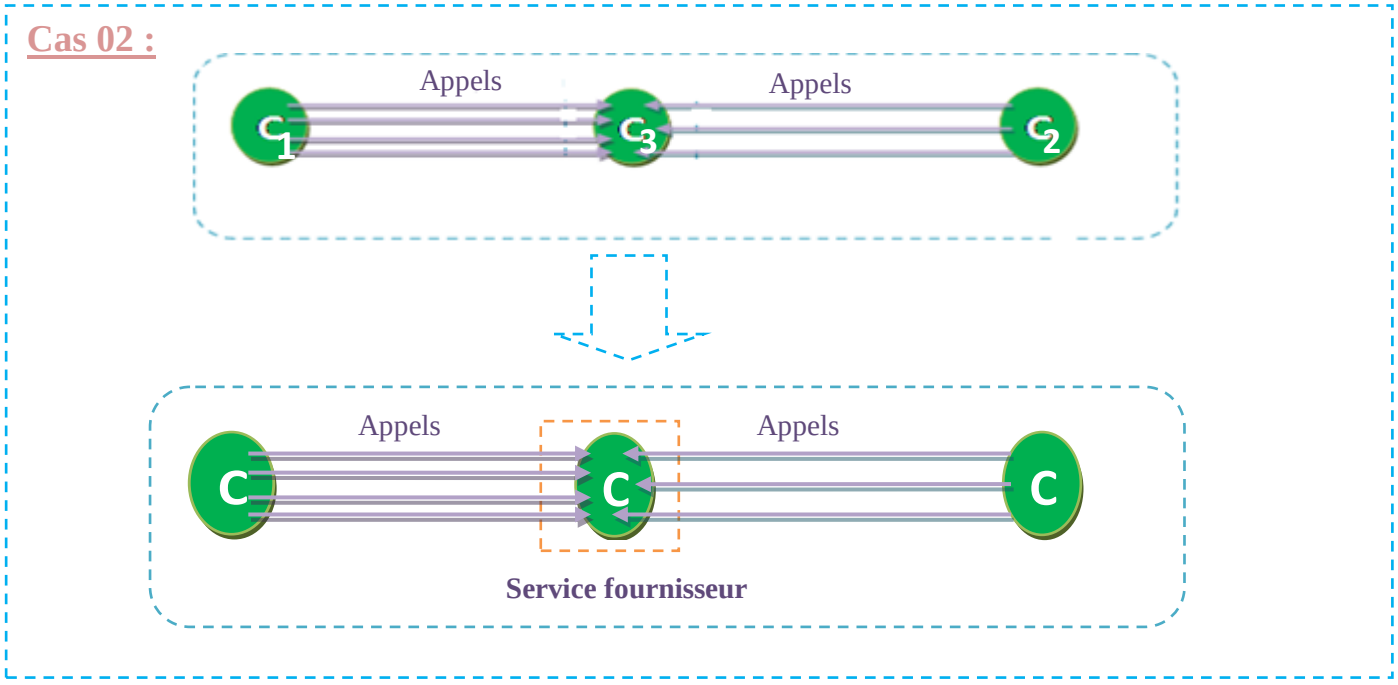


Figure 4.4': *Des exemples de couplage.*

Cas 03 :

Le dernier cas, les classes 3 et 5 sont fortement couplés donc elles seront proposées comme un seul service, la même chose pour les classes 6 et 7.

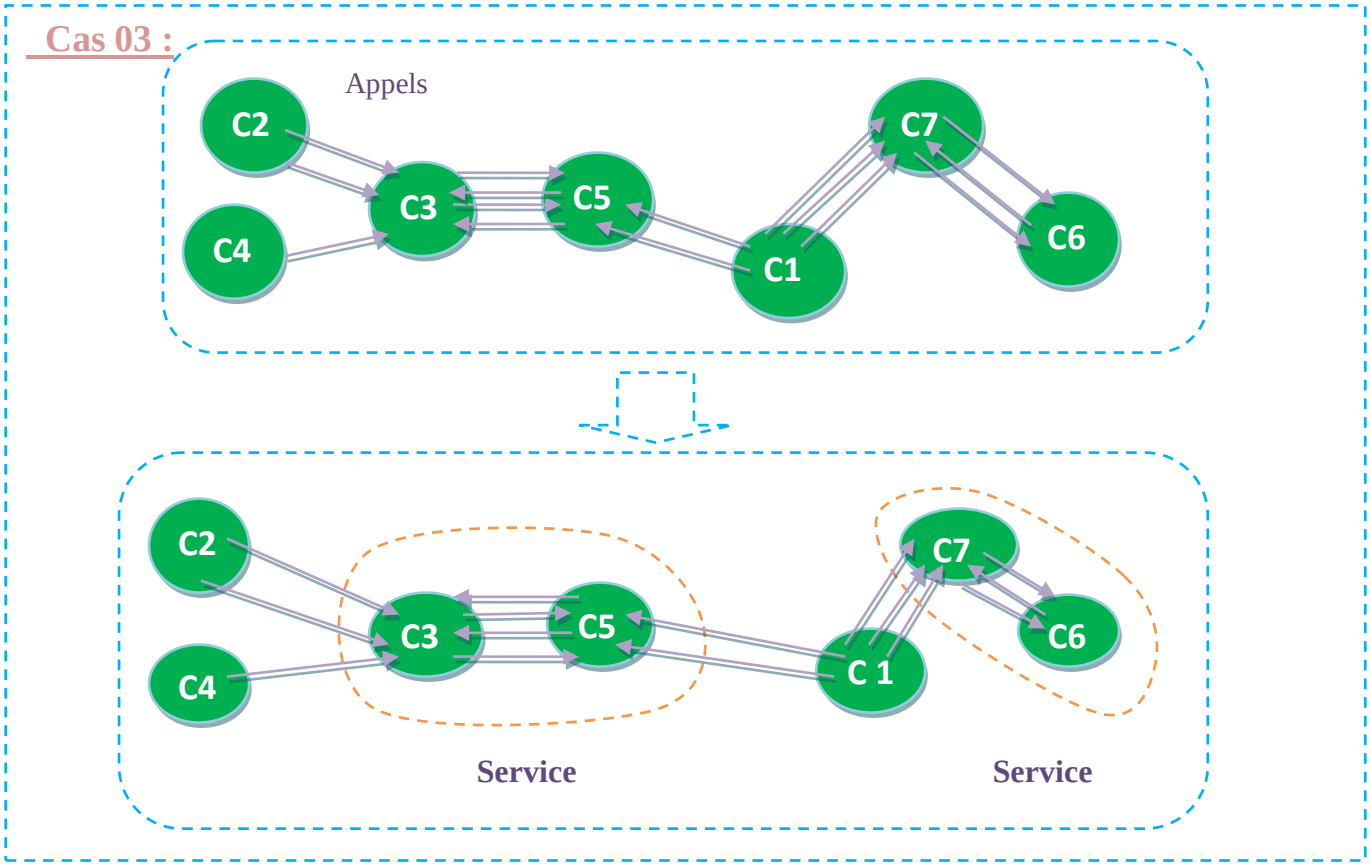


Figure 4.4'' : *Des exemples de couplage.*

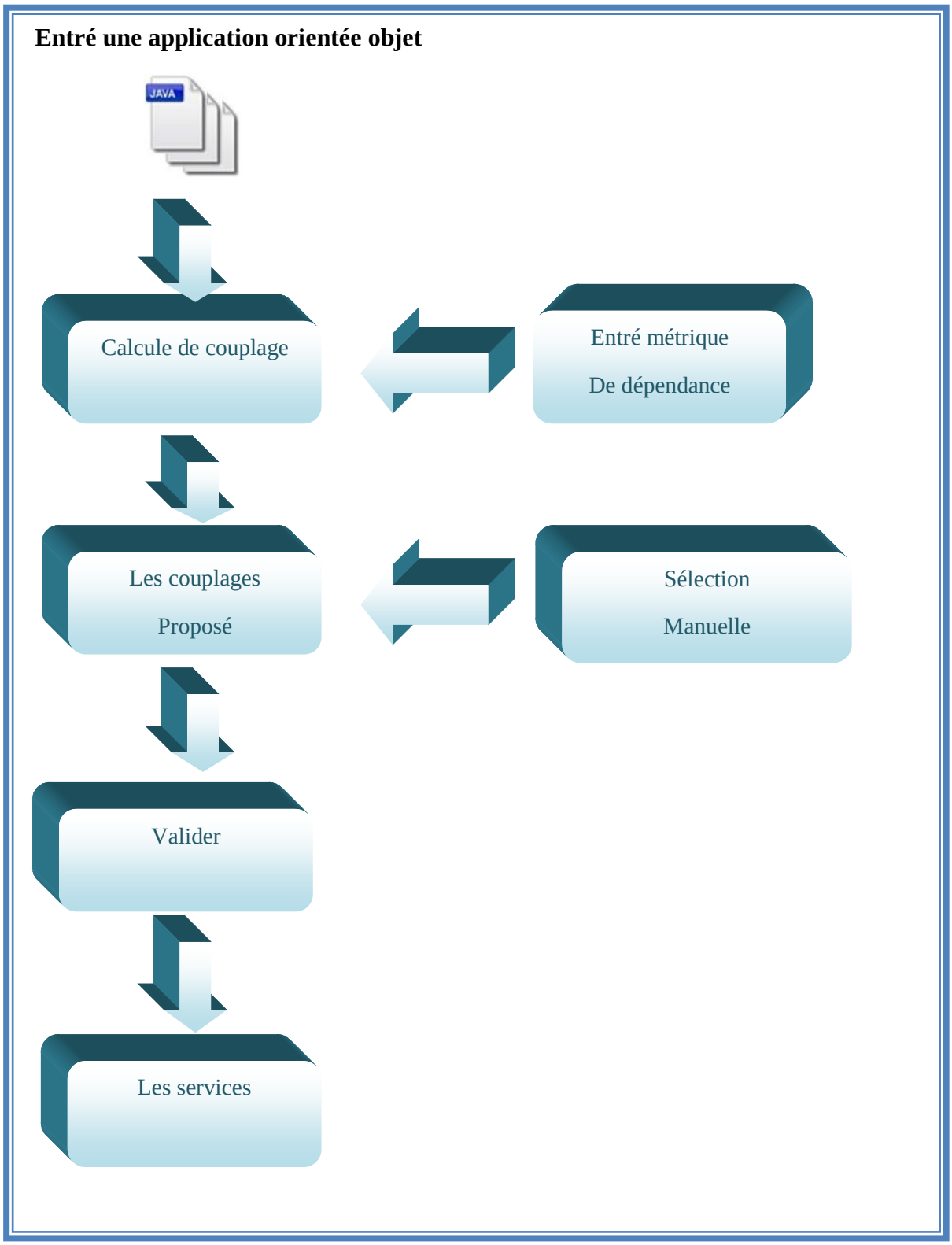


***Figure 4.5** : fonction de la Partie sélection et validation de services*

Étape 03 : Composant de création de servie web

Cette étape procède à la migration des classes vers des services, après validation des services identifiés, pour le faire, nous suivons les étapes suivantes :

1) Création du code du service web: cette étape permet de créer le code de service web (le nouveau code) à partir du code source orienté objet.
2) Étape de compilation : cette étape permet de compiler le nouveau code obtenu.
3) Génération de l'interface WSDL : elle consiste à créer les interfaces qui contiennent la description des services Web (génération des fichiers WSDL).
4)) 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
5) 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.

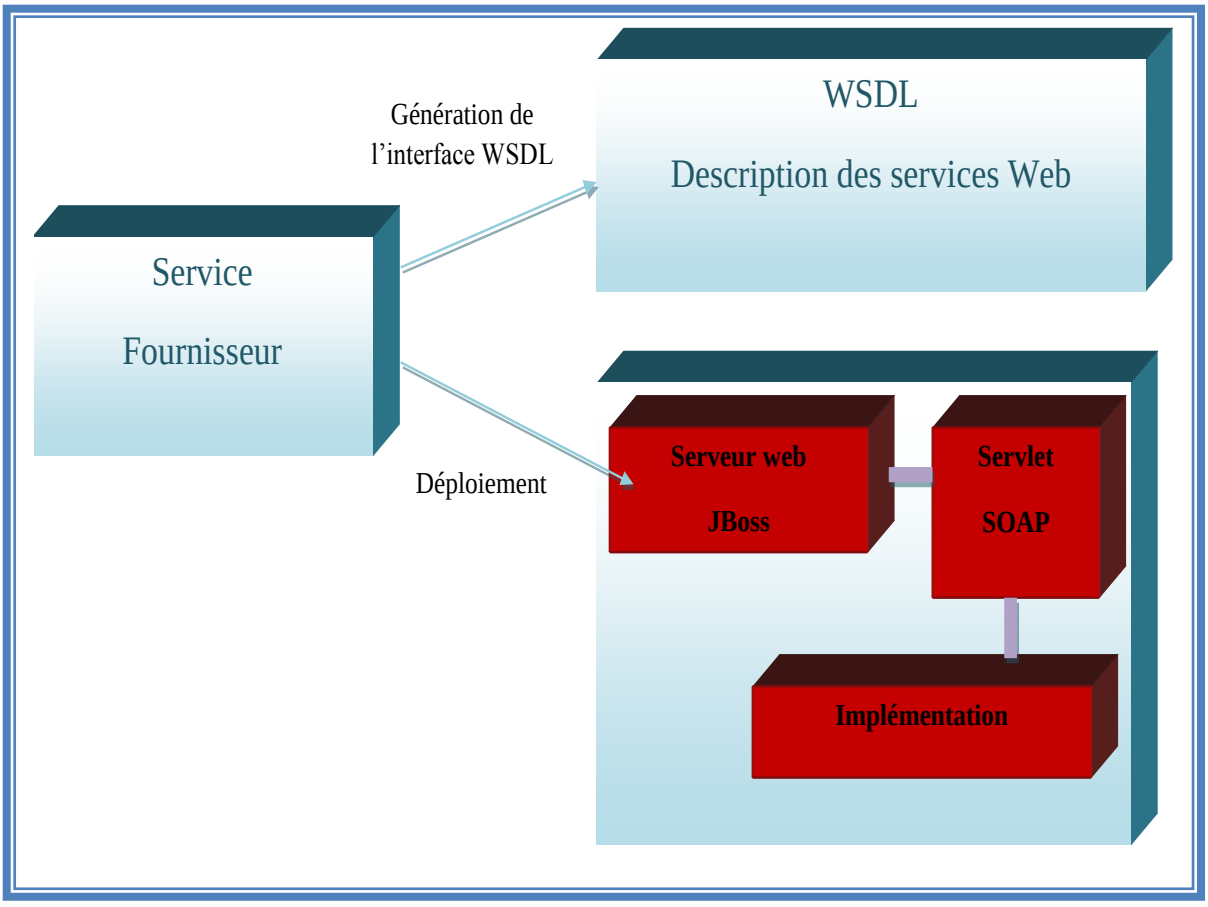


Figure 4.6 : *la structure interne du Composant de création de servie web*

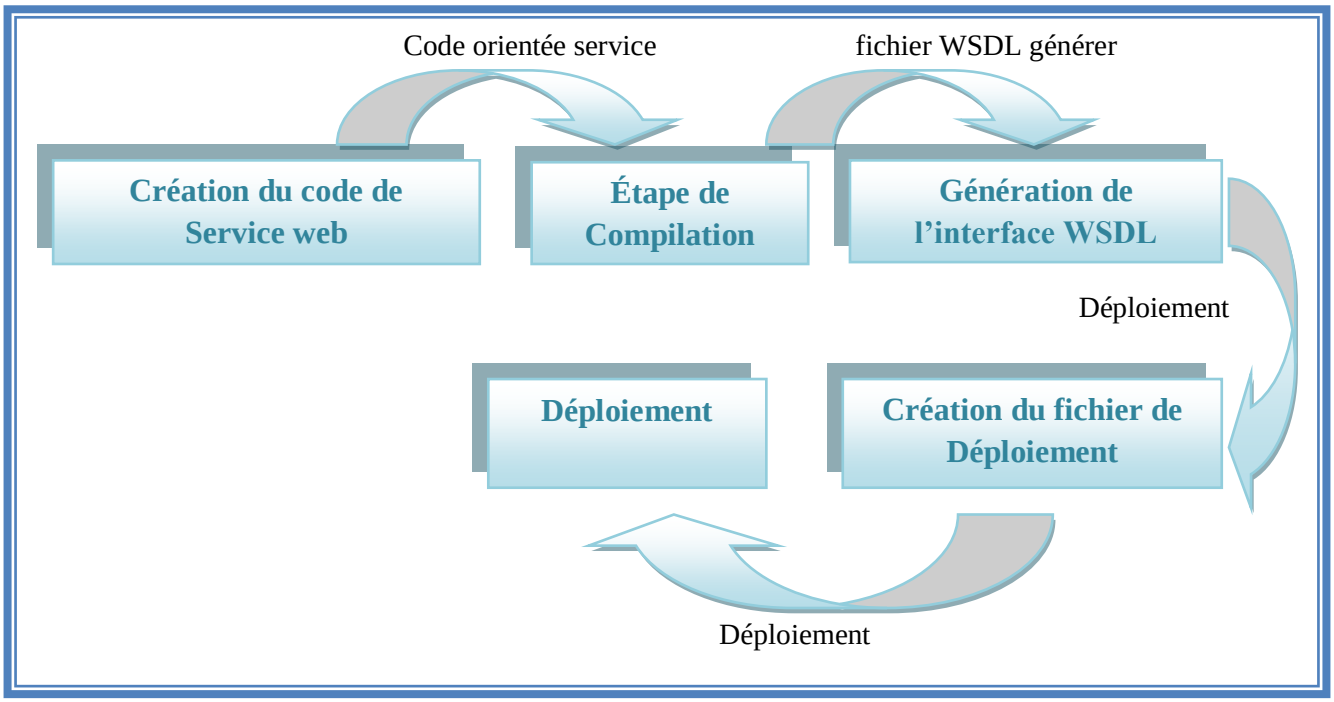


Figure 4.7 : *Etapes du Composant de création de servie web*

5. conclusion :

Nous avons présenté dans ce chapitre l'étude conceptuelle de notre outil ainsi que son architecture générale qui est composé du composant analyse, composant création de service.

Le prochain chapitre sert à présenter les techniques utilisées pour implémenter l'application conçue dans ce chapitre.


Chapitre 05
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 parla présentation de l'environnement logiciel, sur lequel l'application a été réalisée. En suite, 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).

2.3.JBoss Application Server

JBoss Application Server est un serveur d'applications J2EE Libre entièrement écrit en Java, publié sous licence LGPL. Parce que le logiciel est écrit en Java, JBoss Application Server peut être utilisé sur tout système d'exploitation fournissant une machine virtuelle Java

(JVM).Les développeurs du cœur de JBoss ont tous été employés par une société de services appelée " JBoss Inc. ". Celle-ci a été créée par Marc Fleury, le concepteur de la première version de JBoss. Le projet est sponsorisé par un réseau mondial de partenaires et utilise un business model fondé sur le service. En avril 2006, RedHat a racheté JBoss Inc. En février 2007 Marc Fleury quitte le groupe RedHat. JBoss Application Server implémente entièrement l'ensemble des services J2EE.

2.3. JAX-WS(Java API(application programming interface) for XML Web):

Java API for XML Web Services (JAX-WS) offre un ensemble d'outils puissants pour développer une Architecture Orientée Services (SOA).

L'API Java pour les services Web XML (JAX-WS) est une API du langage de programmation Java pour la création de services Web. Il fait partie de la plate-forme Java EE de Sun Microsystems. Comme les autres API Java EE, JAX-WS utilise annotations, introduites dans Java SE 5, pour simplifier le développement et le déploiement des clients et points de terminaison de service Web. Il fait partie de la Web Services Pack de développement Java.

2.4.SoapUI :

SoapUI est un outil graphique qui permet de tester des services web basés sur diverses technologies. Il est disponible en deux versions : une version gratuite et open source et seconde version payante. Il est également disponible sous forme de plugin pour les IDE Netbeans, IntelliJ IDEA et Eclipse. SoapUI est développé entièrement en Java et utilise Java Swing pour son GUI, il fonctionne donc sur la plupart des systèmes d'exploitation et en plus il est disponible sous licence GNU.L'outil gère respectivement les services web basés sur les technologies telles que le **HTTP (S)**, **HTML**, **SOAP (WSDL)**, **REST**, **AMF**, **JDBC et JMS**. Il permet de mettre en place une suite de tests qui peuvent être lancé d'une traite du côté client, permet de tester les services web en mode bouchon mais aussi d'effectuer des tests de charge. Il permet entre autre de fournir une hiérarchie des services web, de lister les différentes méthodes disponible, les paramètres attendus, de réaliser des requêtes et de récupérer les réponses … C'est l'un des meilleurs outils de test unitaires concernant les services web.

4. Configuration des outils

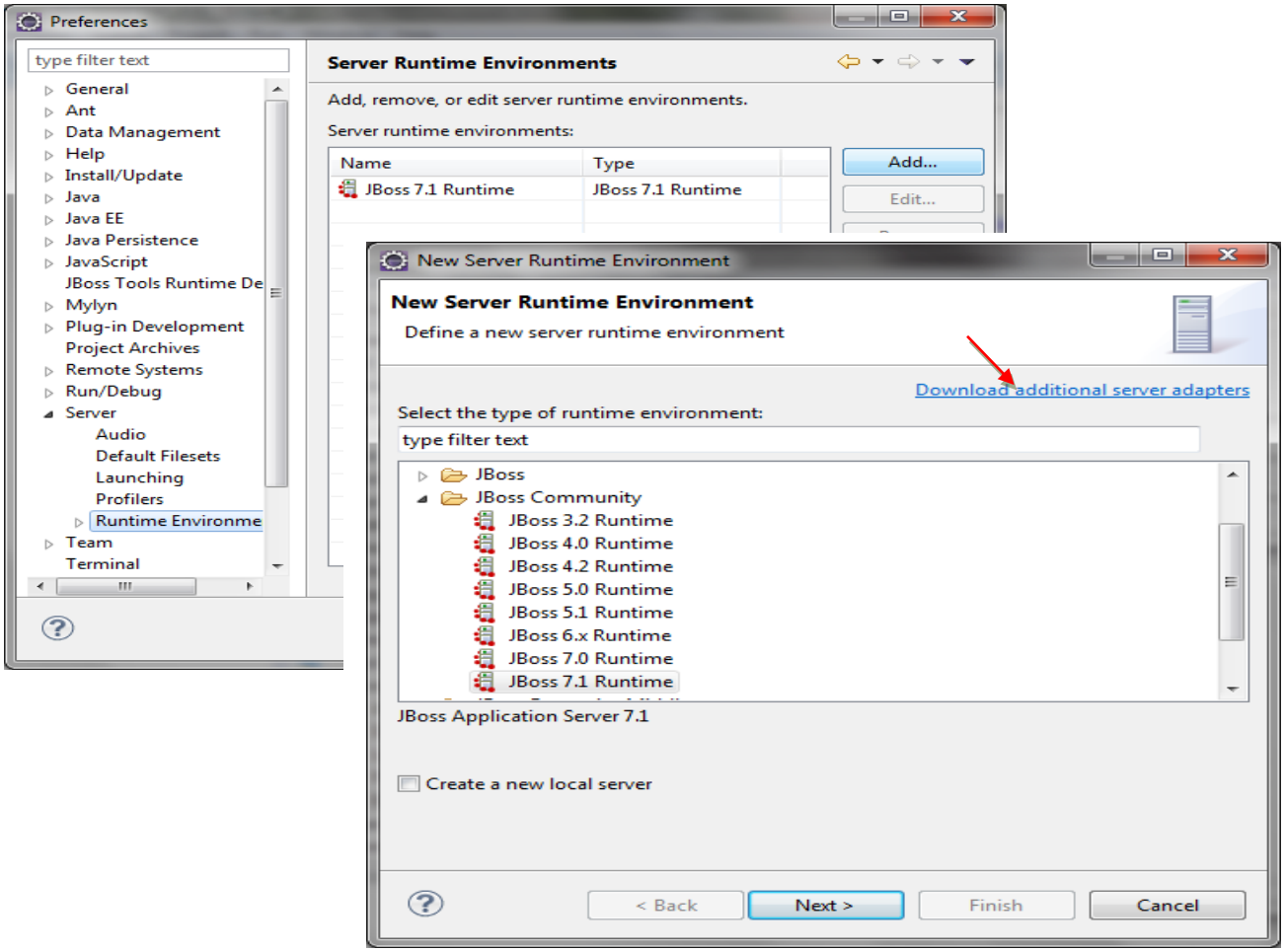
4.1. La configuration de serveur JBoss

Pour configurer le serveur JBoss il suivre les étapes suivant

1) Eclipse -- Window-- Preferences.

2) Server – Start.

3) Vérifier si le server et déploie.



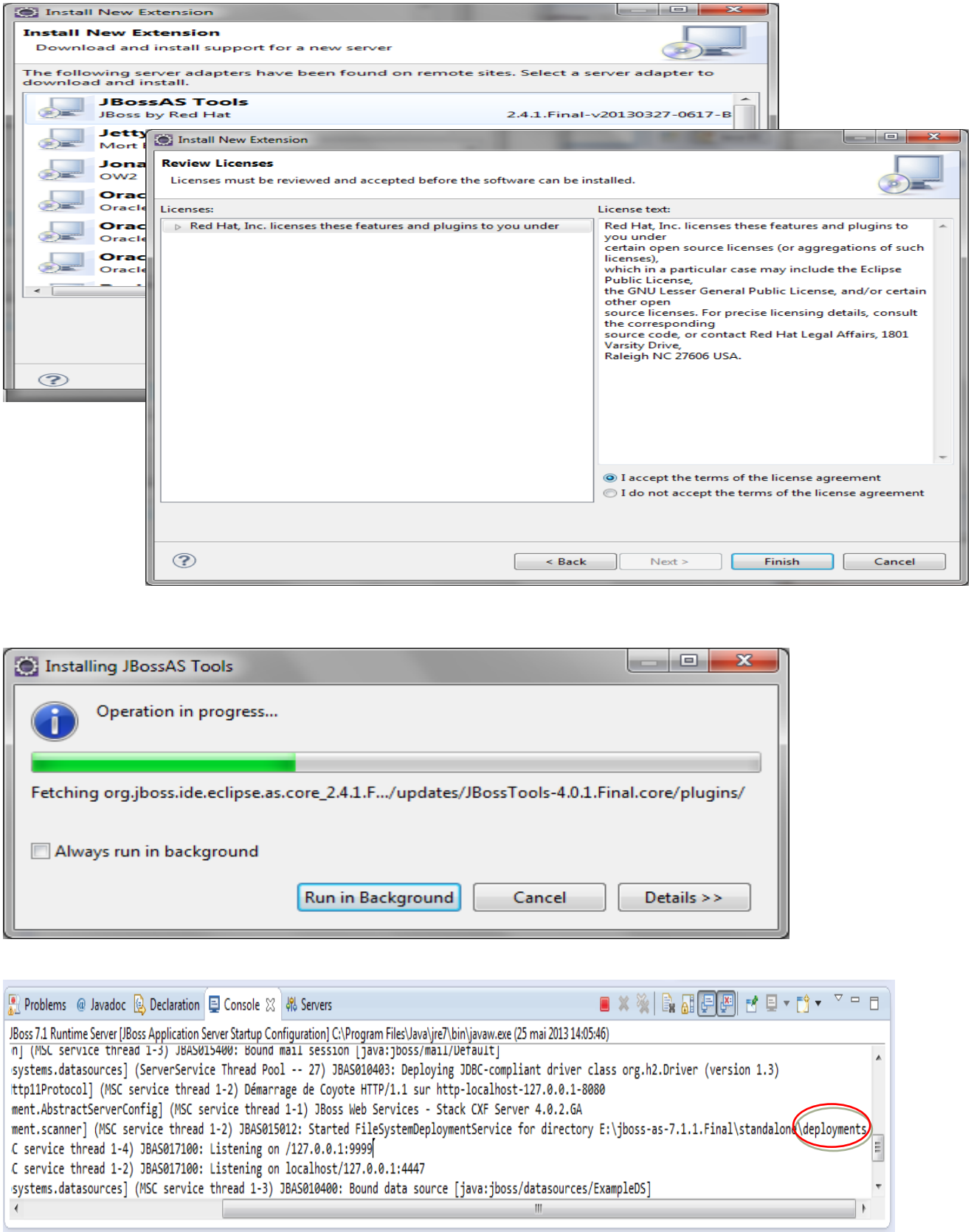


Figure 5.1 *: La configuration de serveur JBoss*

5. 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 de couplage.
3)) le package qui contient les classes principales.
4) le package qui contient les classes de création des projets dynamiques.
5) le package qui contient les classes de création des web service.

5.1. Package analyseur :

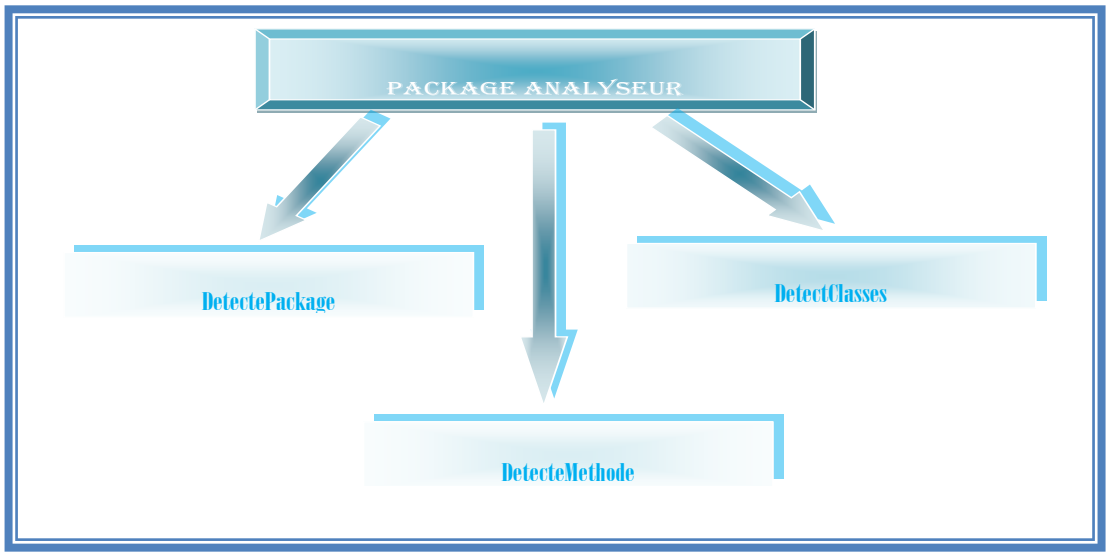


Figure5.2 : *Package analyseur*

Ce package contient 3 classes la première classe est :

a. **Classe DetecteMethode :** Nous Donne les méthodes de chaque classe.
b. **Classe DetectClasses :** Montre les classes qui composent le package.
c. **Classe DetectePackage :** Montre les package de projet.

5.2 Package couplage :

Contient le Logiciel Stan, qui contribue à donner le couplage entre les classes et les packages. La vue Accouplements permet de regarder autour du projet choisi, pour voir toutes ses dépendances entrantes et sortantes. Nous pouvons récupérer les dépendances vers et à partir des classes, des packages, des arbres de paquets ou les bibliothèques.

En plus des dépendances directes, on peut éventuellement visualiser les dépendances intermédiaires visibles entre les objets affichés.

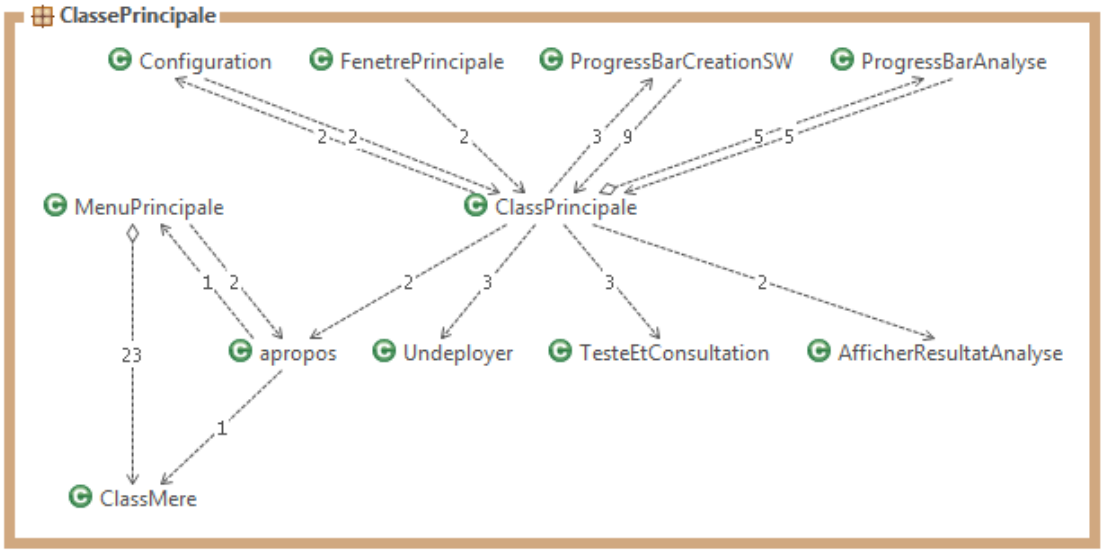


Figure 5.3 : *Couplage entre les classes*

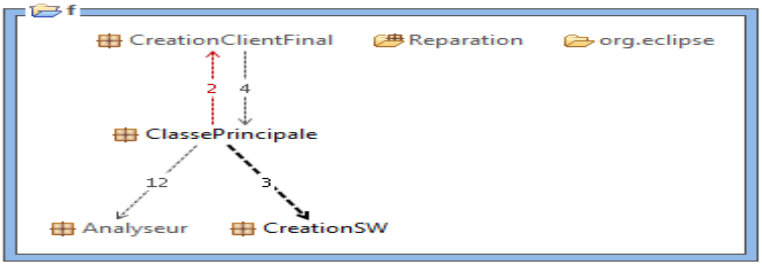


Figure 5.4 : *Couplage entre les package*

5.3. Package principale :

Ce package contient l'interface de notre projet et appels à tous les d'autre package.

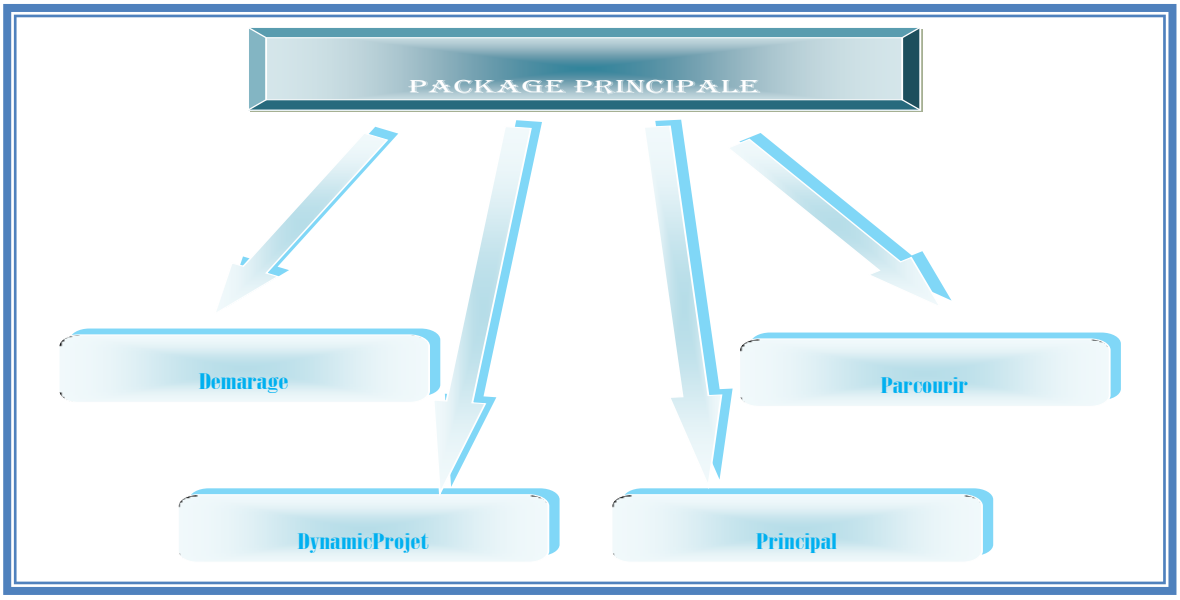


***Figure 5.5** : Package CreationDynamicProjet*

Ce package contient 4 classes la première classe est :

a. **Classe Demarage :** Cette classe présente l'interface pour l'application par d'autre façon c'est la première vue de l'application.
b. **Classe DynamicProjet :** Juste une interface
c. **Classe Parcourir :** Interface parcourir le projet et le transférer en dynamicproject.
d. **Classe Principal :** Contient tous les menus.

5.4. Package CreationDynamicProjet :

Ce package fait le transfert de projet générale ver un projet dynamic

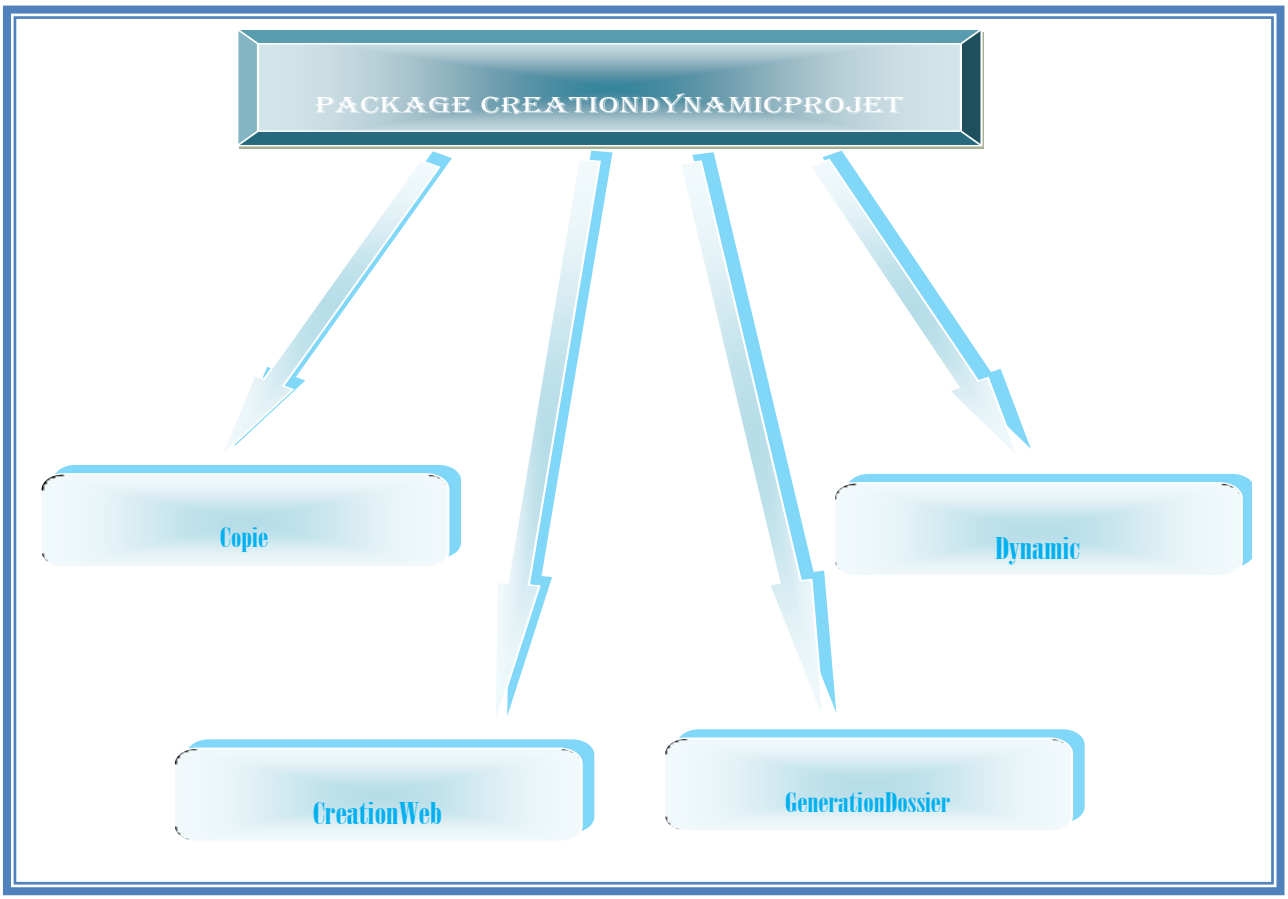


Figure5.6: *Package CreationDynamicProjet*

a. **Copie :** Classe qui contient un algorithme pour copie les fichiers.

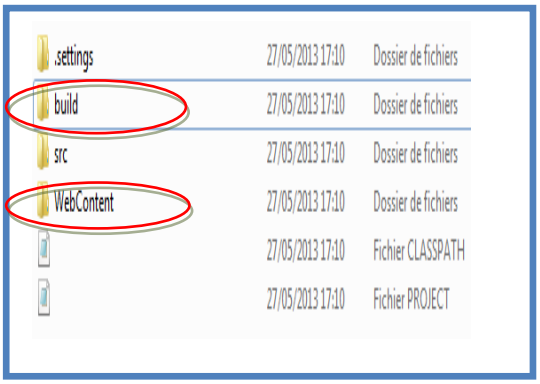
b. **CreationWeb :** Classe pour créer le fichier web.xml

```
  <web-app      xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
instance\"        xmlns=\"http://java.sun.com/xml/ns/javaee\"
xmlns:web=\"http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd\"
xsi:schemaLocation=\http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd\
id=\"WebApp_ID\" version=\"3.0\">
```

C'est un descripteur de déploiement pour définir la méthode de mappage des URL sur les servlets, déterminer les URL qui nécessitent une authentification ainsi que d'autres informations. il se trouve dans le fichier d'archives WAR de l'application, sous WEB-INF/. web.xml fait partie de la norme Servlet pour les applications Web.

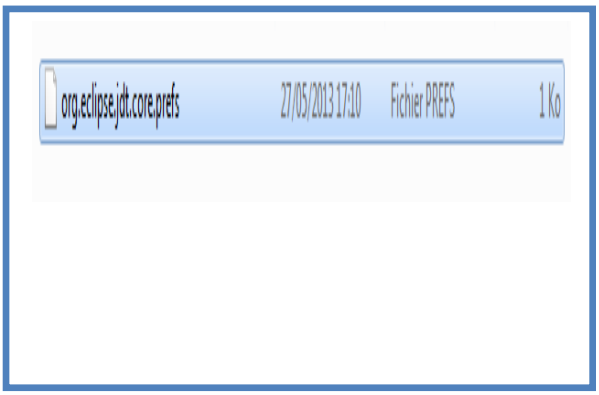
c. **GenerationDossier :** Classe pour générer les dossiers nécessaires
d. **Dynamic :** La classe pour faire exécuter toute les classes du package dynamicproject chacun a son endroit.

Projet générale

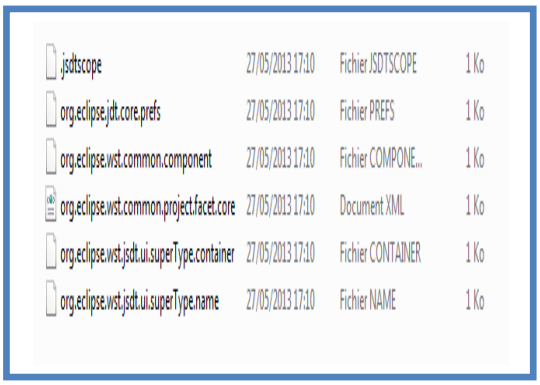


Projet dynamic

Voilà un exemple comment le fichier .settings a changer.



Strings dans un projet générale



Strings dans un projet dynamic

5.5. Package WebService :

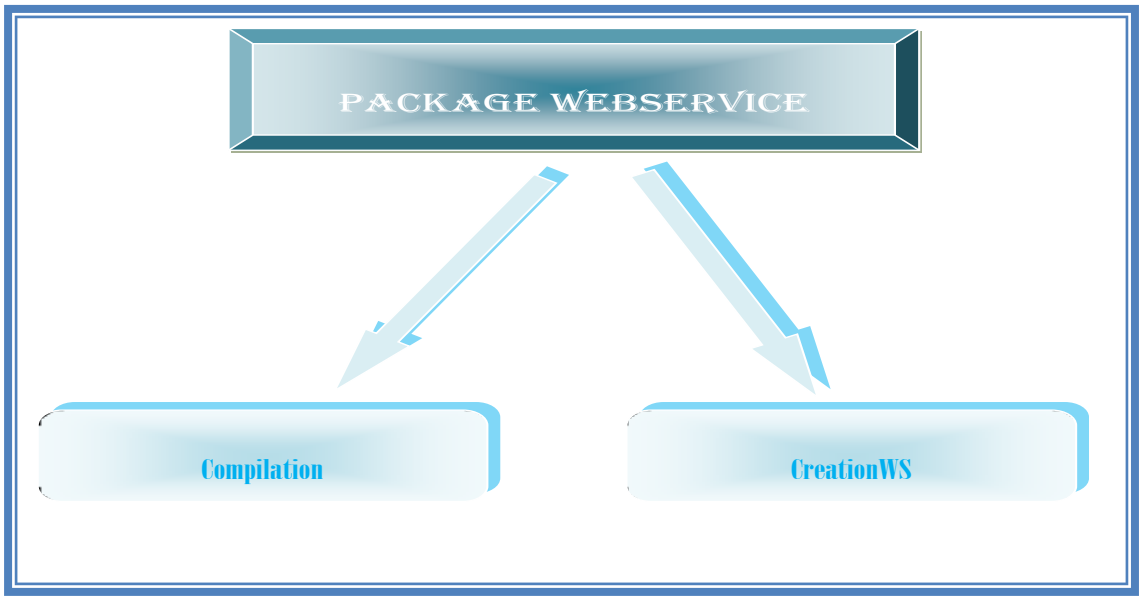


***Figure 5.7** : Package WebService*

a. **Compilation :** Classe pour compiler les classes du projet

b. **CreationWS :** Classe pour modifier les classes pour le faire un web service

```
public class HelloWorld {

    public String sayHello (String  name =
"name")  {
        return "Hello " + name + "!";
    }
}
```

Classe java

```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService()
public class HelloWorld {


    @WebMethod
    public String sayHello(@WebParam(name =
"name") String name) {
        return "Hello " + name + "!";
    }
}
```

service web

6. La présentation de l'application

On commence par la fenêtre principale

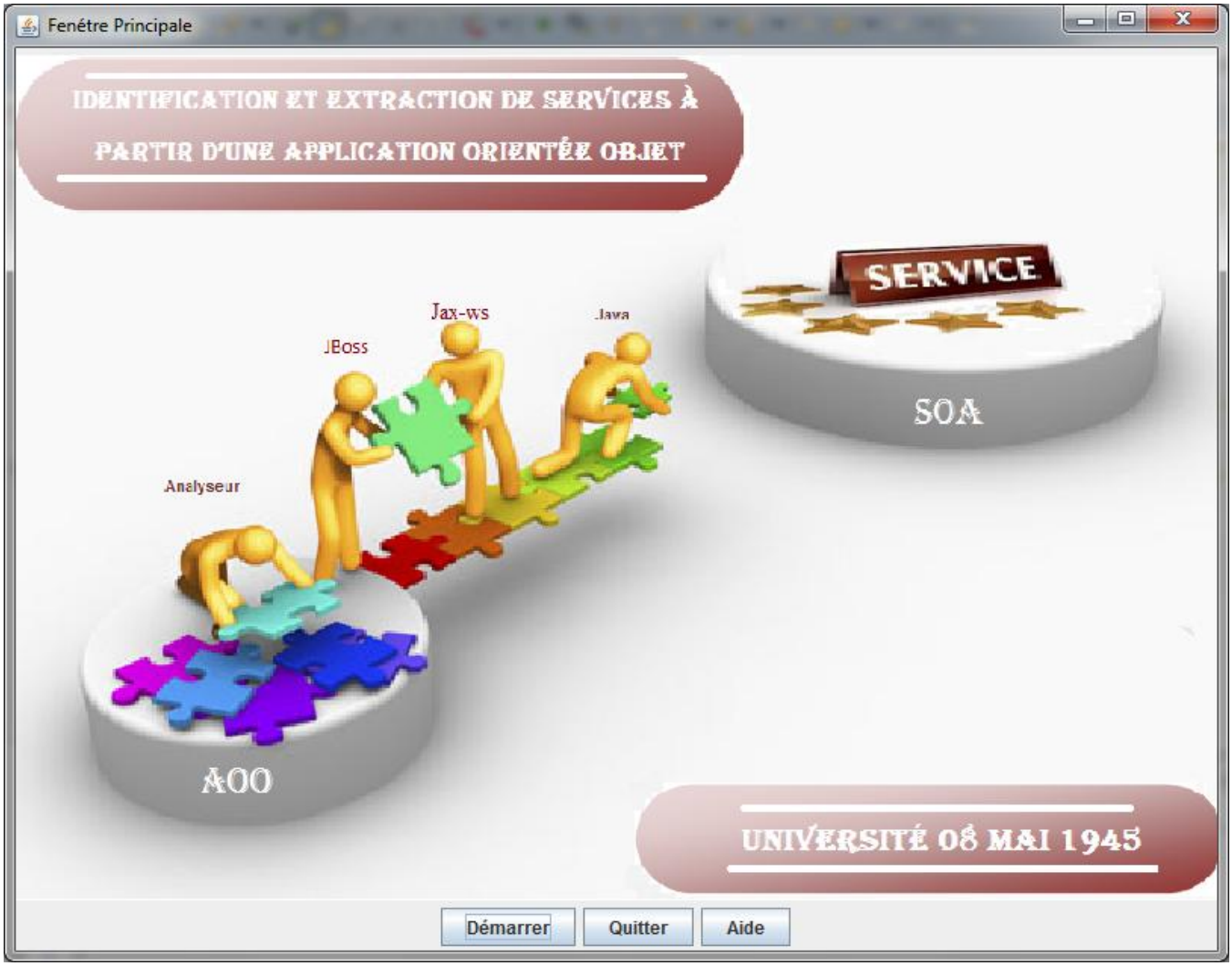


***Figure 5.8** : fenêtre principale*

Cette fenêtre contient trois boutons :

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

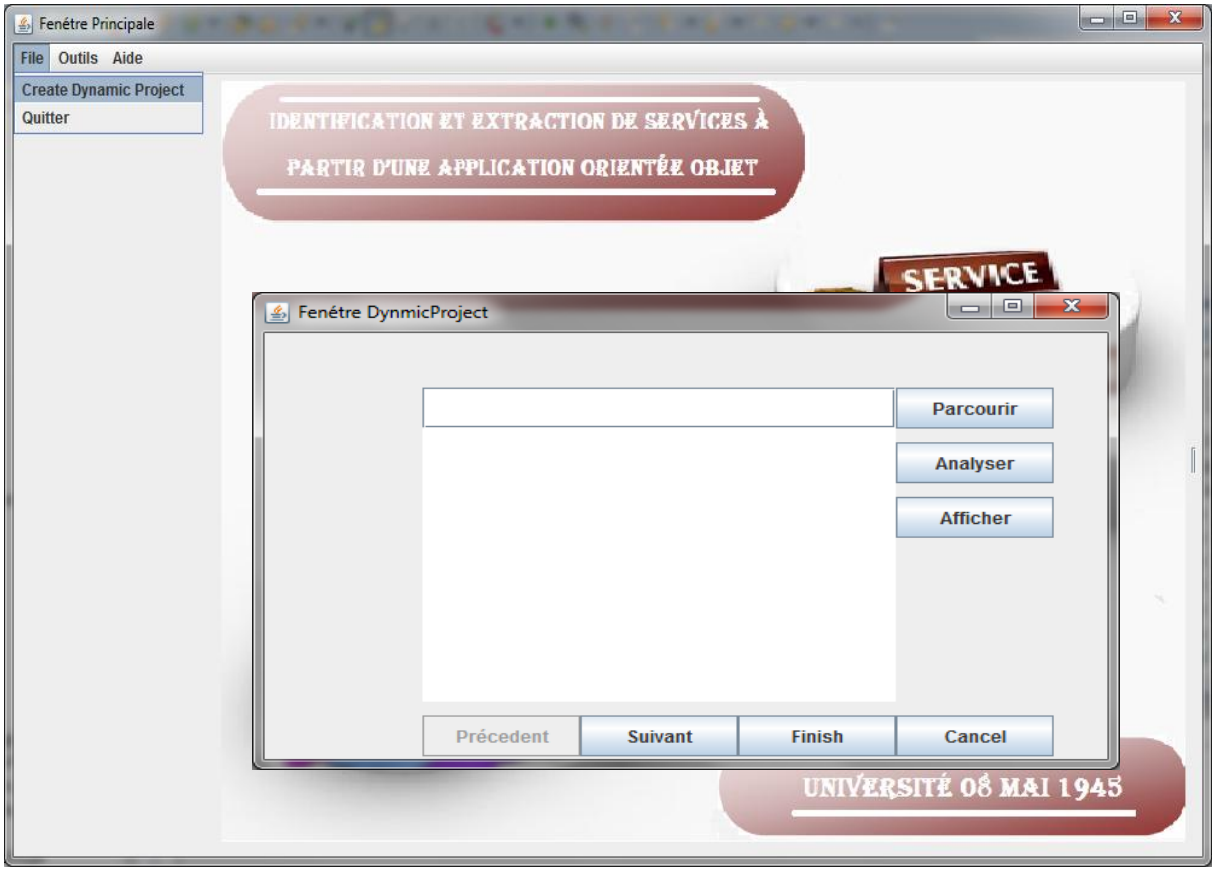


***Figure 5.9**:Parcourir et transférer ver projet dynamic*

Cette fenêtre contient 08 boutons qui sont :

1. parcourir : permet d'ouvrir une boite de dialogue qui vous permet de sélectionner un projet.
2 .Suivant : permet de sélectionner le chemin de projet.

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

4. Précédent : Retour à la page précédente

5. Afficher : affiche les classes qui composent le projet.

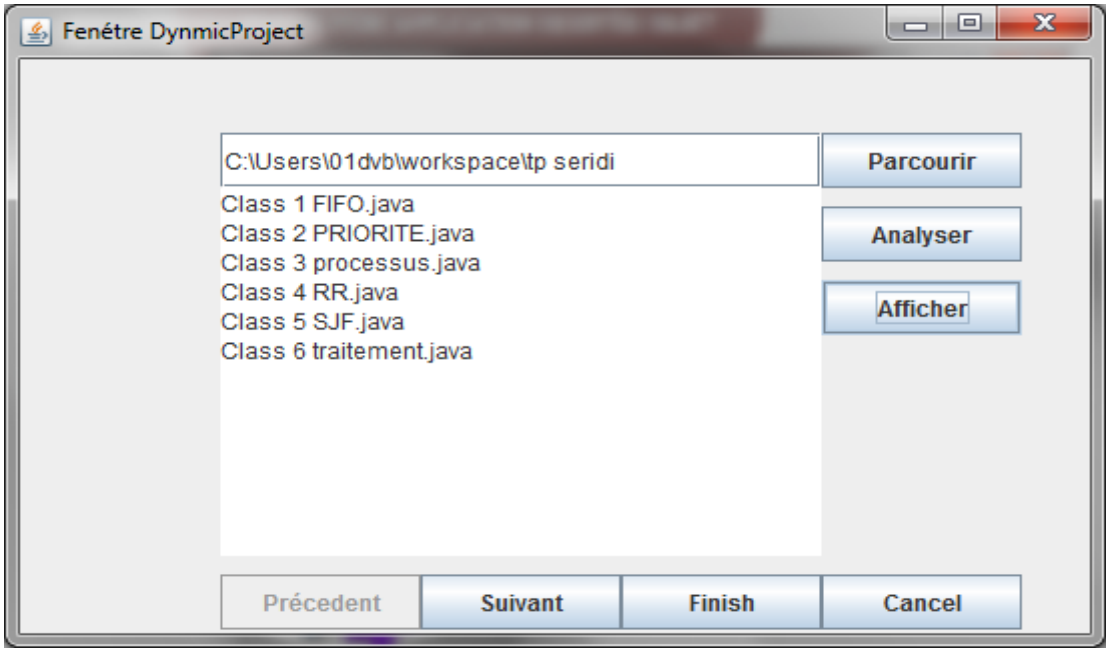


Figure 5.10 *: Afficher*

6. Analyser : ce bouton démarrer le stan.

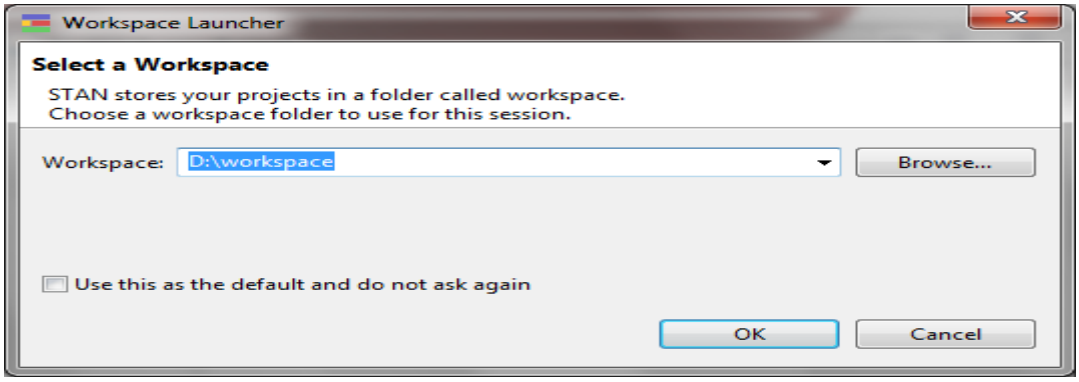


Figure 5.11 *:* Démarrer le stan

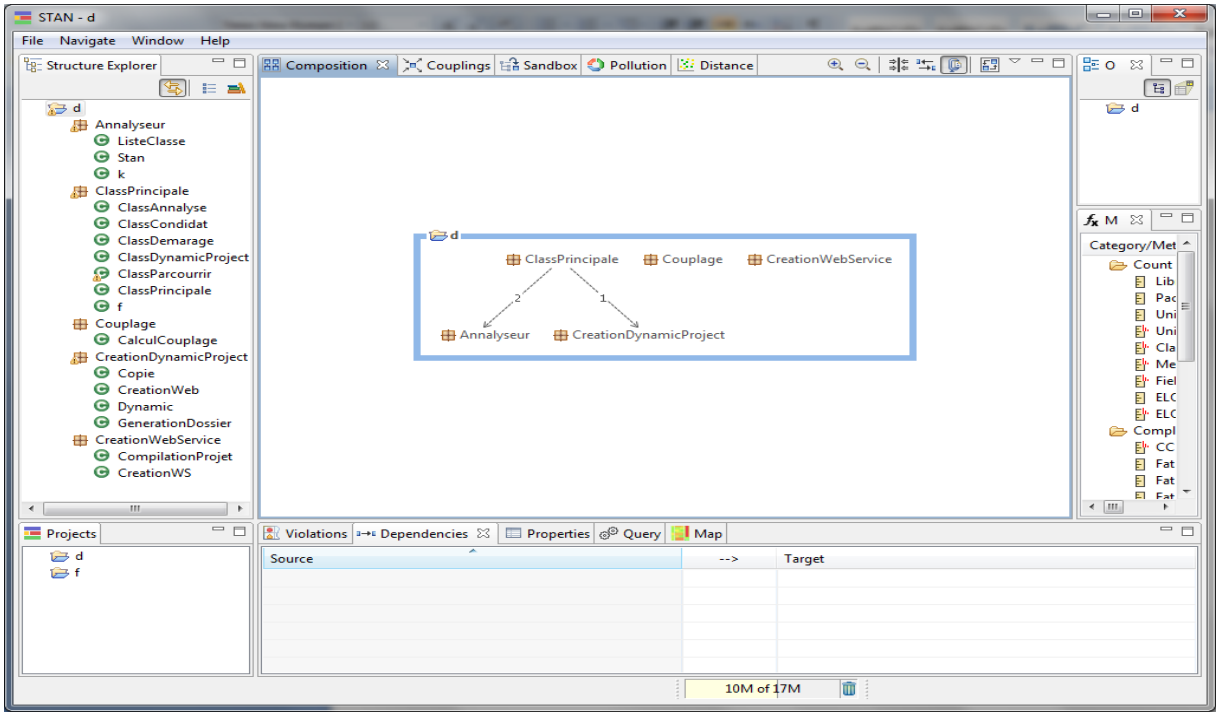


***Figure 5.12**:* Le couplage entre les package

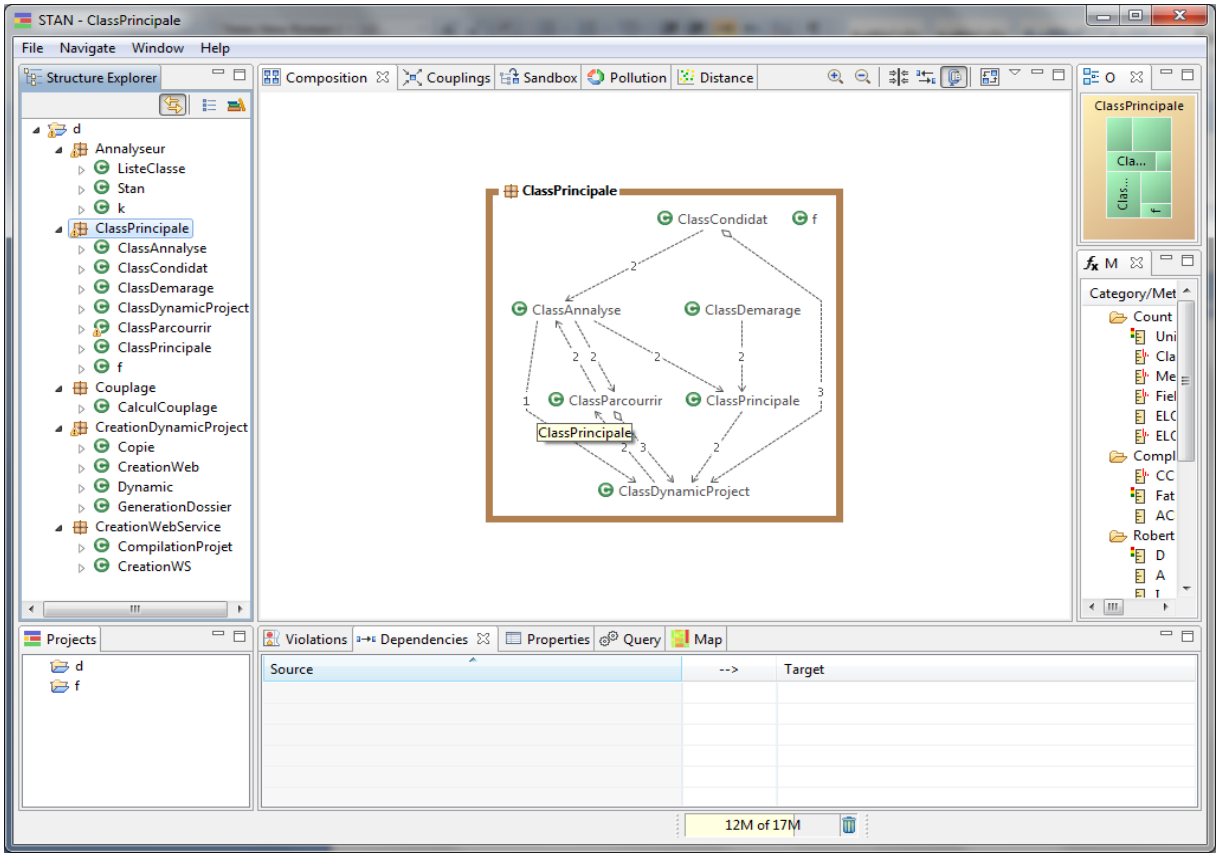


***Figure 5.13** :* Le couplage entre les classes

7. Conclusion :

Dans ce chapitre nous avons présenté les outils utilisé pour développer notre outil de migration, à savoir l'éclipse, JavaEE ,jboss, Jax_WS. Les détails d'implémentation ont été aussi présentés.

Des imprime écrans ont aussi été insérés afin de montrer les différentes options de notre outil.

Conclusions générales et perspectives

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, nous avons réalisé un outil qui permet d'analyser le projet afin d'extraire les informations pertinentes pour la migration. Une étape suivante consiste à générer le code du service web jax-ws et cela par la modification du code source initial puis générer les fichiers WSDL pour chaque service web en utilisant les fichiers compilés«.class » et enfin déployer les services web, après les avoir testé.

Le résultat final du processus de migration est un ensemble de services qui peuvent être réutilisés pour construire de nouvelle application.

Comme perspectives, nous pouvons proposer d'améliorer la fonction de groupage des classes de façon à prendre plus de paramètres et de permettre d'automatiser la partie identification des services.

Webographie

[1]http://www-igm.univ-mlv.fr/~dr/XPOSE2004/woollams/definition.html

[3] http://genexo-a-m.googlecode.com/files/SOA%20final.doc

[5]http://www.zdnet.fr/actualites/soa-comprendre-l-approche-orientee-service-39206712.htm

[6]http://glossaire.infowebmaster.fr/xml/

[7] http://msdn.microsoft.com/fr-fr/library/bb469912.aspx#_Toc478383487

[9] http://emmanuel.illy.pagesperso-orange.fr/UDDIPresentation.html

[10] http://msdn.microsoft.com/fr-fr/library/bb469924.aspx#wsdlexplained_topic03

[12]http://blog.xebia.fr/2009/04/29/soa-du-composant-au-service-lautonomie/

[13]http://www.microsoft.com/france/entreprises/plateforme-applicative/scenarios/details/business-processandsoa.aspx

[14]http://www.cetic.be/Architectures-Orientees-Services?lang=fr

[15]http://www.entreprise-business.com/architecture-orientee-service-soa

[16]http://www.enterprise-concept.com/fr/consultance/business-approaches/soa

[17]http://www.commentcamarche.net/contents/web-services/soa-architecture-orientee-services.php3

[18]http://www.journaldunet.com/solutions/expert/11257/migration-vers-les-architectures-soa---pieges-et-realites.shtml.

[20]http://www.journaldunet.com/solutions/0311/031110_soa.shtml
[21]http://blog.xebia.fr/2007/08/16/mise-en-oeuvre-dune-soa-les-cles-du-succes/

[25]http://www.commentcamarche.net/contents/web-services/soa-architecture-orientee-services.php3

[27]http://www.noratek.netwp-contentuploads20081213514-dlv-18228_rpt_urbanisation-soa-100.pdf

Bibliographie

[2]Youssef BELAID, NFE107 Urbanisation & Architecture des Systèmes d'Information, Centre d'enseignements de Grenoble , Année Universitaire: 2008-2009.
[4]Adolphe Francois & Julien Marmel & Dominique Perlat & Olivier Printemps, SOAP Simple Object Access Protocol ,24/05/2013
[8]Lionel Tricon , Initiation au couple gagnant WSDL/SOAP, *Article publié dans GNU/Linux Magazine France,numéro 76 (Octobre 2005).*

[11]Rim Bejaoui& Ivan Maffezzini, SOA : LES IMPACTS SUR LE CYCLE DE VIE DU PROJET DE DÉVELOPPEMENT LOGICIEL, Université du Québec à Montréal Canada, (2006).

[19]Gilbert Raymond ;SOA : Architecture Logique :Principes, structures et bonnes pratiques ;Version 2.1 avril 2011.

[22] Séridi Ali, Seriai djamel-Abdelhak , Bourbia Riad : Approches Pour L'Evolution Des Systèmes Patrimoniaux Vers UneArchitecture Orientée Service, WOTIC 2011 : The Fourth Workshop on Information Technologies and Communication Casablanca, Morocco Oct 13-15 2011

[23]G. Lewis, E. Morris, D. Smith, L.Wrage, L. O'Brien, Service-Oriented Migration and Reuse Technique (SMART), Proceedings of 13th IEEE International Workshop on Software Technology and Engineering Practice (WSTEP'05), September 2005

[24]Oracle, Oracle IT Modernization Series: The Types of Modernization, an Oracle White Paper;September2008.

[26] Service-Oriented Architecture Governance for the Services Driven Enterprise (Marks, 2006)

[28] Fabrice Rossi ,Services Web, Universit´e Paris-IX Dauphine ;(2013)

[30]A.Benzeltout & W.Yahyaui: Migration d'une application orientée objet vers une architecture orietée service ,mémoire de fin d'études Master, Juin 2012.

[31]M. Brodie and M. Stonebraker: Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach, Morgan Kaufmann San , Francisco,1995.

[32]Amjad Umar and Adalberto Zordan. Reengineering for service oriented architectures: A strategic decision model for integration versus migration. J. Syst. Softw. 82, 3 (March 2009)

[33]Bisbal, J.; Lawless, D.; Bing Wu; Grimson, J.; , "Legacy information systems: issues and directions," Software, IEEE , vol.16, no.5, pp.103-111, Sep/Oct 1999.

[34]Erradi, A.; Anand, S.; Kulkarni, N.; , "Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture," Services Computing, 2006. SCC '06. IEEE International Conference on , vol., no., pp.257-260, 18-22 Sept. 2006.

[35]Marchetto, A., Ricca, F.: From objects to services: toward a stepwise migration approach for java applications. STTT 11(6) (2009) 427-440

[36]He Yuan Huang, Hua Fang Tan, Jun Zhu, and Wei Zhao. A Lightweight Approach to Partially Reuse Existing Component-Based System in Service-Oriented Environment. In Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems (ICSR '08), Hong Mei (Ed.). Springer-Verlag, Berlin, Heidelberg

[37]A. Almonaies, J.R. Cordy and T.R. Dean, "Legacy System Evolution Towards Service-Oriented Architecture", Proc. SOAME 2010, Interrnational Workshop on SOA Migration and Evolution, Madrid, Spain, March 2010, pp. 53-62.

[38]E.J. Chikofsky and II Cross, J.H., "Reverse engineering and design recovery: a taxonomy," Software,IEEE, vol. 7, no. 1, pp. 13–17, Jan 1990

[39]W.P.Stevens, G.J.Myers and L.L.Constantine. Structured Design. In IBM Systems Journal, Vol. 13, No. 2, pages 115-139, May 1974.

[40] J.M. Bieman and B.K. Kang. Cohesion and reuse in an object-oriented system. Proceedings of the Symposium on Software Reusability (SSR'95), Seattle, WA, pp. 259-262, April 1995.

[41]R. S. Pressman, Software Engineering, A practitioner's approach, Fifth edition, McGraw Hill, 2001

[43]Harry M. Sneed : Integrating legacy Software into a Service oriented Architecture, AneCon GmbH, Vienna, Austria IEEE 2006.

[44]Riadh BEN HALIMA: Conception, implantation et expérimentation d'une architecture en bus pour l'autoréparation des applications distribuées à base de services web, Thèse de doctorat,L'université de toulouse France, 14 Mai 2009.