REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTRE DE L'ENSEIGNEMENT SUPERIEURE
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE SAAD DAHLAB DE BLIDA
FACULTE DES SCIENCES
DEPARTEMENT D'INFORMATIQUE

Rapport de fin d'étude

En vue de l'obtention d'un diplôme de Master2 en informatique (LMD)

THEME

Approche Architecture Logicielle pour la spécification des Systèmes de Systèmes

Réalisé par :

ARABI SOUFIANE

AYACHI ILYES

Promotrice :

Mme : CHERFA IMENE

Présidente jury : ARKAM Meriem

REMERCIEMENT

Avant toute chose, nous tenons à remercie le Dieu le tout puissant de nous avoir donné la volonté, la patience et le courage de poursuivre et d'achever ce travail dans les bonnes conditions.

Nos remerciements également a notre promotrice ***Mme CHERFA****, pour nous avoir encadré, diriger et conseiller tout au long de ces mois de travail ainsi pour son aide précieuse lors de la correction de ce mémoire ;*

Nous espérons que nos professeurs de l'USDB trouveront ici, l'expression de notre gratitude, qu'ils soient vivement remerciés des efforts considérables qu'ils ont fournis pour notre formation ;

Nos remerciements s'adressent aussi à Mehdi et Idir et Hassan pour leurs encouragements et leur gentillesse.

Nous n'oublierons pas le soutien constant et les encouragements de notre grande famille.

En fin, on tient à remercier les membres de jury qui vont faire l'honneur d'apprécier notre travail.

Merci

Dédicace

Je dédie ce modeste travail à mes très chers parents qui m'ont toujours soutenu pendant toute ma vie, je leur souhaite le bonheur et la bonne santé.

A toute ma grande famille ; mes Frères:

Abdallah, Abderraouf

A toute ma famille oncles tantes cousins cousines ;

A mon collègue de binôme Ilyes et sa famille;

Et a tout mes amis, et en particulier à :

Aboubakr, Hamed, Haroun.

Soufiane

Dédicace

Je dédie ce modeste travail à mes très chers parents qui m'ont toujours soutenu pendant toute ma vie.

A mes Sœurs et toute ma famille ;

A mon collègue de binôme Soufiane, et sa grande famille;

A tous mes collègues et mes amis.

A toute personne porte des sentiments pour moi ;

Merci à tous ceux qui m'ont aidé de prés ou de loin.

Ilyes

Sommaire

Chapitre III: Business Processes Execution Language BPEL

PARTIE II : Conception et mise en œuvre

Chapitre IV: Conception

Chapitre V: Implémentation

Liste des figures

Liste des tableaux

Résumé

La complexité des logiciels et la criticité des systèmes, dépendant du logiciel, ont augmenté à un rythme effarant. En particulier, les systèmes à logiciel prépondérant ont évolué rapidement pour passer de systèmes autonomes, dans le passé, à des systèmes faisant partie d'un réseau dans le présent, pour devenir des systèmes eux-mêmes composés de systèmes dans le futur à venir.

Les systèmes de systèmes offrent des capacités globales sur des systèmes indépendants (appelés sous-systèmes du SoS) les uns des autres, sujets à des domaines d'utilisation divers et variés, et pouvant évoluer pour répondre à de nouvelles formes de besoins d'utilisation.

Notre travail consiste à effectuer la spécification et la réalisation d'un SoS. Après documentation sur les différentes approches, notre choix s'est porté sur une approche orientée service, et le langage d'orchestration BPEL.

Mots Clés : Complexité, Systèmes de Systèmes SoS, Approches orientées services.

Abstract

Software complexity and criticality of systems dependent on software, increased at an alarming rate. In particular, software-intensive systems have evolved rapidly from autonomous systems in the past, systems that are part of a network in the present, to become systems themselves composed of systems in the coming future. Systems systems offer comprehensive capabilities independent systems (called subsystems SoS) each other, subject to many and varied areas of use, and can evolve to meet new forms of usage needs.

Our job is to study the various components oriented and service-oriented approaches to see if they are suitable for system specification system and choose the model best suited for the SoS among the models studied in different approaches.

Keywords: Complexity, Autonomous systems, Systems of Systems SoS, Independent systems, Components oriented Approache, Services-oriented approaches.

Introduction Générale

Introduction générale :

1. Contexte générale :

Récemment, il ya eu un intérêt croissant pour une classe de systèmes complexes dont les constituants sont eux-mêmes complexes. L'optimisation de la performance, la robustesse et la fiabilité parmi un groupe émergent de systèmes hétérogènes afin de réaliser un objectif commun est devenu l'objet de diverses applications, y compris les systèmes militaire, les systèmes de sécurité, de l'aéronautique, de l'espace, de la fabrication, de l'industrie des services, et la gestion des catastrophes etc. Il ya un intérêt croissant dans la réalisation des synergies entre ces systèmes indépendants pour atteindre la performance globale du système désiré.

La technologie SoS est censée mettre en œuvre et analyser des systèmes grands, complexes, indépendants et hétérogènes qui travaillent en coopération. Le concept SoS présente un point de vue de haut niveau et explique les interactions entre chacun des systèmes indépendants.

2. Problématique :

L'environnement d'un système de systèmes, constitué de tous ces sous-systèmes, est donc hétérogène et dynamique. Techniquement parlant, les sous-systèmes doivent opérer ensemble, ou inter opérer, dans la réalisation de la mission du SoS.

Du point de vue managérial, l'architecte d'un SoS n'a autorité que sur la façon dont les différents systèmes constituants travaillent ensemble. L'architecte n'a pas de contrôle direct sur les systèmes impliqués dans le SoS. Les systèmes individuels remplissent des objectifs pour d'autres fins, et continuent à fonctionner, pour ces fins, s'ils sont déconnectés du SoS.

La problématique auquel doit répondre ce travail est : Comment modéliser les sous systèmes, leurs interactions, et le système global, tout en sachant qu'un SoS est lui-même un système et peut, éventuellement, être un constituant d'un SoS plus complexe.

3. Objectifs :

L'objectif de ce travail est d'effectuer la spécification d'un système de système. Ce qui signifie qu'il faut trouver un moyen pour effectuer la coordination entre les différents sous-systèmes, pour que le système de systèmes atteint sa mission, sans se préoccuper du fonctionnement interne d'un sous-système.

Pour atteindre un tel objectif, nous avons suivre ces étapes :

1. Etudier les différentes approches orientées composants et orientées services pour voir s'ils sont adaptés à la spécification d'un système de système.
2. Choisir le modèle le mieux adapté pour les SoS, et l'implémenter.

4. Organisation du mémoire :

Le présent mémoire est structuré comme suit :

- La première partie est composée de trois chapitres :
  - Nous présentons dans le premier chapitre les systèmes de systèmes et leurs caractéristiques ainsi que leur architecture.
  - Dans le deuxième chapitre nous présentons les approches à composants et les approches à service.
  - Nous présentons dans le troisième chapitre le processus BPEL et ses composants.

- La deuxième partie composée de deux chapitres :
  - Dans le quatrième chapitre nous présentons la conception de notre système de systèmes.
  - Dans le cinquième chapitre nous présentons l'implémentation de notre système de systèmes.
- Notre mémoire se termine par une conclusion et quelques perspectives

Partie I

ETAT DE L'ART

Sommaire :

1. Les Systèmes de systèmes.
2. Les approches à composants et les approches à service.
3. Business Processes Execution Language (BPEL).

CHAPITRE I

Les systèmes de systèmes

I. Introduction:

Le développement des applications demande une compréhension totale de l'application et de l'architecture. La complexité des applications est toujours plus importante. Dans ce premier chapitre nous allons essayer de donner une introduction globale aux systèmes de systèmes, leur définition, leur architecture et leurs caractéristiques.

Ce chapitre va donc être le premier maillon d'une longue chaine qui va suivre.

II. Les systèmes de systèmes :

1. Définitions de système de systèmes :

Définition 1 :

L'intégration du système de systèmes est une méthode pour poursuivre le développement, l'intégration, l'interopérabilité et l'optimisation des systèmes pour améliorer la performance dans les futurs scénarios de champ bataille. **[1]**

Définition 2 :

Il existe des systèmes de systèmes quand il ya une présence de la majorité des cinq caractéristiques suivantes : l'indépendance opérationnelle et de gestion, la répartition géographique, le comportement émergent, et le développement évolutif. **[2]**

Définition 3 :

Les systèmes de systèmes sont des systèmes concourants et distribués à grande échelle qui sont composés des systèmes complexes. **[3]**

A partir de ces définitions et d'autre dans la littérature nous choisissons cette définition :

Un système de systèmes est une collection de tâches orientés ou de systèmes dédiés qui mettent en commun leurs ressources et leurs capacités afin de créer un nouveau système, plus complexe qui offre plus de fonctionnalités et de performances que la somme des systèmes constitutifs.**[4]**

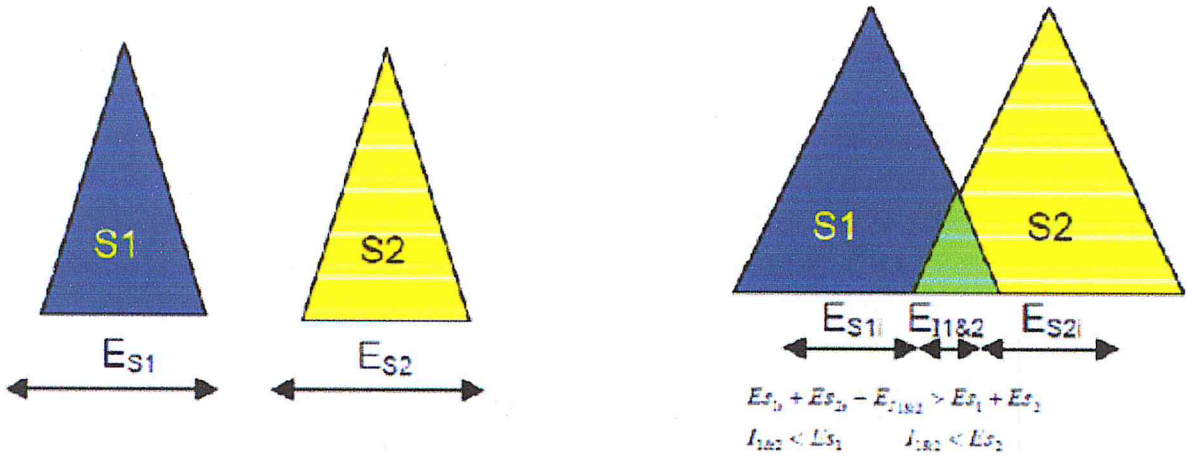

Figure I.1: hiérarchie. **[5]**

2. Les caractéristiques de SoS :

Il n'y a pas encore une définition unique généralement acceptée pour SoS. Cependant, un grand nombre de chercheurs ont tenté de documenter les caractéristiques de SoS comme **[6]** et **[7]**. Ces caractéristiques permettent de faire la distinction entre les systèmes de systèmes et les systèmes monolithiques complexes. Les caractéristiques sont les suivantes **[8]**:

- **L'indépendance opérationnelle des systèmes constitutifs :**

Si le système de systèmes est démonté dans ses systèmes constitutifs, les systèmes constitutifs doivent être capables de fonctionner indépendamment. Cela signifie que chacun de ces systèmes est indépendant et dispose de son propre usage (Exemple de la force navale militaire et ses composants, tels que des véhicules ou des avions).

- **L'indépendance de gestion des systèmes constitutifs :**

Cela signifie qu'ils sont acquis séparément, puis intégré pour créer le système. Ils ont une existence et une organisation séparée, dans leur acquisition, ainsi que dans leur maintenance. Cela concerne les aspects des équipes de projet et/ou de maintenance.

❖ **Développement évolutif:**

Puisque la configuration du système est évolutive, cela signifie que le système global n'est pas complètement formé. Son développement est évolutif avec des fonctions et des objectifs ajoutés, supprimés et modifiés.

❖ ***Comportement Emergent:***

L'émergence des comportements signifie que le système global propose des propriétés et des fonctionnalités non présentes dans l'un de ses systèmes constitutif, puisque ces comportements émergents ne peuvent pas être situés ou attribués à l'un de ces systèmes constitutif. De plus, ces comportements ne peuvent pas être toujours prévus, ce qui conduit à des difficultés pour la validation du système global. Un exemple de service émergent est le routage IP (protocole Internet), qui est à l'origine de l'Internet. Aucun routeur IP connaît la topologie complète des interconnexions de l'Internet, ou encore la configuration des interconnexions locales dans son propre voisinage. Et pourtant, le routage IP est un processus efficace qu'on peut faire confiance, et qui transfère de manière prévisible les messages sources à la destination prévue. Chaque routeur IP dans le chemin de message, décide quel est le routeur voisin qui constituera le point suivant, même s'il n'a pas connaissance des routeurs ou des chemins potentiels accessibles dans son voisinage immédiat. Le routage IP est donc un service émergent, qui travaille également avec des informations incomplètes, imprécises et obsolètes, tout en offrant une fonctionnalité efficace et prévisible.

❖ **Répartition géographique:**

La répartition géographique de ces systèmes constitutifs signifie qu'ils sont situés dans des endroits différents; cette extension géographique est relative et dépend largement sur les technologies disponibles et les moyens de communication. Les systèmes peuvent échanger des informations, mais ne peuvent pas échanger des quantités importantes d'énergie ou de matière.

3. Architecture des SoS:

Avec la définition du problème, l'une des tâches les plus importantes de l'ingénieur de systèmes consiste à partitionner le problème en petits problèmes plus faciles à gérer et prendre des décisions critiques sur la solution. L'une des décisions les plus importantes est l'architecture.

Bien qu'il soit impossible de comprendre toutes les caractéristiques et les conséquences de l'architecture à la fois que le système est conçu, il est possible de réaliser une architecture de système qui maximise la capacité du système pour répondre aux besoins des utilisateurs.

L'architecture d'un système complexe qui se compose d'un certain nombre de systèmes collaboratifs indépendants n'est pas différente de l'architecture d'un système simple. Les deux commencent par la définition d'un problème et la conception de la solution. Le processus de conception est le même, Mais par rapport à la conception de systèmes simples, la complexité de la conception d'un système de systèmes (SoS) est ardue. **[9]**

La Figure I.2 montre le modèle de base SoS. Les éléments de système d'un SoS sont eux-mêmes des systèmes. Ils répondent à leurs propres besoins et résolvent leurs propres problèmes. Ils ont leurs propres propriétés émergentes. En fait, ils ont leur propre but d'existence. Mais ils font aussi partie d'un système plus grand « SoS » qui porte lui-même un besoin et a des propriétés émergentes, résultant de l'interaction de systèmes au sein de SoS. La nécessité de maintenir l'autonomie tout en fonctionnant en même temps dans le contexte SoS augmente considérablement la complexité d'un SoS et est au cœur du défi de l'architecture SOS.

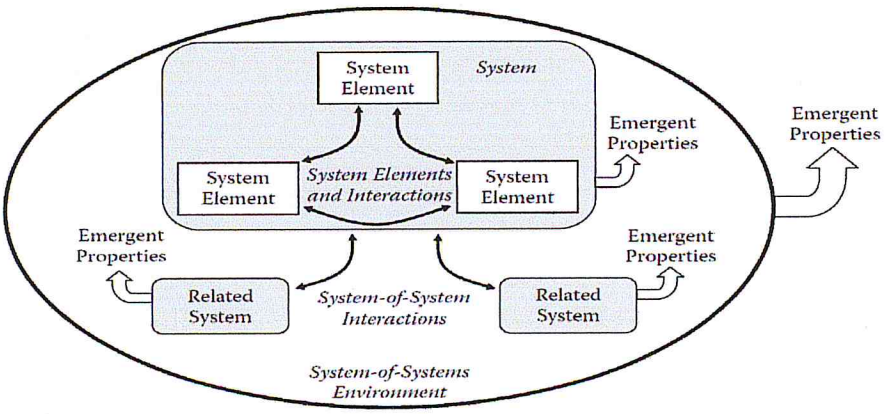

Figure I.2: Le modèle de base SoS. **[9]**

3.1 Définition de l'architecture d'un SoS :

« Une architecture est la structure des composants, leurs relations, et les principes et les lignes directrices régissant l'évolution de leur conception au fil du temps. L'architecture d'un SoS est un framework technique persistant pour gérer l'évolution d'un SoS au fil du temps ».[10]

3.2 Les objectifs de l'architecture d'un SoS :

Les principaux objectifs de l'architecture d'un SoS sont les suivants :

- Aborder le concept des opérations pour le SoS.
- Englober les fonctions, les relations et les dépendances des systèmes constitutifs.
- Décrire la fonctionnalité de bout en bout, le flux de données et les communications.
- Fournir le framework technique pour évaluer les changements nécessaires dans les systèmes ou d'autres options pour traiter les exigences.
- Fournir une vision intégrée de l'ensemble des systèmes dans le SoS.

Une architecture SoS ne porte pas sur les systèmes constitutifs, mais sur la façon dont ils travaillent ensemble. Elle ne traite pas les détails des systèmes individuels ; plutôt, elle définit la façon dont les systèmes fonctionnent ensemble pour répondre aux besoins de l'utilisateur et traite l'implémentation des systèmes individuels uniquement lorsque la fonctionnalité est essentielle pour les questions intersectorielles du SoS. **[10]**

3.3 Les types d'architecture d'un SoS :

Les types d'architecture d'un SoS sont les suivants :

- **L'architecture physique :** Ceci définit les composants physiques (systèmes constitutifs) dont le SoS sera composé.
- **L'architecture fonctionnelle :** Ceci décrit la fonctionnalité que les systèmes individuels contribuent au SoS, fournissant une image fonctionnelle du système et détaillant l'ensemble des fonctions à

remplir au sein de SoS ainsi que les relations entre les fonctions.

Une caractéristique d'une bonne architecture est sa capacité de fournir un framework qui s'adapte aux changements des incréments de développement des SoS, permettant des changements dans certains domaines, tout en assurant la stabilité dans d'autres.

Le développement d'une architecture n'est pas une activité optionnelle lors du développement d'un SoS, puisque la conception d'un SoS se compose de l'architecture de SoS ainsi que des changements à la conception des systèmes constitutifs qui leur permettent de travailler ensemble en respectant l'architecture. **[10]**

3.4 Les défis de l'architecture d'un SoS:

Il ya des propriétés généralement associés à SoS qui présentent des défis à l'architecte de logiciels comprennent:

- **Les comportements émergents :** Les SoS présentent une fonctionnalité qui n'est présente dans aucun des systèmes constitutifs individuels. Les constituants interagissent pour créer de nouvelles fonctionnalités potentielles.
- **Des cycles de vie très longs et la présence des composants hérités :** Les cycles de vie des systèmes constitutifs sont susceptibles d'être gérés en utilisant différents processus, et sont très peu susceptibles d'être synchronisé.
- **Les composants sont gérés indépendamment :** Les systèmes constitutifs évoluent en dehors de SoS, ce qui force par fois le SoS à s'adapter.
- **L'absence d'une autorité de prise de décision centrale.**
- **Les limites du SoS sont floues :** les Limites d'un système à un seul niveau sont généralement faciles à définir. Par sa nature, le SoS tolère l'inclusion des systèmes constitutifs indépendants tiers. Les systèmes indépendants qui constitueront l'environnement opérationnel donné pour un seul système peuvent être considérés comme un environnement SoS, ou comme des systèmes constitutifs **[10]**

III. Conclusion :

Dans ce chapitre nous avons parlé sur les systèmes de systèmes, leurs caractéristiques et leur architecture.

Dans le chapitre suivant nous concentrerons sur Les approches à composants et les approches à service.

CHAPITRE II

Les approches à composants et les approches à service

I. Introduction:

Les approches classiques de programmation rendent relativement difficiles des propriétés comme l'évolutivité et la réutilisation dans les logiciels.
Diverses technologies sont apparues pour permettre une meilleure construction des applications. La programmation orienté-objet a donné certaines réponses à ce problème
Mais elles n'étaient pas suffisantes.
Dans ce premier chapitre nous allons parler sur les différentes approches qui peuvent représentés une solution de notre problématique.

II. L'approche à composants :

L'approche à composants est la continuité de l'approche orientée objet.
Les composants logiciels permettant plus de réutilisabilité et d'encapsulation de morceaux logiciels. La plupart des modèles composants qui existent, sont d'une façon ou d'une autre l'extension d'un modèle objets ou basé sur les méthodes orientées objets.

Avant que le terme composant ne soit populaire des termes comme procédure, fonction, module et objet ont été utilisés dés les premiers jours en développement logiciel pour identifier les fonctionnalités du logiciel. En générale les notions évoquées par ces termes ont été introduites et utilisées avec objectifs la réutilisabilité, la gestion de la complexité du logiciel et le développement de système plus flexibles. **[11]**

1. Définition d'un composant

Un composant est un module logiciel autonome qui permet la construction d'une application par composition et il est réutilisable dans plusieurs applications et permet d'exporter des attributs, méthodes, et propriétés.
Un composant peut être configuré statiquement et / ou dynamiquement, peut être installé sur différentes plates-formes et capable de s'auto-décrire. **[12]**

➢ **Description d'un composant :**

La description d'un composant comprend :

- Ses interfaces d'invocation
- Les interfaces qu'il utilise

• Une interface de configuration

• L'ensemble des contraintes de déploiement

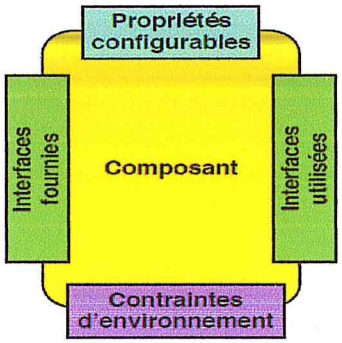

Figure II.1: La description d'un composant. **[12]**

2. Les modèles de composants :

a) Enterprise Java Beans (EJB) :

Le modèle de composants Enterprise Java Beans (EJB) a été introduit par Sun Microsystems en 1998 comme une extension de modèle de composants Java beans (coté client) **[13]** en tant que partie de la Java 2 Enterprise Edition (J2EE) plate-forme.

EJB est un modèle de composants côté serveur pour le développement d'applications dans le langage de programmation Java. **[14]**

L'architecture EJB :

L'architecture EJB identifie les éléments suivants:

- **Beans** : un bean est un composant logiciel et appelé un bean entreprise.
- **Conteneurs**: un conteneur est une encapsulation d'un composant (et ses composantes).

L'espace d'exécution des conteneurs et des composants appelé une "Structure d'accueil».

- **Clients.**
- **Serveurs.**

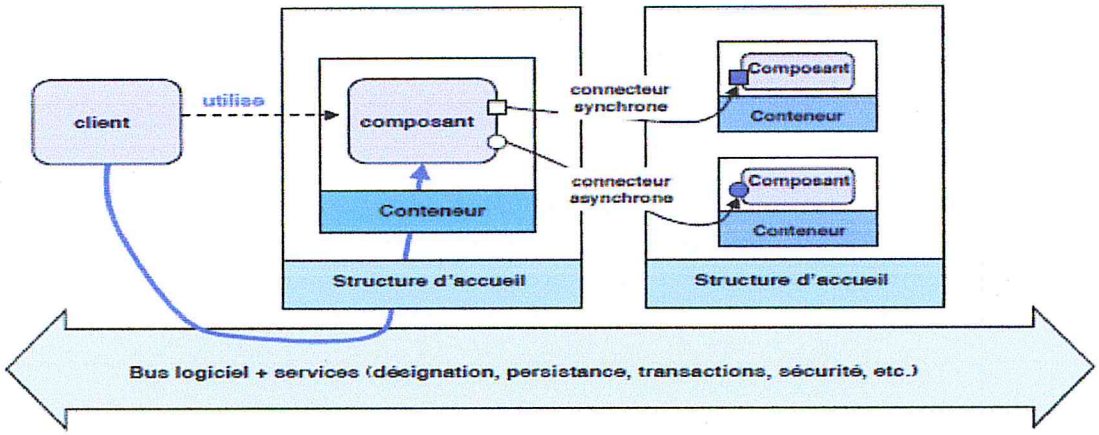

Figure II.2: Conteneurs et Structures d'accueil. **[15]**

Un bean entreprise réside à l'intérieur d'un conteneur. Le conteneur est constitué d'un environnement de déploiement pour l'entreprise bean.

Un serveur EJB fournit un environnement d'exécution pour un ou plusieurs conteneurs. Le serveur gère les ressources du système de bas niveau et les inscrire dans des conteneurs en fonction des besoins.

L'application client interagit avec le bean en utilisant deux interfaces qui sont générés par le conteneur, l'interface d'accueil et de l'interface de l'objet.

Lorsque le client appelle une opération utilisant ces interfaces, le conteneur intercepte chacun d'appels et inserts les services de gestion. **[14]**

Le conteneur fournit un certain nombre de services pour chaque bean, comme la gestion du cycle de vie, la gestion d'état, la sécurité, la gestion des transactions et la gestion de la persistance. Ces services appellent des méthodes fournies par le bean (méthodes collback). **[15]**

L'interface "EJB home" fournit un accès aux services de cycle de vie. Les clients peuvent utiliser cette interface pour créer, détruire ou trouver une instance de bean existant. L'interface "EJB object" permet d'accéder aux méthodes d'application de l'entreprise bean comme le montre la **Figure II.3**. Cette interface représente le point de vue du client sur bean et il expose toutes les interfaces liées à l'application pour l'application cliente, sauf ceux qui sont utilisés par le conteneur EJB pour contrôler et gérer cet objet.

Pour accéder à un bean entreprise, il faut définir une interface distante en utilisant le Java Remote Method Invocation (RMI). **[14]**

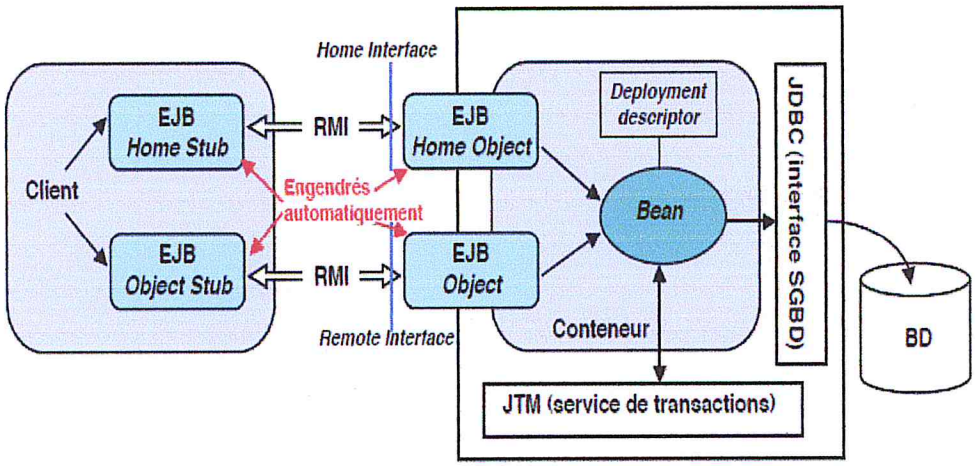

Figure II.3 : Schéma d'exécution des EJB. **[15]**

b) Le modèle de composants CORBA (CCM) :

Le CCM est un *framework* de composants de type serveur, mais qui, semble-t-il, peut facilement être utilisé du côté client. Le CCM se décompose en deux niveaux : un niveau basique, qui permet essentiellement de « componentiser » des objets CORBA, et un niveau étendu, beaucoup plus riche et intéressant. Tout comme les EJB de Sun **[16]**, auxquels correspond la version basique du CCM en Java, la technologie CCM repose sur l'utilisation de conteneurs pour héberger les instances de composants et faciliter leur déploiement. La spécification du CCM **[17]**, qui représente plus de 1000 pages, est découpée en quatre modèles et un méta modèle :

- **Le modèle abstrait :** offre aux concepteurs le moyen d'exprimer les interfaces (fournies et utilisées) et les propriétés d'un type de composant. Pour cela, le langage IDL *(Implementation Definition Language)* a été étendu pour prendre en compte les nouveaux concepts introduits dans le CCM, essentiellement la prise en compte des interfaces multiples : les *ports*.

- **Le modèle de programmation :** spécifie le langage CIDL *(Component Implementation Definition Language)* pour définir la structure de l'implantation d'un type de composant, ainsi que certains de ses aspects non-fonctionnels (persistance, transactions, sécurité). L'utilisation de ce langage est associée à un *framework*, le CIF *(Component Implementation Framework)*, qui définit comment les parties fonctionnelles (programmées) et non-fonctionnelles (décrites en IDL/ CIDL et générées) doivent coopérer. Il inclut aussi la manière dont l'implantation d'un composant interagit avec le conteneur.

- **Le modèle de déploiement :** définit un processus qui permet d'installer une application sur différents sites d'exécution de manière simple et automatique. Ce modèle s'appuie sur l'utilisation de paquetages de composants, ainsi que de descripteurs OSD (Open Software Description, un vocabulaire XML), les paquetages étant déployables et composables.

- **Le modèle d'exécution :** définit l'environnement d'exécution des instances de composants. Le rôle principal des conteneurs est de masquer et prendre en charge les aspects non fonctionnels des composants qu'il n'est alors plus nécessaire de programmer. **[18]**

Architecture générale :

Elle se compose de :

- **ORB (Object Request Broker)** est le noyau de transport des requêtes aux objets. Il intègre au minimum les protocoles GIOP (General Inter-ORB Protocol) et IIOP (Internet Inter-ORB Protocol). L'interface au bus fournit les primitives de base comme l'initialisation de l'ORB.
- **SII (Static Invocation Interface)** est l'interface d'invocations statiques permettant de soumettre des requêtes contrôlées à la compilation des programmes. Cette interface est générée à partir de définitions OMG-IDL (*Le langage de description d'interface*).

OMG-IDL est le langage qui permet de rendre compatible l'utilisation de plusieurs écrites dans différents langages de programmation.

- **DII (Dynamic Invocation Interface)** est l'interface d'invocations dynamiques permettant de construire dynamiquement des requêtes vers n'importe quel objet CORBA sans générer/utiliser une interface SII.
- **IFR (Interface Repository)** est le référentiel des interfaces contenant une représentation des interfaces OMG-IDL accessible par les applications durant l'exécution.
- **SSI (Skeleton Static Interface)** est l'interface de squelettes statiques qui permet à l'implantation des objets de recevoir les requêtes leur étant destinées. Cette interface est générée comme l'interface SII.
- **DSI (Dynamic Skeleton Interface)** est l'interface de squelettes dynamiques qui permet d'intercepter dynamiquement toute requête sans générer une interface SSI. C'est le pendant de DII pour un serveur.
- **OA (Object Adapter)** est l'adaptateur d'objets qui s'occupe de créer les objets CORBA, de maintenir les associations entre objets CORBA et implantations et de réaliser l'activation automatique si nécessaire.
- **ImplR (Implementation Repository)** est le référentiel des implantations qui contient l'information nécessaire à l'activation. Ce référentiel est spécifique à chaque produit CORBA. **[19]**

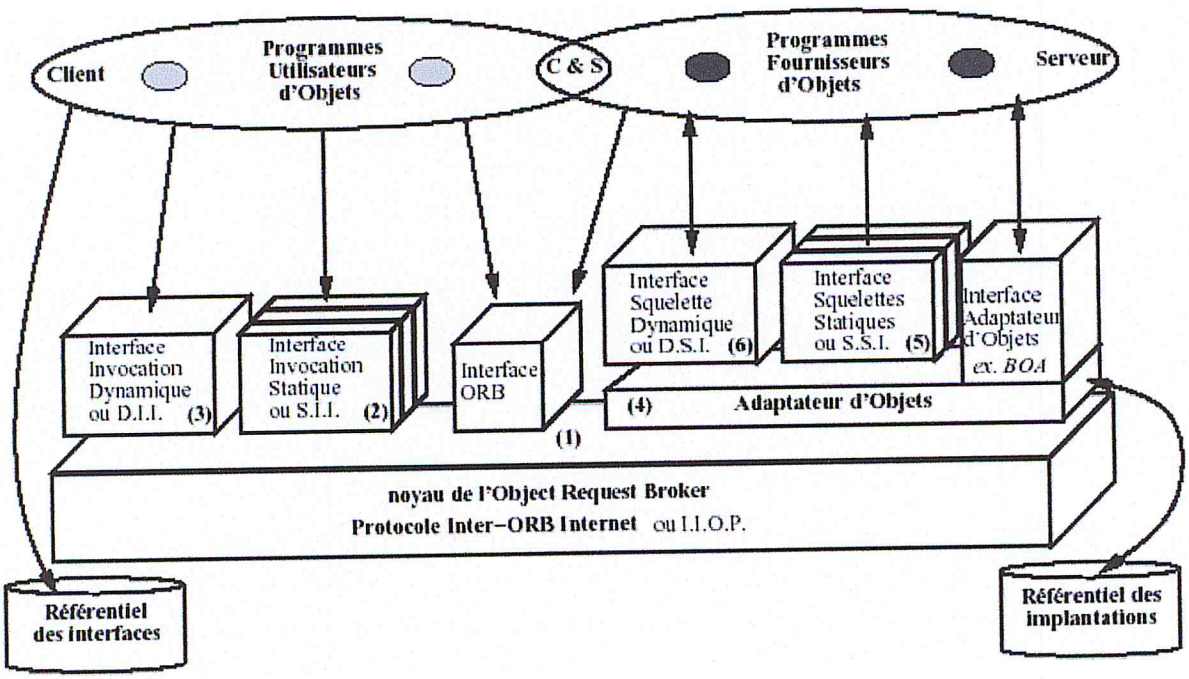

Figure II.4 : L'architecture Corba. **[19]**

c) COM, DCOM (de Microsoft) :

1. COM (Component Object Modèle) :

Le Component Object Model est un standard publié est conçu par Microsoft **[20]**, Il implémente des « connexions » entre les différents composants logiciels : c'est un bus logiciel.

Il est indépendant du langage de programmation du système d'exploitation et de la plate-forme. **[21]**

COM est une extension du modèle OLE (Object Linking and Embedding). Ce dernier est un outil pour la manipulation de « documents composés » comme des composants.

Il a résulté de l'évolution progressive de l'environnement MS-Windows vers les composants.

COM a été crée pour permettre la communication entre les applications Windows dans un environnement à base de composants. **[11]**

Caractéristiques des objets COM :

Pour utiliser COM dans un programme, il faut implémenter des objets COM qui ont les caractéristiques suivantes :

- **L'encapsulation :**

Un objet COM est un paramétrage de données et/ou de fonctions enveloppées dans une entité identifiable unique.

Tout ce qui se trouve dans un objet COM est caché à l'extérieur de l'objet à l'exception des interfaces.

- **Les interfaces :**

- IID (Interface Identifier) : IID est un type de GUID (Globally Unique Identifier) entier de 16 octets qui permet d'identifié.
  - Le nombre de fonctions dans la table.
  - Leur ordre.
  - Leurs signatures.

- Les trois premières méthodes d'une interface sont prédéfinies, en signature comme en signification.
  - QueryInterface() : Négociation d'interface pour la découverte dynamique.
  - Addref() et Release() : Comptage de référence pour la gestion de la durée de vie de l'objet.

La mère de toutes les interfaces « unknown » contient seulement ces trois méthodes.

➢ **Polymorphisme :**

C'est la possibilité pour un objet de répondre correctement aux mêmes demandes en entrée que d'autres objets différents.

➢ **Réutilisation ou héritage :**

Uniquement au niveau de leurs interfaces. **[21]**

2. DCOM (Distributed COM):

Le modèle DCOM est une extension du modèle COM. IL a étendu les communications interprocessus permis par le modèle COM pour que ce soit au travers d'un réseau. Il se base sur le protocole RPC (*Remote Procedure Call*) pour implémenter un système d'appel de procédures vers des objets distants. **[11]**

d) La plate-forme .NET :

Les composants .NET **[22]** sont la dernière évolution que Microsoft a fait sur son modèle de composant. .NET est plutôt un espace qui comprend le CLR (*Common langage Runtime*), des *framework* interfacés et basés sur les classes et empaquetés dans des assemblées et un ensemble d'outils. Le CLR est une implémentation des spécifications de la CLI (*Common Langage Interface*) en rajoutant l'interopération de COM+ et les services d'accès aux plates-formes Windows. **[11]**

Maintenant C++ et Visuel Basic .NET, les assemblées sont les unités de déploiement, de visionnement et de gestion dans .NET autrement dit, ils sont les composants logiciels .NET.

D'un point de vue technique .NET vise trois niveaux :

- Les services WEB.

- La plate-forme de déploiement (serveurs et clients).
- Et la plate-forme de développement.

Les services Web visent la capacité de programmation de l'internet. Cette programmation d'internet est différente du web traditionnel. Pour cela Microsoft a introduit des services comme .NET *passport* et .NET *alerts* et d'autres initiatives. **[11]**

Les plate-formes Microsoft sont plutôt des produits de serveur de Windows .NET server.
Ces serveurs sont transformés pour qu'il soit possible d'intégrer et supporter les services Web et de traiter XML.
La plate-forme de développement comprend les CLR, les *frameworks* et les outils.
Les CLR contribuent à fournir une nouvelle infrastructure de composants qui protège le composant de la plate-forme matérielle. Comme JVM, CLR définit un ensemble d'instructions virtuelles pour s'isoler des processeurs particuliers. A l'inverse de JVM, CLR permet une construction des composants qui peuvent être liés fortement à la plate-forme matérielle.
Avec les dizaines de millions de stations de travail tournant sous le système d'exploitation de Microsoft et sachant que chacune de ces stations est susceptible d'abriter des applications DCOM et .NET, ces derniers peuvent être considérés comme le système de plus répandu dans le monde et sera surement utilisé longtemps encore. **[11]**

III. L'approche à service :

1. Architecture orientée service (SOA) :

Il existe plusieurs façons de percevoir et de définir une Architecture Orientée Services. La plupart de ces définitions se concentrent sur l'aspect technique de la SOA, quoique d'autres présentent des caractéristiques métiers. Voici une liste de définitions proposées et issues de plusieurs sources. Ces définitions sont intéressantes puisqu'elles illustrent plusieurs points de vue sur la SOA.

❖ Nous allons commencer par une définition très courte qui est celle du W3C :
"SOA is a set of components which can be invoked, and whose interface descriptions can be published and discovered **[23]**."

Cette définition du W3C présente avec une manière très simpliste ce qui peut être fait avec un service ou avec sa description. Elle ne s'est pas occupée de la notion d'architecture ni de la manière avec laquelle le service peut être conçu. **[24]**

- Une définition technique a été présentée dans:

"A Service Oriented Architecture (SOA) is a software architecture that is based on the key concepts of service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation." **[25]**

Les auteurs mettent le point sur l'aspect technique de la SOA qui consiste en une application frontale qui utilise un ou plusieurs services. Ces services sont publiés dans des registres de services et la communication entre ces services est assurée par un bus de service. **[24]**

- Une définition de SOA d'un point de vue 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]**

Cette définition prête une grande attention à l'orientation métier dans la SOA de manière que nous pouvons avoir l'impression que la SOA est un concept purement métier et que le niveau technique n'existe que pour le maintien des réseaux et la communication entre les services. **[24]**

En se basant sur ces différentes définitions, nous choisissons la définition suivante:

La SOA est un style d'architecture qui permet la réorganisation du Système d'Information. Elle permet l'encapsulation des fonctionnalités d'un système d'information en un ensemble de services faiblement couplés appartenant à la fois au niveau métier et au niveau technique de l'entreprise.

Les services, munis d'un contrat d'utilisation et d'une interface de description, seront publiés dans des registres de services afin qu'ils puissent être invoqués par d'autres services. **[24]**

Cette définition donne une double vision à l'Architecture Orientée Services une vision métier et une vision technique. Cette séparation de préoccupation a pour but de garantir l'agilité de l'entreprise.

2. Les services web :

2.1 Définition des services Web :

Pour qu'une architecture orientée services SOA soit efficace, il faut une compréhension claire du service à long terme. La technologie des services Web est la technologie de connexion la plus susceptible d'architectures orientées services. **[27]**

Un service Web est un composant logiciel modulaire, complet, fonctionnel et auto-descriptif. Un service Web est disponible à travers le Web grâce à la publication (ou l'annonce). Il peut être facilement localisé, invoqué et consommé à travers le Web. **[28]**

La Figure II.5 introduit les bases des services Web : **SOAP** (*Simple Object Access Protocol*), **WSDL** (*Web Services Description Language*) et **UDDI** (*Universal Description Discovery and Integration*) que nous présentons dans ce qui suit.

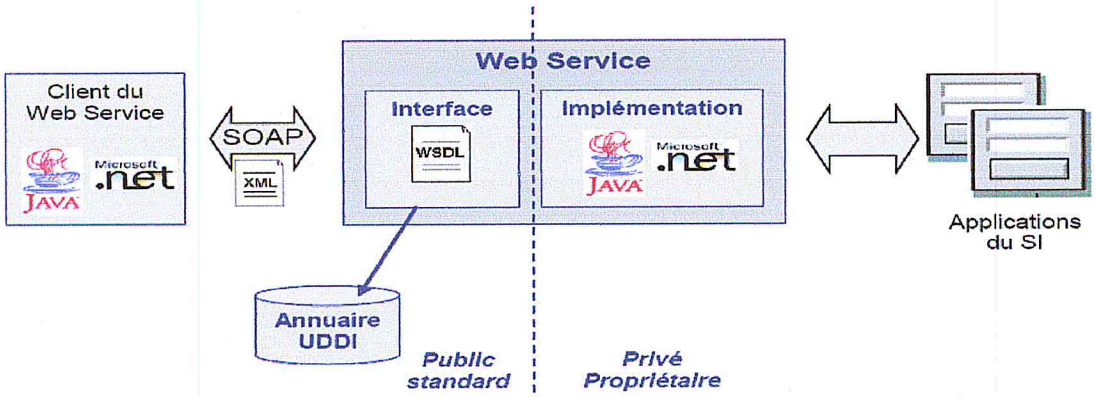

Figure II.5 : Le concept de base des Services Web. **[28]**

À travers ce qui a été présenté, les outils standards des services Web permettent :

- De publier des ressources grâce à UDDI
- De rechercher des ressources grâce à UDDI et WSDL
- D'instancier des ressources grâce à une description WSDL
- À deux applications de communiquer et d'échanger des données grâce à SOAP. **[28]**

2.2 SOAP :

SOAP **[29]** est une recommandation W3C qui le définit comme un protocole léger destiné à l'échange d'informations structurées dans un environnement distribué et décentralisé.

SOAP est basé sur XML afin de mettre en place un mécanisme extensible d'échanges de messages à travers une variété de protocoles. Il a été conçu dans l'optique d'être indépendant du modèle de programmation de l'application ou d'une sémantique particulière.
SOAP vise deux buts : la simplicité et l'extensibilité. SOAP peut être utilisé dans tous les styles de communication : synchrone ou asynchrone, point à point ou multipoints, intranet ou internet. **[28]**

La structure d'un message SOAP se divise en trois parties :
- L'enveloppe SOAP (*envelope*) : marque le début et la fin du message et elle est subdivisée en deux sous parties : la partie en-tête et la partie corps du message.
- L'en-tête (*SOAP Header*) : permet d'ajouter à un message SOAP des attributs spécifiques qui ne sont pas acceptés par toutes les parties et qui peuvent donc être « négociés » au cas par cas.
- Le corps (SOAP *Body*) : permet de décrire les données spécifiques à l'application. Il s'agit essentiellement du nom de la procédure et des valeurs des paramètres d'appel dans le cas d'une demande de service ou des valeurs des paramètres de retour après l'exécution du service. **[30]**

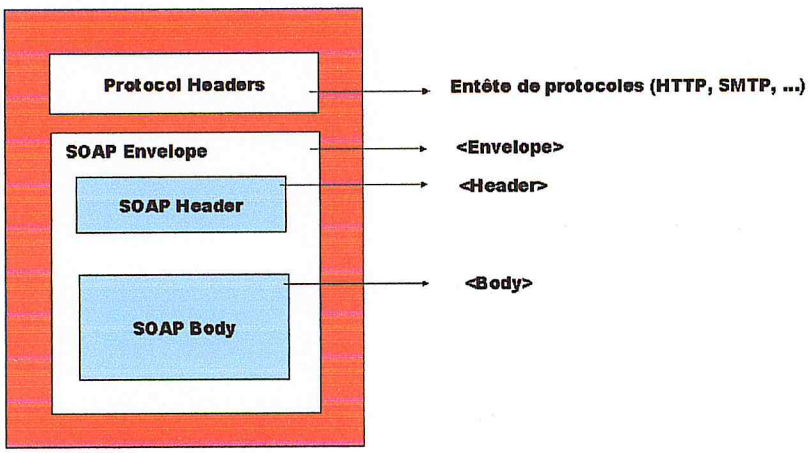

Figure II.6 : Structure d'un message SOAP. **[31]**

Scenario d'utilisation de SOAP :

Le client envoie des messages à l'application de service web correspondant à des requêtes SOAP enveloppés dans des requêtes HTTP. De même, les réponses de l'application de service web sont des réponses HTTP qui renferment des réponses SOAP (voir la **figure II.7**). **[32]**

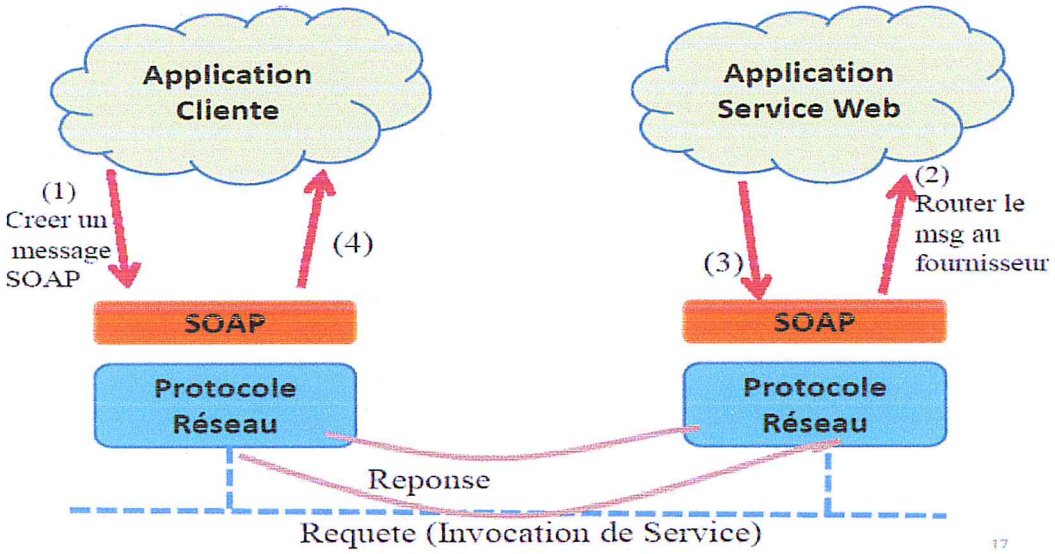

Figure II.7 : Scenario d'utilisation de SOAP. **[33]**

2.3 WSDL :

WSDL **[34]** ou Web Service Description Language, une recommandation W3C, est un format de description des services Web également basé sur XML.

WSDL définit les services réseau sous forme d'un ensemble d'opérations et de messages décrits de manière abstraite et reliés à des protocoles et à des serveurs réseaux concrets.
WSDL est extensible afin de décrire des opérations et les messages correspondant à ces opérations, en fonction du format de ces messages et des protocoles utilisés pour communiquer.

La spécification WSDL 1.0 définit les éléments suivants :

- **Types :** Un conteneur de définitions des types de données. Ce conteneur comprend quelques types système tels que XSD.
- **Message :** Une définition abstraite et typée des données échangées entre services Web.
- **Opération :** Une description abstraite d'une action implémentée par un service Web.
- **Port Type :** Un ensemble abstrait d'opérations réalisées par une ou plusieurs extrémités.
- **Binding :** Un protocole et une spécification de format de données concrets pour un *port-type* donné.
- **Port :** Représente une extrémité (endpoint) définie comme une combinaison de liaison (binding) et une adresse réseau.

• **Service :** une collection de ports. **[28]**

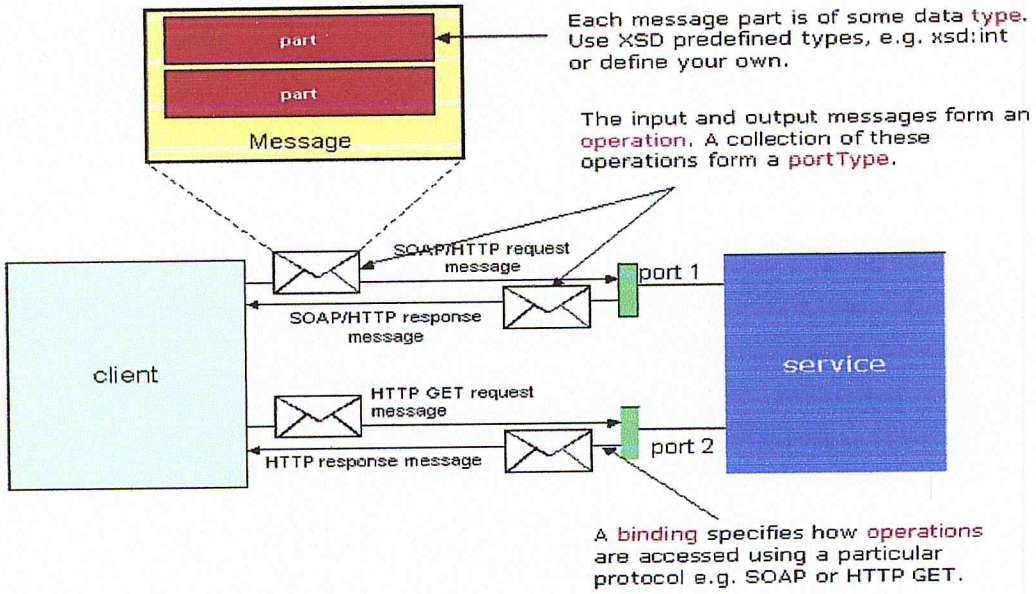

Figure II.8: Différents éléments d'un fichier WSDL. **[35]**

La Figure II.9 montre la description d'un service Web d'adaptation qui prend en entrée l'URL d'une image et retourne sa largeur et sa hauteur.

```
<definitions targetNamespace="http://server.async/" name="GetImageHWImplService">
<types>
<xsd:schema>
        <xsd:import namespace=http://server.async/
        schemaLocation="http://localhost:8084/GetImageHW/GetImageHW?xsd=1"/>
</xsd:schema>
</types>

<message name="getHW">
        <part name="imageURL" element="tns:getHW"/>
</message>

<message name="getHResponse">
        <part name="height" element="tns:getHResponse"/>
</message>

<message name="getWResponse">
        <part name="width" element="tns:getWResponse"/>
</message>

<portType name="GetImageHWImpl">
        <operation name="getHW">
                <input message="tns:getHW"/>
                <output message="tns:getHResponse"/>
                <output message="tns:getWResponse"/>
        </operation>
</portType>

<binding name="GetImageHWImplPortBinding" type="tns:GetImageHWImpl">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <operation name="getHW">
                <soap:operation soapAction=""/>
                <input>
                        <soap:body use="literal"/>
                </input>
                <output>
                        <soap:body use="literal"/>
                </output>
        </operation>
</binding>

<service name="GetImageHWImplService">
        <port name="GetImageHWImplPort" binding="tns:AddNumbersImplPortBinding">
        <soap:address location="http://localhost:8084/GetImageHW/GetImageHWImpl"/>
        </port>
</service>
```

Figure II.9 : Exemple de description WSDL. **[28]**

2.4 UDDI :

UDDI **[36]** est un annuaire d'entreprises accessible par le web, il a pour but de permettre d'automatiser les communications entre prestataires et clients.

Les entreprises publient les descriptions de leurs services web dans l'annuaire UDDI sous forme de fichiers WSDL. Les clients peuvent ainsi rechercher plus facilement les services web dont ils ont besoin en interrogeant le registre UDDI.

Lorsqu'un client trouve dans l'annuaire UDDI une description de service web qui lui convient, il télécharge son fichier WSDL depuis le registre UDDI. Ensuite, à partir des informations inscrites dans le fichier WSDL, notamment la référence vers le service web, le client peut invoquer les fonctionnalités du service web **(Figure II.10)**. **[37]**

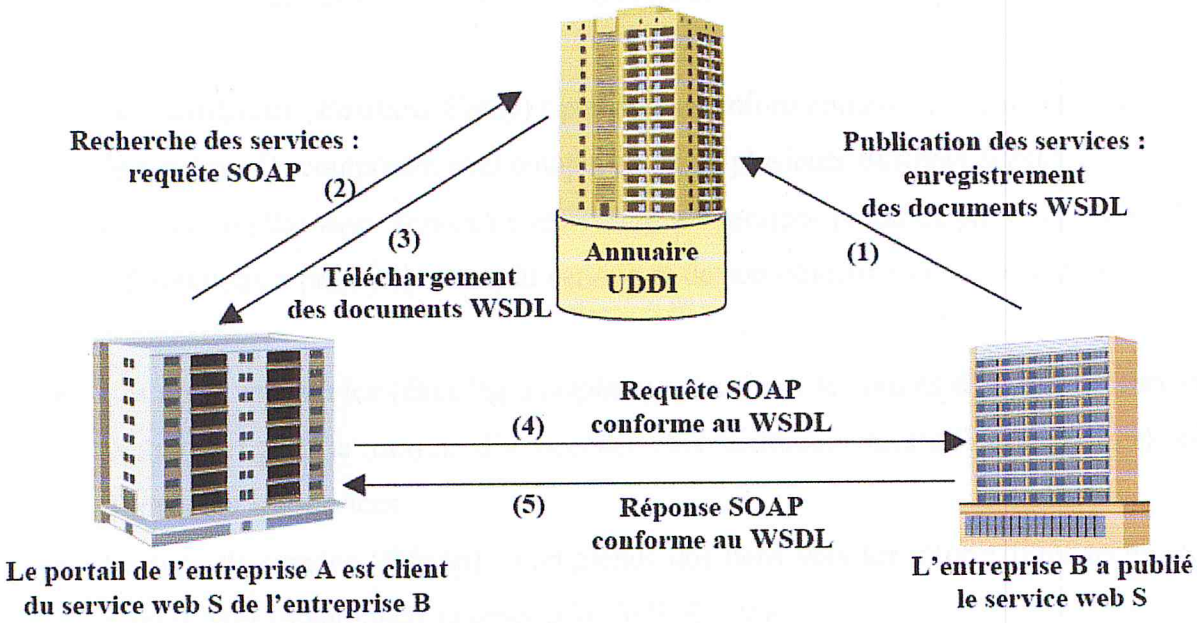

Figure II.10 : Le protocole de découverte des services web via les annuaires UDDI.**[37]**

Les données enregistrées au sein d'un UDDI sont organisées autour de cinq structures de données principales qui sont (voir **Figure II.11**) :

CHAPITRE III

Business Processes Execution Language (BPEL)

toutes les informations jugées pertinentes pour identifier l'organisation (telles que son nom, son adresse physique). Le futur client du service retrouve dans les pages blanches les informations que le fournisseur a renseignées dans l'élément *Business Entity* lors de la publication.

- **Les pages jaunes (*Yellow Paper*)** : Les pages jaunes d'UDDI détaillent la description de l'organisation faite dans les pages blanches en répertoriant les services proposés. Dans cette section, sont décrits : la catégorie de l'entreprise, le secteur d'activité dans lequel exerce l'entreprise, les services offerts par cette organisation, le type de services et les conventions d'utilisation (prix, qualité de service, etc).
- **Les pages vertes (*Green Paper*)** : Les pages vertes comportent les informations techniques liées aux services Web et basées sur leur description WSDL. **[38]**

3. Types de composition de services Web :

La plupart des travaux portant sur la composition de services Web reconnaissent deux types de composition : l'*orchestration* et la *chorégraphie* de services. Cependant, selon les travaux, les définitions des types de composition diffèrent. Pour **[39]** et **[40]**, l'orchestration et la chorégraphie sont des moyens de concevoir la composition, tandis que dans **[41]**, l'orchestration et la chorégraphie sont des points de vue de la composition de services. Dans notre travail, nous retenons le fait que l'orchestration et la chorégraphie sont des moyens de concevoir la composition et les désignons comme des types de composition.

Dans cette section, nous allons présenter l'orchestration et la chorégraphie de manière plus détaillée. Nous allons commencer par la présentation d'une définition de l'orchestration et de la chorégraphie, en expliquant la différence entre ces deux termes. Nous allons ensuite détailler deux langages/standards d'orchestration et deux langages/standards de chorégraphie de services web.

3.1 Chorégraphie :

3.1.1 Définitions :

La chorégraphie permet de décrire la composition comme un moyen d'atteindre un but commun en utilisant un ensemble de services Web. La collaboration entre chaque service Web de la collection (faisant partie de la composition) est décrite par des flots de contrôle.**[41]**

Pour concevoir une chorégraphie, les interactions entre les différents services doivent être décrites. La description de chaque service Web intervenant dans la chorégraphie inclut la description de sa participation dans le processus. De ce fait, ces services peuvent collaborer à l'aide de messages échangés afin de savoir si tel ou tel service peut aider dans l'exécution de la requête.**[39]**

La chorégraphie est aussi appelée composition dynamique. En effet, l'exécution n'est pas régie de manière statique comme dans une composition de type orchestration. Dans une chorégraphie, à chaque pas de l'exécution (*i.e.* à chaque étape de la composition), un service Web choisit le service Web qui lui succède et implémente ainsi une partie de la chorégraphie. La composition de type chorégraphie n'est pas connue, ni décrite à l'avance. **[38]**

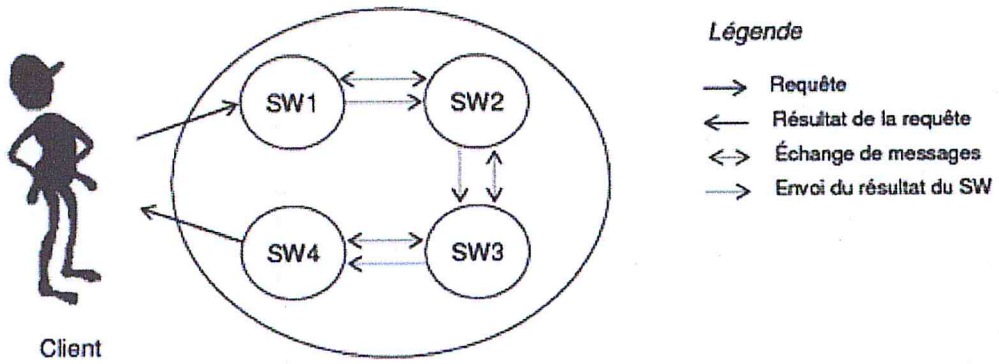

Figure II.12 : Vue générale de l'exécution d'une composition de services web de type chorégraphie. **[38]**

La **Figure II.12** permet d'illustrer une vue générale d'une composition de services Web de type chorégraphie. Le client (logiciel ou humain) établit une requête qui est satisfaite par l'exécution automatique de quatre services Web (SW1, SW2, SW3 et SW4). La requête de l'utilisateur est transmise au premier service Web (SW1) qui est exécuté. Le SW1 découvre ensuite le service Web lui succédant. Une fois le service découvert, les deux services (SW1 et SW2) échangent des messages afin de vérifier si leur communication est viable dans le cadre de la requête. Si les échanges de messages sont concluants, le résultat de l'action du SW1 est transmis au SW2 qui l'utilise comme paramètre d'entrée. Le processus d'implémentation de la composition est identique pour chaque étape (SW2 et SW3). Le SW4 termine le processus et le résultat de son action est transmis au client. **[38]**

3.1.2 Les langages de chorégraphie:

Dès la naissance du terme chorégraphie, de nombreux langages ont été développés pour assurer la coordination des services web. Parmi lesquels, nous pouvons citer :

3.1.2.1 WSCI - Web Services Choreography Interface:

Le langage WSCI **[42]** est une Initiative proposée par Intalio **[43]**, SAP **[44]**, Sun **[16]**, BEA **[45]**. L'objectif consiste à prendre en compte la collaboration d'application à application. Cette initiative s'est concrétisée par une note au World Wide Web Consortium (W3C), le 8 août 2002.

3.1.2.1.1 Concepts du langage :

WSCI est un langage XML, permettant de décrire la chorégraphie entre les services web. Ainsi, les interfaces statiques des services web sont décrites en WSDL, et la chorégraphie entre eux est décrite en WSCI. Le langage WSCI contient les concepts suivant :

a) L'interface :

Le but de WSCI est de décrire les détails du comportement d'un service web, en termes de dépendances temporelles et logiques entre les messages que ce service web échange avec d'autres services web, dans le contexte d'un scénario. Ce comportement est décrit dans un ou plusieurs processus contenus dans l'interface. Un service web peut avoir plusieurs interfaces lui permettant de jouer différents scénarios.

b) Les activités et leur chorégraphie :

Les activités peuvent être atomiques (activité d'envoie et/ou de réception d'un message, ou activité d'attente d'une durée définie) ou complexes (composées d'autres activités). L'activité complexe définit la chorégraphie des activités dont elle est composée. Plusieurs types de chorégraphie existent dans WSCI : l'exécution séquentielle, l'exécution parallèle, la boucle, et l'exécution conditionnelle.

c) Les processus :

Le processus contient une partie du comportement du service web, il représente l'unité de réutilisation dans WSCI. Deux types de processus sont définis dans WSCI: les processus définis au niveau de l'interface, et les processus définis à l'intérieur des activités complexes.

d) Les propriétés :

Les propriétés dans WSCI sont l'équivalent des variables dans les langages de programmation. Les propriétés sont utilisées pour éviter, dans le fichier WSCI, les références explicites vers les messages abstraits décrits en WSDL.

e) La corrélation de messages :

Dans WSCI, une conversation représente un échange de messages entre deux ou plusieurs services web, participant à un scénario. Un service web peut être engagé dans plusieurs conversations en même temps, avec le même service ou avec des services différents.
La corrélation est le mécanisme par lequel un message, reçu par le service, est associé à une conversation particulière.

f) Les exceptions :

WSCI permet de déclarer des exceptions dans la définition de contexte, et de définir un ensemble d'activités que le service web aura à exécuter lorsque cette exception se produit. Les exceptions déclarées peuvent être : la réception d'un message d'exception, la production d'une erreur ou le dépassement de la date limite de fin.
Ainsi, lorsqu'une exception se produit, cela n'arrête pas la chorégraphie en entier, mais seulement le contexte où l'exception s'est produite.

g) Les transactions :

WSCI permet d'associer une transaction dans le contexte. Ainsi, les activités contenues dans cette transaction seront exécutées de la manière "tout ou rien". Les transactions sont soit atomiques, soit emboîtées (composées d'autres transactions). **[37]**

3.1.2.2 WSCL - Web Services Conversation Language :

Le langage WSCL **[46]** est une soumission de Hewlett-Packard au World Wide Web Consortium (W3C) **[47]**, sous forme d'une note, le 14 mars 2002. L'objectif consiste à décrire la séquence d'interactions possibles avec un service web particulier.
Le langage WSCL apporte une approche légèrement différente des précédentes, car il se concentre sur la description de conversations entre paires de services web. Il est donc plus léger que WSCI.
WSCL utilise un procédé pour décrire la conversation et son exécution est décentralisée.

3.1.2.2.1 Concepts du langage :

WSCL est un langage XML, il permet de représenter simplement les interactions entre deux services web.

La spécification WSCL est composée de quatre éléments principaux :

- Les ***schémas*** des documents XML échangés au cours d'une conversation. Ces schémas ne font pas partie du document de spécification WSCL, ce sont des documents séparés que l'on référence par un URL (*Uniform Resource Locator*) dans la spécification de la conversation.
- Les ***interactions*** modélisant les actions de la conversation comme des échanges de documents entre deux participants. Il existe cinq types d'interactions dans WSCL :
  - ✓ "Send" : émission d'un message.
  - ✓ "Receive" : réception d'un message.
  - ✓ "SendReceive" : émission puis réception d'un message.
  - ✓ "ReceiveSend" : réception puis émission d'un message.
  - ✓ "Empty" : ne contient pas de message à échanger, il est utilisé pour modéliser le début et la fin d'une conversation.
- Les ***transitions*** spécifiant l'ordonnancement les relations entre les interactions. Chaque transition spécifie l'interaction source et l'interaction de destination, et éventuellement le type de message de l'interaction source.
- La ***conversation*** donne la liste de toutes les interactions et les transitions composant la conversation, ainsi que quelques informations additionnelles, comme le nom de la conversation, et l'interaction qui démarre la conversation, et celle qui la termine. **[37]**

WSDL se charge de décrire les services web, et WSCL se charge de décrire les conversations entre deux services web. Ainsi, ces deux descriptions sont séparées, afin de permettre la réutilisation (une même conversation WSCL peut avoir lieu entre différentes paires de services web décrites en WSDL, et un service web décrit en WSDL peut participer à plusieurs conversations WSCL). Cette approche différencie WSCL des autres approches (tels que XLANG **[48]** et WSFL **[49]**). **[37]**

3.2 Orchestration :

3.2.1 Définitions :

L'orchestration est un ensemble de processus exécutés dans un ordre prédéfini afin de répondre à un but. Ce type de composition permet de centraliser l'invocation des services Web composants. Chaque service est décrit en termes d'actions internes. Les contrats entre deux services sont constitués selon le processus à exécuter. **[41]**

L'orchestration est un ensemble d'actions à réaliser par l'intermédiaire de services Web. Un moteur d'exécution est un service Web jouant le rôle de chef d'orchestre, gère l'enchaînement des services Web par une logique de contrôle. Pour concevoir une orchestration de services Web, il faut décrire les interactions entre le moteur d'exécution et les services Web. Ces interactions correspondent aux appels, effectués par le moteur, d'action(s) proposée(s) par les *services Web composants*. **[40]**

L'orchestration de services Web consiste en la programmation d'un moteur qui appelle un ensemble de services Web selon un processus prédéfini. Ce moteur définit le processus dans son ensemble et appelle les services Web (tant internes qu'externes à l'organisation) selon l'ordre des tâches d'exécution. **[39]**

L'orchestration peut être vue comme une composition ascendante : les services Web utilisés dans la composition existent au préalable et sont appelés selon un enchaînement prédéfini afin de réaliser un processus précis. La **Figure II.13** illustre l'orchestration. La requête du client (logiciel ou humain) est transmise au moteur d'exécution (*Moteur*). Ce dernier, d'après le processus préalablement défini, appelle les services Web (ici, *SW1*, *SW2*, *SW3* et *SW4*) selon l'ordre d'exécution.

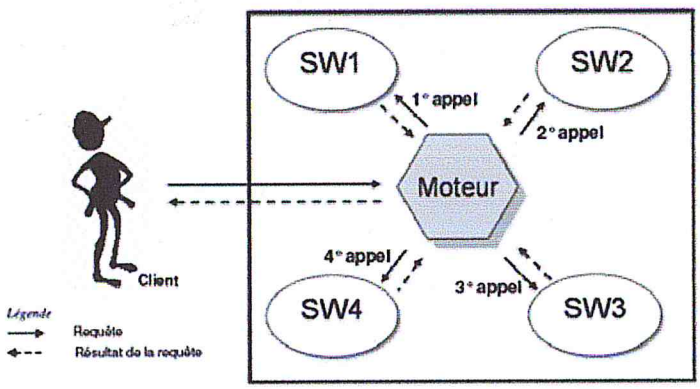

Figure II.13 : Vue générale de l'orchestration. **[38]**

En d'autres termes, l'orchestration de services Web exige de définir l'enchaînement des services Web selon un canevas prédéfini, et de les exécuter selon un script d'orchestration. Ces derniers (le canevas et le script) décrivent les interactions entre services Web en identifiant les messages, et en spécifiant la logique et les séquences d'invocation. Le module exécutant le script d'orchestration de services Web est appelé un moteur d'orchestration. Ce moteur d'orchestration est une entité logicielle qui joue le rôle d'intermédiaire entre les services en les appelants suivant le script d'orchestration. **[38]**

3.2.2 Langages d'orchestration :

Dès la naissance du terme orchestration, de nombreux langages ont été développés pour assurer la coordination et le bon fonctionnement de services web. Parmi lesquels, nous pouvons citer : XLANG, WSFL, BPML, JVCL ou BPEL.

3.2.2.1 BPEL4WS - Business Process Execution Language for Web Services:

3.2.2.1.1 Concepts du langage:

Le langage BPEL4WS **[50]** est une spécification d'IBM **[51]**, Microsoft **[52]**, et BEA **[45]**. Elle remplace les précédentes spécifications XLANG de Microsoft, et WSFL d'IBM.

Le modèle de procédé BPEL4WS forme une couche au-dessus du WSDL. Il définit la coordination des interactions entre l'instance de procédé et ses partenaires.

BPEL4WS contient les caractéristiques d'un langage structuré en blocs du XLANG, ainsi que les caractéristiques d'un graphe direct de WSFL. **[37]**

Avec BPEL4WS, il est possible de décrire les processus métiers de deux manières distinctes :

Les processus métiers exécutables : ce sont des processus qui composent un ensemble de services web existants et précisent l'algorithme exact d'exécution des activités. En effet, les processus exécutables sont importants, car ils présentent les détails exacts du fonctionnement, de remplir l'écart entre les spécifications du processus métier et le code responsable de leur exécution.

Les processus métiers abstraits : ce sont des processus qui décrivent l'échange de messages publics entre le processus métier et son environnement (services invoqués) sans spécifier les détails techniques des flux de processus. Les processus métiers abstraits sont généralement définis pour décrire le comportement externe du processus métier avec son environnement (partenaires). **[53]**

a) Les partenaires :

La relation entre le procédé et un partenaire est une relation "poste à poste" (peer-to-peer). Le partenaire est en même temps le consommateur d'un service que le procédé produit, et le producteur d'un service que le procédé consomme.

Le lien de partenariat (partner link) définit le rôle que joue chacun des deux partenaires dans une conversation. Chaque lien de partenariat a un type (partnerLinkType).

b) Les propriétés de message :

BPEL4WS ajoute des propriétés aux messages échangés entre partenaires. Un exemple de ces propriétés est la "corrélation", qui permet de faire le lien entre les messages appartenant à la même conversation.

c) Les activités :

Le procédé dans BPEL4WS est constitué d'activités, liés par un flot de contrôle. Ces activités peuvent être *basiques* ou *structurées*.

Quelques activités basiques sont : *"invoke"* pour invoquer une opération dans un service web ; *"receive"* pour attendre un message d'une source externe ; *"reply"* pour répondre à une source externe ; *"assign"* pour copier les données d'une place à l'autre.

Les activités structurées sont composées d'autres activités basiques et structurées. Quelques types d'activités structurées sont : *"sequence"* pour définir un ordre d'exécution ; *"while"* pour les boucles ; *"pick"* pour attendre l'arrivée d'un événement ; *"flow"* pour l'acheminement parallèle.

d) Les données :

Le procédé dans BPEL4WS a un état, cet état est maintenu par des *"variables"* contenant des données. Ces données sont combinées afin de contrôler le comportement du procédé, pour cela nous utilisons les *"expressions"*. Les expressions permettent d'ajouter des conditions de transition ou de jointure au flot de contrôle.

Dans BPEL4WS il n'y a pas de flot de données, BPEL4WS se sert des variables pour passer une donnée d'une activité à une autre, à l'aide de l'affectation. [37]

3.2.2.2 BPML - Business Process Modeling Language:

Le langage BPML a été développé par l'organisation BPMI.org *(Business Process*

Management Initiative). Ce langage s'intéresse plutôt aux procédés métier (Business Process). Il gère la coordination de toute sorte de participants, et non pas seulement de services web.

3.2.2.2.1 Concepts du langage :

Un processus métier BPML est un enchaînement d'activités simples, complexes, et de processus incluant une interaction entre participants dans le but de réaliser un objectif métier. Les éléments de définition d'un processus sont :

a) Les messages :

Le message est l'unité d'interaction du processus. Les interactions de messages modélisent le stockage et la recherche de données, l'invocation de méthodes, et la gestion d'éléments de travail.

b) Les participants :

Les participants peuvent être des systèmes, des applications, des services web, des utilisateurs humains, des partenaires commerciaux, et d'autres processus.

Il existe deux types de participants dans BPML : les participants statiques (où l'identité et le comportement sont connus à l'avance), et les participants dynamiques (ajoutés au moment de l'exécution).

c) Les activités :

L'activité est l'unité d'exécution du processus. Les participants et les processus sont représentés en BPML comme des activités productrices et consommatrices de messages. Il existe deux classes d'activités dans BPML : les tâches simples, et les tâches complexes composées à partir de tâches simples.

d) Les transactions :

L'activité peut posséder un attribut supplémentaire afin de spécifier si elle est exécutée dans un contexte transactionnel. Il est également possible de définir des compensations pour les transactions longues partielles. Les transactions dans BPML peuvent être emboîtées.

BPML définit deux modèles de transactions : coordonnées (pour les délais courts), et étendues (Pour les délais longs).

e) Les exceptions :

BPML définit un système de gestion d'exceptions, en ajoutant un dispositif de récupération d'erreur au moteur de procédé.

f) Les règles métier :

Les règles du processus gouvernent le choix des tâches, la gestion de la consommation ou de la production de messages, et la réaction aux erreurs. Elles sont du style :

Si {conditions} alors {actions}

Les conditions portent sur les variables globales du processus ou sur les données échangées entre participants. Les actions déclenchent d'autres tâches BPML. [37]

1. Avantages de l'orchestration :

L'orchestration est considérée comme la meilleure approche de coordination de services Web puisqu'elle offre plusieurs avantages compétitifs, parmi lesquels nous citons :

- ✓ **Avantage organisationnel :** réduction de l'écart entre la conception et l'implémentation.
- ✓ **Avantage de gestion :** réduction des couts de mise en œuvre en proposant l'utilisation de normes et de standards ouverts.
- ✓ **Avantage stratégique :** Meilleure flexibilité : les services Web peuvent être intégrés sans soucis parce qu'ils n'ont pas "conscience" d'appartenir à une composition.
- ✓ **Avantages techniques :** Portabilité, simplicité et réutilisabilité du code. [54]

i. Synthèse et discussion :

L'objectif de notre travail est d'effectuer la spécification d'un SoS, en utilisant une approche architecture logicielle. Afin de trouver une bonne approche pour réaliser le SoS, nous avons effectué une étude comparative entre les différents travaux déjà présentés.

Pour mener cette comparaison, nous avons fixé un certain nombre de propriétés des SoS, et voir quel est l'approche qui couvre ces propriétés. Notons que ces propriétés ont été déduites des caractéristiques des SoS, ainsi que de leurs objectifs.

Un langage qui sert à effectuer la spécification d'un SoS doit permettre la description des différents sous-systèmes, de leurs relations, le flux de données entre les différents sous-systèmes, de la mission du SoS, ainsi que la coordination entre les sous systèmes.

Approches / Propriétés	Orientée Composants				Orientée Services			
	EJB	Corba	COM	.Net	WSCI	WSCL	BPEL	BPML
Sous systèmes	+	+	+	+	+	+	+	+
Relation	-	-	+	+	+	+(limité)	+	+
Flux de données	+	+	-	+	+	+(limité)	+	+
Mission SoS	+	+	+	+	-	-	+	+
Coordination	-	-	-	-	-	-	+	+

Tableau II.1 : comparaison entre les approches étudiées

En analysant cette comparaison, nous remarquons que les approches orientées service utilisant le mécanisme d'orchestration couvrent un maximum de propriétés. De ce fait, pour effectuer la spécification du SoS, nous allons utiliser une approche orientée service avec orchestration. Nous avons deux langages d'orchestration qui sont BPEL et BPML, mais vu la popularité du langage BPEL, nous allons l'utiliser pour spécifier notre SoS.

II. Conclusion :

Dans ce chapitre nous avons parlé sur les approches à composants et les approches à services et les différents types de composition (la chorégraphie et l'orchestration) et ses différents langages.

Dans le chapitre suivant nous allons parler sur le langage BPEL d'une manière détaillé.

CHAPITRE III

Business Processes Execution Language (BPEL)

I. Introduction :

Les architectures orientées services présentent en elles-mêmes bien des avantages. Un de leurs atouts majeurs reste de faciliter la gestion des processus métiers. Cette gestion des processus métiers passe par leur représentation et leur construction. Tel est l'objet de BPEL (Business Processes Execution Language).
Le processus BPEL permet de représenter et gérer graphiquement les processus métiers.
Les processus métiers occupent une place centrale dans la mise en œuvre et l'évolution des systèmes. En effet, la maîtrise des processus métiers est un des points clé pour la réussite des projets.

BPEL offre la possibilité d'interagir avec des services web tiers et ainsi de faire communiquer tout type d'application ayant une interface service web, de mettre en place des processus complexes (traitements parallèles, interactions humaines, etc.), de gérer simplement les exceptions, etc.

On a parlé dans le chapitre précédent sur les composants du processus BPEL. Dans ce chapitre, nous allons détailler les composants du langage BPEL, car c'est le langage qui a été choisi pour notre travail.

II. Exemple générique d'un processus BPEL :

Pour ses clients, un processus BPEL prend l'aspect d'un service Web. La définition d'un processus BPEL consiste à définir un nouveau service Web qui est une composition de services existants.

L'interface du nouveau processus BPEL (ou du service Web composite) utilise un ensemble de port types à travers lesquels il propose ses opérations comme n'importe quel service Web. Pour invoquer un service Web décrit avec BPEL, il suffit d'invoquer le service Web composite résultant. La figure qui suit schématise un processus BPEL.
La Figure III.1 met en scène un processus BPEL qui attend la requête d'un client grâce à la balise *<receive>*, et qui lui répond via la balise *<reply>*. Le processus BPEL compose deux services Web (service Web 1 et 2). Les invocations de ces services Web sont réalisées par le biais des balises *<invoke>* et d'autres balises (par exemple des balises de séquence ou de condition) que nous ne détaillons pas dans cette figure. **[28]**

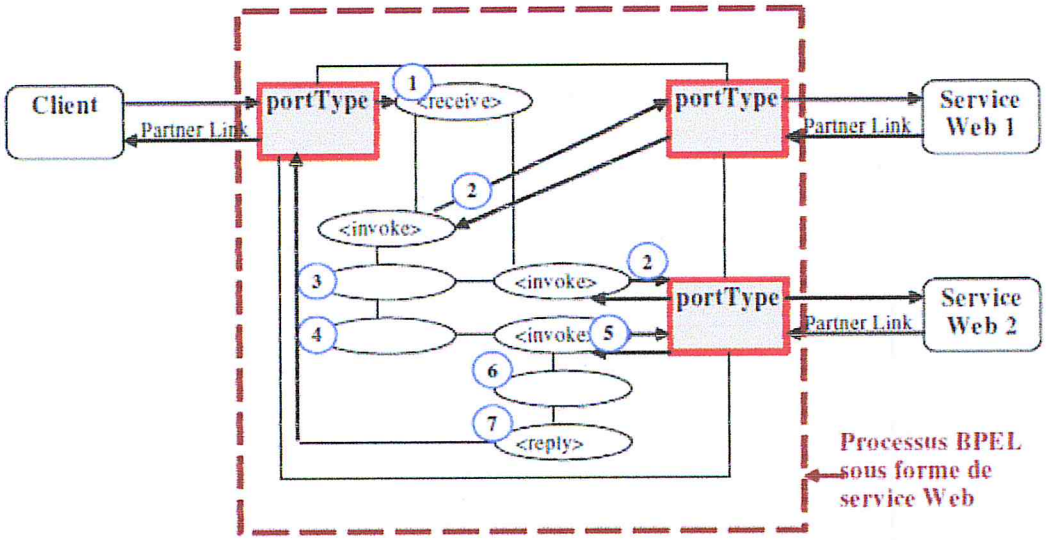

Figure III.1 : Exemple générique d'un processus BPEL. **[28]**

III. Les composants de BPEL :

La force du langage BPEL se manifeste dans la structure de son processus qui est basé sur un fichier XML et qui est composé de trois blocs principaux : les services web qui participent dans la composition du processus (les partenaires), les variables manipulées dans le processus et les activités de traitement qui composent le processus métier.

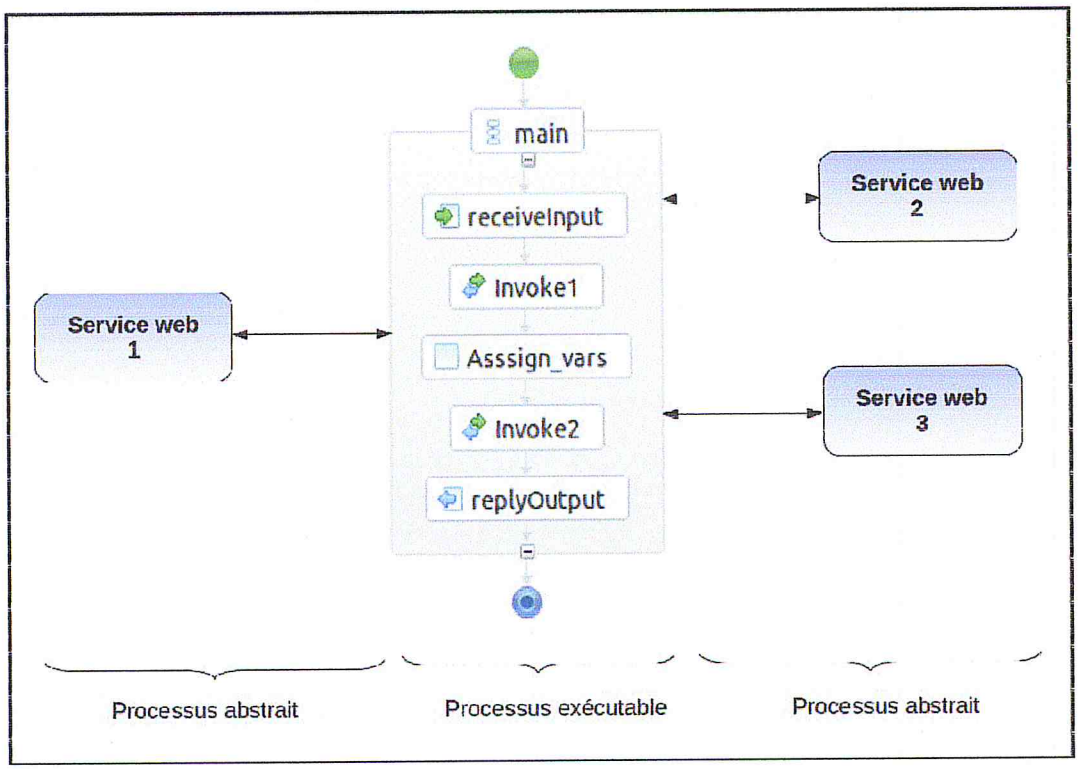

Figure III.2 : Types de processus BPEL. **[53]**

1. Les *partnerlinks :*

Les *partnerslinks* sont les différentes parties (services web) qui interagissent avec le processus BPEL. Chaque *partnerLink* spécifie un ou deux attributs principaux à savoir :

- *MyRole* : indique le rôle du processus métier.
- *PartnerRole* : indique le rôle du partenaire.

Le *partnerLink* peut spécifier un seul rôle dans le cas des opérations synchrones, bien qu'il spécifie deux rôles pour les opérations asynchrones (voir la **Figure III.3**).

```
<partnerLinks>
        <bpel:partnerLink name="Partnerlink"
        partnerLinkType="tns:PLT"
        PartnerRole="PRole"
        MyRole="MRole"/>
</partnerLinks>
```

Figure III.3 : Structure générale du Partnerlink. **[53]**

Chaque *partnerlink* est lié à un *partnerLinkType* (PLT) spécifique qui le caractérise. Il est défini au niveau du fichier WSDL comme élément fils de l'élément *<WSDL:definition>* et il représente le genre des messages WSDL échangés entre deux services web. Un PLT caractérise cet échange en définissant les rôles joués par chaque service et en spécifiant le *portType* fourni par le service afin de recevoir les messages appropriés à l'échange (voir la **Figure III.4**).

```
<plnk:partnerLinkType name="PLT">
        <plnk:role name="role1">
          <plnk:portType name="PT1"/>
        </plnk:role>

        <plnk:role name="Role2">
          <plnk:portType name="PT2"/>
        </plnk:role>
</plnk:partnerLinkType>
```

Figure III.4 : Structure générale du *partnerLinkType*. **[53]**

2. Les variables :

Les variables sont utilisées pour stocker les messages échangés entre le processus BPEL et ses partenaires ou pour stocker les informations relatives à l'état du processus. Afin de spécifier une variable, il faut spécifier son nom (*name*) et son type (*type*). Le type est représenté par l'un des trois attributs :

- *messageType* : dans le cas où la variable stocke un message WSDL.
- *element* : Dans le cas où la variable stocke un élément (type complexe) du schéma XML.
- *type* : Dans le cas où la variable stocke un type simple du schéma XMI. [28] (voir la **Figure III.5**).

```
<variables>
        <bpel:variable name="varName"
        messageType="tns:msgType"
        />
</variables>
```

Figure III.5 : Structure générale de variable. **[53]**

3. Les activités :

Dans le langage BPEL, la logique des interactions entre un service et son environnement est décrite à travers un ensemble d'activités de base et structurées

a) Les activités de base:

Les activités de base de BPEL sont des activités élémentaires. Elles sont décrites comme suit :

- ✓ ***invoke*** : Cette activité permet au processus métier d'invoquer un service web partenaire. Cette invocation peut être réalisée de manière synchrone ou asynchrone. En effet, lorsque le processus d'affaire invoque une opération sur le service web, il définit une variable d'entrée « *inputvariable* » qui contient un ensemble de paramètres comme des messages d'entrée avec les services web. Ensuite, lorsque le service renvoie la réponse, elle sera stockée dans une variable de sortie « *outputvariable* ».

- ✓ ***receive :*** Cette activité permet à un processus BPEL de se bloquer et d'attendre la réception d'un message à partir d'un partenaire.

- ✓ ***reply*** **:** Cette activité permet à un processus de répondre à un message qu'il aurait reçu par l'activité *receive*. La combinaison d'un *receive* et d'un *reply* permet de réaliser une opération de question/réponse en mode synchrone.

- ✓ ***assign :*** Cette activité fournit une méthode pour manipuler les données telle que la copie ou l'affectation du contenu entre les variables ou expressions. Cette activité peut contenir plusieurs nombres d'affectations élémentaires en spécifiant pour chaque affectation la source (<from>) et la destination (<to>).

- ✓ ***validate*** **:** cette activité permet de valider la valeur des variables dans leurs schémas de définition.

- ✓ ***empty :*** Cette activité permet d'insérer une instruction non-opérationnelle qui ne fait rien dans un processus. **[53]**

b) Les activités structurées :

Les activités structurées sont des activités composées. Elles décrivent l'ordre des flux de contrôle et peuvent être formées par d'autres activités (de base ou structurées) d'une manière récursive. Trois catégories de flux de contrôle dans les activités structurées : traitements séquentiels, parallèles et sélectifs.
Les activités structurées de BPEL sont décrites comme suit :

- ✓ ***If :*** Cette activité permet d'ajouter une logique conditionnelle à la définition du processus BPEL. L'utilisation des éléments *elseif* ou *else* qui sont facultatifs permet la définition d'une ou plusieurs branches conditionnelles au sein de l'activité If.

- ✓ ***Pick :*** Cette activité permet d'attendre la réception de nombreux messages, le premier message arrivé détermine le chemin qui sera exécuté. L'activité *pick* peut contenir une minuterie qui spécifie le délai d'attente soit comme une date précise ou comme une

période de temps. Ce chemin sera choisi si aucune entrée correspondant à l'un des messages reçus pendant un délai d'attente n'est reçu.

- ✓ ***While*** **:** Cette activité permet de réaliser une boucle *while* traditionnelle. En effet, elle exécute les activités d'une façon itérative tant que la condition est à vrai.

- ✓ ***Foreach*** **:** Cette activité permet d'exécuter pour un nombre bien déterminé l'activité scope interne. Les itérations d'exécution peuvent se produire en parallèle ou en séquence. Le nombre d'itérations à effectuer est déterminé par une valeur de départ et une valeur finale.

- ✓ ***RepeatUntil*** **:** Cette activité permet d'exécuter son contenu une ou plusieurs fois tant que les conditions spécifiques égales à vrai après l'exécution de chaque itération.

- ✓ ***Sequence*** **:** Cette activité permet d'exécuter de façon séquentielle une ou plusieurs activités BPEL. La séquence se termine elle-même lorsque la dernière activité du processus est achevée.

- ✓ ***Wait*** **:** Cette activité permet d'attendre une période de temps donnée ou jusqu'à un certain temps spécifié pour exécuter un bloc d'activités.

- ✓ ***Scope*** **:** Cette activité se compose d'un ensemble d'activités imbriquées qui peuvent avoir leurs propres états internes comme des *partnerlinks* et variables locales, des gestionnaires d'erreur, des gestionnaires de compensation qui sont tous visibles localement. Elle est analogue à un bloc interne dans un langage de programmation.

- ✓ ***Flow*** **:** Cette activité permet d'exécuter en parallèle plusieurs processus. Ces différents processus peuvent être indépendants. Aussi, elle ne termine son exécution que si tous les processus sont achevés en assurant une synchronisation totale. **[53]**

c) Les exceptions (Faults) :

Quand le processus BPEL s'exécute, des erreurs peuvent se produire lors de l'invocation de services web ou même lors de l'exécution du processus lui-même. Pour cela, BPEL fournit un ensemble d'activités pour gérer les exceptions. Ces activités sont les suivantes :

- ✓ ***Exit :*** Cette activité force l'arrêt de l'exécution du processus avant qu'il ne soit fini et sans notification de l'état du processus BPEL.

- ✓ ***Fault handler :*** Cette activité déclare les erreurs qui peuvent être générées et contiennent une ou plusieurs activités dont chacune permet de détecter une sorte spécifique d'erreurs afin de les résoudre.

- ✓ ***Throw :*** Cette activité permet de jeter une exception pour être capturée et résolue par l'activité FaultHandler déclarée dans l'activité scope.

- ✓ ***Rethrow :*** Nous utilisons l'activité *rethrow* lorsque l'activité *fault handler* ne peut pas gérer l'exception courant et nous voulons la propager à un gestionnaire d'exception externe de l'activité scope. L'activité *rethrow* n'est disponible que dans l'activité *fault handler*, car une seule exception existante peut être relancée.

- ✓ ***Compensate :*** Cette activité permet de restaurer (Roll-back) la transaction d'une activité. Elle se produit quand un processus ne peut pas terminer plusieurs opérations, après avoir déjà terminé les autres.

- ✓ ***CompensateScope :*** Cette activité est utilisée pour démarrer l'activité *compensate* dans l'activité scope pour annuler sa transaction dans le cas où elle serait terminée avec succès **[53]**

IV. Serveurs BPEL :

Un serveur BPEL fournit un environnement d'exécution aux processus d'entreprise BPEL. BPEL est très relié aux services Web et aux plates-formes qui les supportent telles que J2EE ou .Net de Microsoft.

Plusieurs serveurs commerciaux ont vu le jour tels que :

- ❖ Oracle BPEL Process Manager. **[55]**
- ❖ Microsoft BizTalk. **[56]**

Il existe également des serveurs BPEL open source tels que :

- ❖ ActiveBPEL Engine. **[57]**
- ❖ Apache Agila **[58]** précédemment connu sous le nom de Twister. **[28]**

V. Conclusion :

Dans ce chapitre, nous avons évoqué d'une manière détaillée le processus BPEL et ses différents composants, car c'est le langage qui a été choisi. Dans notre travail, chaque sous système est considéré comme un processus BPEL.

Dans le chapitre suivant nous présenterons la conception et la modélisation de notre système.

Partie II

Conception et mise en œuvre

Sommaire :

CHAPITRE IV

CONCEPTION

I. Introduction

La conception a pour but de définir de façon très précise les fonctions du logiciel, à partir des besoins exprimés et des contraintes générales définies en phase de spécification des besoins.

II. Cycle de vie suivi:

Le cycle de vie d'un logiciel, désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage permet de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre. [59]

Le cycle en V est le modèle choisit, c'est un paradigme du développement informatique. Il est représenté par un V dont la branche descendante contient toutes les étapes de la conception du projet, et la branche montante toutes les étapes de tests du projet. La pointe du V représente la réalisation concrète du projet « le codage ».

En effet, chaque étape d'une branche a son pendant dans l'autre branche, c'est à dire qu'une étape de conception correspond à une étape de test qui lui est spécifique. [60]

Contrairement au modèle en cascade dont les étapes de testes sont effectuées après l'étape de codage, on trouve dans le modèle en V que chaque étape de test peut être élaborée dès que la phase de conception correspondante est terminée. Donc, par rapport à un logiciel fait en utilisant le modèle en cascade, le nombre de défauts dans le logiciel effectué en utilisant le modèle en V est moins.

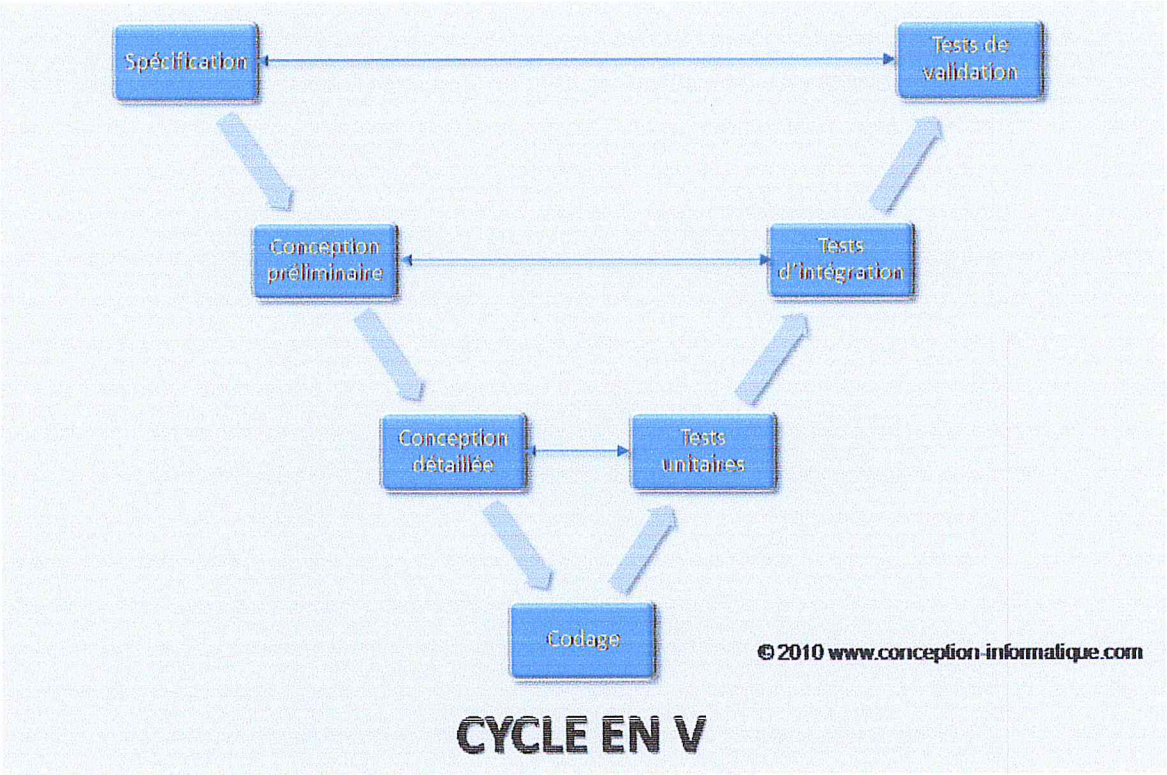

Figure IV.1 : Modèle du cycle de vie en V. [60]

Le cycle de vie du logiciel comprend généralement au minimum les étapes suivantes :

1. Spécification :

La spécification est souvent le document d'entrée du projet, c'est une suite d'exigences, si possible regroupées par thèmes, pour permettre d'esquisser un début d'architecture et aussi un plan de test. Chaque exigence devrait être testable.

2. Conception :

La conception regroupe les activités d'étude qui suivent la spécification, et ce jusqu'au codage.

La conception englobe :

- **la modélisation**: Elle permet de définir et de visualiser le système ou une partie de celui-ci, aussi bien de manière statique que dynamique. La modélisation n'est pas une étape en soi, elle peut être utilisée à tout endroit du projet.
- **l'architecture, ou conception préliminaire**: indispensable à partir d'une certaine taille de projet, elle définit l'ensemble des briques constitutives de l'application et leurs interfaces. **[60]**
- **La conception détaillée**: Le système conçu est divisé en plus petites unités ou modules et chacun d'eux est expliqué de façon que le programmeur puisse commencer à coder directement. Le document de conception détaillée contiendra une logique fonctionnelle détaillée du module, en pseudo-code :
  - ✓ Tous les détails de l'interface.
  - ✓ Les listes de message d'erreur.
  - ✓ Les entrées et les sorties d'un module.
  - ✓ …. **[61]**

La conception est donc une étape à ne pas négliger, elle n'est pas plus importante qu'une autre au niveau contractuel, mais techniquement une conception loupée détruira un projet de manière sûre et certaine.

3. Codage :

Le codage consiste à mettre en forme la conception détaillée. La tâche parait simple mais le codeur doit savoir lire une conception détaillée, et surtout doit connaitre le langage utilisé.

4. Tests unitaires :

Un test unitaire consiste à rédiger un scénario codé dans le langage de développement, qui peut être lancé, automatiquement ou non, à tout moment.

5. Tests d'intégration :

Les tests d'intégration consistent à tester le comportement de l'application en intégrant progressivement toutes les briques du logiciel.

6. Tests de validation :

Les tests de validation consistent à dérouler les tests permettant d'affirmer que le logiciel répond aux exigences de la spécification. Ce sont souvent des scénarios déroulés automatiquement. **[60]**

III. Etude de cas :

Dans le cadre de cette étude, on a choisi le projet de l'agence de voyage SOS (SOS TA) **[62]** pour illustrer l'application de l'approche orienté service (SOA) sur SOS.

Nous présentons d'abord une spécification informelle de SOS TA avant de fournir une formulation plus détaillée basée sur UML.

L'agence de voyage est un système de systèmes qui comporte les systèmes suivants : l'agent de voyage, compagnie aérienne, réservation d'hôtel.

Cette agence doit permettre la réservation des chambres d'hôtels et des billets d'avion. Chaque hôtel possède plusieurs chambres et il est caractérisé par un numéro, un nom et une adresse.

Une compagnie aérienne assure plusieurs vols, et elle est caractérisée par un numéro et un nom.

Le client peut réserver son voyage via des services web en spécifiant le lieu de départ et d'arrivée et la date de départ et d'arrivée ainsi qu'il peut annuler une réservation en utilisant le numéro de billet et le numéro de réservation d'hôtel.

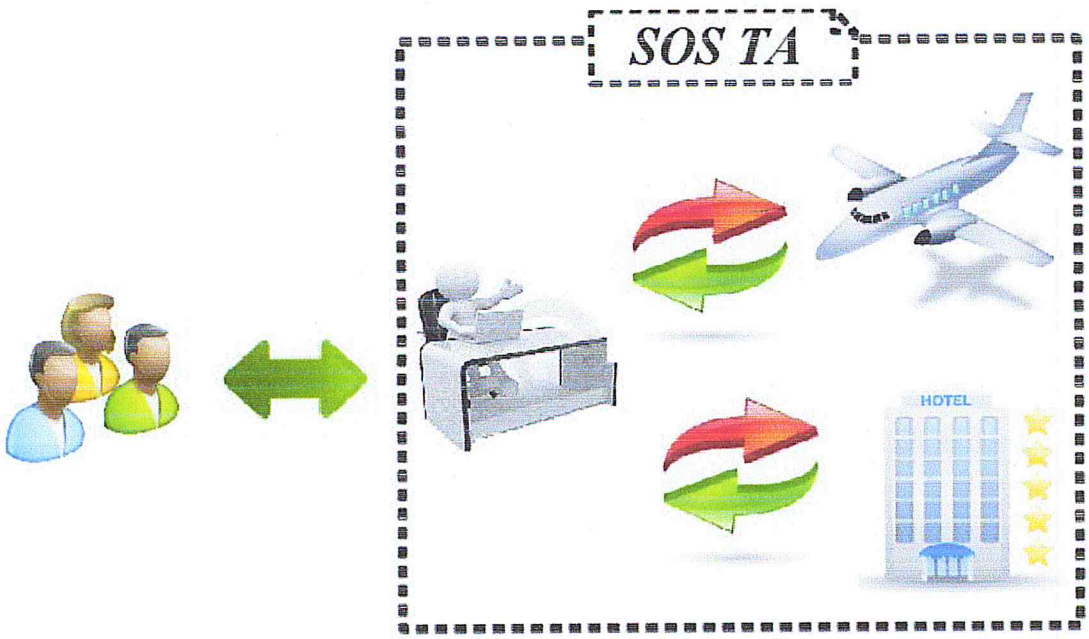

Figure IV.2: Architecture SOS TA.

IV. Le langage UML :

1. Définition

UML ou Unified Modeling Language, est un langage de modélisation qui est né au milieu des années 90 de la fusion de trois méthodes objets: OMT (Object Modeling Technique), BOOCH1 (GRADY BOOCH le concepteur de la méthode) et OOSE (Object Oriented Software Engineering).

C'est à partir de 1997 que l'OMG2 (Object Management Group) qui standardise les technologies de l'objet, s'est attachée à la définition d'un langage commun unique, utilisable par toutes les méthodes objets dans toutes les phases du cycle de vie et compatible avec les techniques de réalisation du moment, d'où la naissance d'UML. UML offre des éléments pour décrire les différents aspects d'un système: les diagrammes. **[63]**

2. Les modèles d'objet UML :

D'après **[64]**, Plusieurs modèles d'objet UML sont utilisés pour comprendre le SOS et ses systèmes constitutifs ainsi que pour identifier/comprendre les fonctions d'un système qui peuvent être employées pour développer de nouvelles capacités.

Ces modèles commencent par des modèles de « boîte noire » pour comprendre le SOS et ses constituants à un niveau élevé et sont évolués aux modèles de « boîte blanche » qui saisissent les informations internes sur les systèmes constitutifs requis pour comprendre les options des différentes capacités de SOS.

Ces modèles de SOS, décrits dedans incluent :

❖ **Les Modèles de boîte noire :**

- **Les diagrammes de contexte** : Ils peuvent être utilisés tout au long du cycle de vie des systèmes, mais ils sont particulièrement utiles pour la compréhension et l'ingénierie des exigences pour un système. **[65]**
- **Les diagrammes de cas d'utilisation**: Le diagramme de cas d'utilisation représente la structure des grandes fonctionnalités nécessaires aux utilisateurs du système. C'est le premier diagramme du modèle UML, celui où s'assure la relation entre l'utilisateur et les objets que le système met en œuvre. **[59]**
- **Les diagrammes de séquence:** Illustre la séquence de requêtes et les réponses qui en découlent dans l'environnement SOS. **[64]**

❖ **Les modèles de boite blanche :**

- **Les entités des systèmes constituants:** Décrit les principales fonctions d'un seul système qui peuvent être effectuées par ce dernier.
- **Les entités d'interface:** Décrit chaque interface dans le SOS et l'information qui passe sur cette interface.
- **Les entités Entrée / Sortie (I / O):** Décrit les détails de chaque type d'élément de données qui passe sur les différentes interfaces. [64]

3. Spécification des besoins :

3.1 Identification des ressources :

Les systèmes constituants de SOS TA sont décrits comme suit :

L'agent de voyage : C'est le système qui émit des requêtes de réservation ou d'annulation vers la compagnie aérienne et le système de réservation d'hôtel lors de la réception d'une requête de client.

La compagnie Aérienne : fournit des réservations de vols dans des dates prévues.

Réservation d'hôtel : fournit des réservations de chambres dans des dates prévues.

3.2 Diagramme de contexte :

Les systèmes constitutifs qui interagissent avec l'agent de voyage sont illustrés dans le diagramme de contexte de l'agent de voyage représenté dans la **Figure IV.3.** Il est utilisé pour comprendre les systèmes constitutifs à haut niveau du point de vue de l'agent de voyage.

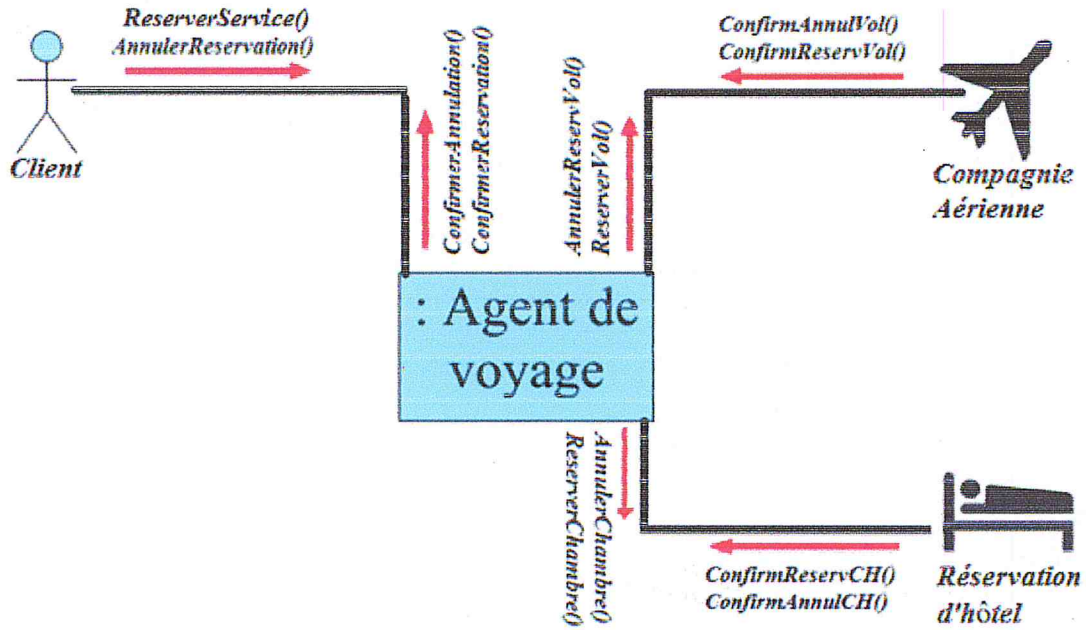

Figure IV.3: Diagramme de contexte de l'agent de voyage.

3.3 Diagramme de cas d'utilisation :

La figure IV.4 illustre un diagramme de cas d'utilisation à haut niveau pour les principales missions de SOS TA. Cette figure identifie deux cas d'utilisation : Réservation d'un service et Annulation d'une réservation. Il montre également les acteurs qui participent à chaque cas d'utilisation. Étant donné qu'il s'agit d'une réservation ou d'une annulation, c'est le client qui initie chaque activité à travers l'agent de voyage.

Chaque diagramme de cas d'utilisation est ensuite analysé et élaboré par la rédaction d'une spécification de cas d'utilisation. La spécification de cas d'utilisation contient une brève description du cas d'utilisation.

Remarque : Les descriptions détaillées vont être organisées dans des tableaux de la façon suivante :

Elément	Signification
Cas d'utilisation	Le nom de cas d'utilisation

Acteur	L'acteur qui réalise ce cas d'utilisation
But	Le but de cas d'utilisation
Description	Une explication de cas d'utilisation
Pré condition	Les conditions qu'elles doivent être vérifiées afin de démarrer le cas d'utilisation
Post condition	Les résultats de cas d'utilisation
Exception	Les informations entrées par l'acteur

Tableau IV.1 : Modèle de représentation des descriptions détaillées des cas d'utilisations.

A. Diagramme de cas d'utilisation global :

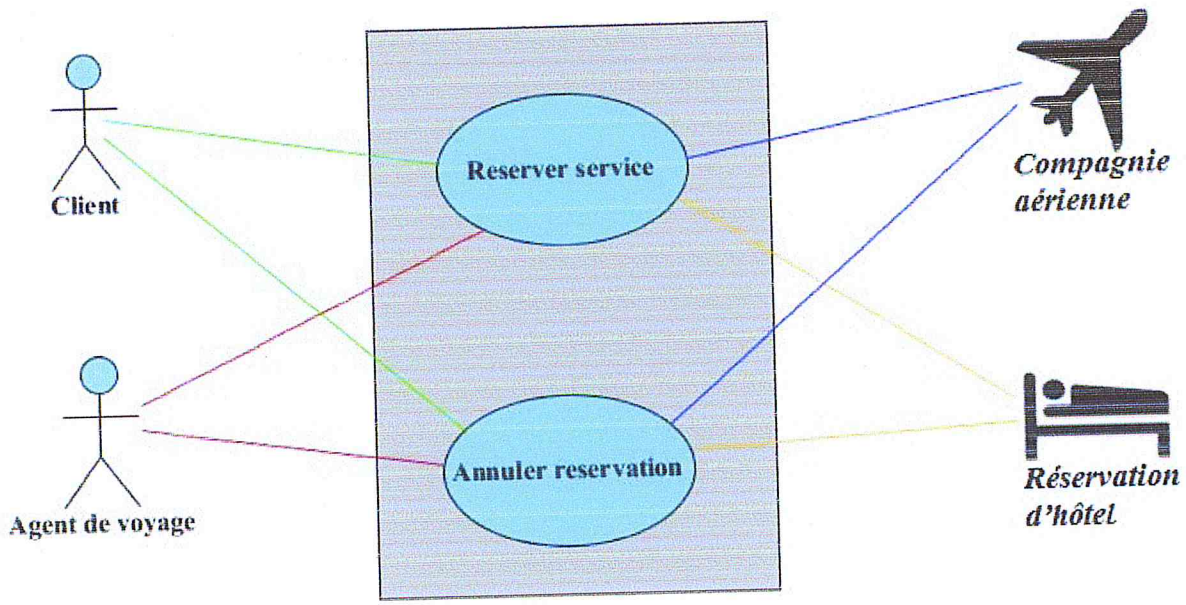

Figure IV.4 : Diagramme de cas d'utilisation global.

➢ **Réservation d'un service :**

Elément	Signification
Cas d'utilisation	Réservation d'un service
Acteur	Client, Compagnie aérienne, Réservation d'hôtel, Agent de voyage.
But	Permet la réservation d'un service choisit par le client.
Description	La réservation doit passer par le système de l'agent de voyage.
Pré condition	Le service ne doit pas être déjà réservé
Post condition	Le client obtient sa réservation.
Exception	Pas d'exception.

Tableau IV.2 : Description du cas d'utilisation « Réservation d'un service »

➢ **Annulation d'une réservation :**

Elément	Signification
Cas d'utilisation	Annulation d'une réservation
Acteur	Client, Compagnie aérienne, Réservation d'hôtel, Agent de voyage.
But	Permet l'annulation d'une réservation d'un service choisit par le client.
Description	L'annulation d'une réservation doit passer par différentes systèmes.
Pré condition	Pour annuler une réservation il faut qu'elle soit déjà réservée.
Post condition	Le client annule la réservation voulu.
Exception	Pas d'exception.

Tableau IV.3 : Description du cas d'utilisation « Annulation d'une réservation »

B. Diagramme de cas d'utilisation détaillé d'une réservation d'un service :

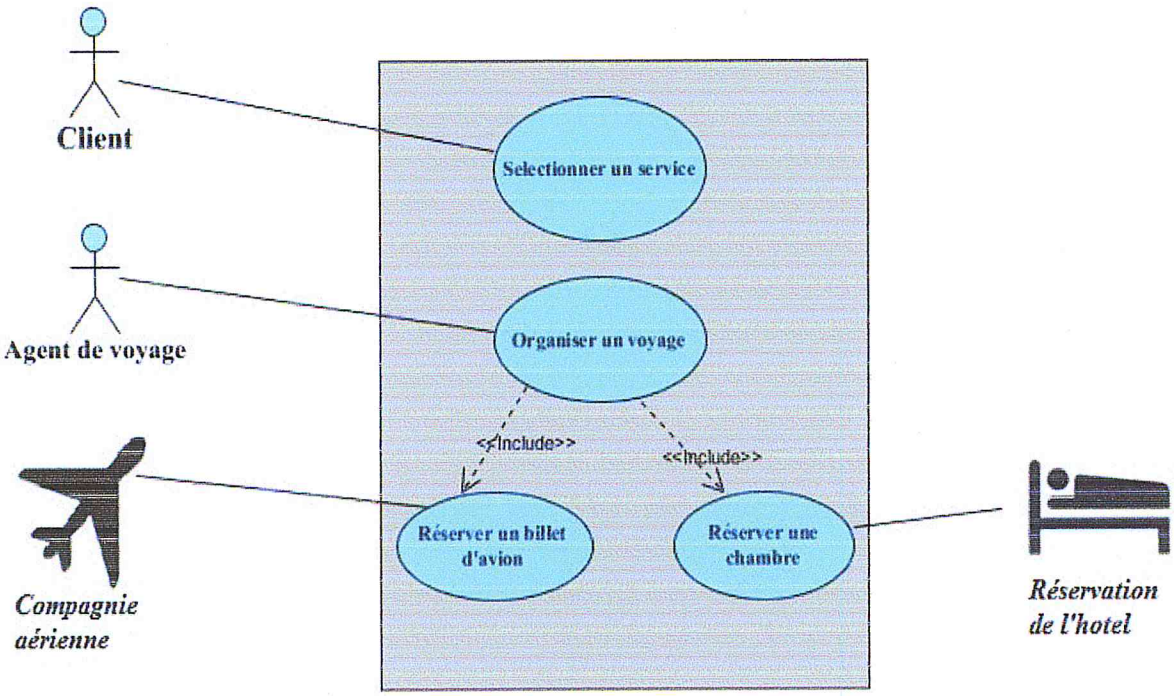

Figure IV.5 : Diagramme de cas d'utilisation détaillé d'une réservation d'un service.

➢ **Réservation d'un billet d'avion :**

Elément	Signification
Cas d'utilisation	Réservation d'un billet d'avion.
Acteur	Compagnie aérienne.
But	Permet la réservation d'un billet d'avion.
Description	La réservation d'un billet d'avion doit passer par l'agent de voyage et la compagnie aérienne.
Pré condition	Le billet n'est pas encore réserver.
Post condition	Le client obtient un billet d'avion.
Exception	Pas d'exception.

Tableau IV.4 : Description du cas d'utilisation « Réservation d'un billet d'avion »

➢ **Réservation d'une chambre :**

Elément	Signification
Cas d'utilisation	Réservation d'une chambre.
Acteur	Réservation d'hôtel.
But	Permet la réservation d'une chambre.
Description	La réservation d'une chambre doit passer par l'agent de voyage et la réservation d'hôtel.
Pré condition	La chambre n'est pas occupée.
Post condition	Le client obtient une chambre dans l'hôtel.
Exception	Pas d'exception.

Tableau IV.5 : Description du cas d'utilisation « Réservation d'une chambre »

C. Diagramme de cas d'utilisation détaillé de l'annulation d'une réservation :

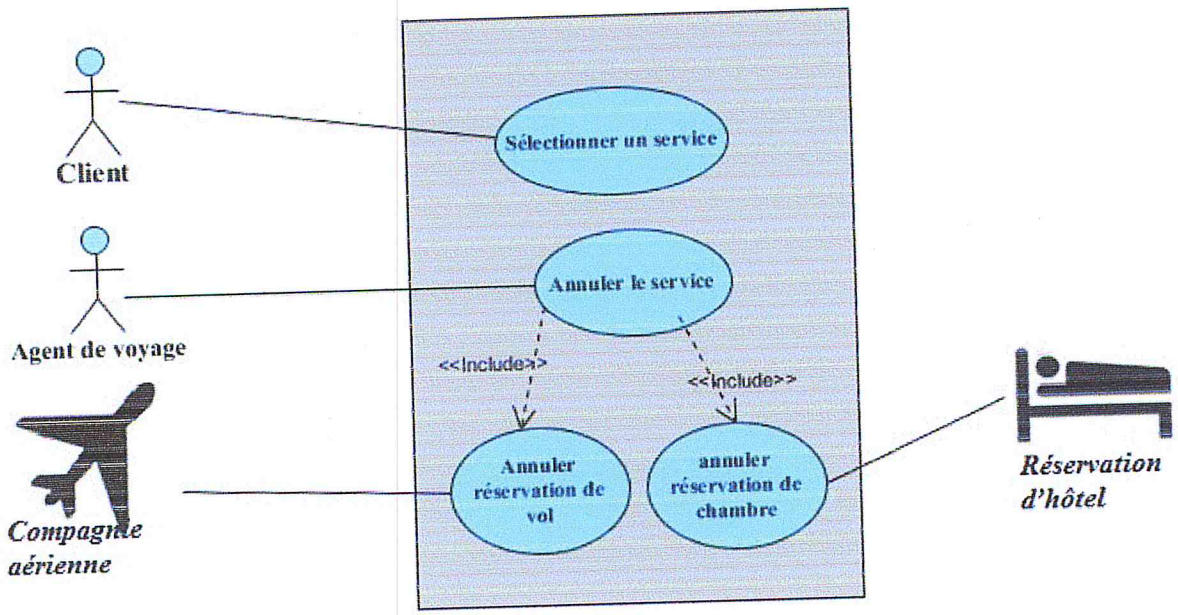

Figure IV.6 : Diagramme de cas d'utilisation détaillé de l'annulation d'une réservation.

➢ **L'annulation d'une réservation de vol :**

Elément	Signification
Cas d'utilisation	Annulation d'une réservation de vol.
Acteur	Compagnie aérienne.
But	Permet d'annuler une réservation d'un billet d'avion.
Description	L'annulation de réservation d'un billet d'avion doit passer par l'agent de voyage et la compagnie aérienne.
Pré condition	Pour annuler une réservation de vol il faut qu'elle soit déjà réservée.
Post condition	Le client annule sa réservation de vol.
Exception	Pas d'exception.

Tableau IV.6 : Description du cas d'utilisation «L'annulation d'une réservation de vol »

➢ **L'annulation d'une réservation de chambre :**

Elément	Signification
Cas d'utilisation	Annulation d'une réservation de chambre.
Acteur	Réservation d'hôtel.
But	Permet d'annuler une réservation de chambre.
Description	L'annulation de réservation d'une chambre doit passer par l'agent de voyage et la réservation d'hôtel.
Pré condition	Pour annuler une réservation d'une chambre il faut qu'elle soit déjà réservée.
Post condition	Le client annule sa réservation de chambre.
Exception	Pas d'exception.

Tableau IV.7 : Description du cas d'utilisation «L'annulation d'une réservation de chambre»

4. Conception :

4.1 Conception préliminaire :

Dans cette partie on va décrire l'architecture de SOS TA ou bien son organisation fondamentale incorporé dans ses systèmes, leurs relations les uns aux autres et à l'environnement. Comme on a montré dans les sections précédentes, on a basé dans notre conception sur l'architecture orienté service (SOA) en utilisant la composition des services web avec orchestration basant sur le processus métier BPEL qui est aussi un service.

On va proposer que chaque service soit un système constitutif de SOS TA ainsi que le processus métier BPEL qui est le système de l'agent de voyage.

4.1.1 Diagramme de séquence :

Les diagrammes de séquence sont utilisés principalement pour concevoir , documenter et valider l'architecture, les interfaces et la logique du système en décrivant la séquence d'actions qui doivent être effectuées pour accomplir une tâche ou un scénario . Les diagrammes de séquence UML sont des outils de conception utiles, car ils offrent une vue dynamique du comportement du système.[66]

Donc les diagrammes de séquence sont extrêmement utiles comme outils d'ingénierie de SOS TA pour concevoir son architecture

4.1.2 Présentation des scenarios et diagrammes de séquence de SOS TA :

- **Réservation d'un service :**
  - **Scénario :**

1 . Le client envoie une demande de réservation d'un service.

1.1 : Le système de l'agent de voyage envoie une requête de réservation d'une chambre à l'hôtel.

1.2 : Le système de l'hôtel confirme la réservation de chambre.

2 : Le système de l'agent de voyage envoie une requête de réservation d'un vol.

3 : Le système de Compagnie aérienne confirme la réservation de vol.

3.1: L'agent de voyage confirme la réservation voulu par le client.

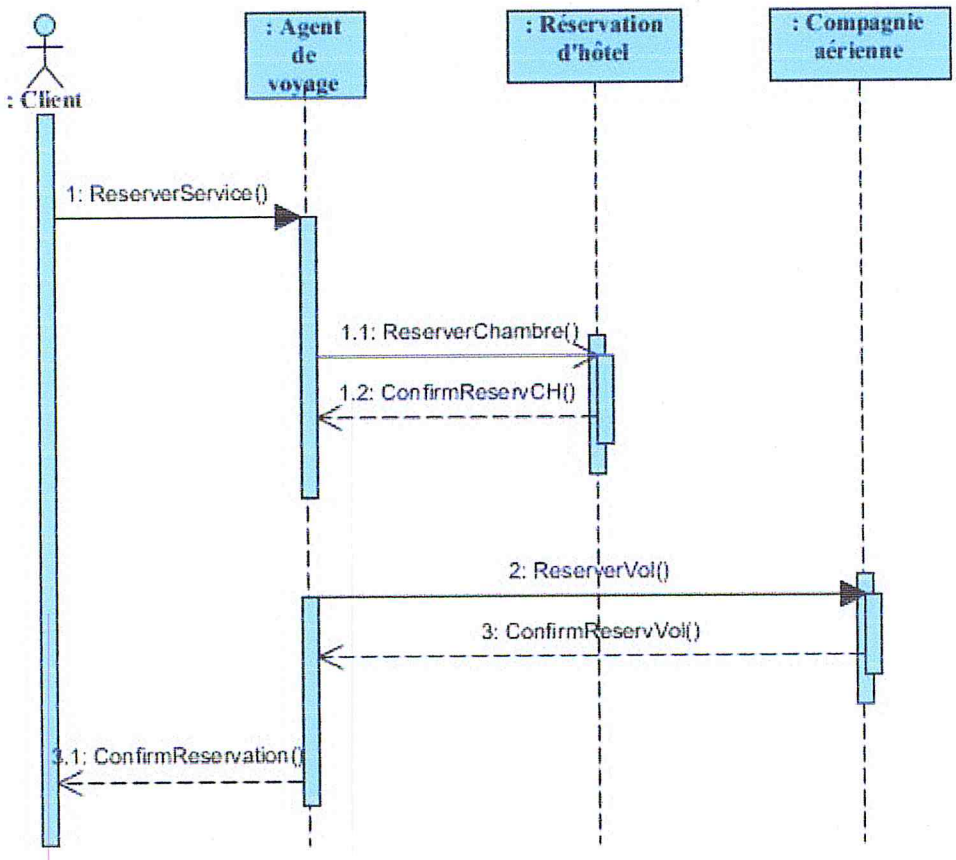

Figure IV.7: Diagramme de séquence de réservation d'un service.

❖ **Annulation d'une réservation :**

➢ **Scénario :**

1 : Le client envoie une demande d'annulation d'un service.

1.1 : Le système de l'agent de voyage envoie une requête à l'hôtel pour annuler la réservation d'une chambre à condition que cette chambre soit déjà réservée.

1.2 : Le système de l'hôtel confirme l'annulation de réservation de chambre.

1.3 : Le système de l'agent de voyage envoie une requête au système de compagnie aérienne pour annuler la réservation d'un vol à condition qu'il soit déjà réservé.

1.4 : Le système de Compagnie aérienne confirme l'annulation de réservation de vol.

2 : L'agent de voyage confirme l'annulation de réservation voulu par le client.

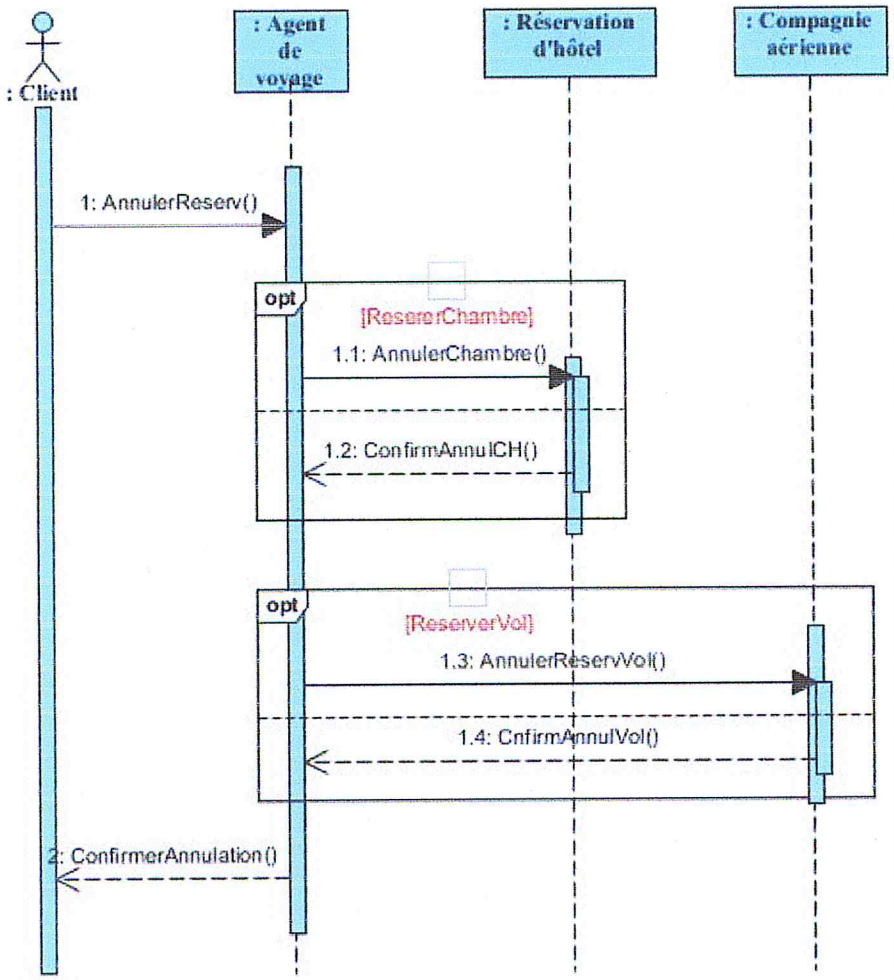

Figure IV.8: Diagramme de séquence de l'annulation d'un service.

4.2 Conception détaillée :

Dans cette section nous présentons le processus de modélisation des interfaces et les données qui sont transmises sur chaque interface ou bien dans un autre sens les modèles de boite blanche. Ces modèles évaluent la capacité d'un système pour interagir avec un autre.

La première étape pour comprendre et gérer l'interopérabilité au sein SOS TA est de comprendre l'information qui circule à travers chaque interface dans les deux cas (réservation, annulation) et le format des éléments de données qui font partie de cette information.

- **Réservation :**

Les figures IV.9, IV.10 et IV.11 montrent le schéma d'interface pour l'agent de voyage, les diagrammes pour les systèmes constitutifs participant au cas d'utilisation de réservation d'un service, et les entités d'E/S associées à chaque système.

La figure IV.9 représente l'interface de l'agent de voyage lors d'une réservation, cette interface reçoit des informations sous forme de paramètres d'opérations pour les différents systèmes.

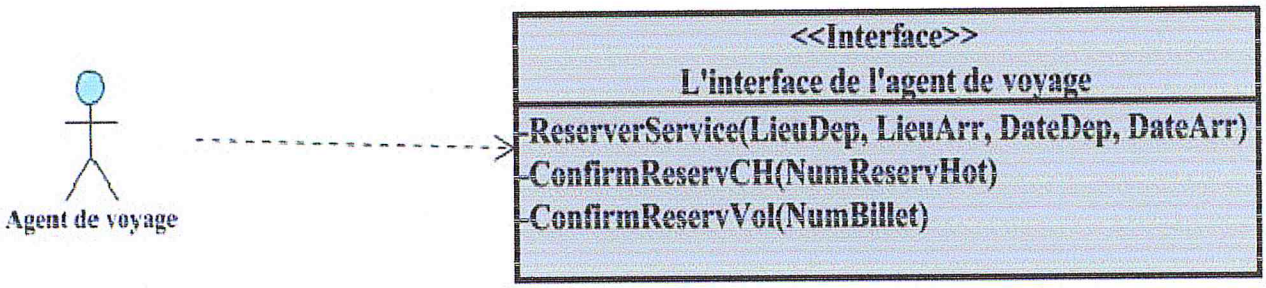

Figure IV.9: Interface de l'agent de voyage lors d'une réservation d'un service.

La figure IV.10 représente l'entité du système Compagnie aérienne avec l'opération RéserverVol() et les différentes informations échangées sous forme d'entités Entrée / Sortie. Nous avons comme entrées : NumComp, NomComp, LieuDep, LieuArr, DateDep, DateArr, et on trouve comme sortie : NumBillet.

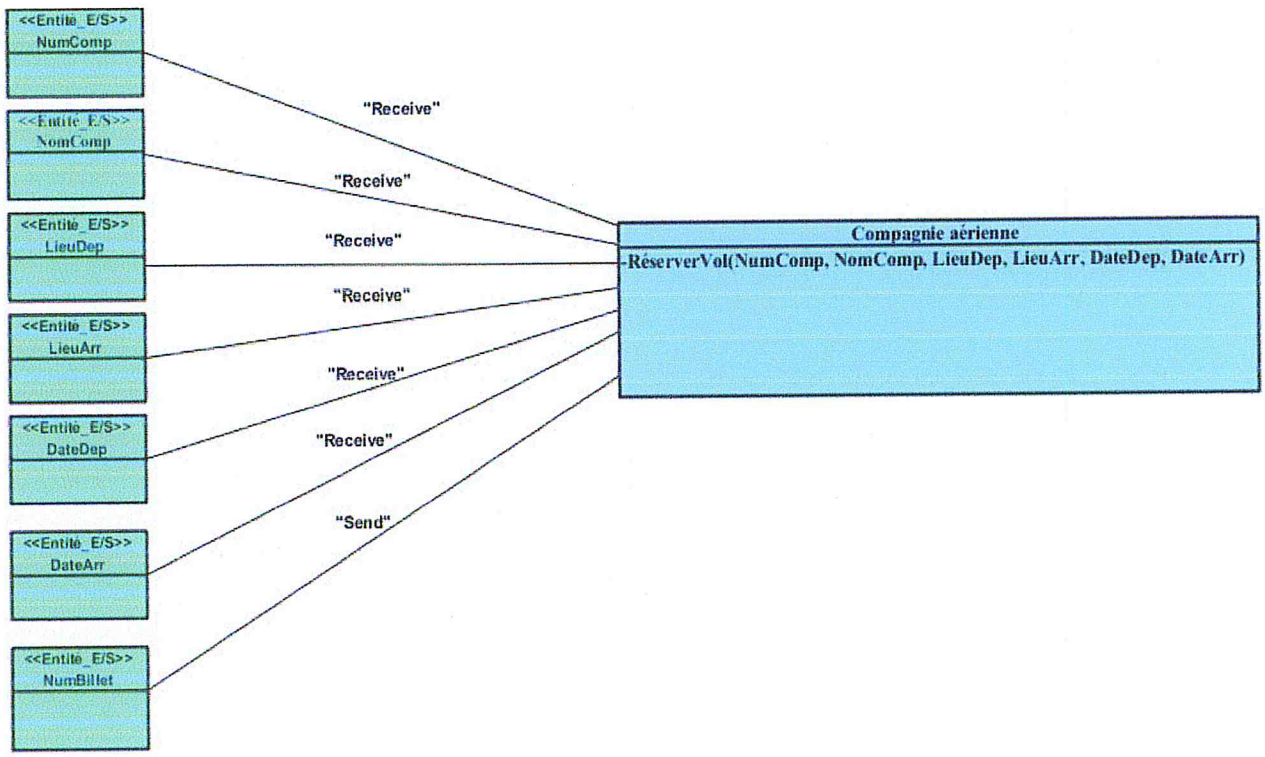

Figure IV.10: Entité de Compagnie aérienne dans le cas d'une réservation de service et les entités d'E/S associées.

La figure IV.11 représente l'entité du système Réservation d'hôtel avec l'opération ReserverChambre() et les différentes informations échangées sous forme des entités Entrée / Sortie tel qu'on a comme entrée : NumHot, NomHot, AdrHot, DateDeb, DateFin, et on trouve comme sortie : NumReservHot.

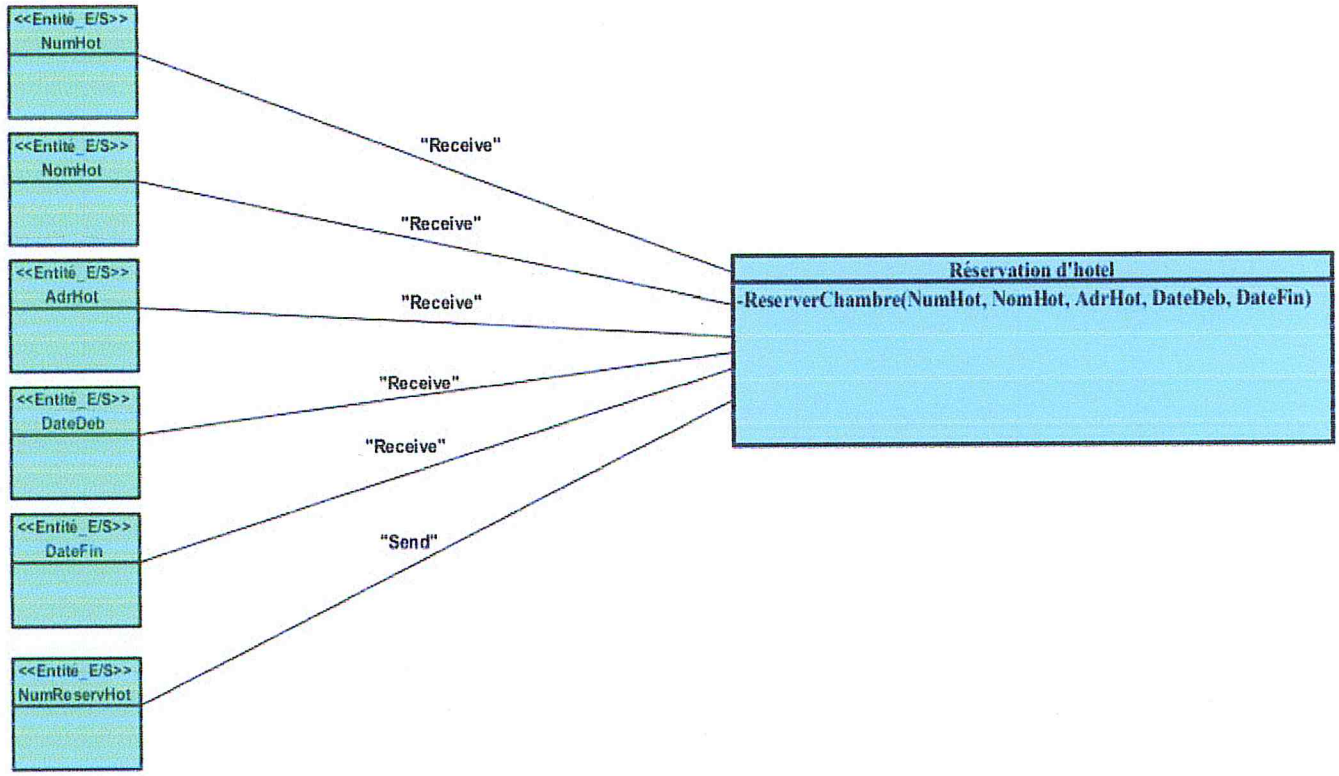

Figure IV.11: Entité de Réservation d'hôtel dans le cas d'une réservation de service et les entités d'E/S associées.

❖ **Annulation :**

Les figures IV.12, IV.13 et **IV.14** montrent le schéma d'interface pour l'agent de voyage, les diagrammes pour les systèmes constitutifs participant au cas d'utilisation d'une annulation, et les entités d'E/S associées à chaque système.

La figure IV.12 représente l'interface de l'agent de voyage lors d'annulation, cette interface reçoit des informations sous forme de paramètres d'opérations pour les différents systèmes.

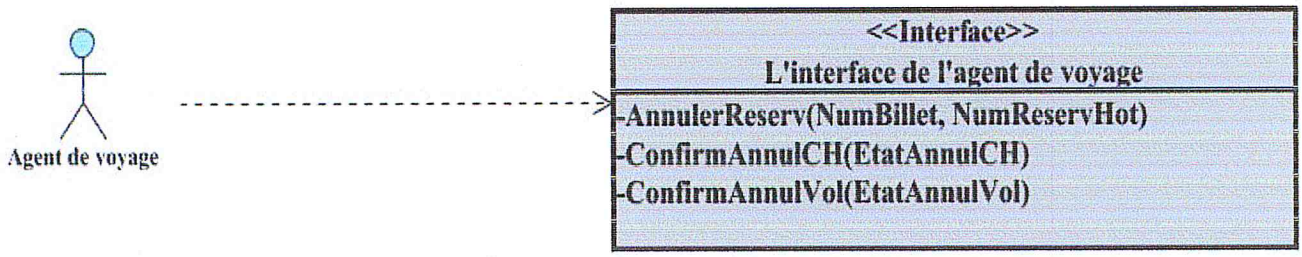

Figure IV.12: Interface de l'agent de voyage lors d'annulation d'une réservation.

La figure iV.13 représente l'entité du système Compagnie aérienne avec l'opération AnnulRéserverVol() et les différentes informations échangées sous forme des entités Entrée / Sortie. Nous avons comme entrées : NumBillet, et on trouve comme sortie : EtatAnnulVol.

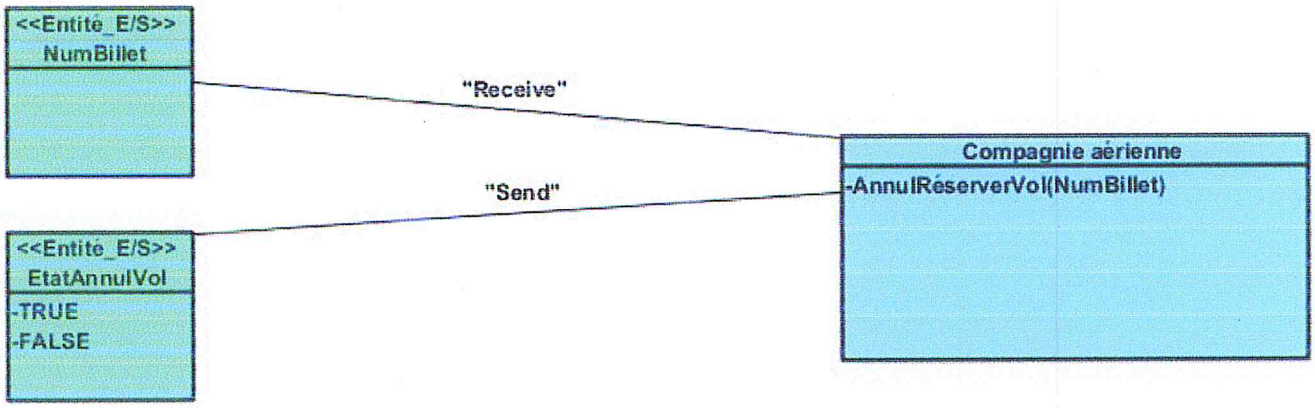

Figure IV.13: Entité de Compagnie aérienne dans le cas d'une annulation d'une réservation et les entités d'E/S associées.

La figure IV.14 représente l'entité du système Réservation d'hôtel avec l'opération AnnulerChambre() et les différentes informations échangées sous forme des entités Entrée / Sortie tel qu'on a comme entrée : NumReservHot, et on trouve comme sortie : EtatAnnuleCh.

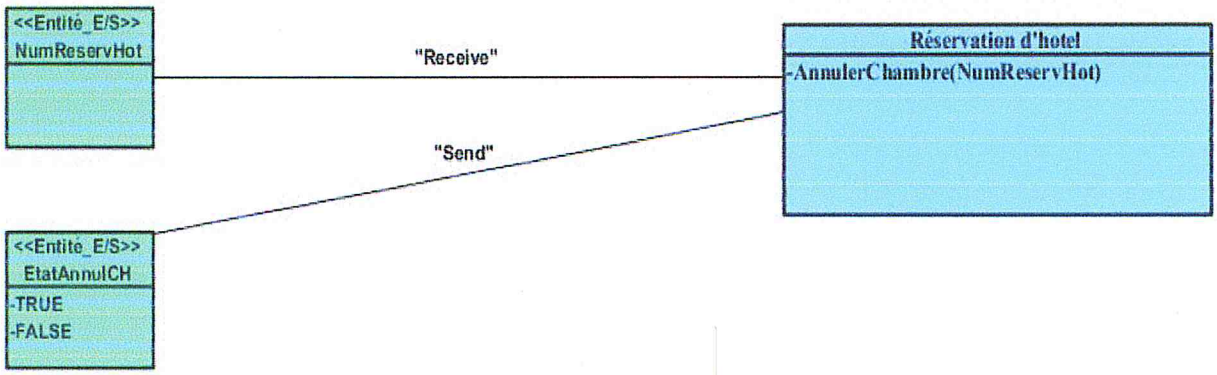

Figure IV.14: Entité de Réservation d'hôtel dans le cas d'une annulation d'une réservation et les entités d'E/S associées.

V. Conclusion :

Dans ce chapitre nous avons présenté une conception détaillée de notre système de systèmes basé sur l'architecture orienté service (SOA) en utilisant le langage UML, cette étude conceptuelle nous a permet de mettre en évidence les étapes nécessaires pour la création d'un système de système.

La prochaine étape consiste donc en la réalisation d'un système de système basé sur les outils que nous utilisons pour l'implémenter.

CHAPITRE V

Implémentation

I. Introduction

Après l'expression des besoins et la modélisation du système de systèmes à développer, nous allons dans cette partie présenter son implémentation et sa mise en œuvre. En première partie, nous présentons l'environnement de travail, les outils de développement utilisées, Ensuite nous présentons notre application.

II. Environnement de travail

1. Langages utilisés :

Le choix du langage de programmation représente une étape très importante dans la réalisation de n'importe quelle application.

Pour la réalisation de notre projet nous avons utilisé :

1.1 Le langage de programmation JAVA :

Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton de Sun Microsystems.

Il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitation (Windows, Linux, Macintosh, Solaris). [67]

Nous avons choisi le langage JAVA pour les raisons suivantes :

- Un langage orienté objet qui met à la disposition du développeur plusieurs paquetages prêt à l'utilisation (javax.swing, java.io....etc).
- Offre une encapsulation, la généricité et la réutilisation de notre code métier plus facilement car il offre beaucoup de souplesse.
- La disponibilité de la documentation et de l'assistance (forums).
- Java est un langage à haute sécurité.
- Java est un langage simple.
- Java est portable. [68]
- On peut l'utiliser avec BPEL et SOA.

1.2 Extensible Markup Language XML :

Extensible Markup Language (XML) est un format de texte simple, très souple dérivé de SGML (ISO 8879). Conçu à l'origine pour répondre aux défis de l'édition électronique à grande échelle,

XML joue également un rôle de plus en plus important dans l'échange d'une grande variété de données sur le Web et ailleurs. [69]

2. OUTILS

Dans notre application SOA, nous avons utilisé l'environnement Netbeans, ainsi que le serveur Glassfish que nous présentons ci-dessous.

2.1 NetBeans 6.7.1 :

Nous avons utilisé l'éditeur de NetBeans avec la version 6.7.1 (www.netbeans.org) pour la réalisation de notre implémentation.
NetBeans IDE prend en charge les normes de services Web à partir de la plus récente Java EE 7 et Java EE 6 du cahier des charges et aussi la plus ancienne de Java EE 5 et la Spécification précédente.
On peut créer et consommer des services Web utilisant des fonctionnalités telles que l'Assistant base de génération de code et à des insertions de code de l'éditeur.
NetBeans IDE rendent facile à créer sur la base des projets d'application Web Java EE avec JSF 2.2 soit (Facelets), JSP ou servlets. L'éditeur prend en charge la complétion de code, la navigation et la refactorisation pour les fichiers de mappage. [70]

2.2 GlassFish V2.1 :

GlassFish est un projet de serveur d'application Java créée par Sun Microsystems qui permet à plusieurs développeurs de générer des technologies de l'entreprise qui sont pratique et évolutive, ainsi que des services supplémentaires qui peuvent être installés en fonction des préférences. C'est un logiciel libre, sous une double licence en vertu de la Licence publique générale GNU (GPL) et le développement commun et de la licence de distribution (CDDL). GlassFish a été acquis par Oracle en 2010
GlassFish a été développé sur la base d'un code source qui a été publié par Sun et le système de persistance TopLink d'Oracle. Le projet a été lancé en 2005 et la première version qui a soutenu Java EE 5 est sortie en 2006.

L'implémentation de référence de Java EE est GlassFish, il prend en charge JMS (*Java Message Service*), JSP (*Java Server Pages*), Enterprise JavaBeans, RMI (*Remote Methode Invocation*), et les servlets. En raison de sa nature, les développeurs peuvent créer des applications évolutives et portables qui s'intègrent facilement aux systèmes et technologies existants. [71]

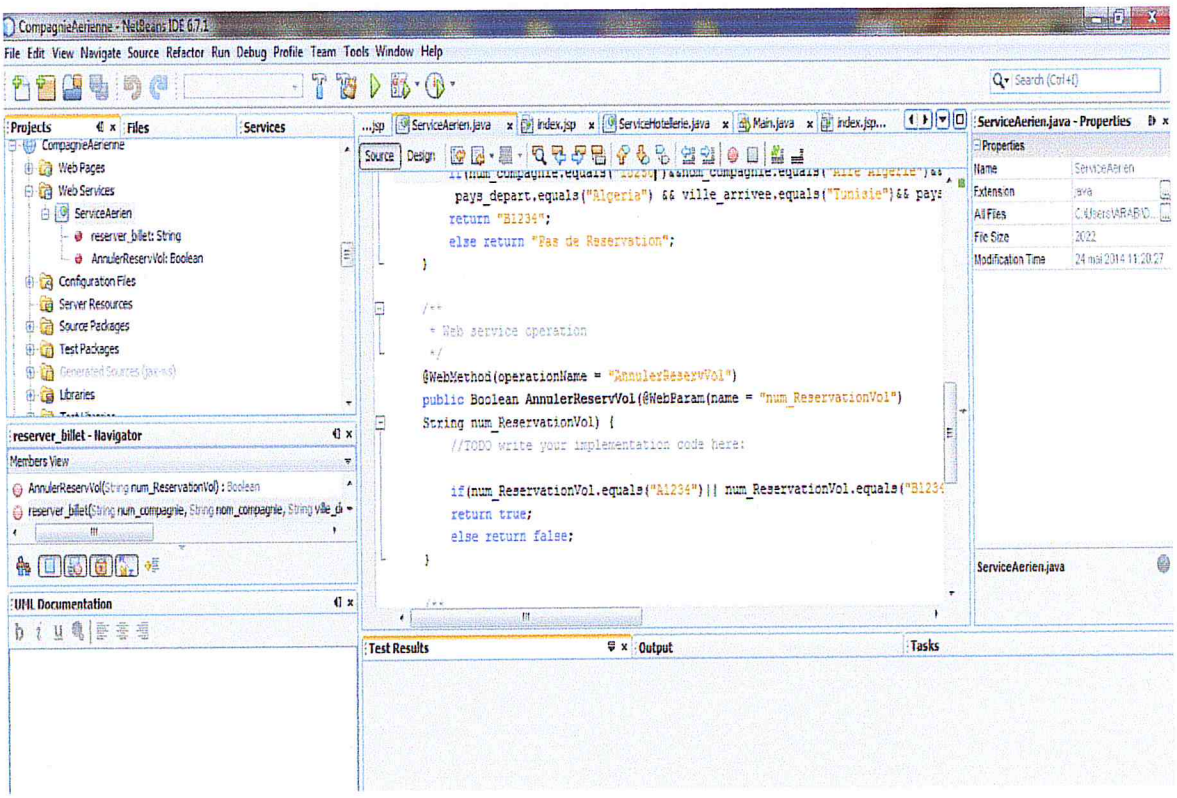

Figure V.1: Interface de l'éditeur NetBeans.

2.3 JBI, OpenESB, et l'outillage SOA NetBeans :

2.3.1 Java Business Integration (JBI) « L'architecture » :

Java Business Integration (JBI) est une norme d'intégration d'entreprise basée sur Java définie par le groupe d'experts de la JSR 208. JBI définit une plate-forme extensible, ouverte et *pluggable* qui permet la collaboration de la technologie d'intégration (tels que les moteurs des processus métiers, les moteurs de transformation des documents, gestion B2B et règles de gestion) avec des

services Web. Les éléments clés de l'architecture JBI, montré dans la figure suivante, comprennent:

- **Service Engine (SE) :**

SE exécute la logique métier dans le run-time de JBI. SE est souvent mis en œuvre comme un type de conteneur, pour que les artifacts peuvent être déployés. Ces artifacts précisent la logique métier dans des langages comme BPEL, XSLT, SQL et Java.

- **Binding Component (BC) :**

Le BC gèrent les interfaces externes dans JBI. Les BCs sont souvent mis en œuvre comme un type de conteneur, à laquelle les artifacts peuvent être déployés. Ces artifacts précisent le protocole de configuration de traitement / transport défini à l'aide de WSDL avec des extensions personnalisées, par exemple : SOAP / HTTP, fichier, JMS, SMTP, etc.

- **Service Assembly (SA) :**

Le SA est le package de déploiement standard pour une application SOA. Il est comme un fichier WAR pour le JBI run time. Il contient des Artifacts pour des composants d'exécution de SE et de BC. Il contient également des métadonnées décrivant le routage, les connexions et les services qui composent l'application SOA.

- **Service Unit (SU) :**

SU est l'artifact de déploiement standard pour une SE ou une BC. Il contient la logique métier, les définitions WSDL, et les métadonnées décrivant les paramètres qui composent le composant SOA. Chaque SU est un fichier jar contenant des objets spécifiques SE. [72]

2.3.2 OpenESB «La mise en œuvre » :

Projet Open ESB met en œuvre un run-time d'Enterprise Service Bus (ESB) en étendant la mise en œuvre de référence de JBI avec les moteurs de service supplémentaires, les composants, les outils et la gestion de liaison et des services de surveillance. Cela permet une intégration facile

des services web pour créer des applications composites de classe entreprise qui sont faiblement couplées. Quelque composants SE et BC :

Service Engines **:** BPEL SE, Encoding SE, SQL SE, XSLT SE..etc.

Binding Components **:** File BC, FTP BC, HTTP BC, JDBC BC, ...etc. **[72]**

2.3.3 NetBeans SOA « Les outils »:

Les outils NetBeans SOA facilitent le développement d'applications SOA pour OpenESB. Il ya trois couches d'outils SOA fournit par NetBeans IDE. Ces couches d'outils sont les suivants:

- **NetBeans Editors :**

Les outils NetBeans SOA fournit de nombreux éditeurs pour spécifier la logique métier, par exemple, BPEL, XSLT, et SQL, et les définitions de services Web, par exemple, WSDL, XSD et XML.

Ces éditeurs sont utilisés pour construire des objets de données nécessaires à des *artifacts* de l'unité de *service assembly*.

- **Les projets de *Service Unit (SU)* :**

Les outils NetBeans SOA fournit des types de projet de *service unit* spéciaux pour de nombreux moteurs de service OpenESB, par exemple, BPEL, XSLT, et SQL. Chaque type de projet SU contient des scripts personnalisé pour la construction de l'unité de service dans un format requis par type spécifique de moteur de service.

- **Les projets d'Application composite (CompApp) :**

Les outils NetBeans SOA fournit des types de projet de l'application composite pour définir des services assemblés d'OpenESB. Un projet « *CompApp* » peut lier à zéro ou plusieurs projets SU, et générer des *artifacts* de déploiement supplémentaires nécessaires pour construire un service Assemblée (SA) complet pour le déploiement. **[72]**

3. Présentation de l'application :

3.1 Le Processus BPEL :

L'application consiste en des processus BPEL de réservation et d'annulation orchestrant l'ensemble des services hôtellerie et aérien pour une agence de voyage.

Nous considérons que les paramètres d'entrée relatifs au client de ce processus sont :

La date de départ, la date d'arrivée, la ville de départ et celle d'arrivée. Quant au paramètre de sortie de ce processus est sous la forme d'une offre finale informant le client sur les détails de son voyage.

L'agence fournit leurs services en utilisant des services web, il est alors possible de composer le service web de l'application web compagnie aérienne et de l'application web réservation d'hôtel.

Chaque application web doit contenir au moins un service web et chaque service web doit contenir au moins une opération.

La **Figure V.2** montre un exemple d'une application web qui contient un service web.

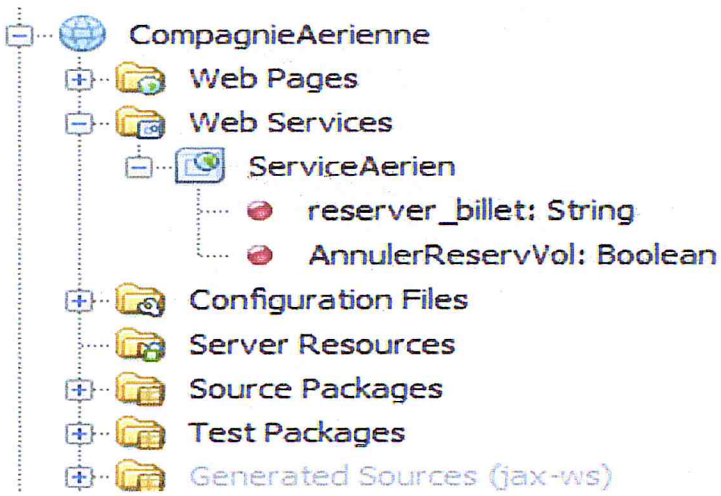

Figure V.2 : Exemple d'un projet application web.

En effet, le processus BPEL invoque le service aérien et le service hôtellerie.

La Figure V.3 montre le code de base du processus de Réservation, qui implémente l'opération «ReservationOperation».

Nous allons juste présenter des simples invocations élémentaires aux services web partenaires. Pour ce faire, le processus déclare trois liens des partenaires qui connectent la composition

respectivement au client, au service web de réservation d'un billet et au service web de réservation d'une chambre d'un hôtel. Il déclare aussi des variables pour supporter les messages requête et réponse du client, de même ceux relatifs aux invocations des opérations «ReserverBillet» et «ReserverChambre» des services web correspondants. Ensuite, nous trouvons l'activité « receive » qui relie l'opération «ReservationOperation» à une activité « assign » pour copier les données.

En plus, les activités « invoke » suivantes appellent le service aérien et le service hôtellerie.

L'activité « reply » envoie l'offre finale au client.

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="agence_De_Voyage"
    targetNamespace="http://enterprise.netbeans.org/bpel/Agence_De_Voyage/agence_De_Voyage"
    xmlns:tns="http://enterprise.netbeans.org/bpel/Agence_De_Voyage/agence_De_Voyage"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Trace"
    xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Editor"
    xmlns:sxeh="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/ErrorHandling" xmlns:sxed2="http://www.sun.com/wsbpel/2.0/process/executable/S
    <import namespace="http://j2ee.netbeans.org/wsdl/Agence_De_Voyage/Reservation" location="Reservation.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"
    <import namespace="http://enterprise.netbeans.org/bpel/ReserverBilletServiceWrapper" location="ReserverBilletServiceWrapper.wsdl" importType="http://sch
    <import namespace="http://ReserverBillet/" location="ReserverBilletService.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/ServiceHotellerieServiceWrapper" location="ServiceHotellerieServiceWrapper.wsdl" importType="http
    <import namespace="http://ServiceHotellerie/" location="ServiceHotellerieService.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <partnerLinks>
        <partnerLink name="ServiceAerien" xmlns:tns="http://enterprise.netbeans.org/bpel/ReserverBilletServiceWrapper" partnerLinkType="tns:ReserverBilletLi
        <partnerLink name="ServiceHotellerie" xmlns:tns="http://enterprise.netbeans.org/bpel/ServiceHotellerieServiceWrapper" partnerLinkType="tns:ServiceHo
        <partnerLink name="PartnerLink1" xmlns:tns="http://j2ee.netbeans.org/wsdl/Agence_De_Voyage/Reservation" partnerLinkType="tns:Reservation" myRole="Re
    </partnerLinks>
    <variables>
        <variable name="ReserverChambreOut1" xmlns:tns="http://ServiceHotellerie/" messageType="tns:ReserverChambreResponse"/>
        <variable name="ReserverChambreIn1" xmlns:tns="http://ServiceHotellerie/" messageType="tns:ReserverChambre"/>
        <variable name="ReserverBilletOut1" xmlns:tns="http://ReserverBillet/" messageType="tns:ReserverBilletResponse"/>
        <variable name="ReserverBilletIn1" xmlns:tns="http://ReserverBillet/" messageType="tns:ReserverBillet"/>
        <variable name="ReserverChambreOut" xmlns:tns="http://ServiceHotellerie/" messageType="tns:ReserverChambreResponse"/>
        <variable name="ReserverChambreIn" xmlns:tns="http://ServiceHotellerie/" messageType="tns:ReserverChambre"/>
        <variable name="ReserverBilletOut" xmlns:tns="http://ReserverBillet/" messageType="tns:ReserverBilletResponse"/>
        <variable name="ReserverBilletIn" xmlns:tns="http://ReserverBillet/" messageType="tns:ReserverBillet"/>
```

```
        <variable name="ReservationOperationIn" xmlns:tns="http://j2ee.netbeans.org/wsdl/Agence_De_Voyage/Reservation" messageType="tns:ReservationOperation
    </variables>
    <sequence>
        <receive name="Receive1" createInstance="yes" partnerLink="PartnerLink1" operation="ReservationOperation" xmlns:tns="http://j2ee.netbeans.org/wsdl/
        <if>
        <reply name="Reply1" partnerLink="PartnerLink1" operation="ReservationOperation" xmlns:tns="http://j2ee.netbeans.org/wsdl/Agence_De_Voyage/Reservati
    </sequence>
</process>
```

Figure V.3: Extrait du code de base du processus BPEL de l'agence de voyage

La **Figure V.4** illustre graphiquement cette description plus clairement.

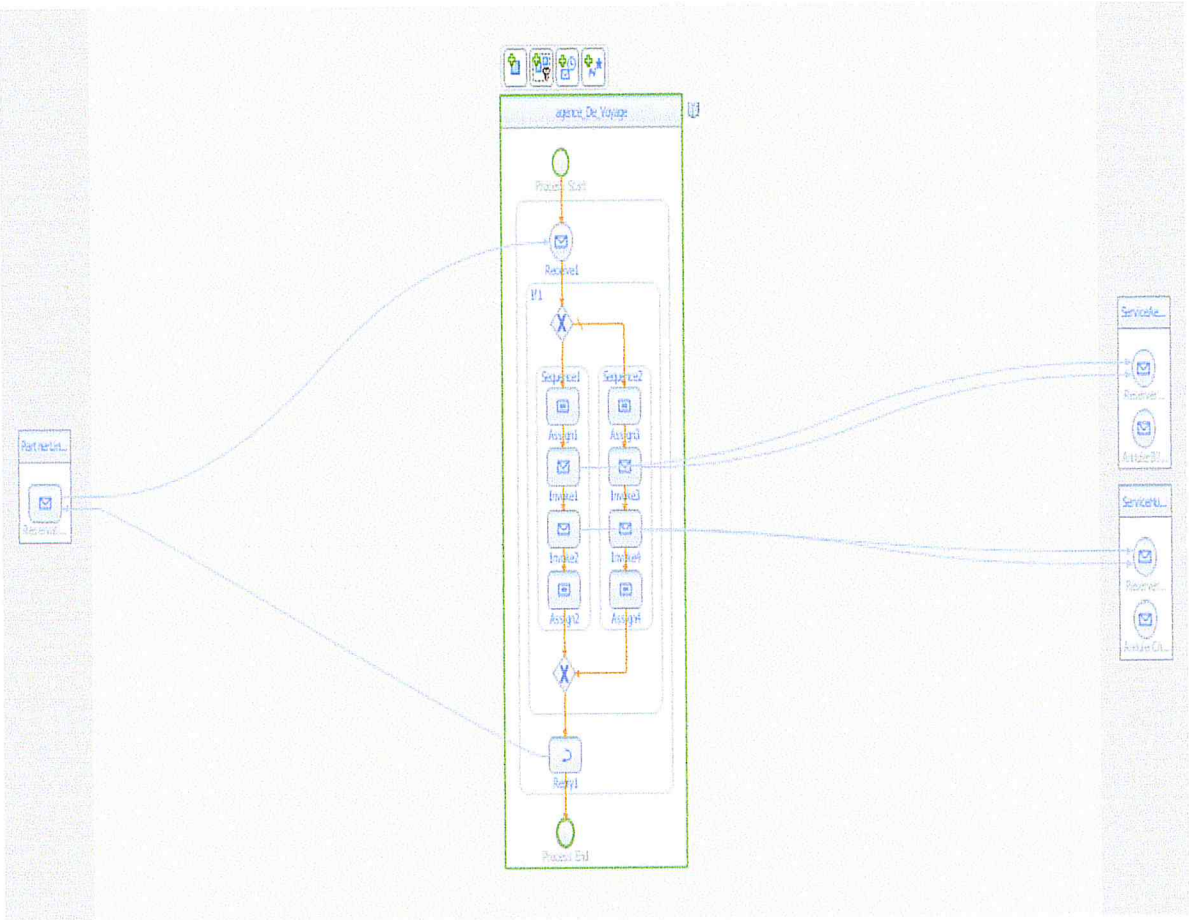

Figure V.4 : Illustration graphique du processus BPEL

3.2 L'application Composite

Un projet BPEL n'est pas directement déployable.

Il doit être rajouté avec les applications web en tant que des modules JBI dans un projet "Application Composite" qui sera ensuite déployée.

La **Figure V.5** illustre les modules JBI de notre application.

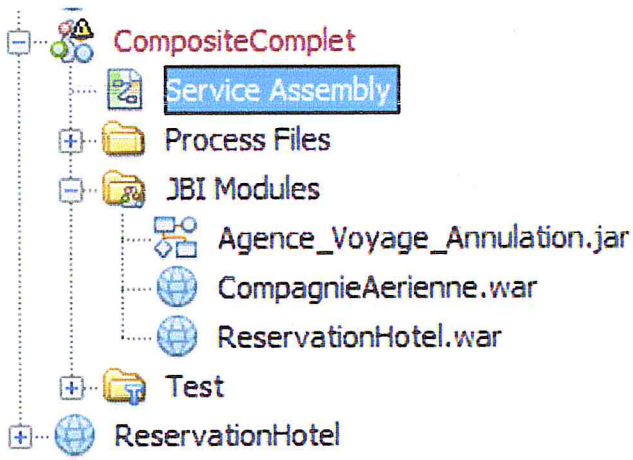

Figure V.5: les modules JBI de l'application composite de l'agence de voyage

On peut voir l'assemblage des différents services de l'application en utilisant le Service Assembly. La fenêtre est divisée en 3 parties: une pour les ports WSDL, une pour les modules JBI, et une autre pour les modules externes (voir la **Figure V.6**).

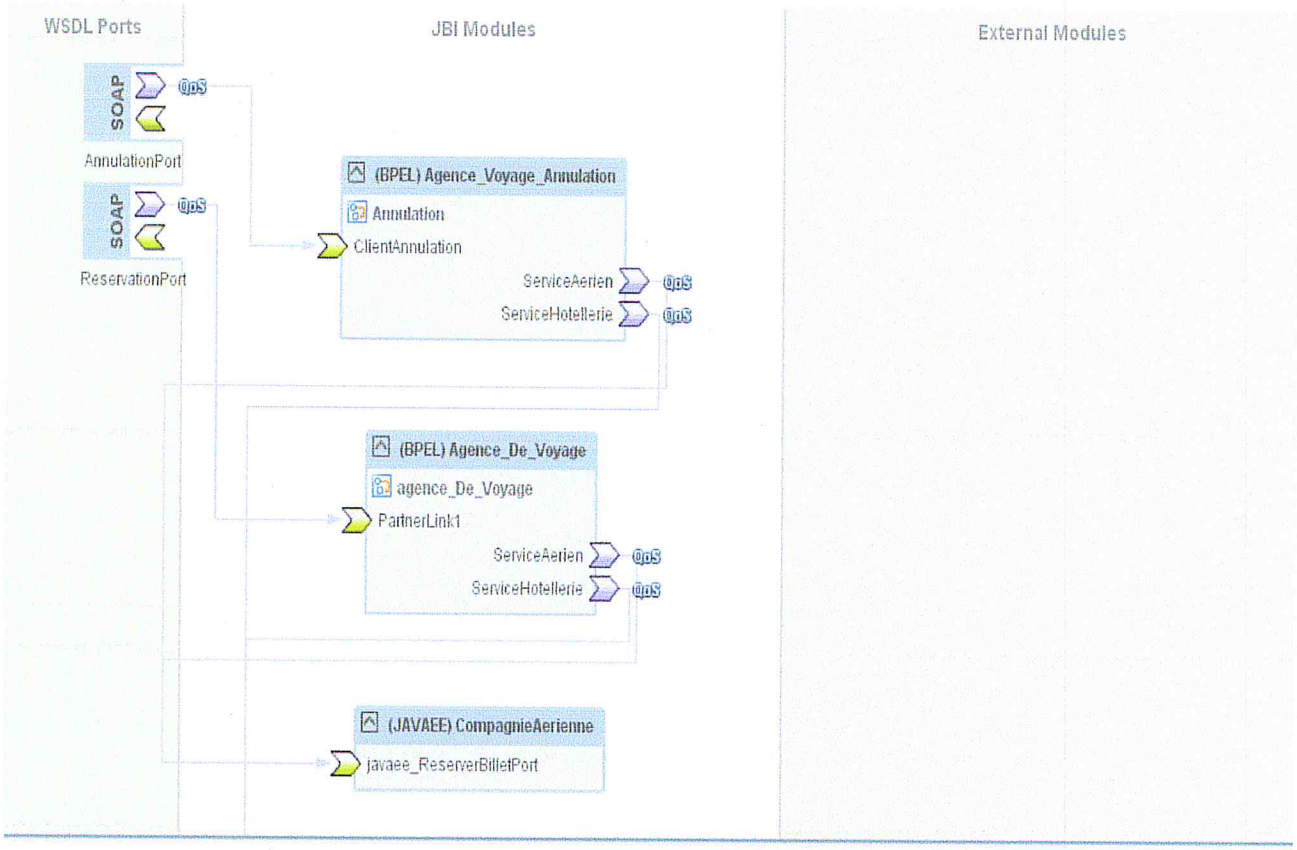

Figure V.6 : l'application composite de l'agence de voyage

Pour déployer notre application composite on utilise le serveur Glassfish V2.1 (voir la **Figure V.7**).

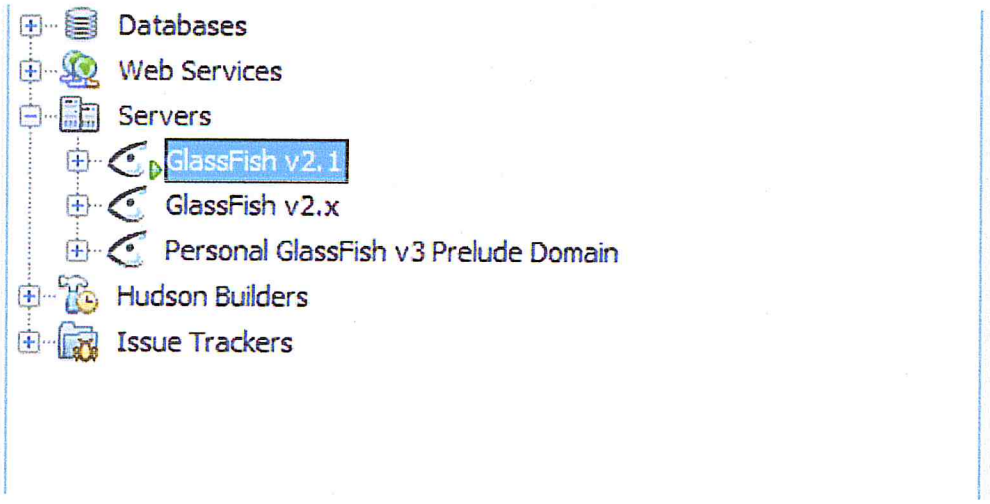

Figure V.7: Le serveur GlasFish V2.1

3.3 Le test de l'application composite

Une fois notre application composite déployée, il est possible de la tester.

Dans le répertoire Test on doit choisir un nouveau « Test Case », pour le processus BPEL (Réservation ou Annulation).

Notre test contient deux fichiers XML qui sont : le fichier input et le fichier output, on doit remplir le fichier input par les informations du client (Réservation ou Annulation).

Si on lance le test pour la première fois, une fenêtre d'erreur va apparaître (voir la **Figure V.8**).

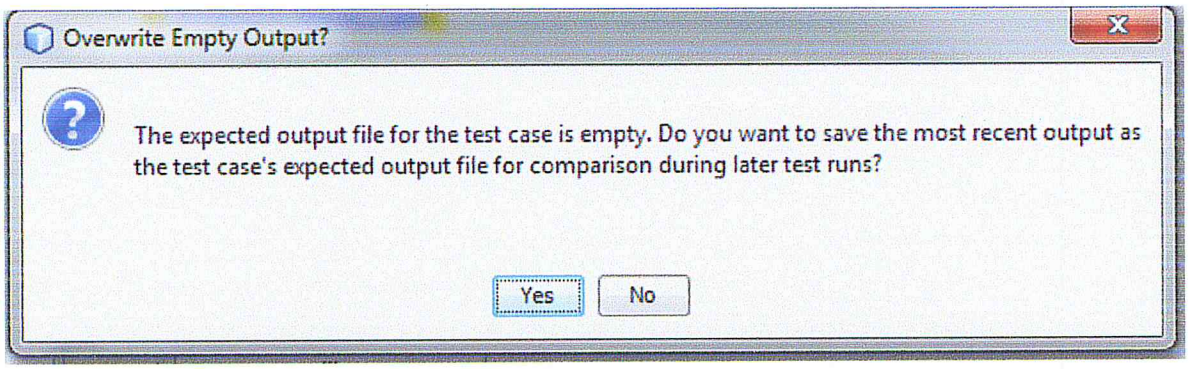

Figure V.8 : La fenêtre d'erreur

Cette fenêtre indique que le test n'est pas réussi, car la sortie obtenue ne correspond pas à la sortie attendue. En effet, le fichier output, étant resté vide, n'a pas montré le résultat correct que nous attendions. En cliquant sur « Yes », ce fichier sera rempli avec le résultat obtenu, et une deuxième exécution du test devra indiquer que le test est réussi (voir la **Figure V.9**). Le fichier Output retourné est illustré dans **la figure V.10**

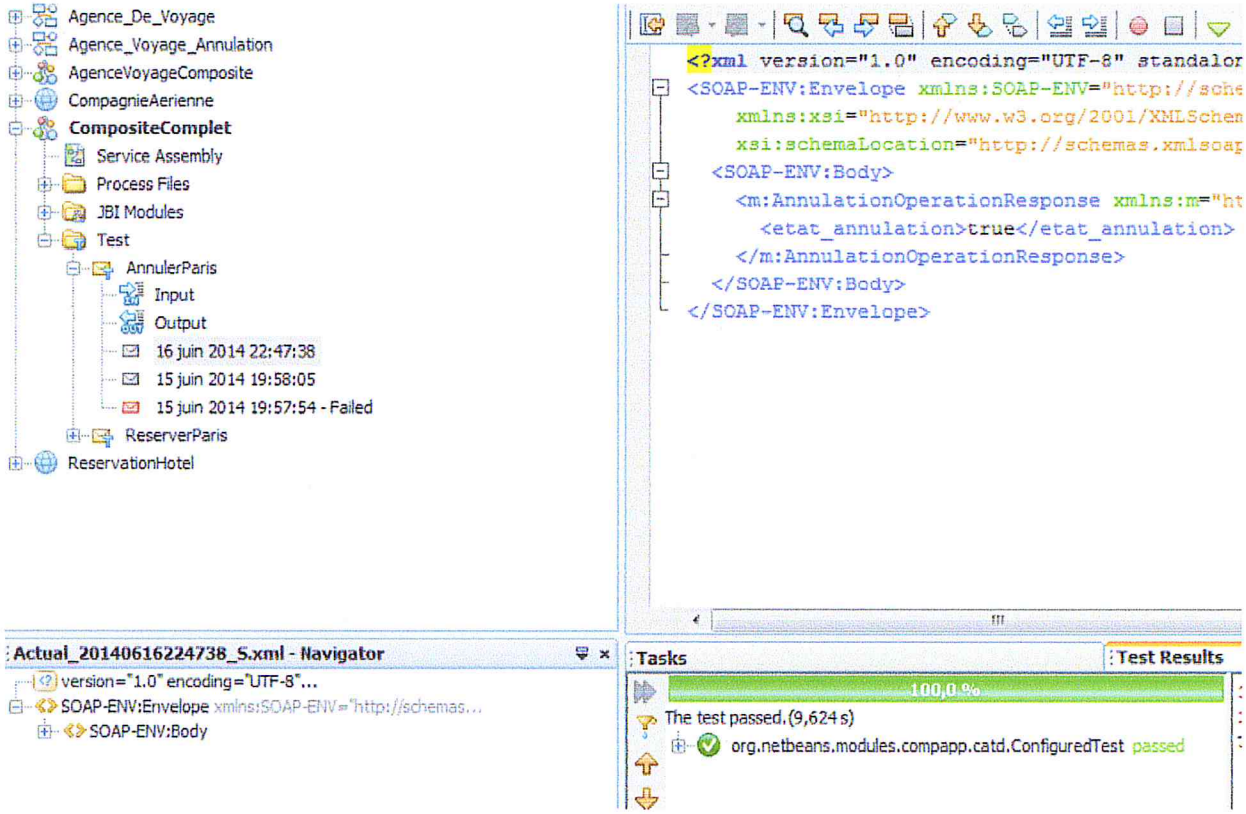

Figure V.9 : Application composite : Test Réussi

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:AnnulationOperationResponse xmlns:m="http://j2ee.netbeans.org/wsdl/Agence_Voyage_Annulation/Annulation">
      <etat_annulation>true</etat_annulation>
    </m:AnnulationOperationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure V.10 : Exemple d'un fichier output avec le résultat obtenu

III. Conclusion :

Dans ce chapitre, nous avons présenté l'éditeur qu'on a utilisé et on a exposé l'environnement de développement choisi (langage de programmation) pour notre système (système de système), dans le quel repose notre application.

Enfin, nous avons présenté la vue réelle de notre application, et des principaux modules qui la font fonctionner.

CONCLUSION GENERALE

Conclusion générale:

Les travaux présentés dans ce mémoire se situent dans le domaine des architecture logicielle, ils portent exactement le volet de spécification des Systèmes de Systèmes.

Les Systèmes de Systèmes sont une approche de supervision et de contrôle global ou les systèmes constituants sont caractérisés comme des sous-systèmes du SoS.

Le travail qui nous a été demandé consiste à trouver une approche efficace pour la spécification et la réalisation d'un SoS.

Afin d'élaborer ce projet, nous avons commencé le travail par rechercher les travaux qui ont été proposés en architecture logicielle, et qui peuvent être utilisés dans le contexte des SoS. Les résultats trouvés se divisent entre deux grands paradigmes d'architecture logicielle qui sont l'orienté composants, et l'orienté service. Après avoir mené une étude comparative, nous avons décidé d'opter pour une approche orientée service, utilisant l'orchestration de BPEL.

Afin de pouvoir réaliser un SoS. Nous avons choisi un exemple de SoS comprenant deux sous-systèmes, et nous avons commencé par la définition des besoins, en faisant la collecte d'information nécessaire à notre étude préalable, ensuite nous avons entamé l'étude conceptuelle suivant le modèle en V, qui traite les différentes phases à savoir la conception détaillée, l'implémentation, pour aboutir enfin à la réalisation de l'application.

Notre projet de fin d'études nous a été d'un grand apport sur plusieurs plans, principalement le plan conception, Nous avons appris l'aspect essentiel dans les applications qui se basent sur l'architecture orienté service. Sur le plan technique nous avons acquis une certaine expérience dans le domaine de développement par les technologies suivantes : Java, SOA.

L'apport que nous avons essayé d'apporter au domaine de Système de Systèmes, qui est considéré comme nouveau, est très petit, ce domaine est très vaste et peut être amélioré, corrigé ou complété.

La solution qu'on a présenté et le projet réalisé dans ce mémoire n'est en réalité qu'une ouverture vers d'autre travaux car notre système de systèmes peut encore évoluer.

Plusieurs perspectives pour ce travail sont à envisager :

- L'augmentation du niveau de complexité de l'architecture du SoS (Système composé plusieurs services Web).
- L'utilisation du BPML comme un langage d'orchestration de services Web.

BIBLIOGRAPHIE

Bibliographie :

[1] Pei, R. S., Systems of systems integration (SoSI) a smart way of acquiring Army C4I2WS systems, *Proceedings of the Summer Computer Simulation Conference*, pp 134–139, 2000.

[2] Jamshidi, M., Theme of the IEEE SMC 2005, Waikoloa, Hawaii, USA. October 2005.

[3] Carlock, P. G. and R. E. Fenton., System of systems (SoS) enterprise systems for information-intensive organizations. *Systems Engineering* 4(4):242–261, 2001.

[4] Stanley N. H., Modeling and Simulation for Systems and System-of-Systems Engineering, November 2013.

[5] Mary A. A., Ron B., Hao C., Hòa G., Harcharanjit S., Steve T., NECSI: Complex Physical, Biological and Social Systems Project, the Characteristics and Emerging Behaviors of System of Systems, 2004.

[6] MAIER M.W., "Architecting principles for systems-of-systems", *6th Annual International Symposium of the International Council of System Engineering*, Boston, 1996.

[7] SAGE A.P., CUPPAN C.D., "On the systems engineering and management of systems of systems and federations of systems", *Information, Knowledge, Systems Management*, vol. 2, p. 325-345, 2001.

[8] Dominique L, Jean-René R, Systems of Systems, 2010.

[9] Jamshidi M., SYSTEMS OF SYSTEMS ENGINEERING Principles and Applications, 2009.

[10] Simon P. , Jon H., Richard P., Claire I., Alvaro M., Finn O., Luis D., Stefan H., Anders K., Juliano I., Marcio C., Jan P., COMPASS project, Comprehensive Modelling for Advanced Systems of Systems, Report on Modeling Patterns for SoS Architectures (D22.3) , 2013.

[11] Al shabani I., Un framework pour les composants logiciels distribués et parallèles, (thèse doctorat), Université des Sciences et Technologies de Lille (France), décembre 2006.

[12] Yann V., Modèles de composants et Java Beans, Université de Versailles St-Quentin-en-Yvelines, France.

[13] Page d'accueil d'EJB : http://docs.oracle.com/javaee/5/tutorial/doc/bnblr.

[14] Zoran S., A Method for Component-Based and Service-Oriented Software Systems Engineering, (these doctoral), Delft University of Technology, 2005.

[15] Krakowiak S., Composants logiciels Exemples : Java Beans, Enterprise Java Beans, Université Joseph Fourier, Projet Sardes (INRIA et IMAG-LSR), 2005-2006.

[16] page officiel de SUN Microsystems : http://www.oracle.com/us/sun/.

[17] OMG, *CORBA 3.0 New Components Chapters*. Object Management Group, Novembre 2001.

[18] Projet ACCORD, Le modèle de composants CORBA, livrable 1.1-4, mai 2002.

[19] Adnane CABANI, Etude de la thèse de Philippe Merle CorbaScript, CorbaWeb Propositions pour l'accès à des objets et services distribués, Juin 2003.

[20] site officiel de Microsoft: http://www.microsoft.com.

[21] Yves L., Présentation de l'architecture COM component object model DCOM /ACTIVEX, Conseil Audit de systèmes d'information CISA, Avril 2003.

[22] site officiel de .NET : http://www.microsoft.com/net.

[23] Web Services Glossary available at: http://www.w3.org/TR/ws-gloss.

[24] Chaari S., Interconnexion des processus Interentreprises : une approche orientée services, Thèse de doctorat, Institut National des Sciences Appliquées, Lyon France, Décembre 2008.

[25] Banke, K. and Slama, D., *Enterprise SOA: Service-Oriented Architecture Best Practices* Prentice Hall, 2004.

[26] Eric A. Marks, Michael Bell, Service Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology, June 2006.

[27] Web Services, Service-Oriented Architectures, and Cloud Computing : http:// www. service-architecture.com/articles/web-services/serviceoriented architecture soa definition.html

[28] Kazi-Aoul Z. I., Une architecture orientée services pour la fourniture de documents multimédia composés adaptables, thèse doctorat, l'École Nationale Supérieure des Télécommunications Au département Informatiques et Réseaux,Paris France, Janvier 2008.

[29] Spécification de SOAP : http://www.w3.org/TR/soap/.

[30] Boukadi K., Coopération interentreprises à la demande :Une approche flexible à base de services adaptables (Thèse Doctorat), l'École Nationale Supérieure des Mines de Saint-Étienne , 2009.

[31] Clapié S., Renard A, , Les Services web, 2003 .

[32] le projet Apache Axis, la formation d'ingénieur à l'université Marne la Vallée, Paris : http://www-igm.univ-mlv.fr/~dr/XPOSE2003/axis_seng/soap.html.

[33] EL Ghers S., Architecture Logicielle (Support de cours), 2014.

[34] Page d'accueil de WSDL : http://www.w3.org/TR/wsdl.

[35] Marmel J., Francois A., Perlat D., Printemps O., SOAP Simple Object Access Protocol.

[36] Site officiel d'UDDI : http://www.uddi.org/.

[37] JAMAL S., Environnement de procédé extensible pour l'orchestration Application aux services web, l'Université Joseph Fourier de Grenoble, 2005.

[38] Céline Lopez-Velasco, Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation, l'Université Joseph Fourier de Grenoble, 2008.

[39] Peltz C., Web Services Orchestration and Choreography. IEEE Computer, 2003, vol.36, n°10, pp.46-52.

[40] Benatallah B., Dijkman R., Dumas M., Maamar Z. Service Composition: Concepts, Techniques, Tools and Trends. **In:** Stojanovic Z., Dahanayake A., Eds. Service Oriented Software Engineering: Challenges and Practices. Idea Group Inc (IGI), 2005, pp.48-66.

[41] Barros A., Dumas, M., Oaks, P. Standards for Web Service Choreography and Orchestration: Status and Perspectives. **In:** Procs of the 3rd International Conference the Business Process Management (BPM 2005), 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management, Nancy, France, Sept. 2005, pp.1-15.

[42] Page d'accueil de WSCI : http://www.w3.org/TR/wsci.

[43] site officiel d'Intalio **:** http://www.intalio.com/

[44] site officiel de SAP : http://www.sap.com/

[45] site officiel de BEA : http://fr.bea.com/

[46] Page d'accueil de WSCL : http://www.w3.org/TR/2002/NOTE-wscl10-20020314/

[47] site officiel de W3C : http://www.w3.org/

[48] Page d'accueil de XLANG : http://msdn.microsoft.com/enus/library/aa577463

[49] Page d'accueil de WSFL : http://xml.coverpages.org/wsfl.html

[50] Page d'accueil de BPEL : http://www.128.ibm.com/developerworks/library/specification/ws-bpel.

[51] site officiel d'IBM : http://www.ibm.com/.

[52] site officiel des recherches de Microsoft : http://research.microsoft.com.

[53] SELLAMI W., Une approche formelle pour la vérification des propriétés non fonctionnelles d'orchestration des services web, Université de Sfax, 2011.

[54] DRISS M., Approche multi-perspective centrée exigences de composition de services Web, 2011.

[55] page d'accueil d'Oracle BPEL Process Manager : http://www.oracle.com/ technology/products /ias/bpel/index.html.

[56] page d'accueil de Biztalk : http://microsoft.com/biztalk/.

[57] Site officiel d'Active bpel : http://activebpel.org.

[58] Site officiel d'Apache Agila : http://wiki.apache.org/agila/.

[59] Laurent Audibert, UML 2 - de l'apprentissage à la pratique, 2006.

[60] LECLERC P,. Conception Informatique, www.conception-informatique.com/cycle-en-v

[61] V Model :http://en.wikipedia.org/wiki/V_Model%28software_development%29#Module design.

[62] Dependable Systems of Systems, Case Study Problem Analysis (DMS3).

[63] KRELIFA Halla, ZENILE Amina, Intégration des sources de métadonnées hétérogènes: XML, RDF et RuleML par l'approche médiateur, 2012.

[64] Daniel J. Epstein, Jo Ann Lane., System of Systems Capability to Requirements, 2012.

[65] Dr Stuart Burge, The Systems Engineering Tool Box, 2011.

[66] http://www.edrawsoft.com/uml-sequence.php.

[67] NADJI K., MEZZI M., Conception et réalisation d'un système de construction d'Ontologies par fusion des Ontologies existantes, Mémoire de Master, Université de SAAD Dahlab, Blida, Juillet 2011.

[68] METIDJI S., MECHIEKH F., Conception et réalisation d'un médiateur de métadonnées: XML, RDF, RuleML, Mémoire de Master, Université de SAAD Dahlab, Blida, Juillet 2011.

[69] Page d'accueil de XML : http://www.w3.org/XML/.

[70] Java EE and Web Application Development : https://netbeans.org/features/java-on-server/.

[71] Cory Janssen, Glassfish : http://www.techopedia.com/definition/27238/glassfish.

[72] Introduction to JBI, OpenESB, and NetBeans SOA Tooling : http://wiki.netbeans.org/SoaFaqJbiOpenESBSoaTooling.