1. Introduction Générale

Les web services sont devenus une technique incontournable pour construire des systèmes distribués faiblement couplés. L'architecture Orientée service a été largement employée dans plusieurs domaines tel que dans les systèmes e-business, e-gouvernement, systèmes automobiles, services multimédia, finances et dans beaucoup d'autres domaines.

Deux styles différents de web services ont été identifiés dans la littérature ; les Services web de types SOAP qui reposent sur le protocole SOAP (à base XML) pour assurer une communication hétérogène. Et les Services de type REST qui reposent sur le protocole Http.

La composition de service est une méthode de construction de logiciels en utilisant les services déjà établis. Par composition, on peut construire des services plus complexes avec des entités logiciels faiblement couplées. Souvent, les développeurs composent des services de même type (SAOP avec SOAP ou REST avec REST) à cause des contraintes architecturales de chaque style de service. Toutefois, la composition de service de différentes architectures est requise afin de profiter de tous les services développés.

Dans ce travail, il nous semble très utile de développer un modeleur de composition de services de type SOAP et REST. Le modeleur assure l'interfaçage entre les deux architectures. Ce modeleur garantie le passage des données échangés entre services hétérogènes d'une façon cohérente. Ainsi, nous proposons dans ce manuscrit, la conception et la réalisation du modeleur.

Ce mémoire est organisé en quatre chapitres à savoir ;

1. Chapitre 1 : présente l'architecture orienté Service, et les Service Web de Type SOAP.
2. Chapitre 2 : présente l'architecture REST et la composition de service
3. Chapitre 3 : dédié à la modélisation du modeleur de composition de services web.
4. Chapitre 4 : présente les détails de réalisation du modeleur proposé.

Ce mémoire termine avec une conclusion générale.

2. Introduction

Avec la grande utilisation du web, les chercheurs ont développé des logiciels pour assurer et simplifier la communication entre les machines et les applications connectées via le réseau, ces logiciels sont appelés «web services».

Dans ce chapitre, nous présentons les concepts de base de l'architecture orientée service (AOS). Comme nous allons voir l'une des approches d'implémentation d'AOS qui consiste à utiliser les Web service. Nous focalisant dans ce chapitre uniquement sur le premier type des web service qui est le type SOAP.

3. Architecture de service orienté

3.1 Définition d'architecture de service orienté (SOA[1])

On trouve plusieurs définitions de l'architecture SOA dans la littérature on a choisi les suivantes :

- Selon Nicolai M.Josuttis:

« *SOA is not a concrete architecture it is something that leads to a concrete architecture. You might call it a style, paradigm, concept, perspective, philosophy, or representation. That is, SOA is not a concrete tool or framework you can purchase. It is an approach, a way of thinking, a value system that leads to certain concrete decisions when designing concrete software architecture*. » [1]

Nicolaï M.Josuttis veut dire que SOA n'est pas une architecture, mais c'est un paradigme qui nous conduit à une architecture. C'est une approche, une voie de pensée qui nous conduit à des certaines décisions.

- Selon OASIS:

« A SOA ecosystem is a network of discrete processes and machines that, together with a community of people, creates, uses, and governs specific services as well as external suppliers of resources required by those services. »[2]

Après la traduction on extrait qu'un écosystème SOA est un réseau de processus et de machines discrètes qui crée, utilise et réagit des services spécifiques ainsi que des fournisseurs externes de ressources requises par ces services.

[1] Service Oriented Architecture

- Selon Thomas Erl:

« *SOA establishes an architectural model that aims to enhance the efficiency, agility, and Productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing*. »[3]

Après la traduction, SOA est un modèle d'architecture. Son but est de progresser l'efficacité, l'agilité et la production en positionnant les services comme le primaire, c.-à-d. avec quel solution est représenté dans un support de la réalisation des buts stratégiques associées avec SOA.

- Selon Todd Biske:

« *A SOA, therefore, is quite simply, an architecture that utilizes the core concepts of service providers and service consumers to define a system*. »[4]

Comme Todd Biske a dit que cette architecture est vraiment simple. Elle utilise le cœur des concepts de deux services : le fournisseur et le consommateur pour définir un système.

SOA est un style d'architecture destiné à fournir un couplage lâché entre les composants d'un système. Un service est une fonction métier sans état qui accepte des demandes et renvoie des réponses à travers une interface bien définie.

3.2 Caractéristiques de l'architecture SOA

L'architecture orienté service architecture est caractérisée par :

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

3.3 Acteurs de l'architecture SOA

L'architecture orienté service est basée sur trois acteurs principaux (FigureI.1) définis comme suit :

- **Service fournisseur** : On peut l'appelé fournisseur il met le service web en application et il le rend disponible pour tout le monde sur internet.
- **Service consommateur :** C'est un client demandeur ou un consommateur qui fait la demande d'un service web bien précis pour répondre à ses besoins.
- **Service annuaire :** Le registre fournit un endroit où le consommateur peut trouver des nouveaux services web et le fournisseur dispose une description pour des nouveaux services web.

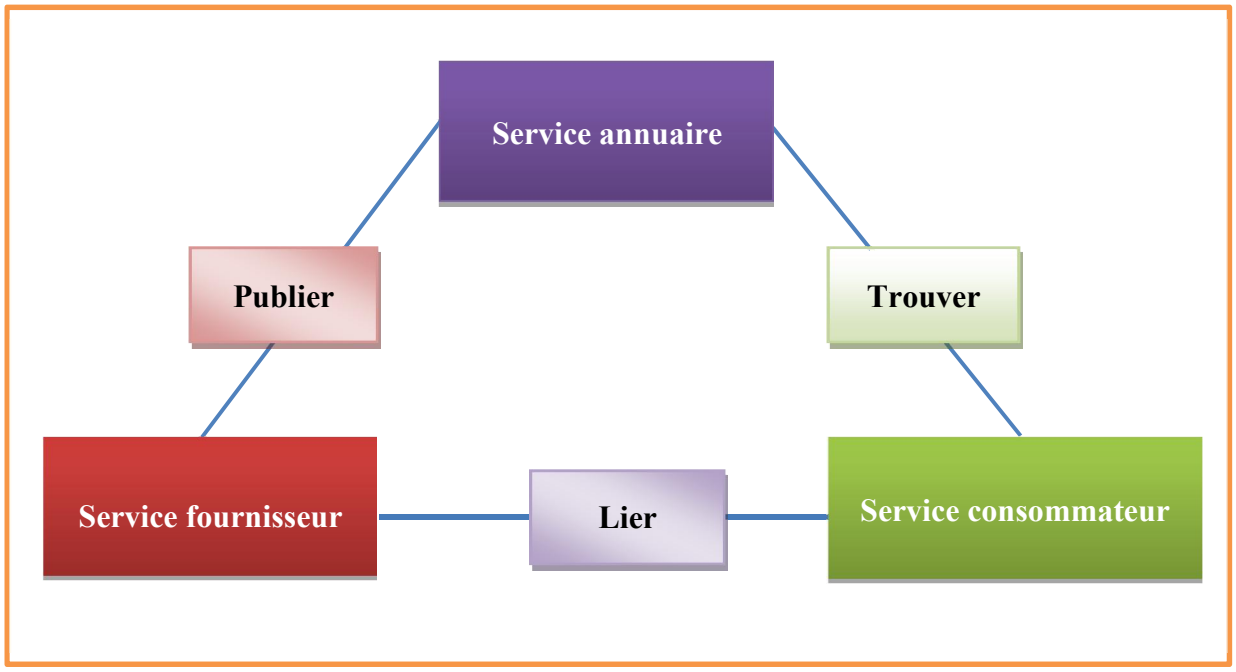

Figure I.1 : Principaux acteurs de SOA**.**

3.4 Concepts de l'architecture orienté service

L'architecture orientée service est axée autour de trois concepts fondamentaux qui sont : [1]

3.4.1 Services

C'est-à-dire une fonction que l'on peut interroger à l'aide d'une requête et qui fournit une ou plusieurs réponses.

3.4.2 Interopérabilité

Il permet la propagation des fonctionnalités des services web via des systèmes hétérogènes.

3.4.3 Couplage faible

Couplage faible est le concept qui assure l'évolutive, la flexibilité et la tolérance. Son but est de minimiser les dépendances. Quand on minimise les dépendances on va aussi minimiser l'influence sur les autres systèmes.

3.5 Approches de SOA

Deux approches sont définies pour implémenter l'architecture SOA tel qu'il est présenté dans la figureI.2 **[1]** :

3.5.1 Bottom-Up :

Cette approche est inventée par Krafzig and Slama [5]. Son principe est de commencer par un petit groupe et on jouter jusqu'à on obtient une grande entreprise.

3.5.2 Top-down :

Cette approche aussi est inventée par Krafzig and Slama. On peut dire que c'est l'adverse de Bottom-Up. Top-down est la décomposition d'un système ou problème jusqu'à l'obtient d'un petit service de base**. [1]**

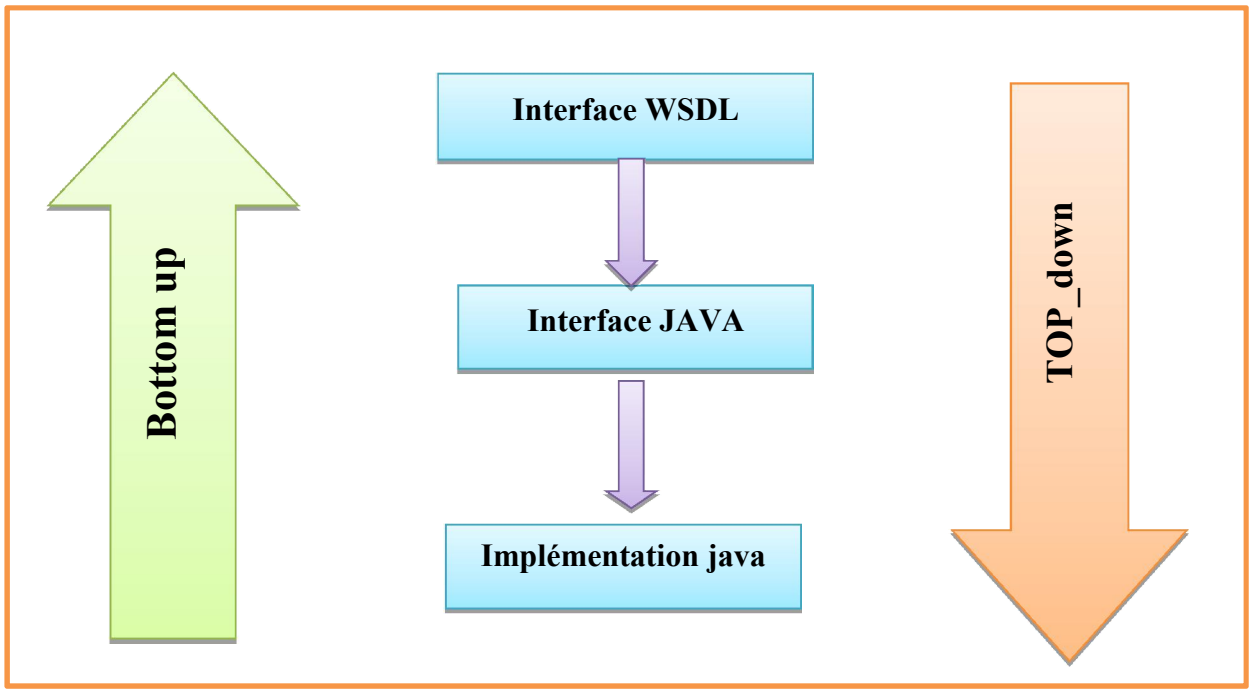

Figure I.2 : Approches de développement SOA

4. Service Web

Dans les sections suivantes, nous présentons les concepts des services web qui proposent une solution pour implémenter l'architecture orientée service.

4.1 Définition des Services Web

Le service web est un programme informatique permettant la communication et l'échange de données entre des applications et des systèmes hétérogènes basé sur un ensemble des protocoles et standard.

Plusieurs définitions ont été attribuées aux services web, on peut citer :

- Selon la définition de W3C 2 [8] :

«A web service is a computing system that exchanges XML messages with other systems using HTTP[3] as the communication protocol. In addition, a web service is an independent computing unit that implements a single business requirement. The ultimate goal for web service developers, however, is to chain as many of them as possible to solve problems of increasing complexity. And because they are available over the Internet, they offer an attractive scalable computing architecture»**.**

Ça veut dire que les services web sont des systèmes d'ordinateur qui échangent des messages de type XML avec d'autre système en utilisant http comme un protocole de communication. En plus le service web est une unité indépendante qui implémente un unique besoin.

Le meilleur but pour les développeurs des services web est de les chainer pour assurer la communication pour la raison de résoudre les problèmes de complexité.

- D'après les auteurs Doug Tidwell, James Snell et Pavel Kulchenko:

« *A web service is a network accessible interface to application functionality, built using standard Internet technologies. In other words, if an application can be accessed over a network using a combination of protocols like HTTP, XML[4], SMTP[5], or Jabber, then it is a web service. Despite all the media hype around web services, it really is that simple. Web services are nothing new. Rather, they represent the evolution of principles that have guided the Internet for years*. » [6]

[2]World Wide Web Consortium
[3]HyperText Transfer Protocol
[4]Extensible Markup Language
[5]Simple Mail Transfer Protocol

D'après la dernière définition : le service web est une interface qui accède aux applications via l'internet en utilisant les technologies standards de net.

D'autre mot, si une application qui peut accéder via le net en utilisant des protocoles comme http, XML, SMTP etc. Donc c'est un service web.

Les services web ne sont qu'une représentation des évolutions de principes qui guider l'internet durant ces années passées.

- Selon Martin Kalin

« *A web service is thus a distributed application whose components can be deployed and executed on distinct devices*. » **[7]**

Apres la traduction, le service web est une application distribuée qui peut être déployée et exécutée sur des différentes machines.

D'après toutes ses définitions on a sorti avec un résumé que ces services sont des logiciels qui permet la communication et le transfert des données entre des systèmes hétérogènes.

4.2 Caractéristiques des services web :

Les caractéristiques des services web de type SAOP sont :

- Accessible via le web.
- Décrit à partir de XML.
- Indépendant de plate-forme
- Conçus pour résoudre des problèmes bien définit et complexe.

4.3 Technologies de l'architecture orienté service

Les services web sont appliqués les contraintes de SOA en utilisant ces technologies suivantes :

4.3.1 SOAP

Le langage SOAP (Simple Object Access Protocol) est un Langage à base XML dédié à l'échange d'information entre les services Web.

a) Définition du SOAP

Le langage SOAP est défini Comme suit :

- D'après W3C:

«*SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses* ».[11]

Si on reprend la définition de W3C le soap est un protocole léger pour l'échange des informations dans un système distribué. Il est basé sur XML qui fournit trois (03) parties : Une enveloppe qui définit le cadre pour la description des messages, des règles de codage pour exprimer les instances de type de données et les conventions pour la représentation des appels et les réponses.

SOAP est un protocole de communication, supportant un RPC (Remote procedural Call) ou encore des messages de type document.

Sa caractéristique principale c'est qu'il est entièrement basé sur le langage XML pour définir la structure du message (l'enveloppe) et les données véhiculées.

Le fait que SOAP utilise des protocoles de transport standards de l'Internet est une autre caractéristique essentielle. [9]

Le SOAP est un protocole d'architecture SOA. Il est produit de Microsoft et IBM, sa première version a été acceptée par le W3C en 2000. Il permet l'échange des documents de type XML. Il permet aussi l'envoie des messages via le protocole de transport http.

On résumant ces précédentes définitions on aura que le but de SOAP qui échange les différents types des informations via le net sur des différents environnements en suivant trois (03) majeur éléments:

- La spécification de l'enveloppe SOAP.
- Les règles de codage des données.
- La convention de RPC.

b) Structures de message SOAP :

Chaque message SOAP contient trois (03) éléments, une enveloppe (Figure I.3), un entête et un corps.

Un message soap est composé d'une enveloppe. Ce dernier contient deux parties une entête et un corps.

- L'entête

L'entête ou bien le header c'est un élément optionnel s'il existe il doit être le premier élément dans l'enveloppe.

- Le corps

Le corps c'est un élément nécessaire dans l'enveloppe où il doit contenir des demandes et des réponses de SOAP.

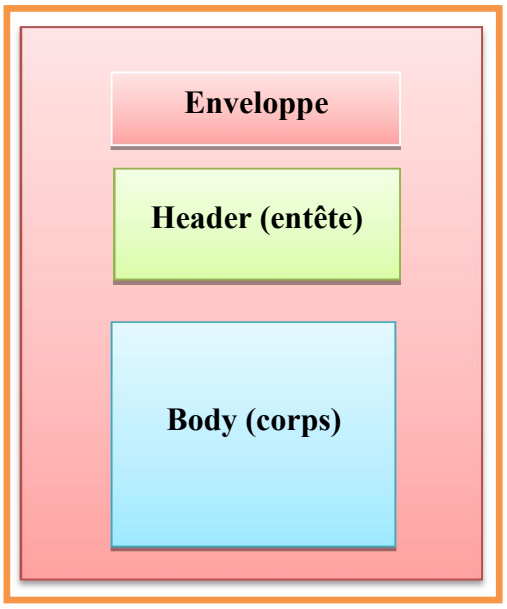

Figure I.3 Schéma d'un message SOAP

c) Exemple de Messages SOAP

Soit un service web qui fait l'addition entre deux nombres, la figure 1.4 représente le message SOAP de la requête. La réponse de ce message est représentée à la figure I.5 sous la forme d'une enveloppe SOAP de type réponse :

```
<?xml                                                version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Bodyxmlns:m="http://www.example.org/add">
                                                     <m:Getaddition>
    <m:nub1>1</m:nub1>
      <m:nub2>3</m:nub2>'
      </m:Getaddion>
</soap:Body>
</soap:Envelope>
```

Figure I.4 Exemple de Message SAOP de type Requête

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Bodyxmlns:m="http://www.example.org/add">
 <m:GetadditionResponse>
 <m:res>3</m:res>
 </m:GetadditionResponse>
</soap:Body>
</soap:Envelope>
```

Figure I.5 Exemple de Message SAOP de type Réponse

4.3.2 WSDL

Le langage WSDL (Web Service Description Language) est utilisé pour la description des services Web.

a) Définition de WSDL

«*WSDL is used to define service interfaces. In fact, it can describe two different aspects of a service: its signature (name and parameters) and its binding and deployment details (protocol and location).* »

La définition précédente veut dire que : WSDL est utilisé pour définir les interfaces des services web. Il décrit aussi deux différents aspect de service. Ces signatures qui se représentent le nom et les paramètres et ses détails de liaison et du déploiement.

b) Eléments de WSDL

Le WSDL se compose d'un ensemble d'élément décrivant le type de données utilisée par le service web. Il propose une syntaxe XML pour la définition des documents qui décrivent des services web. Chaque document est devisé en sept (07) parties :

- **Type**

Il définit les structures des messages échangés avec un service web et il décrit les types de données utilisées entre les clients et les serveurs.

- **Message**

Il définit le format des messages échangés.

- **Opération essentielle**

  Il définit les opérations invoquées sur les services web.

- **Type de port**

  Il regroupe un ensemble d'opération. Chaque opération peut être :

- Un message d'entrée.
- Un message de sortie.
- Un message d'erreur.

- **Liaison ou (binding)**

  Il spécifie une liaison entre le type de port et un protocole de communication.

- **Port**

  Il précise le point d'accès à un service web.

- **Service**

  Il indique les adresses de port de chaque liaison.

c) Avantages de WSDL

Parmi les avantages de WSDL on trouve : [5]

- WSDL rend plus facile l'écriture et le maintien des services en fournissant une approche plus structurée**.**
- WSDL est plus facile à consommer des services web en réduisant la quantité de code (et des erreurs potentielles) qu'une application client doit mettre en œuvre.
- WSDL facilite la mise en œuvre des changements qui seront moins susceptibles de casser SOAP applications clientes.

d) EXEMPLE de WSDL

La Figure I.6 représente Le document WSDL correspondant à un service Web d'addition.

4.3.3 UDDI

Universal Description Discovery and Integration (UDDI) est définie comme suit :

« *UDDI is a technical pecification for describing, discovering, and integrating web services. UDDI is therefore a critical part of the emerging web service protocol stack, enabling companies to both publish and find web services*. » [10]

```
<?xml version="1.0" encoding="UTF-8"?>
        <wsdl:definitionstargetNamespace="urn:Calculette"  xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="urn:Calculette"  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <wsdl:message name="Getaddion Request">
        <wsdl:part name="nub1" type="xsd:int"/>
        <wsdl:part name="nub2" type="xsd:int"/>
        </wsdl:message>
        <wsdl:message name="Getaddion Response">
        <wsdl:part name="res" type="xsd:int"/>
        </wsdl:message>
        <wsdl:portType name="Addition">
        <wsdl:operation name="Getaddion " parameterOrder="nub1nub2">
        <wsdl:input message="tns:Getaddion Request"/>
        <wsdl:output message="tns:Getaddion Response"/>
        </wsdl:operation>
        </wsdl:portType>
        <wsdl:binding name="AdditionSoapBinding" type="tns:Addition">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="Getaddion ">
        <soap:operationsoapAction=""/>
        <wsdl:input name="additionnerRequest">
        <soap:bodyencodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  namespace="urn:Calculette"
use="encoded"/></wsdl:input>
        <wsdl:output name="additionnerResponse">
        <soap:bodyencodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  namespace="urn:Calculette"
use="encoded"/>
        </wsdl:output>
        </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="Getaddion Service">
        <wsdl:port binding="tns:AdditionSoapBinding" name="Getaddion ">
        <soap:address location="http://localhost:8080/axis/services/Getaddion "/>
        </wsdl:port>
        </wsdl:service>
        </wsdl:definitions>
```

Figure I.6 Exemple d'un document XML

a) Techniques UDDI

- Modèle de données UDDI : C'est un schéma qui décrit les services web et les entreprises.
- API UDDI : Une API SOAP pour la recherche et la publication des données UDDI.

Les informations de registre UDDI sont regroupées en trois pages :

- Les pages blanches **:** ces pages contiennent les listes des organisations, leurs informations de contacts et les services qu'elles fournissent.

- Les pages jaunes **:** sont des pages pour la classification des entreprises des services web selon des taxonomies.
- Les pages vertes **:** pour la description d'invocation d'un service web.

4.4 Fonctionnement d'un service web

Le fonctionnement des services web ou bien de type SOAP est expliqué dans les étapes suivantes :

- Le WSDL fait la description du service web.
- Il publie dans l'annuaire UDDI.
- Si un client a besoin d'un service web
  - Il demande l'UDDI.
  - L'annuaire fournit au client la description du service web pour l'invocation.
  - Le client invoque le service web après sa localisation.
    - ✓ Pour invoquer un service web on doit envoyer un message soap « la requête ».
    - ✓ Le web service répondre à cette requête par un message soap aussi « la réponse ».

La figure I.4 représente le fonctionnement du service web en utilisant les trois (03) techniques :

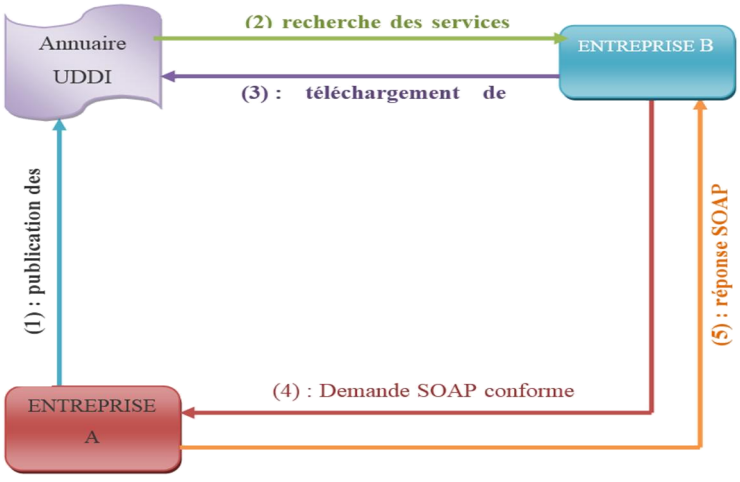

Figure I.7 : Fonctionnement d'un service web

4.5 Avantages du service web

Parmi les avantages des services web on trouve :

- Un meilleur développement et adaptabilité
- Plus de flexibilité au niveau de la réutilisabilité des composantes et l'adaptation au changement.
- Une grande tolérance aux pannes, et une maintenance plus aisée.

- Composition de services permet d'offrir des services à forte valeur ajoutée aux clients.
- Interactions faiblement couplées

4.6 Inconvénients du Service Web

Parmi les inconvénients des services web on trouve :

- La Multiplication de la masse d'information véhiculée.
- Le Surcharge de traitement.
- La Complexité d'implémentation.

5. Conclusion

Dans ce chapitre, nous avons présenté en premier lieu l'architecture orientée service (SOA). Ensuite, nous avons présenté les Web services de type SOAP et leurs technologies sous-jacentes.

Dans le prochain chapitre, Nous allons présenter l'architecture REST et les services web développés sous cette architecture.

1. Introduction

Le présent chapitre est divisé en deux parties. La première partie est consacrée à la présentation de l'architecture REST et ses services web. La deuxième partie est réservée à la description des Composition des services web de type SOAP ou REST.

2. Architecture REST

REST (**REpresentational State Transfer**) est un style d'architecture crée par Roy. T. Fielding en 2000 dans sa thèse de doctorat "Architectural Styles and the Design of Network-based Software Architectures .[12]

2.1 Définition de REST

On a plusieurs définitions sur l'architecture REST, on cite quelque unes :

- **Selon Roy Thomas Fielding: «***REST is a hybrid style derived from several of the network-based architectural styles and combined with additional constraints that define a uniform connector interface***»** [12].

Roy Thomas Fielding a créé cette architecture « REST » selon sa définition le REST est un modèle hybride dérivé de plusieurs modèles basés sur les concepts réseau.

- **Selon Martin Kalin**: « *REST is a style of software architecture for distributed hypermedia systems; that is, systems in which text, graphics, audio, and other media are stored across a network and interconnected through hyperlinks*. »[7]

On conclut du point de vue de " Martin Kalin" que REST est un style d'architecture pour les systèmes hypermédia distribués, c'est un système auquel le texte, les graphiques, l'audio et autre média sont stockés sur un réseau et reliés entre eux par des hyperliens.

2.2 Contrainte d'architecture REST

La conception de l'architecture REST est basée sur les contraintes suivantes [12] :

2.2.1 Le modèle vide

Le modèle vide représente simplement un ensemble vide de contraintes décrit un système dans lequel aucune frontière distincte n'existe entre les composants. C'est le point de départ pour notre description de REST.

2.2.2 Le modèle client-serveur

C'est la première contrainte ajoutée au REST pour séparer les concepts. Cette séparation permet aux composants d'évoluer indépendamment.

2.2.3 Le modèle sans état

C'est un modèle concerne l'interaction entre le client et le serveur pour chaque requête du client vers le serveur doit contenir toutes les informations nécessaires et il ne peut pas profiter d'aucun contexte sur le serveur.

Cette contrainte assurer la visibilité, la fiabilité et faculté de montée en charge. Mais cette contrainte diminue la performance du réseau en répétant les données (surplus par interaction) envoyées dans une série de requêtes. La figure II.1 représente un schéma du modèle sans état

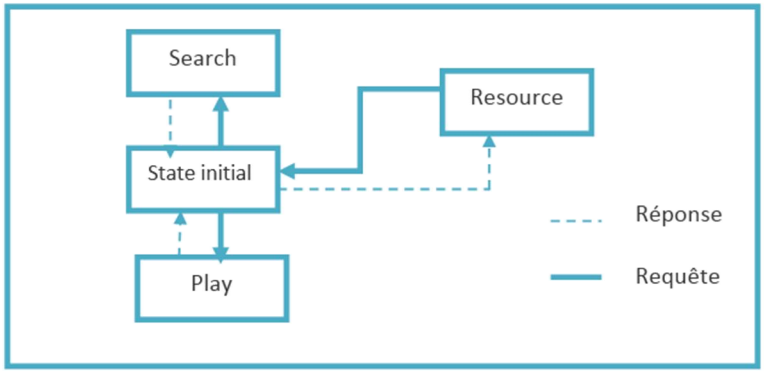

Figure II.1 Schéma représentant un modèle sans état

2.2.4 Le modèle cache

Cette contrainte est basée sur la mise en cache des donnés d'une réponse de façon implicite ou explicite pour améliorer la performance de réseau.

2.2.5 La contrainte interface uniforme

Le point fondamental qui distingue le modèle d'architecture REST des autres modèles est la mise en exergue d'une interface uniforme entre les composants.

Cette contrainte permet la simplicité de système, l'amélioration de la visibilité d'interaction et la mise en œuvre sera être découplées des services qu'elles fournissent. Mais elle n'est pas optimale pour d'autre forme d'interaction architecturale.

2.2.6 Le modèle Système en couche

Le modèle de système en couches permet à une architecture d'être composé de couches hiérarchiques en contraignant le comportement des composants, pour offrir un équilibrage de charges pour les services.

2.2.7 La contrainte code à la demande

Un client a le droit d'accès à un ensemble de ressources, mais il ne sait pas comment ils sont traités. Il envoie une requête au serveur traitant qui est distant et il l'exécute localement.

Les avantages de code à la demande incluent la possibilité d'ajouter des fonctionnalités à un client déployé, ce qui permet d'améliorer l'extensibilité.

Mais le code à demande réduit la visibilité, ce manque de visibilité conduit à des problèmes de déploiement si le client ne peut pas faire confiance aux serveurs c'est pour ça cette contrainte est optionnelle.

2.3 Principes de REST

L'architecture REST repose sur les principes décrits dans les sous sections :

2.3.1 Adressage

L'adressage est l'idée que tous les objets et les ressources de votre système est accessible par un identifiant unique [13]. Cela semble à une évidence, mais si on pense à ce sujet, l'identité de l'objet normalisé n'est pas disponible dans de nombreux environnements.

Ce n'est pas une grosse affaire pour une application, mais avec la nouvelle popularité de la SOA, on se dirige vers un monde où les applications disparates doivent être intégrer et interagir. Ne pas avoir quelque chose d'aussi simple que le service d'adressage normalisé ajoute toute une dimension complexe aux efforts d'intégration.

2.3.2 Interface

Le principe de REST d'une interface limitée est peut-être la pilule la plus difficile pour un développeur COBRA ou SOAP expérimenté à avaler. L'idée derrière cela est qu'on s'en tient à l'ensemble fini des opérations du protocole d'application lequel vous distribuez vos services sur.

Cela signifie qu'on n'a pas un paramètre "action" dans URI et on utilise uniquement les méthodes de HTTP pour **les services** web. HTTP a un petit ensemble fixe de méthodes opérationnelles.

a) GET

Le but de la commande GET (Figure II.2) est de récupérer une représentation d'une ressource. Cette méthode ne devrait jamais causer d'effets secondaires. Le risque d'abuser le

« GET » est que les ressources seront modifiés ou involontaire conséquences pourraient corrompre ou affecter les ressources de façon indéterminable.

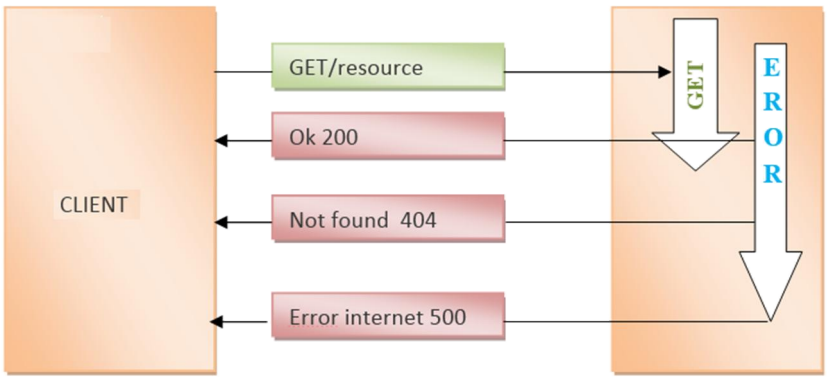

Figure II.2 Méthode http GET

b) POST

POST Cette méthode a essentiellement deux objectifs : l'un qui s'inscrit dans les contraintes de REST, et l'autre qui va REST extérieur et introduit un élément de style RPC. [18]

La méthode POST est conçu pour :

- Annoter des ressources existantes ;
- Afficher un message à un babillard, forum de discussion, liste de diffusion, ou un groupe similaire des articles.
- Fournir un bloc de données, tel que le résultat de la soumission d'un formulaire, à un traitement de données processus ;
- Extension d'une base de données via une opération d'ajout.

Le mécanisme de POST : [18]

- Un client Java effectue une requête à l'URI « http://restfuljava.com/ étudiants / Jane,»
- La demande POST porte la charge utile sous la forme d'un fichier XML.
- Le serveur reçoit la demande et le cadre de REST permet à manipuler le code dans le pour stocker la représentation
- Une fois le stockage de la nouvelle ressource est terminée la réponse est renvoyée : Si c'est un succès, nous envoyons un code de 200 ; sinon nous envoyons le cas échéant code d'erreur.

c) DELETE

La méthode DELETE supprime une ressource. Le serveur devrait retourner une réponse indiquant le succès ou l'échec de l'opération. Les réponses à la méthode DELETE ne sont pas mises en cache.

- Un client soumet une demande de suppression d'une ressource. DELETE / utilisateurs / john HTTP/1.1 Hôte : www.example.org
- Le serveur crée une nouvelle ressource et retourne une représentation indiquant l'état de la tâche.
- Le client peut interroger l'URI http://www.example.org/task/1 pour connaître l'état de la demande.

```
HTTP/1.1 202 Accepted
Content-Type: application/xml;charset=UTF-8
<status xmlns:atom="http://www.w3.org/2005/Atom">
<state>pending</state>
<atom:link href="http://www.example.org/task/1" rel="self"/>
<message xml:lang="en">Your request has been accepted for  processing.</message>
<created>2009-07-05T03:10:00Z</ping>
<ping-after>2009-07-05T03:15:00Z</ping-after>
</status>
```

Figure II.3 Exemple de la méthode Delete

d) PUT

Cette méthode est pour créer des nouvelles ressources uniquement lorsque le client peut contrôler une partie de l'URI. Par exemple, un serveur de stockage peut attribuer une URI racine de chaque client et de laisser les clients à créer de nouvelles ressources à l'aide de cette racine URI comme un répertoire racine sur un système des fichiers.

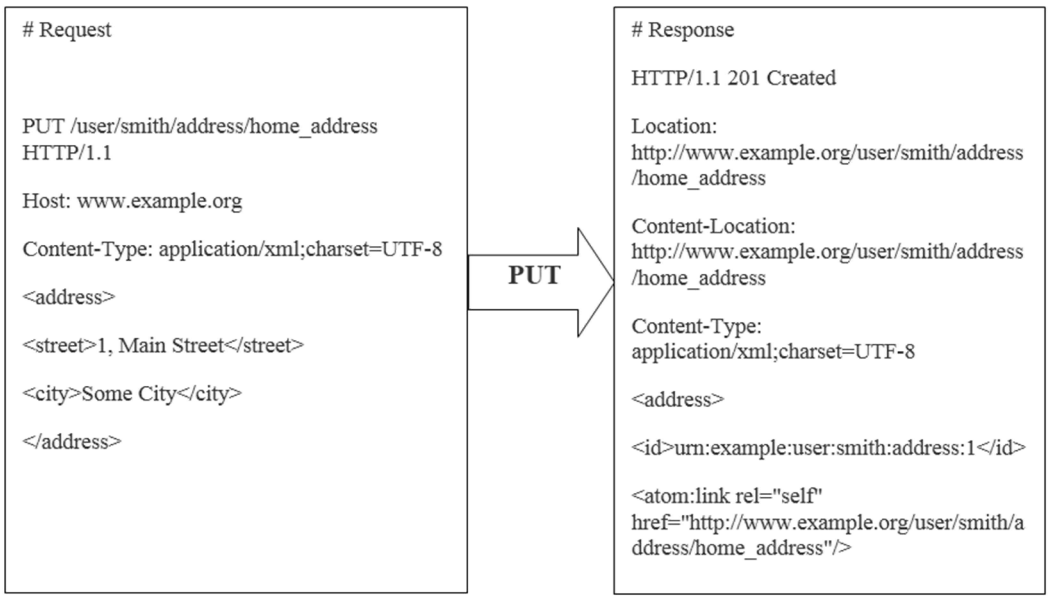

Figure II.4 Exemple de la méthode PUT

e) HEAD

HEAD est exactement comme GET, sauf qu'au lieu de retourner un corps de la réponse, il ne retourne que le code de réponse et les en-têtes associés à la demande.

2.3.3 Représentation des ressources

Parce que REST et HTTP ont une approche multidimensionnelle à l'adressage la de méthode choix, et le format de données, on a un protocole beaucoup plus découplées qui permet à un service d'interagir avec une grande variété de différents clients d'une manière cohérente.

2.4 Fonctionnalités des services RESTFUL

Parmi les fonctionnalités et les spécificités des Services de type REST, on trouve entre autres :

1. Dans une demande, l'appariement d'un verbe HTTP telles que GET avec un URI comme **http://.../**
2. Le service utilise des codes de statut HTTP tels que 404 (ressource non trouvée) et 405 (méthode non autorisé) pour répondre aux mauvaises demandes.
3. Si la demande est bonne, le service répond avec une représentation XML capturant l'état de la ressource demandée. Jusqu'à présent, les honneurs de services GET demandent, mais les autres verbes CRUD seront ajoutés à la prochaine révision.
4. Le service ne profite pas des types MIME.

5. La mise en œuvre des services RESTFUL ne contraint pas de la même manière comme un service SOAP basé précisément parce qu'il n'y a pas de contrat de service formel.
6. La mise en œuvre est flexible mais bien sûr même pour un ad hoc.

2.5 Web Application Description Language (WADL)

WADL est une description XML d'une application web RESTful déployé. Il contient le modèle des ressources déployées, leur structure, types de supports, les méthodes HTTP, etc. Dans un sens, WADL est un analogue de la WSDL (Web Service Description Language) qui décrit les services Web SOAP. WADL est cependant spécifiquement conçu pour décrire les ressources Web RESTful.

3. Composition des services web

Un seul service web ne peut pas répondre aux besoins des utilisateurs par conséquence on nécessite de combiner un ensemble de service web pour réaliser et répondre aux besoins des utilisateurs.

3.1 Définition

La composition de services désigne une interaction entre deux ou plusieurs services en vue d'accomplir des objectifs déterminés. [14]

La composition requiert la description et l'organisation de l'interaction entre les services. Elle nécessite la gestion de plusieurs aspects comme les échanges des données entre les services, les pannes ou les erreurs éventuelles, le contexte d'interaction, le degré d'automatisation des taches, etc.

Un environnement d'outil pour la composition des services doit fournir au moins les modules suivants : [14]

a) **Le module de conception**

Ce module offre une interface graphique pour spécifier un service composite. Le module peut traduire une conception d'un service composite à un langage de description.

Outils de conception les plus avancés peuvent soutenir la vérification et / ou la simulation automatisée des modèles de services composites sur la base d'un langage formel.

b) Environnement d'exécution

Il est responsable de l'exécution d'un service composite et du routage des messages entre ses composantes. Il est également responsable de la surveillance, la faute et la gestion

des exceptions. L'environnement d'exécution peut supporter la sélection dynamique et la liaison de services

3.2 Approches de composition des services web

La composition peut être décrite sous deux angles :

3.2.1 Orchestration

Une orchestration assemble les services web dans un processus métier exécutable qui doit être exécuté par un moteur d'orchestration.

L'orchestration de services Web (Figure II.5) consiste à la programmation d'un moteur qui appelle un ensemble des 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

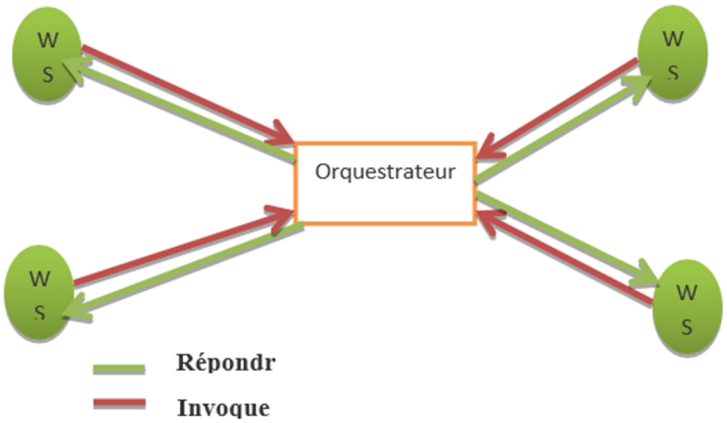

Figure II.5 Schéma représentant orchestration de service

a) Exemple d'orchestration

La figure II.6 représente un exemple de gestion de commande

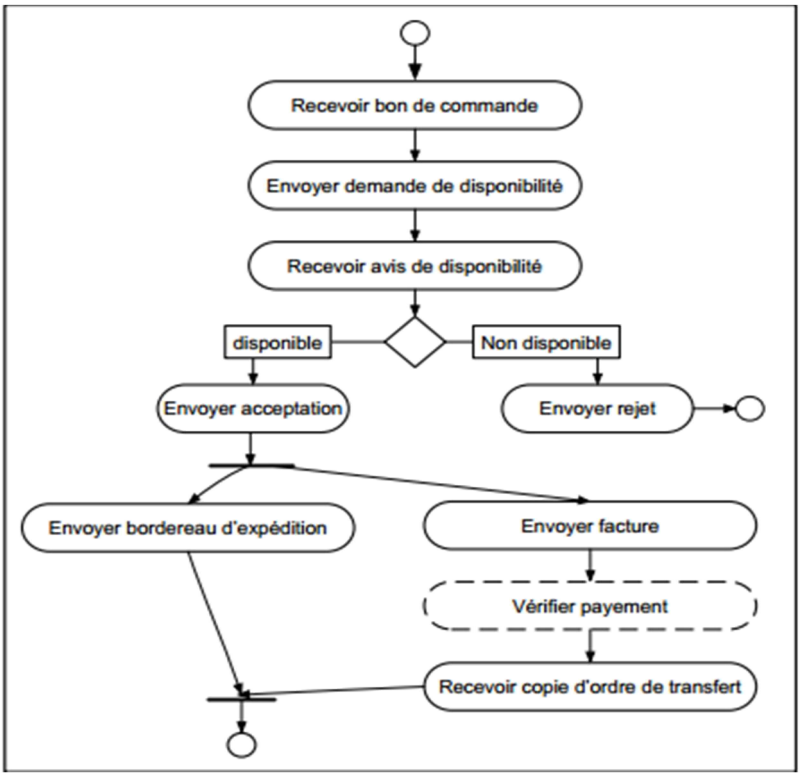

Figure II.6 schéma d'activité d'exemple d'orchestration

3.2.2 Chorégraphie

La chorégraphie (Figure II.7) illustre les différents échanges de messages entre les participants.

La chorégraphie n'implique pas un contrôle centralisé, le contrôle est partagé entre les participants en interaction. Une orchestration représente un processus exécutable à exécuter par un moteur d'orchestration en un seul endroit, alors que la chorégraphie en essence représente une description de la façon de distribuer le contrôle entre les participants collaborent, à l'aide des plusieurs moteurs pour faire le travail.

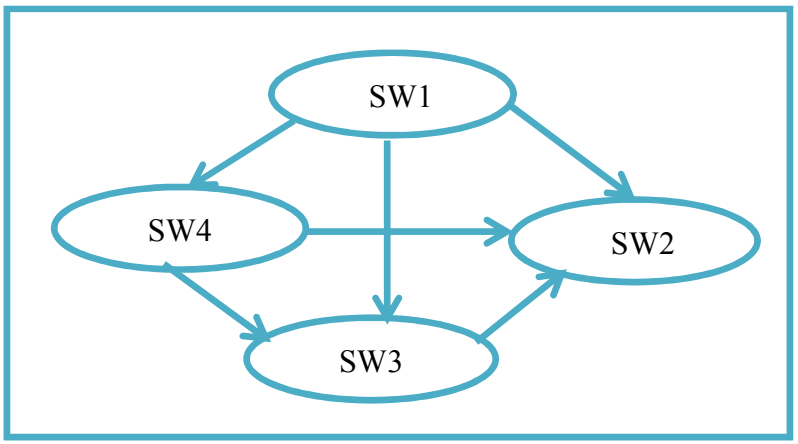

Figure II.8 Illustration d'une chorégraphie de web services

3.3 Exigences techniques pour l'orchestration et la chorégraphie

On définir les exigences techniques pour l'orchestration des services Web en les point suivantes.

3.3.1 Flexibilité

La flexibilité peut être atteinte en prévoyant une séparation claire entre le processus logique et les services web invoqués.

Cette séparation peut être réalisée grâce à un moteur d'orchestration qui gère l'ensemble du flux de processus avec cette flexibilité.

3.3.2 Activités basiques et structurés

Un langage d'orchestration doit soutenir les activités à la fois pour communiquer avec d'autres services web et de manipulation sémantique de flux de travail. On peut penser à une activité de base comme un composant qui interagit avec quelque chose externe au processus lui-même. Par contre, les activités structurées gèrent l'ensemble de flux de processus, précisant quelles activités doivent courir et dans quel ordre.

3.3.3 Composition récursive

Un processus métier unique peut interagir avec de multiples Services Web. Cependant, un processus métier peut lui-même être exposé comme des services web, permettant des processus d'affaires à être agrées pour former un niveau supérieur de processus. En plus les deux services web orchestrés et chorégraphiés doivent soutenir certaines exigences de base pour la gestion de l'intégrité et de la cohérence des interactions.

3.4 Langages utilisés pour la composition des services web :

Il y a quelque langage pour la composition des services web :

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

b) **BPML :** est un métalangage qui a été développé par l'organisation BPMI.org (Business Process Management Initiative). Un processus métier BPML est un enchaînement d'activités simples, d'activités complexes et de processus incluant une interaction entre participants dans le but de réaliser un objectif métier.

c) **WSCL** : **Web Services Conversation Language** permet de décrire les conversations du niveau de la logique du métier ou les processus publics basé sur la définition d'un

service web. Les définitions de conversation en WSCL sont elles-mêmes des documents XML et peuvent donc être interprétées par des infrastructures de services web et des outils de développement.

d) **BPEL : Business Process Execution Language** représente un langage de programmation standard aidant les entreprises à définir la manière de combiner des services web [15]. La figure II.9 représente la structure d'un processus BPEL synchrone.

e) **BPEL4WS :** Business Process Execution Language for Web Services ou tout simplement BPEL est une spécification d'IBM, Microsoft, et BEA. Elle remplace les précédentes spécifications XLANG de Microsoft, et WSFL (Web Services Flow Language d'IBM.

Le modèle de procédé BPEL forme une couche au-dessus de WSDL. Il définit la coordination des interactions entre l'instance de procédé et ses partenaires. Les procédés dans BPEL exportent et importent les fonctionnalités en utilisant des interfaces de services web uniquement. BPEL contient les caractéristiques d'un langage structuré en blocs (block structure de langage) du XLANG, ainsi que les caractéristiques d'un graphe direct de WSFL.

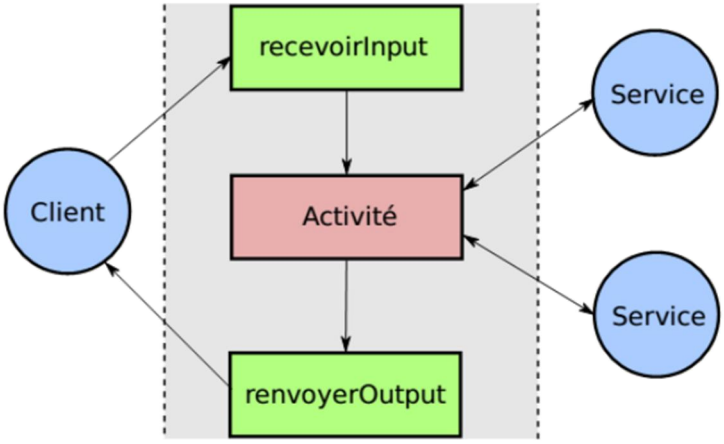

Figure II.9 Structure d'un processus BPEL synchrone

3.5 Concepts du processus métier

La possibilité de spécifier des conditions exceptionnelles et de leurs conséquences, y compris les séquences de récupération, est au moins aussi important pour les processus métier comme la possibilité de définir le comportement dans le "tout va bien" affaire.

L'interaction de longue durée comprennent plusieurs unités, souvent imbriqués de travail, chacun avec ses propres exigences en matière de données. Les processus d'affaires nécessitent souvent la coordination des partenaires croix du résultat (succès ou échec) des unités de travail aux différents niveaux de granularité.

3.6 Fonctionnement d'un processus métier exécutable :

Le fonctionnement s'appuie sur WSDL en supposant que toutes les interactions externes du processus d'affaires sont produites par des opérations de service Web. Cependant, les processus d'affaires WS-BPEL représentent les interactions de longue durée dans lequel chaque interaction a un début et une fin.

Un processus BPEL est constitué d'activités mises bout-à-bout par des structures de contrôle de flux :

- **Invoke :** permet d'appeler un service.
- **Receive :** permet de recevoir un message (utilisé notamment pour initier le processus).
- **Reply** : permet de renvoyer un message.
- **Throw** : permet d'émettre une exception au même titre que dans d'autres langages de programmation.
- **WSDL** permet également de décrire des messages d'erreur qui peuvent transiter.
- **BPEL** permet d'exploiter cela facilement (il y a des constructions de type catch).
- **Terminate** permet de terminer le processus.
- **Wait** permet de suspendre le processus pendant une durée déterminée.
- **Assign** permet d'effectivement manipuler des données, comme par exemple copier des valeurs entre variables ou utiliser des expressions XPath 1.0 pour effectuer des traitements comme manipuler des strings etc.

4. Conclusion

Dans ce chapitre, nous avons présenté les concepts de base de l'architecture REST et les web Service REST. Nous avons présenté également des généralités sur la composition de Service Web, approches Dans le prochain chapitre, nous proposons une solution via un

modeleur de composition à la composition des services web de type SOAP, et les services web de type REST

1. Introduction

Les chapitres précédents ont été consacrés aux concepts de base des architectures orientées services, REST. Nous avons également présenté les notions de base des web services Soap et leur composition.

Dans ce chapitre, nous proposons une architecture d'un modeleur de composition de services REST et SOAP.

2. Description du Modeleur

Le modeleur proposé doit assurer à ses utilisateurs, qui sont des concepteurs et/ou développeurs, une composition de services web de différente architecture. Comme il est mentionné dans l'état de l'art, les services SOAP se diffèrent complètement de services REST, car les premiers utilisent le protocole SOAP lors de communication et les autres utilisent le protocole http. Ainsi, le modeleur doit construire une interface de communication entre les deux concepts, tel qu'il est schématisé dans la figure III.1. Le fonctionnement du modeleur proposé est décrite dans la section suivante.

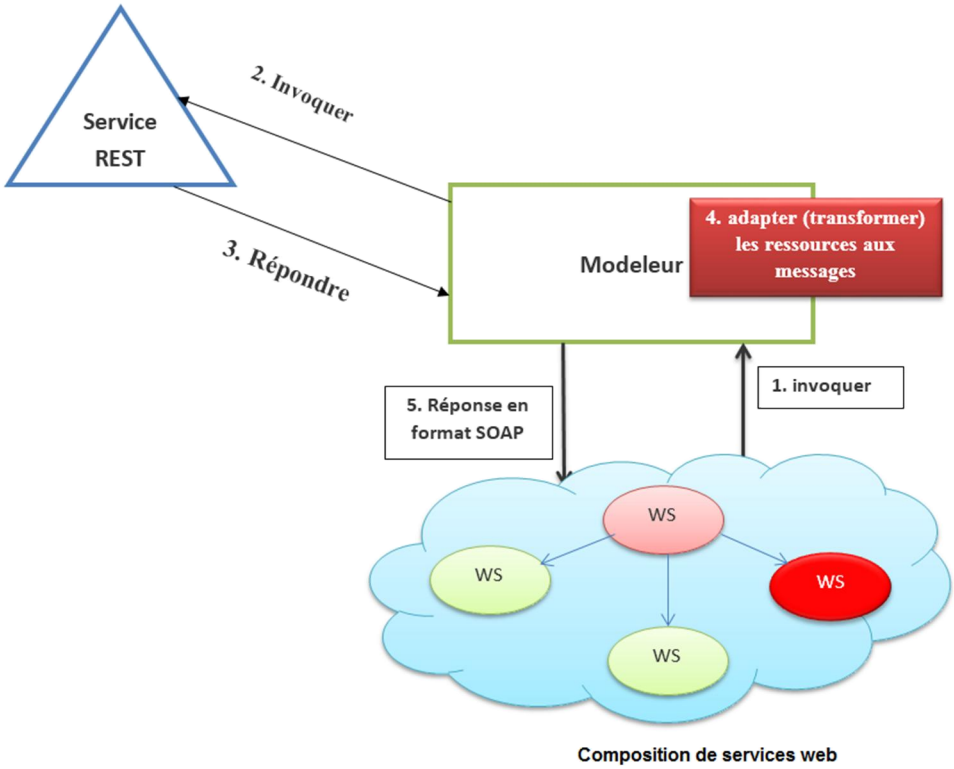

Figure III.1 Schéma de fonctionnement du modeleur proposé

3. Fonctionnalités du Modeleur

Pour utiliser le modeleur, le concepteur doit préalablement :

1. Définir l'orchestration de services web SOAP en utilisant le langage PBEL.
2. Déterminer les services REST à intégrer dans l'orchestration, soit pour utilisation soit pour recouvrement d'un service SOAP défaillant.
3. Déterminer les correspondances entre les services REST et SAOP en cas de recouvrement/ remplacement de service défaillant.

Les modeleur offre a son utilisateur de réaliser l'ensemble de fonctions représentées dans le digramme de cas d'utilisation suivant (Figure III.2) :

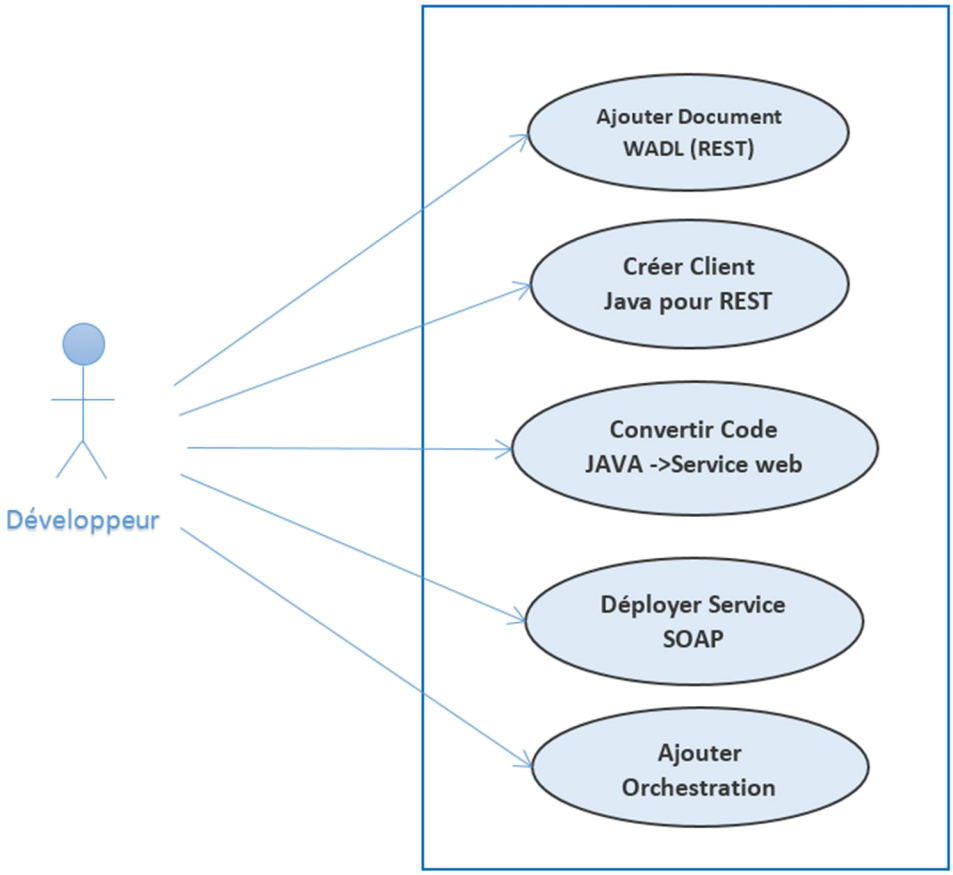

Figure III.2 Diagramme de Cas d'utilisation du Modeleur

4. Architecture du Modeleur

L'analyse fonctionnelle présentée via le diagramme de cas d'utilisation (Figure III.2) nous a conduites de proposer l'architecture présentée dans la figure au-dessous (Figure III.3).

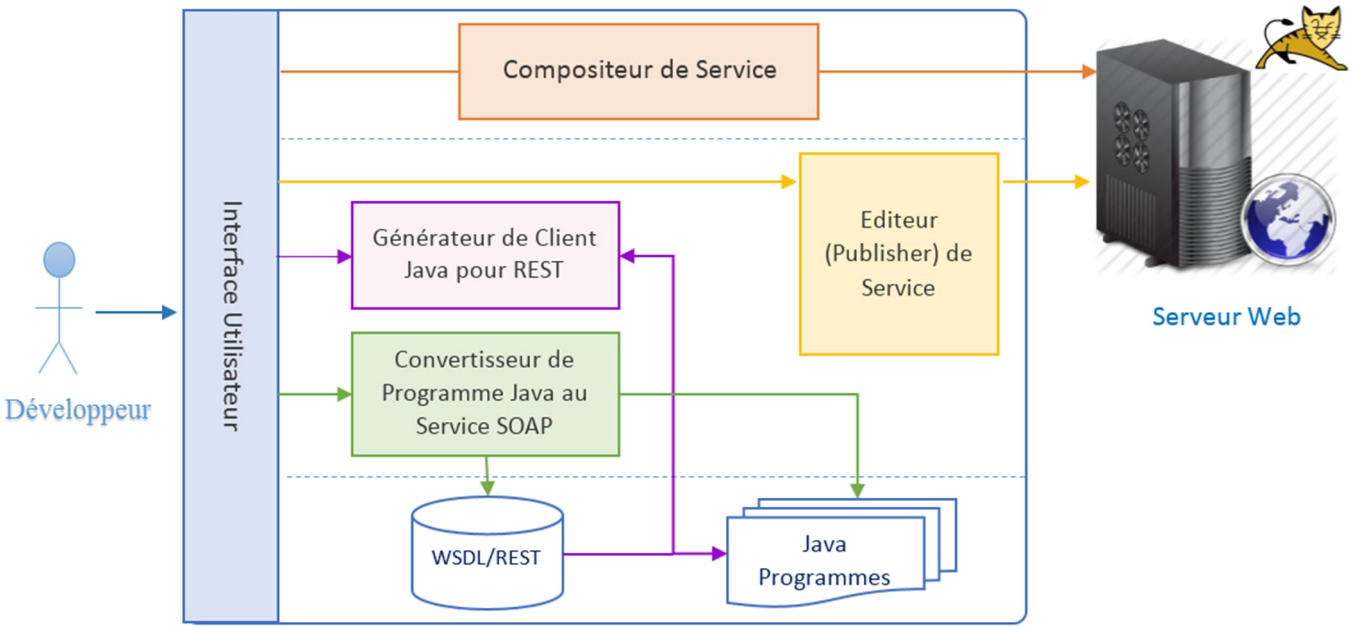

Figure III.3 Architecture du Modeleur

Notre modeleur sera composé de cinq composantes principales.

4.1 Compositeur de Services

La première composante dans le modeleur est le module compositeur de services. Ce module est une interface avec le moteur BPEL (Apache ODE[6]). Le moteur permet d'exécuter une ou plusieurs orchestrations de service Web « business process ».

Le module permet à utilisateur d'introduire le code d'orchestration se forme d'un fichier BPEL. L'utilisateur peut par la suite modifier le code d'orchestration en fonction de ces nouveaux besoins. Il peut également substitue un ancien service par un (plusieurs) service(s) REST, après la génération du service miroir (service SOAP client du REST). Ainsi, des mises à jour peuvent être effectuées sur le BPEL d'origine.

Le compositeur de service finalement n'exécute que des orchestrations WS-BPEL. Les services manipulés sont :

- Services d'origines SOAP
- Service REST invoqué par des Services miroirs SOAP.

[6] Apache Orchestration Director Engine, Home page : www.ode.apache.org

L'interfaçage entre le module compositeur et l'engin ODE d'apache est schématisé dans la figure III.4.

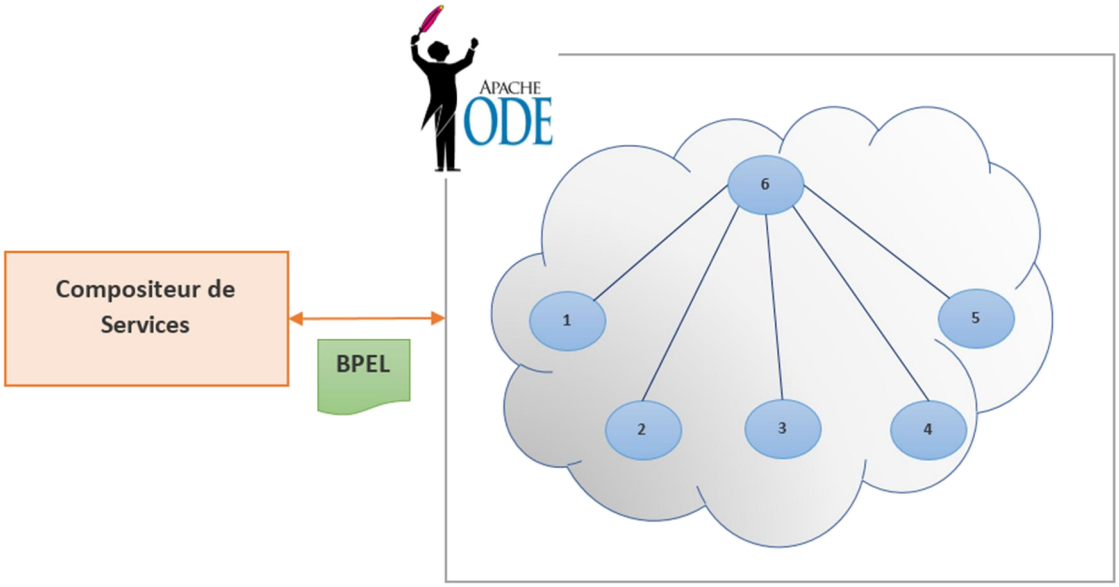

Figure III.4 Interfaçage entre le Compositeur et l'engin Apache PDE

Le compositeur doit impérativement contrôle le flux d'exécution de l'orchestration. L'utilisateur peut 1) démarrer, 2) interrompre momentanément, 3) arrêter ou 4) redémarrer une orchestration de services web.

4.2 Générateur du client REST

Ce module est un générateur du client REST écrit en Java. L'utilisateur détermine le client REST à intégrer dans son orchestration en fournissant le modeleur par un fichier WADL correspondant au service REST. Par la suite, le générateur produit une classe Java dont les méthodes sont des méthodes publiques où chacune manipule une ressource REST. Le travail du module Générateur du code client REST est divisé en deux phases :

- Analyse : c'est une phase d'extraction des ressources, Opération http de chaque ressource, et les paramètres de chaque opérations.
- Génération du code Java : c'est la phase de génération d'un code client pour chaque ressource.

La description de chaque phase est présentée dans les sous-sections suivantes.

4.2.1 Analyse du WADL

Le générateur analyse le fichier wadl, qui est un fichier a base XML doté d'une syntaxe précise. L'analyse permet au générateur d'extraire les ressources disponibles dans les

services REST. Chaque ressource est associé à une opération http (GET, POST, PUT, DELETE). Le fichier WADL contient les paramètres à envoyer lors de manipulation de la ressource. Ainsi les opérations disponibles dans le service REST sont identifiées (Opération http + Ressource + (Paramètres)).

4.2.2 Génération du code Java

Dans la deuxième phase, le module génère un code client pour chaque ressource REST. Tout dépend de l'opération http d'origine, une méthode public Java est produite. L'ensemble de méthodes est regroupé dans une classe Java cliente au service REST à utiliser.

Figure III.5 Exemple de génération du code Client pour une ressource REST

La signature des opérations est formulée comme suit

- Mot clé **Public**
- **Type de retour** selon le paramètre du retour lors d'invocation de la ressource REST. Une conversion des type est obligatoire (XSD:Integer devient Int...etc.)
- L'**identifiant** de la méthode java contient le nom (get, set, delete) suivant le type d'opération http (Get, Post, Delete, Put), concaténer avec le nom de la ressource.
- Les **paramètres** sont écrits en respect au paramètre extraits du fichier WADL.

Le core de la méthode implémente une invocation de la ressource demandée suivant l'opération http utilisée et les paramètres à fournir tel il est décrit dans la spécification WADL.

La figure III.5 illustre la génération du code client pour une ressource REST à partir de la description WADL associée.

4.3 Convertisseur Java – Service Web

Le module Convertisseur est responsable sur la conversion d'une classe Java vers un web service de type SOAP. En effet, ce module est utilisé dans le modeleur pour rendre les clients des services REST accessibles via le web sous forme d'un web service. Ainsi, une invocation de ce web service, qu'on l'appelle un service miroir, n'est en réalité qu'un appel au service REST. Car les méthodes offertes par le service miroir ne font qu'une invocation aux ressources REST. Le convertisseur compile automatiquement la classe Java correspondante. Ensuite il prépare le code compilé à la publication.

4.4 Editeur de service SOAP

Le module d'éditeur de service SOAP est une composante responsable sur la publication du service SOAP localement. Elle définit une interface avec le serveur TomCat afin de publier les services miroirs. Ainsi lors de l'orchestration, l'utilisateur peut invoquer des services REST en utilisant les services SOAP miroirs publié localement.

4.5 Interface Utilisateur

Le module interface utilisateur représente l'interface graphique (homme-machine) du modeleur. Les utilisateurs (concepteurs/développeurs) accèdent via cette interface aux fonctionnalités proposées par les différents modules vus précédemment. Lors de l'implémentation l'interface doit être facile à utiliser et permettre à ses utilisateurs de contrôler les opérations en cours d'exécution telle l'orchestration de service.

5. Diagramme de classe

La figure III-6 représente le diagramme de classe du modeleur proposé. Chaque module dans l'architecture est implémenté par une ou plusieurs classes java à savoir :

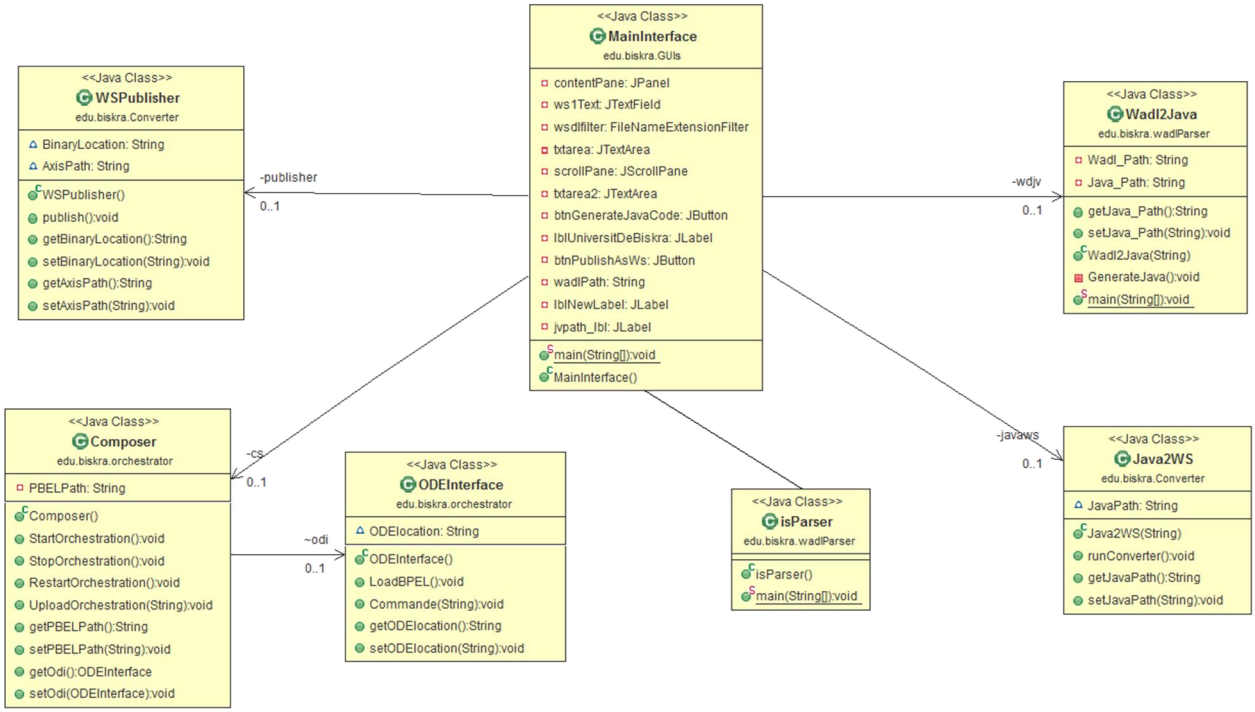

Figure III-6 Diagramme de classe du modeleur proposé

- Module **Interface utilisateur** : sera implémenté par la classe *MainInterface*
- Module **Compositeur de service** : sera implémenté par les classes *Composer* et *OEDInterface*.
- Module **Générateur du client REST** : sera implémenté par les classes *WADL2Java* et *IsParsed*.
- Module **Convertisseur Java-webService** : sera implémenté par la classe *Java2WS*.
- Module **Editeur de Service SOAP** : sera implémenté par la classe *WSPublisher*.

6. Conclusion

Nous avons consacré le présent chapitre à la présentation de la conception du modeleur proposé. Nous avons décrit le contexte et le fonctionnement du modeleur avec ses composantes principales. En outre nous avons donné l'architecture statique du modeleur de

composition de services SOAP et REST afin de l'implémenter. Les détails d'implémentations sont présentés dans le chapitre suivant.

1. Introduction

Précédemment, nous avons proposé une conception d'un modeleur de composition de services Web de différentes architectures (SOAP, REST). Dans ce chapitre, Nous décrivons les détails d'implémentation du modeleur Conçu. Premièrement, nous allons lister les langages, les environnements de développement, les outils et les APIs utilisés lors de la réalisation. Puis, nous présentons les interfaces graphiques du système de conversion REST -> SOAP réalisé qui représente le cœur du modeleur conçu.

2. Langages de programmation utilisés

Pour l'élaboration de système conçu, Nous avons utilisé les langages de programmation suivants :

2.1 Langage Java

Java est un langage de programmation et une plate-forme informatique qui ont été créés par Sun Microsystems en 1995. Beaucoup d'applications et de sites Web ne fonctionnent pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour. [20]

2.1.1 Caractéristiques du langage Java

Les caractéristiques du langage java sont :

a) **Interprété** : La source est compilé en pseudo code ou byte code puis exécuté par un interpréteur Java « Java Virtual Machine (JVM) ».

b) **Portable** : Il est indépendant de toute plate-forme. Le code reste indépendant de la machine sur laquelle il s'exécute. Cette indépendance est assurée au niveau du code source grâce à Unicode et au niveau du byte code.

c) **Orienté objet :** Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application. Java n'est pas complètement objet car il définit des types primitifs (entier, caractère, flottant, booléen,...).

d) **Simple** : Le choix de ses auteurs a été d'abandonner des éléments mal compris ou mal exploités des autres langages tels que la notion de pointeurs (pour éviter les incidents en manipulant directement la mémoire), l'héritage multiple et la surcharge des opérateurs.

e) **Sûr** : La sécurité fait partie intégrante du système d'exécution et du compilateur. Un programme Java planté ne menace pas le système d'exploitation.

f) **Multitâche :** Il permet l'utilisation de threads qui sont des unités d'exécution isolées. La JVM utilise plusieurs threads

2.2 Langage XML

XML est un standard promulgué par le W3C, l'organisme chargé de standardiser les évolutions du Web. On retrouve dans XML une généralisation des idées contenues dans HTML et SGML1.

XML permet de définir des balises et de leur associer une interprétation. Dans HTML, on n'utilise les balises que pour décrire l'aspect graphique que doit revêtir la page dans le navigateur Web. Dans XML, les balises permettent d'associer toutes sortes d'informations au fil du texte.

2.3 Langage WADL

WADL est un format de document XML qui définit un vocabulaire pour décrire les applications web. WADL vise à être le WSDL des services web RESTful. Il permet aux fournisseurs de services afin de documenter les ressources offertes, les méthodes HTTP que les ressources, et les formats de représentation pris en charge. Les clients du service peuvent utiliser WADL tant pour la documentation, la configuration et la génération de code.

Lors de l'exécution, Jersey va générer automatiquement un document WADL pour une application JAX- RS. [13]

L'application WADL est disponible dans / root / application.wadl, où root est l'URI de base de l'application Web déployée. Cette WADL généré inclut toutes les ressources profondes et autant de métadonnées qui peuvent être extrait de la compilation

3. Outils et API utilisés

Les outils et les APIs utilisés pour la réalisation du projet sont :

3.1 Eclipse

La plate-forme Eclipse est structurée comme des sous-systèmes qui sont mis en œuvre dans un ou plusieurs plug-ins. Les sous-systèmes sont construits au-dessus d'un petit moteur d'exécution.

L'environnement Eclipse a beaucoup d'avantages, on cite comme exemple :

- Plate-forme ouverte pour le développement d'applications et extensible grâce aux plug-ins
- Plusieurs versions d'un même plug-in peuvent cohabiter sur une même plate-forme
- Support multi-langages, multi-OS (Win, Linux, Mac)
- Très rapide à l'exécution
- Nombreuses fonctionnalités proposées par le JDT (Java Development Tool)
- Historique local des dernières modifications réalisées.

3.2 Serveur Apache Tomcat

Apache Tomcat est un conteneur libre de servlets et JSP Java EE. Issu du projet Jakarta [22], Tomcat est un projet principal de la fondation Apache. Il est paramétrable par des fichiers XML et de propriétés, et inclut des outils pour la configuration et la gestion. Il comporte également un serveur http. Tomcat est un serveur d'applications Java. Les applications web qu'il est capable d'exécuter sont écrites en Java. Pour traiter les requêtes entrantes, il existe des connecteurs capables de véhiculer les requêtes du serveur web frontal au serveur d'applis.

3.3 Apache Axis 2

La communauté de service Web Apache a introduit un nouveau cadre de service appelé Axis2, avec un certain nombre de nouvelles exigences, ainsi qu'un architecture très flexible et facilement extensible, pour soutenir les normes WS-* en cours ainsi que pour les futures normes.

Apache Axis2 non seulement prend en charge SOAP 1.1 et SOAP 1.2, mais il a également intégré soutien pour le REST de style très populaire de services Web. La logique de l'entreprise mise en œuvre peut offrir à la fois une interface de style WS*. [21]

3.3.1 Caractéristique d'Axis

Les caractéristiques d'Axis sont :

- **Rapid**e : Axis2 utilise son propre modèle d'objet et Stax (Streaming API for XML) de l'analyse à réaliser de façon significative plus grande vitesse que les versions antérieures d'Apache Axe.
- **Axiom** : Axis2 est livré avec son propre modèle d'objet léger. AXIOM le traitement des messages, qui est extensible, effectue hautement, et pratique.
- **Déploiement à chaud** : Axis2 est équipé de la capacité de déploiement du web services et les gestionnaires alors que le système est en marche. En d'autres termes, de nouveaux services peuvent être ajoutés au système sans avoir à arrêter le

serveur. Il suffit de déposer l'archive de services Web requis dans les services répertoire dans le référentiel, et le modèle de déploiement sera automatiquement déployé le service et le rendre disponible pour l'utilisation.

- **Flexibilité** : L'architecture Axis2 donne au développeur la liberté complète à insérer des extensions dans le moteur de traitement d'en-tête personnalisé, système gestion etc.
- **Stabilité** : Axis2 définit un ensemble d'interfaces publiées, qui changent relativement lentement, en comparaison avec le reste de l'axe.

3.4 Document Object Model API

DOM (Document Object Model) est une spécification du W3C (World Wide Web Consortium) définissant la structure d'un document sous forme d'une hiérarchie d'objets, afin de simplifier l'accès aux éléments constitutifs du document XML.

Plus exactement DOM est un langage normalisé d'interface (API, Application Programming Interface), indépendant de toute plateforme et de tout langage.

3.5 Serveur Apache ODE

Apache Orchestration Director Engine est un serveur qui exécute des processus d'affaires (business process) écrites conformément à la norme WS-BPEL. Il parle à des services Web, envoyer et recevoir des messages, manipuler les données et récupérer les erreurs.[23]

ODE prend en charge les exécutions de processus de longues et courtes vie pour orchestrer tous les services qui font partie de l'application.

3.5.1 Les caractéristiques d'Apache ODE [23]

Les caractéristiques du serveur ODE sont :

- Prise en charge à la fois la norme 2.0 OASIS WS-BPEL et l'héritage de spécification BPEL4WS 1.1 du fournisseur.
- Prise en charge de deux couches de communication : l'une fondée sur Axis2 (Web Services de transport http) et un autre basé sur le standard JBI (en utilisant ServiceMix).
- Soutien à la liaison, permettant l'invocation des services Web de type REST WSDL HTTP.
- API de haut niveau pour le moteur qui permet d'intégrer le noyau avec n'importe quelle couche de communication.

- ODE assure un déploiement à chaud des processus PBEL.

3.6 Java Runtime Environment

Cet environnement se compose de la Java Virtual Machine (JVM), de la classe standard de la plate-forme Java et des bibliothèques Java de prise en charge. L'environnement JRE correspond à la partie exécution du logiciel Java et permet d'exécuter ce dernier dans un navigateur Web.

4. Présentation des Interfaces graphiques

4.1 Interface d'accueil

La fenêtre d'accueil du modeleur est une fenêtre de présentation du cadre du projet. Elle permet à l'utilisateur via des buttons d'accéder aux fonctionnalités du modeleur ou de quitter le programme. La figure IV.1 représente la fenêtre d'accueil du modeleur.

Figure IV.1 Fenêtre principale du Système

4.2 Interface du Générateur du code

La fenêtre du générateur du code est présentée dans la figure IV.2. La fenêtre permet à l'utilisateur de spécifier le chemin du fichier WADL qui décrit le Service REST. Et des buttons pour Générée le code Java client du REST.

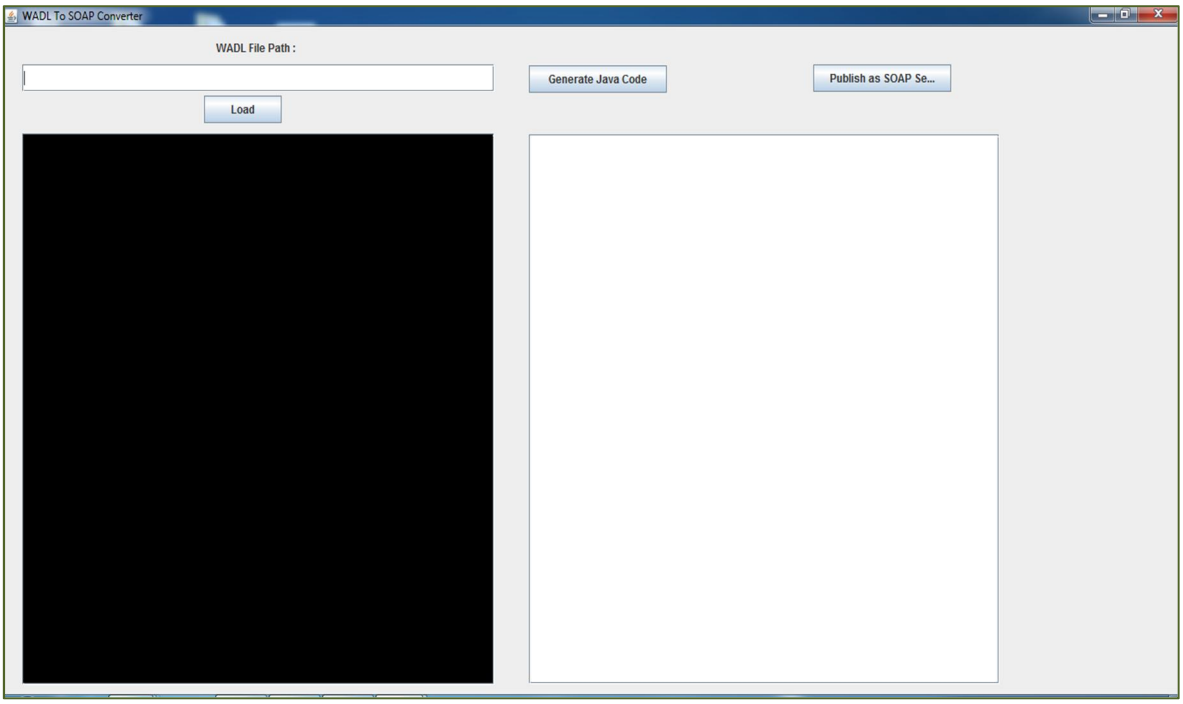

Figure IV.2. Interface du Générateur du Client REST

4.3 Exemple de Génération du code

Le code de génération se repose sur l'analyse de fichier WADL en utilisant le parseur DOM pour les fichiers XML. Les informations les ressources, les paramètres et les méthodes pour construire un code java. Ce dernier contient les méthodes, qui peuvent invoquer le service web REST. Pour illustration, l'utilisateur doit sélectionner un fichier WADL (Figure IV.3).

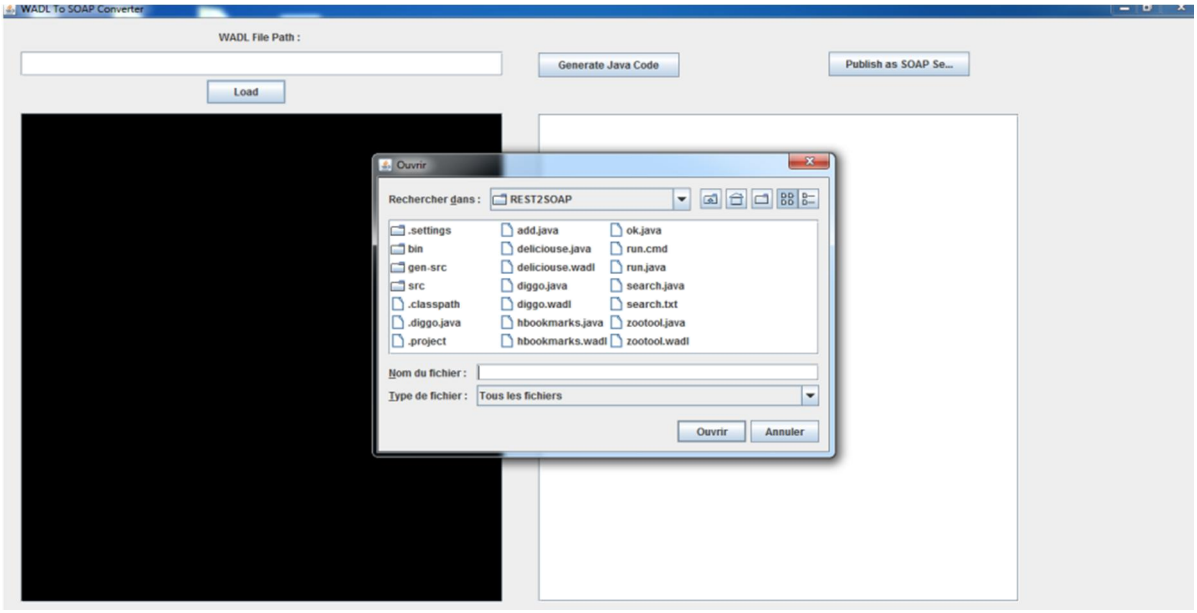

Figure IV.3 Sélection du Fichier WADL

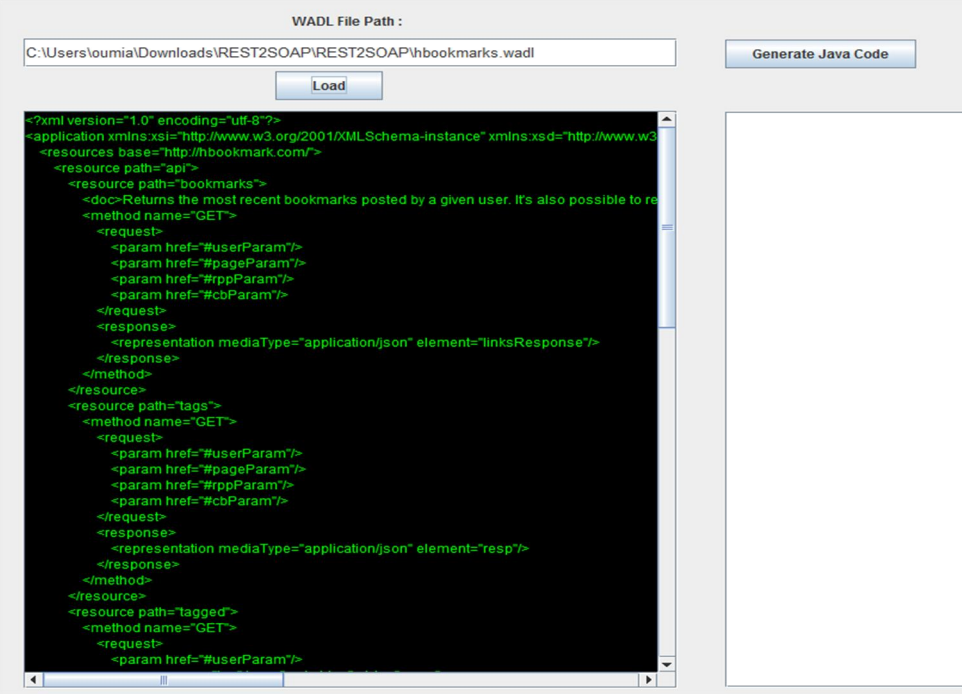

Figure IV.4 Chargement du fichier WADL

Une fois le fichier est chargé l'utilisateur ((figure IV.4) peut invoquer le générateur du code. Le résultat de génération s'afficher dans la même fenêtre et une copie du code généré est sauvegardé dans le même emplacement du fichier WADL (figure IV.5).

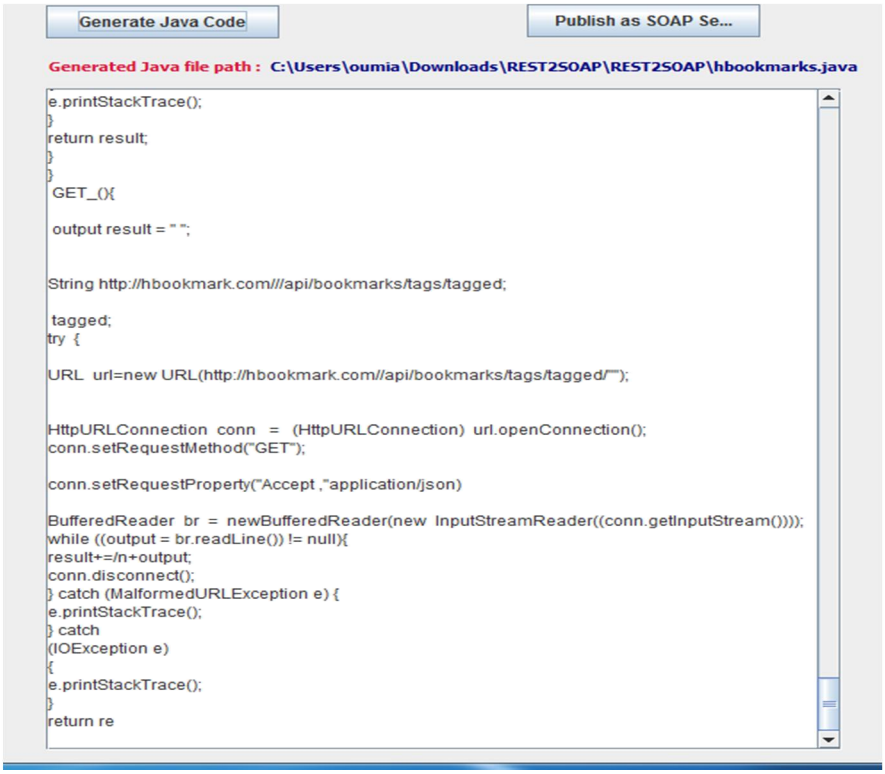

Figure IV.5 Résultat de Génération du Code client

4.4 Conversion et Déploiement du Web Service

Le déploiement consiste à compiler la classe généré, renommer le fichier "nom_class.jws" en "nom_class.java" et compiler le par commande "javac nom_classe.java", après vous devez copier la classe résultante dans le dossier "jwsClasses" ou "Classes" dépendamment du serveur Tomcat.

On peut vérifier que le déploiement est correctement effectué en accédant à l'URL :

« http://localhost:8080/axis2/services/listServices»

5. Conclusion

Dans ce chapitre on a décrit la mise en œuvre de notre modeleur. On a implémenté quelques fonctionnalités du modeleur proposé en architecture. Nous avons focalisé beaucoup plus sur la génération des services miroirs (service SOAP client au REST) à partir de leurs descriptions WADL. En outre, nous avons présenté les langages et outils utilisés pour la mise en œuvre d'un prototype du modeleur proposé.

1. Conclusion Générale

Notre projet s'inscrit dans le domaine de génie logiciel, plus particulièrement l'architecture des services Web. Dans le premier chapitre, nous avons présenté la notion orientée service. Dans le deuxième, on a présenté des généralités sur l'architecture REST et la composition des services web. En troisième chapitre, nous avons présentés la conception de notre modeleur. Finalement, nous avons présenté la réalisation du ce modeleur dans le dernier chapitre.

Le travail qui nous a été associé consiste à permettre l'échange des informations entre deux architecture SOAP et REST dans une orchestration des services web SOAP.

Pour atteindre notre objectif on a créé un modeleur qui nous permet de créer des services SOAP miroirs pour les services REST à utiliser dans des orchestrations. Ainsi, le modeleur assure un échange entre services SOAP et REST via des services intermédiaires créés et hébergés localement. La solution permet aux utilisateurs du modeleur d'adapter des services REST pour utilisation et pour recouvrement en cas de défaillance de service SOAP.

Notre future travail vise à améliorer le modeleur pour :

- Compléter les fonctionnalités non encore réalisées
- Réalisation d'un moniteur d'exécution des orchestrations de service Web.

On a rencontré beaucoup de difficulté lors de la réalisation du générateur du aux spécificités des langages de descriptions et de nombre de langage, outils et de serveur à maitriser.

Référence

[1]. Nicolai M. Josuttis : “ SOA in Practice”, O’Reilly Media, august 2007

[2]. Kin Laskey MITRE corporation; OASIS, Reference Architecture Foundation for service oriented architecture Version 1, Article du Web URL: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra.html, Consulté le 20/04/2014.

[3]. Thomas Erl: “SOA Design Patterns”**,** January 9, 2009.

[4]. Todd Biske: “SOA Governance The key to successful SOA adoption in your organization”, Packt Publishing, 2008, BIRMINGHAM – MUMBAI.

[5]. Carl jones: “Do More with SOA Integration: Best of Packt Integrate, automate, and regulate your business processes with the best of Packt's SOA books”, Packt Publishing, December 2011, BIRMINGHAM.

[6]. Doug Tidwell, James Snell et Pavel Kulchenko: “Programming Web Services with SOAP “, O'Reilly, 2001.

[7]. Martin Kalin: “Java Web Services: Up and Running by Martin Kalin”, O'Reilly,2009, USA

[8]. World Wide Web Consortium, http://www.w3.org/.

[9]. Walid FDHILA : « décentralisation Optimisées et synchronisation des Procèdes Métiers Inter-Organisationnels » thèse de doctorat, l’université Henri Poincaré Nancy1, 2011.

[10]. Ethan Cerami: “Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL”, O'Reilly, February 2002.

[11]. SOAP W3C, SOAP version 1.2 part 1: Messaging framework (second edition), Reference Architecture Foundation for service oriented architecture Version 1, Article du Web URL: http://www.w3.org/TR/soap12-part1/, Consulté le 01/04/2014.

[12]. Roy Thomas Fielding : « Architectural Styles and the Design of Network-based Software Architectures”, these de doctorat, university of california, irvine, 2000.

[13]. Bill Burke: “RESTful Java with JAX-RS”, O'Reilly, Copyright © 2010, USA.

[14]. Elie ABI LAHOUD: « composition dynamique de services : application à laconception et au développement de systèmes d'information dans un environnement distribue » thèse de doctorat, l’université de Bourgogne Laboratoire LE2I - école doctorale E2S, 11 février 2010.

[15]. Demba COULIBALY: « un langage et un environnement de conception et de développement deservices web complexes » thèse de doctorat, l'université paris dauphine Laboratoire LAMSADE, juin 2009.

[16]. YuliVasiliev :" SOA and WS-BPEL Composing Service-Oriented Solutions with PHP and ActiveBPEL"Packt Publishing , Copyright © 2007 , BIRMINGHAM – MUMBAI.

[17]. Céline LOPEZ-VELASCO : « Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation » thèse de doctorat, L'UNIVERSITE JOSEPH FOURIER, le 18 novembre 2008.

[18]. Leonard Richardson and Sam Ruby:"RESTful web services" O'Reilly Media, Copyright © 2007, United States of America.

[19]. Eclipse ; Eclipse documentation – Archived release, Article du Web URL : http://help.eclipse.org/indigo/index.jsp, Consulté le 28/04/2014.

[20]. Oracle, Equipe Java ; Ressource de sécurité Java, Article du Web URL : http://www.java.com/fr/security/, Consulté le 01/05/2014.

[21]. Deepal Jayasinghe ET Afkham Azeez: "Apache Axis2 Web Services 2nd Edition"; Packt Publishing, February 2011, BIRMINGHAM – MUMBAI.

[22]. Tomcat's Corner, Article du Web URL: http://www-igm.univ-mlv.fr/~dr/XPOSE2003/tomcat/tomcat.php?rub=3. Consulté le 16/05/2014.

[23]. The Orchestration Director Engine executes business processes written following the WS-BPEL standard, article du Web URL: http://ode.apache.org/ onsulté le 24/05/2014

Sommaire