Université AbdElhamid Mehri -Constantine 2

Faculté des Nouvelles Technologies de l'Information et de la Communication

Département d'Informatique Fondamentale et ses Applications

Spécialité : Master Réseaux et Systèmes Distribués (RSD)

Polycopié de la matière

Calcul Orienté Services (COSE)

Niveau : M2
Semestre : 3

Préparé par:

Mohamed Gharzouli

Février 2021

Syllabus de la matière

Domaine: Mathématique et Informatique

Filière: Informatique

Spécialité: Réseaux et Systèmes Distribués

Cycle : Master

Niveau: M2

Semestre: 03

Intitulé : Calcul orienté services (COSE)

Unité (UEF6): Services et distribution (SERD)

Crédits: 4

Coefficient: 4

Répartition du volume horaire: Cours : 1h30, TD : 0h, TP: 1h30, travail personnel : 5h

Objectifs de l'enseignement

Ce cours vise à fournir des réflexions récentes sur le calcul orienté service, la technologie et la gestion dans le but de comprendre une partie de plus en plus importante des Systèmes d'information souvent désignés comme «Science Service». Le cours fournit un aperçu de ce domaine et de l'expertise sur certaines des compétences informatiques clés, des technologies et techniques et leur application dans les organisations modernes.

Connaissances préalables recommandées : *Connaissances fondamentales en réseaux*

Contenu de la matière :

1. Introduction au SOC

2. Service-Oriented Architecture

3. Fondements de Service

- Service-oriented middleware
- Service description (interface)
- Service publishing
- Service discovery
- Service binding

4. Composition et Coordination de Services

- Service aggregator
- Service coordination
- Transaction management
- Service adaptation

5: Gestion et Monitoring de Service

- Installation, configuration et déploiment
- Métriques
- Balance de charges
- Gestion de changement

6. Aspects Sémantiques

- Semantique
- Qualité de Service (QoS)
- Characteristiques non fonctionnelles

7. Service-Oriented Engineering

- Service-oriented modeling and design
- Programmation orientée service
- G2nie Logiciel pour SOC
- Service versioning and adaptivity

Préface

La matière « Calcul Orienté Services » ou « Services Oriented computing » essentiellement destiné aux étudiants de *Master 2* en *Réseaux et Systèmes Distribués (RSD),* vise à présenter des concepts de base et des outils nécessaires permettant le développement de systèmes distribués orientées services. Dans cette matière, le terme « service » signifie un composant logiciel. Principalement, ce polycopié présente deux architectures avec leurs technologies de mise en œuvre permettant l'implémentation de ce type de systèmes, à savoir :

- L'architecture orientée services (SOA) et les web services SOAP
- L'architecture Microservices et les web services REST.

Aussi, d'autres concepts sous-jacents sont décrits dans ce document comme : la composition de services, scénarios de développement, QoS …etc.

Il est important de noter que toutes les notions citées dans le syllabus ci-dessus sont présentées dans ce polycopié. Cependant, l'organisation des chapitres a été adaptée parce que quelques concepts (principalement ceux des chapitres 5 et 7) sont traités au niveau de la partie « Travaux Pratiques ». Ces derniers sont présentés sous forme de tutoriaux dans ce polycopié.

Sommaire

Chapitre 1 : Architecture orientée services (Services Oriented Architecture : SOA)

1. Introduction

Les changements dans la gestion des entreprises sont rapides, ce qui influence aussi l'évolution des systèmes d'information qui ont besoins de s'adapter à ces changements et de se conformer aussi au développement des nouvelles technologies. A cause de cette évolution rapide, les systèmes d'information sont devenus hétérogènes, et ils contiennent une gamme de différents systèmes, d'applications, de technologies, et d'architectures **[01]**.

Actuellement, les solutions des nouvelles générations des SI distribués sont tournées vers les processus métiers et les échanges interentreprises: elles prennent en compte les nouveaux modèles économiques créés et promus par Internet et ses technologies. De plus, les progiciels de gestion intégrés de la nouvelle génération apportent une quantité de nouveaux modules organisés autour d'une nouvelle vision pour les systèmes d'information et les systèmes distribués **[16]**.

Dans ce contexte, nous allons présenter, dans ce chapitre, l'architecture SOA qui utilise les « services » pour décrire les composants du système et la manière dont ils interagissent **[02]**. Nous allons décrire comment cette architecture est utilisée pour développer des systèmes distribués en utilisant des composants logiciels (services) répartis sur des différents systèmes d'information.

2. Définition

SOA est une architecture ou modèle qui permet de construire des systèmes d'information distribués basées sur les services **[03]**.

Le service est généralement un composant logiciel exécuté par un **fournisseur** à la demande d'un **client**. Cependant, l'interaction entre les deux est faite par le biais d'un médiateur (qui peut être un bus) responsable de la mise en relation des composants **[04]**.

L'architecture orientée services est une vision pour le système informatique. Ce dernier n'est plus décrit comme un ensemble d'applications mais comme un ensemble de services. Donc, plutôt que de privilégier une architecture applicative basée sur des contraintes techniques, l'architecture orientée service (SOA) propose de découper les fonctionnalités d'une application en services métier, réutilisables dans d'autres applications. En se concentrant sur les services, les applications sont agrégées pour fournir des processus opérationnels plus riches et plus significatifs **[16]**.

Les services d'une architecture SOA répondant, notamment, aux critères suivants:

- **Faiblement couplés** : les applications traditionnelles incluent dans leur code les données métiers de l'entreprise. Elles sont complètement liées aux systèmes pour

lesquels elles ont été conçues. Cette contrainte implique la difficulté de toute demande de modification, qu'elle concerne l'accès aux données, les règles de gestion ou celles de présentation. Un faible couplage permet une scission des aspects métiers du code qui permettra une simple reconfiguration des processus quand les fonctions métiers évoluent.

- **Distribués** : les services qui composent les applications peuvent être physiquement répartis sur des différents systèmes dans l'entreprise, mais aussi au-delà.
- **Invocables et publiables :** les services doivent être invocables et publiables quels que soit les systèmes utilisés.

L'aspect le plus important de l'architecture SOA est qu'elle permet de séparer l'implémentation du service de son interface (Description).

3. Les composants de la SOA

Une architecture de services (SOA) est constituée de 4 composants primaires. Le premier est le prestataire de services (le service réel). Vient ensuite le demandeur du service, autrement dit le composant qui accède au service. Enfin, l'agence de services fournit des services de découverte et d'enregistrement.

Le paradigme "découvrir, interagir et exécuter" comme montré dans la figure 1.1, ce paradigme permet au consommateur du service (client) d'interroger un **annuaire** pour le service qui répond à ses critères. Si l'annuaire possède un tel service, alors il renvoie au client le **contrat** du service voulu ainsi que son adresse. SOA consiste en quatre entités configurées ensemble pour supporter le paradigme découvrir, interagir et exécuter **[02]**.

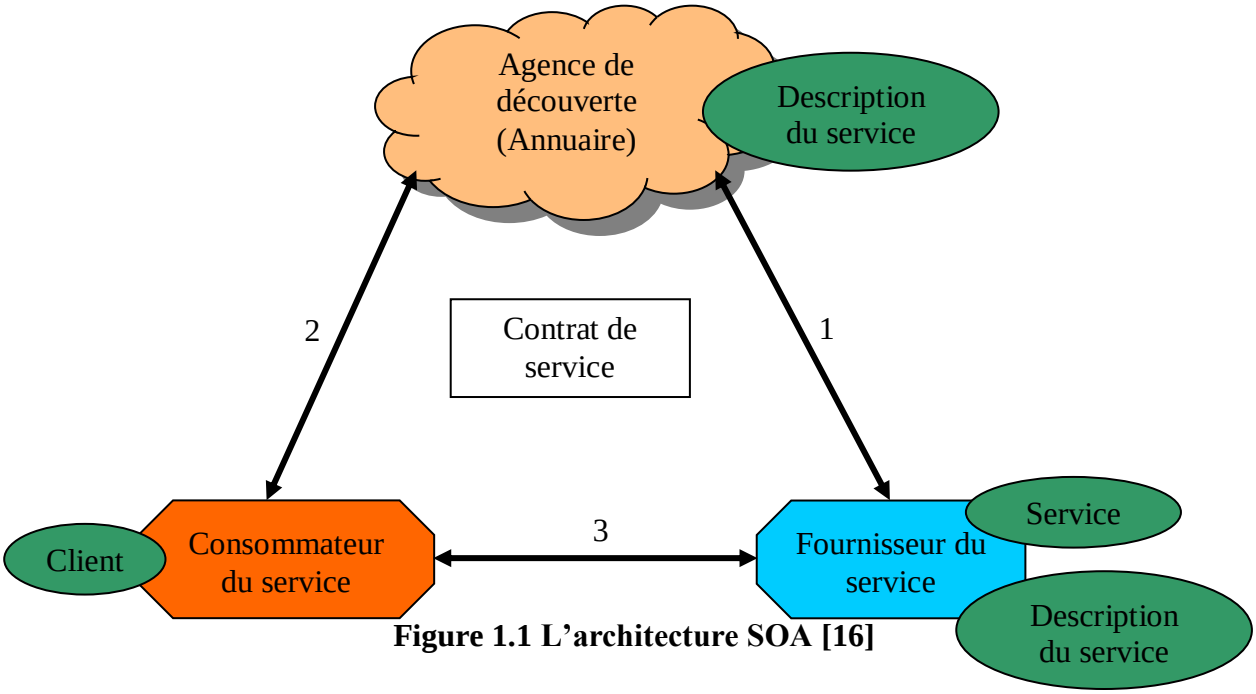


Figure 1.1 L'architecture SOA [16]

3.1. Le consommateur de service

Le consommateur du service : d'une perspective architecturale, c'est l'application qui recherche et invoque ou débute une interaction avec un service. Le consommateur du service peut être une personne qui utilise un navigateur comme il peut être un programme (un autre Web service, par exemple). Nous pouvons dire que dans un modèle d'échange de messages client/serveur, le consommateur du service est le client **[16]**. C'est l'entité qui initie la localisation du service dans l'annuaire, envoie une requête à travers un protocole et reçoit le résultat de l'exécution de la fonction exposée par le service **[02]**.

3.2. Le fournisseur de service

Du point de vue business, c'est le propriétaire du service. D'une perspective architecturale, c'est la plateforme ou l'environnement d'exécution du Web service. D'une autre façon, nous pouvons dire que dans un modèle d'échange de messages client/serveur le fournisseur du service est le serveur **[16]**.

Le fournisseur de service est une entité adressable via un réseau, il accepte et exécute les requêtes venant d'un client. Le fournisseur de service publie le contrat de service dans l'annuaire pour qu'il puisse être accédé par les clients **[02]**.

3.3. L'annuaire de service

Agence de découverte c'est un ensemble de descriptions de services publiés par les fournisseurs de services. Il peut être centralisé ou distribué. Le consommateur du service peut chercher des services et obtenir des informations sur un service donné à partir de sa description **[16]**.

L'annuaire de service sauvegarde les contrats du fournisseur de service et présente ces contrats aux éventuels clients **[02]**.

3.4. Le contrat de service

Le contrat spécifie la manière dont le client de service va interagir avec le fournisseur de service. Il spécifie le format de la requête et la réponse du service **[02]**.

4. Web services SOAP

D'après la définition, SOA est une approche architecturale qui ne fait aucune hypothèse sur la technologie de mise en œuvre. Pour cette raison, la confusion entre SOA et les web services est une erreur. Cependant, la proposition des technologies Web services a été

menée dans l'objectif de répondre au mieux aux enjeux de l'architecture SOA **[05]**. Dont l'objectif de fournir l'interopérabilité des applications, les web services ont été proposés pour fournir les bases technologiques nécessaires pour réaliser cet objectif indépendamment des différentes plateformes, différents systèmes d'exploitation et différents langages de programmation **[05]**, **[06]**.

4.1. Définition

Un web service est une application logicielle identifiée par une URI, qui possède une interface publique définie en utilisant XML. Sa définition peut être découverte par d'autres systèmes. Ces systèmes peuvent interagir avec le web service selon la manière prescrite par sa définition, en utilisant des messages basés sur XML et portés par des protocoles internet **[07]**.

4.2. Les standards des web services

Dans cette section, nous allons présenter trois spécifications des Web services : SOAP, WSDL et UDDI. Ces spécifications pourront être mises en œuvre dans le cadre d'une architecture SOA basée sur les Web service **[07]**.

4.2.1. SOAP (Simple Object Access Protocol)

SOAP est un protocole basé sur XML, qui permet aux applications d'échanger des informations à travers HTTP (ou autres protocoles) **[07]**:

- SOAP est l'acronyme de Simple Object Access Protocol (ou Services Oriented Architecture Protocol)
- SOAP est un protocole de communication entre les applications à travers internet (communication entre clients et services)
- SOAP a un format d'envoi de messages
- SOAP est indépendant de toute plateforme et de tout langage
- SOAP est simple et extensible
- SOAP permet d'éviter les difficultés causées par les pare-feux
- SOAP est un standard du W3C

a) Objectifs de SOAP

SOAP est un protocole XML permettant la communication entre composants, logiciels et applications en s'appuyant sur des protocoles standards de type HTTP, SMTP,…etc. Sa première version SOAP1.1 proposée à W3C en 2000 par UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, Microsoft et SAP. En suite, il fut

standardisé par W3C pour la version SOAP 1.2. SOAP fournit un moyen de communication entre des applications exécutées sur différents systèmes d'exploitation, avec différentes technologies et différents langages **[16]**.

SOAP n'est pas lié à un protocole particulier. Il n'est pas non plus lié à un système d'exploitation ni à un langage de programmation, donc, théoriquement, les clients et serveurs de ces dialogues peuvent tourner sur n'importe quelle plate-forme et être écrits dans n'importe quel langage du moment qu'ils puissent formuler et comprendre des messages SOAP. En tant que tel, il s'agit d'un important composant de base pour développer des applications distribuées qui exploitent des fonctionnalités publiées comme services par des intranets ou Internet.

b) Syntaxe de SOAP

Un message SOAP est un document XML ordinaire qui contient les éléments suivants **[11]** :

- L'élément *Envelope* qui identifie le document XML comme étant un message SOAP
- L'élément *Header* qui est optionnel et qui contient des informations d'entête
- L'élément *Body* qui contient l'appel ainsi que la réponse retournée
- L'élément *Fault* qui est optionnel et qui fournit des informations sur d'éventuelles erreurs survenues lors de l'analyse du message

Tous ces éléments cité ci-dessus sont déclares dans les *namespace* de l'enveloppe SOAP.

https://www.w3.org/2003/05/soap-envelope/ (*http://www. w3.org/2001/12/soap-envelop)*

Et le *namespace* pour le SOAP encoding et les types de données :

http://www.w3.org/2003/05/soap-encoding (***http://schemas.xmlsoap.org/soap/encoding***)

c) Squelette d'un message SOAP

```
< ?xml version= "1.0" ?>
 <soap:Envelope xmlns="https://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Header>
……..
</soap:Header>
<soap:Body>
……..
  <soap:Fault>
…
 </soap:Fault>
<soap:Body>

</soap:Envelope>
```

L'élément Envelope

Cet élément est la racine de tout message SOAP, il définit le document XML comme étant un message SOAP.

Noter l'utilisation du namespace xmlns:soap. Il doit toujours avoir la valeur :

https://www.w3.org/2003/05/soap-envelope/ (*http: //www. w3.org/2001/12/soap-envelop*)

Et il définit l'enveloppe comme étant une enveloppe SOAP.

```
< ?xml version= "1.0" ?>
 <soap:Envelope xmlns="https://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
……................................
Information et message
</soap:Envelope>
```

Le namespace xmlns : soap

Un message SOAP doit toujours avoir un élément Envelope associé au namespace : ***https://www.w3.org/2003/05/soap-envelope/*** (http ://www. w3.org/2001/12/soap-envelope)

Si un autre namespace est utilisé, l'application doit générer une erreur.

L'attribut encodingStyle

Un message SOAP n'a pas d'encodage par défaut. Pour définir les types de données utilisés dans le document, il utilise cet attribut. Ce dernier peut apparaître dans n'importe quel élément SOAP, et il sera appliqué au contenu de cet élément ainsi que tous ses éléments fils.

Syntaxe

```
soap:encodingStyle="URI">
```

Exemple

```
< ?xml version= "1.0" ?>
 <soap:Envelope xmlns="https://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
……................................
Information et message
</soap:Envelope>
```

L'élément Header

C'est un élément optionnel et contient des informations spécifiques à l'application (par exemple des informations sur l'authentification) sur le message SOAP. Si cet élément est présent, il doit être le premier fils de l'élément Envelope.

Note Tous les éléments fils de l'élément Header doivent être qualifiés par un namespace.

```
< ?xml version= "1.0" ?>
 <soap:Envelope xmlns="https://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Header>
<m:Trans xmlns :m="……." soapmustUnderstand="1" >234</m:Trans>
</soap:Header>
……
   ……..
</soap:Envelope>
```

Dans l'exemple ci-dessus, l'élément *header* a comme fils l'élément *Trance* avec la valeur 234. SOAP définit aussi des attributs pour l'élément Header comme *role* et *mustUnderstand*. Dans cet exemple, l'attribut *mustUnderstand* a la valeur 1, il indique si l'élément header doit être traité ou pas par le récepteur. Dans ce cas, la valeur 1 signifie « Oui ».

Syntaxe

```
soap:mustUnderstand="0|1"
```

Exemple

```
< ?xml version= "1.0" ?>
 <soap:Envelope xmlns="https://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Header>
<m:Trans xmlns :m="……." soapmustUnderstand="1" >234</m:Trans>
</soap:Header>
……
   ……..
……
</soap:Envelope>
```

L'élément SOAP Body

Cet élément contient le message envoyé au récepteur. Le fils de l'élément Body peut être qualifié par un namespace.

Exemple

```
< ?xml version= "1.0" ?>
 <soap:Envelope xmlns="https://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Body>
<m:GetPrice xmlns :m="……................." >
< m:Item >Apples</ m:Item>
</m:GetPrice>
<soap:Body>
</soap:Envelope>
```

L'élément SOAP Fault

Un message d'erreur est porté par cet élément. Si un élément Fault est présent, il doit apparaître comme étant fils de l'élément Body. Un élément Fault ne peut apparaître qu'une seule fois dans un message SOAP.

L'élément Fault a au moins deux éléments fils **[11]**:

1. Code (obligatoire).
2. Reason (obligatoire).
3. Node (optionnel)
4. Role (optionnel)
5. Detail (optionnel).

Les codes des erreurs

Pour décrire une erreur l'élément *code* utilise les valeurs suivantes dépendamment de l'erreur qui s'est produite **[11]**:

Element	Description
VersionMismatch	Namespace du bloc Envelope non valide
MustUnderstand	Un élément fils du bloc header qui devait absolument être compris qui ne l'a pas été
DataEncodingUnknown	L'encodage du bloc header ou un fils du bloc body et non valide
Sender	Message mal formé ou non valide
Receiver	Un problem lors du traitement d'un message

4.2.2. WSDL

WSDL est un langage basé sur XML utilisé pour décrire les web services et comment les accéder **[07]**:

- WSDL est l'acronyme de Web Service Description Language
- WSDL est écrit en XML (et XML schéma pour décrire les types de données)
- WSDL est utilisé pour décrire les web services
- WSDL est un standard du W3C

WSDL décrit les web services

Le document décrit le web service. Il spécifie la localisation du web service et les opérations (méthodes) qu'expose ce web service.

WSDL est une recommandation du W3C

WSDL est devenu une recommandation du W3C en Juin 2007 **[12]**.

La structure d'un document WSDL

WSDL décrit un web service en utilisant ces principaux éléments **[15]**, **[16]**:

- **Types:** précise les types de données complexes, pour lequel WSDL emploi XML Schema.
- **Message:** l'abstraction décrivant les données échangées entre services.
- **Operation:** l'abstraction décrivant une action implémentée par un service Web.
- **Port types:** Cet élément définit de manière abstraire une collection d'opérations ou d'actions, chaque opération est déclenchée par une requête, puis génère une réponse.
- **Binding (liaison):** Cet élément spécifie de manière concrète le protocole de communication (exemple : SOAP1.1, HTTP, MIME (*Multipurpose Internet Mail Extension*), …) et le format des donnés pour les opérations et messages définit par un type de port particulier.
- **Port:** Cet élément définit un point de communication unique avec l'adresse réseaux à laquelle elle est liée.
- **Service:** Cet élément définit une collection d'adresses (ports) permettant d'invoquer un service. Il sert à regrouper un ensemble de points de communication. En général ; il correspond à une URL invoquant un service SOAP.

Chaque document WSDL peut être documenté grâce à une balise **<documentation>**. Cet élément est facultatif.

Un document WSDL est divisé en deux parties : l'interface du service et son implémentation. L'interface du service est la partie réutilisable de la définition du service, elle

peut être référencée par de multiples implémentations du service. Cette partie contient les éléments : WSDL : binding, WSDL : portType, WSDL : message et WSDL:type **[16]**.

Dans l'élément WSDL:portType, les opérations d'un service Web sont définies. Ces opérations définissent comment un message XML peut apparaître dans les flux des données entrants et sortants. Une opération est comprise comme une signature d'une méthode dans un langage de programmation OO. L'élément WSDL:message spécifie comment les types de données XML constituent les différentes parties d'un message. L'élément WSDL:message est utilisé pour définir les paramètres entrants et sortants d'une opération. L'utilisation des types de données complexes dans le message est décrite dans l'élément WSDL : types. L'élément WSDL : binding décrit le protocole, le format de données, la sécurité et autres attributs pour une interface d'un service particulier (WSDL : portType) **[16]**.

La définition d'implémentation d'un service est un document WSDL qui décrit comment une interface particulière d'un service est implémentée par un fournisseur donné. Un service Web est modélisé par un élément WSDL:service. Un élément service contient une collection (habituellement une seule) d'éléments WSDL:port. Un port associé un «endpoint» (par exemple une adresse d'un endroit sur le réseau ou une URL) à un élément WSDL : binding d'une définition d'interface d'un service **[16]**.

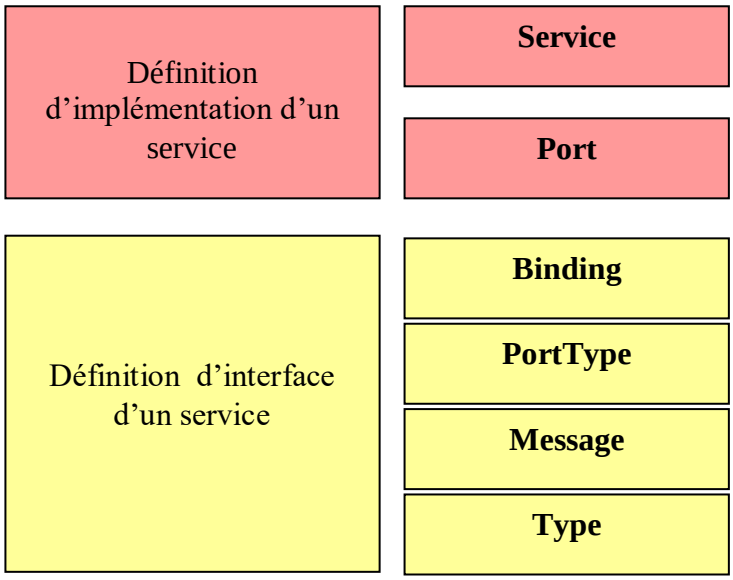


Figure 1.2 : Description WSDL d'un service [29], [16].

La structure principale d'un document WSDL ressemble à :

```
<definitions>
<types>
Définition des types
</types>
<message>
Définition des messages
</message>
<portTypes>
Définition des opérations
</portTypes>
<message>
Définition des liaisons
</message>
<Service>
<port>
…….
Définition des implémentations
</service>
<definitions>
```

4.2.3. UDDI

Initialement définie par Ariba, IBM et Microsoft, UDDI est un protocole d'annuaire permettant aux entreprises de publier et de découvrir, d'une manière standard, des informations relatives aux fournisseurs et aux types de services qu'ils proposent. Ainsi, les clients peuvent savoir quels sont les services fournis par chaque fournisseur et les concepteurs de logiciels clients peuvent apprendre ce qu'ils ont besoin de connaître pour créer ces clients. UDDI est une technologie qui s'articule autour des protocoles HTTP et SOAP, ainsi que du langage XML. Les spécifications UDDI définissent les types d'annuaire de services Web distribues *: pages blanches* (nom de l'entreprises, adresse, contacts), *pages jaunes* (services classés par catégories industrielles) et *pages verts* (information d'implémentation des services Web proposés). Ainsi, UDDI se présente comme un ensemble de bases de données utilisées par les entreprises pour enregistrer leurs services Web ou pour localiser d'autres services Web. Grâce à UDDI, les entreprises peuvent enregistrer des données les concernant, des renseignements sur les services qu'elles offrent et des informations techniques sur le mode d'accès à ces services. Une fois l'enregistrement terminé, les informations sont automatiquement répliquées sur l'ensemble des annuaires. Ce fonctionnement permet aux services d'être découvert par un plus grand nombre d'entreprises **[09], [16]**.

a) Les types de données UDDI :

L'XML schéma d'UDDI fournit 4 éléments obligatoires pour accéder et utiliser un Web service **[08]**, **[16]** (voir la figure ci-dessous)

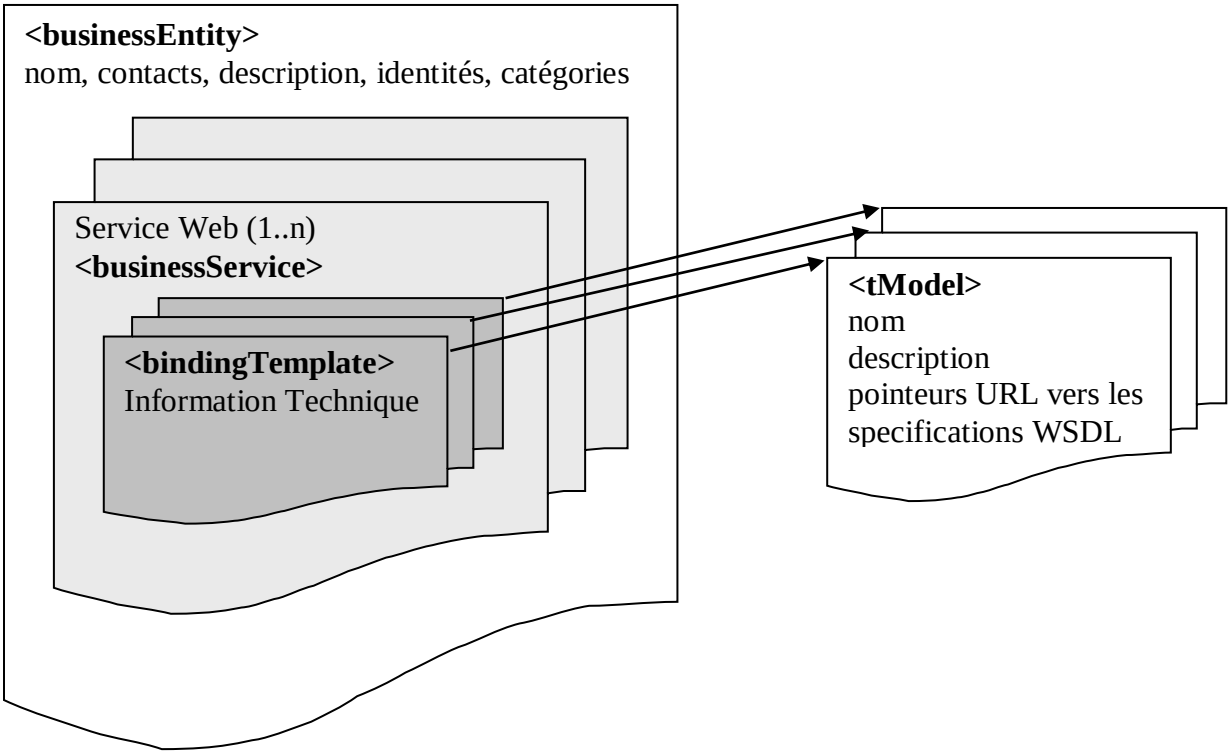


Figure 1.3 : Structure des informations dans UDDI [08].

- **L'entité commerciale : <BusinessEntity>**

Cet élément est la racine du document UDDI décrivant l'enregistrement du ou des web services d'un même fournisseur. Il contient l'identité de ce dernier, son adresse physique et électronique ainsi que des qualifications ou des mots-clés faisant référence aux taxonomies industrielles standards **[10]**.

- **La description des services : <BusinessServices >**

A l'intérieur de l'élément précédent, les services proprement dits sont délimites par la balise *businessService*. Chacun de se sous éléments contient pour l'essentiel le nom et la description du service, sa catégorie dans une taxonomie propre à UDDI, des clés de recherche et des pointeurs vers des classes de liaisons (bindingTemplates) **[10]**.

- **La liaison UDDI : <BindingTemplates >**

Comme dans WSDL, la liaison UDDI regroupe, pour un protocole de communication donné, les données techniques nécessaire à l'exploitation du web services par un programme : adresse IP, noms de domaines et le cas échéant, des informations sur les modalités d'usage du service (hébergement, paramétrage initial, et.). Un même service peut disposer de plusieurs points d'accès, par exemple, selon des protocoles différents (SOAP, SMTP…) **[10]**.

- **Les modèles données *:* <tmodels >**

Le model est une structure « creuse » qui est utilisée dans la description des entités commerciales comme référence à un autre document décrivant un modèle de données ou toute autre information nécessaire aux requêtes de recherche et aux interactions avec l'entité commerciale considère. Dans le cas courant de l'enregistrement d'un web service, par exemple, le *tmodel* pointera, en général, vers le document WSDL décrivant l'interface publique du service. Plus généralement, le tmodel peut renvoyer à d'autres documents spécifiant par exemple, les conventions employées dans les échanges ou bien encore les taxonomies industrielles les attributs peuvent faire référence, etc. **[10]**.

b) Publication d'un document WSDL dans un annuaire UDDI

Pour enregistrer un service dans un annuaire UDDI, il faut publier ses deux documents WSDL : le document interface qui contient la définition du service (<Types>, <Message>, <Portype>, <binding>) et le document implémentation qui contient la description du service lui-même (<Service>, <Port>) où ce dernier importe le document interface **[16]**.

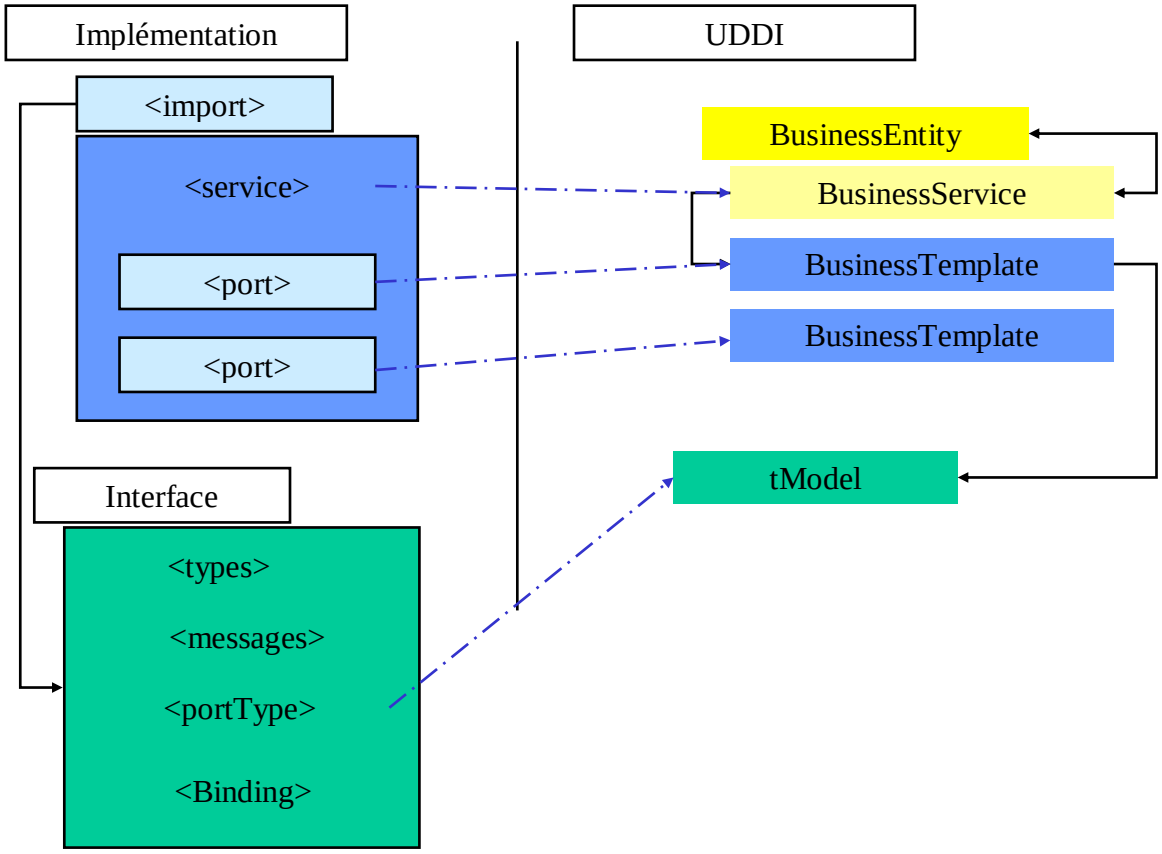


Figure 1.4 : Enregistrement du document WSDL dans un annuaire UDDI [16]

5. Cycle de vie de développement de services Web

Comme nous avons déjà vu, le service Web a trois composants principaux : le demandeur du service, le service d'annuaire d'enregistrement et le fournisseur de services. Le cycle de vie de développement des services Web décrit ci-dessous prend en considération les rôles de fournisseur de services et de demandeur de services, il est défini en quatre phases **[16]**, **[28]**:

- **Construction :** la phase de construction inclut le développement et le test de l'implémentation du service Web, la définition de la description de l'interface et la définition de la description d'implémentation. Il y a trois possibilités pour fournir une implémentation d'un service Web : soit par la création de nouveaux services, la transformation des applications existantes en services Web ou bien par la composition de nouveaux services Web et applications existantes.

Le développement d'un nouveau service Web implique l'utilisation des langages de programmations et les modèles appropriés pour l'environnement de développement du fournisseur du service.

La transformation des applications existantes implique la génération de la définition de l'interface du service et l'enveloppement (Wrapping) de l'application existante pour permettre l'interaction entre l'application existante et le monde extérieur. L'enveloppe (Wrapper) est un logiciel qui communique avec l'application existante telle qu'elle a été conçue et offre à l'extérieur une nouvelle API (Application Programming Interface) où une interface graphique peut être développée pour cette API. Il est intéressant de faire la remarque que ce travail est réalisé pour les *anciennes applications* (*Legacy systems*).

Pour la troisième possibilité qu'est la composition de nouveaux services Web à partir des services Web existants implique l'ordonnancement et l'orchestration de flux des messages entre les programmes directement ou bien avec la technique de Workflow (ou ESB).

- **Déploiement :** la phase de déploiement inclut la publication de la description du service Web dans un service d'enregistrement, déploiement du code exécutable du service Web dans l'environnement d'exécution et l'intégration avec les legacy systems. Pour les services Web qui représentent des applications existantes, le déploiement peut inclure seulement le service Wrapper parce que l'application peut être déjà déployée.

- **Exécution :** dans cette phase le consommateur du service peut trouver la description du service et invoquer toutes les opérations définies dans le service.

- **Gestion :** cette phase couvre la gestion et l'administration de l'application du service Web : sécurité, disponibilité,…etc.

5.1. Processus de développement des services Web côté fournisseur

Il existe quatre méthodes de développement d'un service Web pour un fournisseur de services, l'utilisation de ces méthodes dépend de l'existence de l'application/service et l'interface du service. Ces méthodes sont représentées dans le tableau suivant **[16]**, **[28]**:

	Interface du service n'existe pas	**Interface de service existe déjà**
Nouveau Web service	Green-Feild	Top-Down
Application/service existe déjà	Bottom-up	Meet-in-the-Middle

Les méthodes de développement des services Web pour un fournisseur du service [28].

Les quatre méthodes indiquées dans le tableau sont décrites dans ce qui suit **[16]**, **[28]**:

5.1.1. Scénario Green-Field

Cette méthode décrit comment une nouvelle description de l'interface du service sera créée pour un nouveau service Web. Dans se scénario le processus de développement est comme suit :

Construction :

- Création d'un nouveau service Web : cette étape inclut la conception, le développement et le test du service.
- Création ou génération de l'interface WSDL, qui décrit le service et la manière de son invocation. La description de l'interface du service peut être générée à partir de l'implémentation du service (actuellement la plupart des plateformes de développement fournissent une génération automatique du code WSDL qui décrit le service Web).

Déploiement :

- Publication de l'interface du service : La définition d'interface de service doit être publiée avant que le service ne soit déployé. L'interface du service est utilisée par un demandeur

du service pour déterminer comment accéder à un service. Un service d'enregistrement (UDDI par exemple) est utilisé pour la fonction de publication, ce dernier peut être publique ou privé tout dépend du contexte d'utilisation.

- Déploiement du service Web : le code exécutable pour le service doit être fourni dans l'environnement d'exécution.
- Création de la définition de l'implémentation du service (la description peut être créé avec WSDL). Cette description peut être créée à partir de la façon et où le service a été déployé.
- Publication de la définition de l'implémentation du service dans un service d'enregistrement.

Exécution :

- Exécution du service Web : le service Web est maintenant opérationnel, déployé dans l'environnement d'exécution et publié dans un service d'enregistrement. Il peut donc, être invoqué par les consommateurs.

Scénario Top-Down

Cette méthode est utilisée lorsqu'un nouveau service Web peut être développé en se conformant à une interface existante. Ce type d'interface de service est habituellement une partie d'une norme d'industrie qui peut être implémentée par n'importe quel nombre de fournisseurs de services. Dans se scénario, le processus de développement est comme suit :

Construction :

- Trouver l'interface du service : cette dernière est supposée existante et publiée dans un service d'enregistrement. Donc, la fonction de recherche est faite au niveau d'un service d'enregistrement (annuaire UDDI, par exemple).
- Création d'un gabarit (template) de l'implémentation du service : en utilisant la définition de l'interface du service trouvé, un gabarit d'implémentation du service peut être généré. Ce dernier doit contenir toutes les méthodes et les paramètres qui doivent être implémentés par le service Web correspondant à l'interface.
- Développement d'un nouveau service : Le gabarit créé dans l'étape précédente est utilisé pour concevoir et développer l'application qui représente le service Web. Cette étape inclut le développement et le test du service Web.

Déploiement :

La phase de déploiement de cette méthode (Top-Down) est presque semblable à la phase de déploiement dans la méthode Green-Field. La seule différence c'est que la description de l'interface est déjà publiée.

- Déploiement du service Web.
- Création de la définition de l'implémentation du service.
- Publication de la définition de l'implémentation du service.

Exécution :

- Exécution du service Web.

5.1.2. Scénario Bottom-Up

Cette méthode est utilisée lorsque les fonctionnalités existantes d'une application sont exposées comme un service Web. L'application existante peut être implémentée sous forme d'un EJB (Enterprise Java Beans), servlet, C++, classe java,…etc.

Construction:

- Génération de l'interface du service : l'interface est générée à partir de l'implémentation de l'application qui représente le service Web.

Déploiement :

- Déploiement de service Web.
- Création de la définition de l'implémentation du service.
- Publication de l'interface du service.
- Publication de la définition de l'implémentation du service.

Exécution :

- Exécution du service Web.

5.1.3. Scenario Meet-in-the-Middle

Cette méthode est utilisée lorsque la description de l'interface du service existe déjà et l'application qui représente le service existe aussi.

Le problème qui peut être posé dans cette situation est que les fonctionnalités de l'application existante ne peuvent pas correspondre exactement à la description de l'interface du service Web désiré. Pour régler ce problème, nous pouvons créer un enveloppe (Wrapper) pour l'application existante qui utilise la définition de l'interface du service. Dans se scénario le processus de développement est comme suit :

Construction :

- Trouver l'interface du service qui sera utilisée pour implémenter le service Web.
- Création d'un gabarit (template) de l'implémentation du service en utilisant la définition de l'interface du service trouvé
- Développement du service d'enveloppement (service Warpper).

Déploiement :

- Déploiement de services Web.
- Création de la définition de l'implémentation du service.
- Publication de la définition de l'implémentation du service.

Exécution :

- Exécution du service Web.

5.2. Processus de développement des services Web côté consommateur

Le demandeur du service exécute les mêmes étapes du cycle de vie que le fournisseur du service. Mais, le demandeur du service exécute des tâches différentes pendant chaque phase. Les tâches du temps de construction (build time) pour le demandeur de service sont basées sur la méthode de liaison au service Web.

Il y a deux façons pour se lier à un service donné : liaison statique et liaison dynamique. La liaison statique est utilisée lorsqu'il y a une seule implémentation du service qui peut être utilisée au moment de l'exécution. La liaison dynamique est utilisée quand un demandeur veut invoquer un service Web, mais son emplacement n'est pas connu jusqu'au moment de l'exécution.

Un service Proxy est généré à partir de l'interface du service. Il contient le code nécessaire pour accéder et invoquer un service Web.

Il y a trois scénarios possibles pour la consommation d'un service Web, selon le type de liaison. La liaison statique est utilisée seulement au temps de constructions, cependant la liaison dynamique peut être utilisée au temps de construction et au temps d'exécution. La liaison statique ne peut être utilisée au temps d'exécution parce que le consommateur du service demande toutes les informations nécessaires au temps de construction **[16]**, **[28]**.

	Liaison statique	Liaison dynamique
Construction	Static binding	Build-time dynamic binding
Exécution	[Non applicable]	Runtime dynamic binding

Les méthodes de développement des services Web pour un consommateur du service [28].

5.2.1. Scénario Static binding

Comme nous l'avons déjà indiqué, la liaison statique (static binding) est utilisée lorsqu'il y a une seule implémentation du service qui peut être invoqué au moment de l'exécution. La liaison statique est édifiée au temps de construction en localisant la définition de l'implémentation du seul service Web qui sera invoqué par le consommateur. La définition d'implémentation du service contient une référence à l'interface de service qui sera utilisé pour générer le code Proxy du service. Le Proxy est déployé avec l'application client. Le consommateur peut invoquer les opérations du service Web.

Les étapes suivies dans ce scénario sont **[28]**:

Construction:

- Chercher la définition d'implémentation du service : durant la construction, l'utilisateur du service doit localiser la description de l'implémentation du service dans un service d'enregistrement (un annuaire UDDI universel ou privé). La description de l'implémentation du service contient une référence à la description de l'interface du service et à l'endroit du fournisseur où le service est accessible.
- Générer le Proxy : la définition de l'interface et les informations récupérées sur l'endroit du service sont utilisées pour la génération du Proxy d'implémentation. Le Proxy se conformera à l'interface du service et essayera d'accéder au service Web toujours au même endroit.
- Tester le Proxy : avant le déploiement, le service Proxy devra tester pour vérifier qu'il peut interagir avec le service Web spécifié.

Déploiement :

- Déployer le Proxy : après la phase de test, le Proxy devra être déployé avec l'application cliente dans l'environnement d'exécution.

Exécution

- Invoquer le service Web : exécuter l'application client qui utilisera le Proxy pour appeler le service Web.

5.2.2. Scenario Build-time dynamic binding

Ce type de liaisons est utilisé lorsque le demandeur veut invoquer un type particulier de services Web, mais l'implémentation de ce dernier n'est pas connue jusqu'au moment de l'exécution. Le type de services est défini dans une interface de service.

Les étapes suivies dans ce scénario sont :

Construction:

- Trouver la définition de l'interface du service : l'interface du service contient seulement la définition abstraite des opérations du service Web (réellement, il y a plusieurs implémentations qui fournissent les mêmes opérations et correspondent à l'interface du service. Mais chaque implémentation a sa propre description).
- Générer un Proxy générique: en utilisant la définition de l'interface, un Proxy peut être généré. Il peut être utilisé pour accéder à toute implémentation de la même interface. Il contient le code pour localiser une implémentation du service à travers une opération de recherche dans un service d'enregistrement (un annuaire UDDI par exemple).
- Tester le Proxy : le test peut être accompli en trouvant une implémentation qui correspond à l'interface du service.

Déploiement :

- Déployer le Proxy : le Proxy devrait être déployé avec l'application client dans l'environnement d'exécution, ce dernier a besoin d'accéder au service d'enregistrement au moment de l'exécution pour chercher une implémentation de l'interface du service.

Exécution:

- Trouver la description de l'implémentation du service : avant que le Proxy puisse invoquer le service Web, une implémentation du service doit être localisée dans le service d'enregistrement. Cette opération doit être contenue dans le code du Proxy générique.
- Invoquer le service Web : après qu'une implémentation de service ait été trouvée, le Proxy peut l'utiliser pour invoquer le service.

5.2.3. Scénario Runtime dynamic binding

Ce type de liaisons est semblable au build-time dynamic binding. Une interface du service est utilisée pour générer un Proxy qui peut être utilisé pour invoquer n'importe quelle implémentation de cette interface. La différence entre ces deux types de liaisons est que dans

ce type l'interface du service est trouvée au moment d'exécution. Après que l'interface de service soit trouvée, le Proxy est généré, compilé puis exécuté.

Finalement, trois importantes remarques peuvent être ajoutées pour ce qui concerne le cycle de vie de développement des services Web (côté fournisseur et consommateur).

- Chaque scénario décrit auparavant peut être adapté selon certaines contraintes. Par exemple, nous pouvons éliminer l'étape de publication du service si le développeur ne sera pas utilisé un annuaire UDDI privé.
- Dans certains environnements, l'interface et l'implémentation du service sont implémentées dans un même fichier (par exemple, l'environnement .NET de Microsoft). Dans ce cas, la création de la définition de l'interface et de l'implémentation du service est effectuée dans une seule étape.

Chapitre 2 : Composition de web services

1. Introduction

Dans ce chapitre nous allons présenter la composition de web services et voir sa relation avec les processus métiers, ensuite on présentera le langage BPEL.

2. Processus métiers

Avant de détailler le concept de processus métiers, il est important de clarifier d'abord que le mot « processus » peut être utilisé dans des contextes hétérogènes pour des objectifs extrêmement variés. Indépendamment du domaine, le fais de parler de processus impliquera que l'on s'intéresse aux activités qui le composent, à sa finalité, à certaines ressources qu'il consomme. Sur cette base, chaque spécialité appose son filtre qui permettant de ne retenir que ce qui lui semble pertinent **[13]**. Dans notre cas, on s'intéresse aux processus métiers dans les organisations et leur implication dans les systèmes d'information intra et interentreprises.

2.1. Notion de processus et processus métier

Dans la littérature, nous trouvons plusieurs définitions du terme processus. La raison est que cette notion est suffisamment générale pour être utilisée dans différents domaines scientifiques ou applicatifs. L'étude de ces définitions permet d'avoir une idée claire de ce qui est désigné par un «processus».

Le terme processus métier (processus d'affaire, processus opérationnel et processus d'entreprise) est une traduction française de la notion de business process.

Nous définissons un processus d'une entreprise en tant qu'enchaînement d'activités interactives. Une activité transforme des entrées en sorties par l'influence d'objets de contrôle et en utilisant les ressources requises et disponibles pendant une durée bien définie. Un processus manipule des objets entrants par le moyen de ressources pour fournir des objets en sortie (produits/services) afin de répondre aux besoins d'un client. Le processus est généralement déclenché suite à des événements (internes et/ou externes à l'entreprise). Chaque processus est en communication avec d'autres et peut être décomposé en sous-processus **[09]**.

Aussi, on peut définir le processus métier comme suit : un processus est un ensemble d'activités, une activité est un ensemble de taches, et une tache est une opération élémentaire. Les conditions de lancement d'une instance de processus sont liées aux événements de son environnement **[09]**.

2.2. Les activités

Selon les auteurs de **[14]**, « *Une activité est caractérisée par une fonction qui transforme un flux d'objets entrant en un flux d'objets sortant dès lors qu'un ensemble de prè-conditions est satisfait et sous réserve de disponibilité de ressources et de temps.*

Une activité correspond à la réalisation d'une tâche ou d'un service, éventuellement sous forme d'un ensemble d'opérations, par un pool de ressources : homme, machine, application informatique. Les principales fonctions associées à une activité ont pour but de transporter, transformer, contrôler, stocker un flux entrant sous forme de matière, d'information, d'énergie ou de ressource sous contrôle de paramètres ».

Les conditions de lancement d'une activité sont liées aux contraintes de succession décrites par la structure du réseau d'activités. Dans ce cas, chaque instance d'une activité peut se trouver dans un des états suivants **[14]**:

- à exécuter : une demande ou prévisionnelle est à satisfaire mais les conditions de lancement ne sont pas remplies,
- exécutable : une demande effective ou prévisionnelle est à satisfaire, les conditions de lancement sont remplies, l'activité n'est pas lancée,
- en cours : sa date de début est antérieure à l'instant courant, elle est ni bloquée, ni abandonnée, ni terminée,
- bloquée : son exécution est suspendue en attente d'une décision,
- abandonnée : suite à une anomalie, l'exécution de l'instance n'est pas poursuivie,
- terminée : sa date de fin est antérieure à l'instant courant, le responsable ayant validé son exécution.

Le diagramme suivant illustre le changement d'état d'une activité :

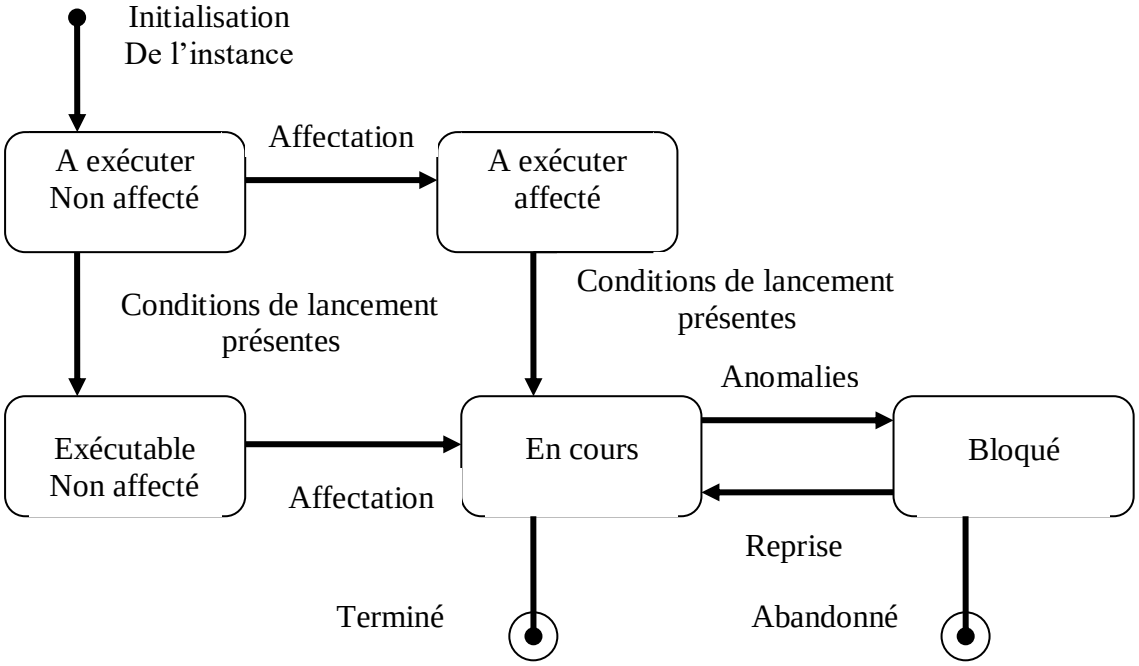


Figure 2.1 : Diagramme d'états-transitions pour une instance d'activité [14].

De nombreux modèles et représentations graphiques permettent de représenter les réseaux d'activité qui composent les processus (réseaux de pétri, state charts, graphes de précédence,…).

3. Composition de web services

Par définition, les Web services, tels qu'ils sont présentés, sont des composants conceptuellement limités à des fonctionnalités relativement simples qui sont modélisées par une collection d'opérations **[32]**. De ce fait, les Web services doivent pouvoir être composés en services que l'on compose jusqu'à ce que le service résultant fournisse un support entier pour les processus métiers **[15]**.

Dans le contexte de la composition des web services, les processus métiers sont ainsi définis comme étant un ensemble d'activités à travers lesquelles les services sont invoqués. Pour le demandeur (client) un processus métier (composition de plusieurs services) est vu comme n'importe quel autre service. Avec la composition, on peut utiliser des services internes et externes dans nos processus **[01]**.

3.1. Orchestration et Chorégraphie

Dépendamment des besoins, la composition des web services peut être faite en utilisant l'une des deux méthodes :

- Orchestration
- Chorégraphie

3.1.1. Orchestration

Cette technique est centralisée, un élément coordinateur (un processus ou un autre web service) prend le contrôle sur les web services participants dans la composition et il gère l'exécution et l'interaction avec les différentes opérations sur les web services impliqués. Ces derniers ne savent pas (et n'ont pas à savoir) qu'ils sont participés dans une composition et qu'ils sont une partie d'un processus métier complexe, seul le processus central de l'orchestration le sait. L'orchestration définit explicitement les opérations et l'ordre d'invocation des web services. L'orchestration est schématiquement décrite dans la figure suivante **[01].**

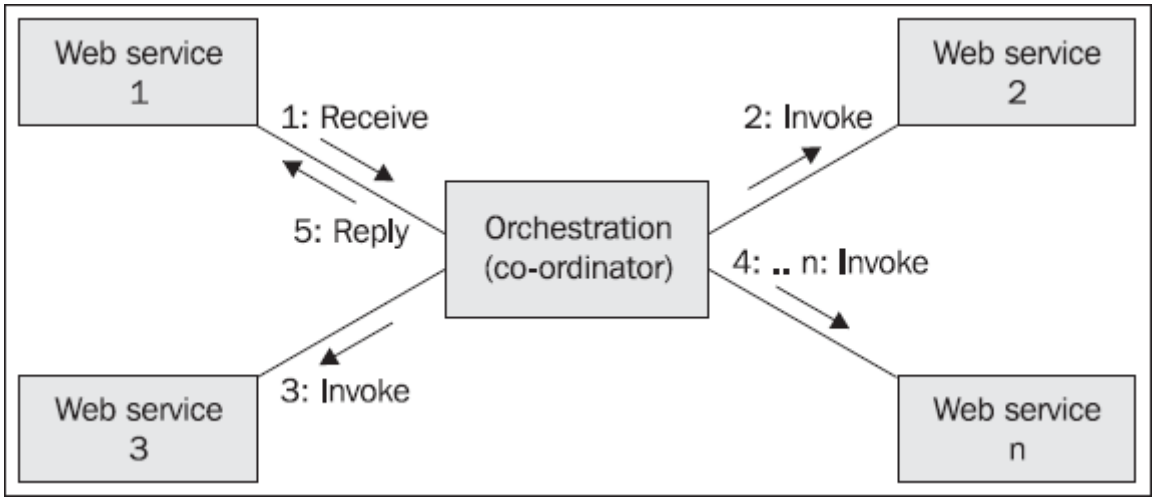


Figure 2.2 : Composition de web services en utilisant l'Orchestration [01]

3.1.2. Chorégraphie

La chorégraphie n'utilise pas un un coordinateur central. Elle est un effort de collaboration entre les services participants qui échangent des messages dans un processus métier publique. Donc, dans une chorégraphie, chaque web service sait exactement quand exécuter ses opérations et avec qui interagir. Tous les services impliqués dans la chorégraphie doivent savoir leurs rôles dans le processus métier, les opérations à exécuter, les messages à

échanger, et le temps d'échange de ces messages. La chorégraphie dans la composition des web services est montrée par la figure ci-dessous **[01]**, **[15].** Il est important de noter que le langage WS-CDL (Web Services Choreography Description Language), est une initiative permettant de décrire les compositions qui utilisent la chorégraphie.

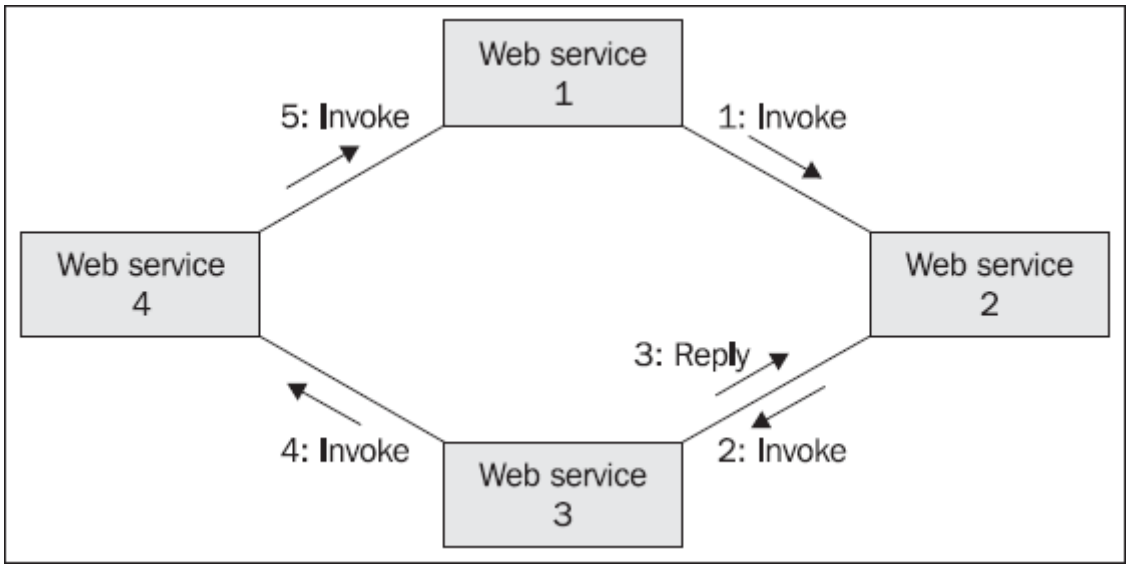


Figure 2.3 : Composition de web services en utilisant La chorégraphie [01]

Du point de vue de la composition des web services, pour exécuter des processus métiers, l'orchestration a un avantage sur la chorégraphie, parmi ces avantages :

- On connaît de façon exacte qui est le responsable de l'exécution du processus métier en entier.
- On peut incorporer les web services, même ceux qui ne savent pas qui sont impliqués dans des processus métiers.
- On peut aussi fournir un scénario alternatif quand il y a des erreurs.

3.2. Besoin d'un langage spécifique à la composition des Web services

La composition des Web services est définie comme étant un processus métier où les services présentent une collection d'activités qui collaborent pour implémenter ce processus **[15]**.

Pour une vue extérieure, le processus métier résultant est un Web service (composite) vu comme n'importe quel autre service basique. Par conséquent, les Web services composites doivent être considérés récursivement comme des Web services et doivent garder les mêmes caractéristiques que les services basiques (services WSDL) à savoir auto-descriptifs, interopérables, et facilement intégrables **[32]**, **[15]**.

WSDL fournit une description technique pour décrire de simples interactions entre le client et le web service. Cette description est basique et c'est une spécification des messages échangés avec des interactions qui peuvent être sans états (*stateless*), synchrones, ou asynchrones. Ces relations sont inadéquates pour décrire des compositions complexes de plusieurs web services qui consistent souvent en des échanges de messages synchrones et asynchrones peuvent être combinés avec des interactions assez longues. Aussi, un autre aspect important est la capacité de décrire la façon dont les erreurs sont traitées. En plus, la composition de services a d'autres exigences, comme le support de plusieurs instances de processus, des processus qui ont un temps d'exécution assez long, etc. Tout ça, fait que l'utilisation de technologies dédiées à la composition est une chose indispensable **[01]**.

La composition de services en des processus métiers peut être réalisée en utilisant un des langages de programmation bien connus (Java, C#,...), mais le problème est que la composition de services diffère un peu de la programmation classique. Avec la composition, on fusionne les fonctionnalités (services) en des services et des processus plus complexes. En d'autres termes on fait de la programmation de haut niveau. Cette programmation signifie la représentation de la logique de transition d'état du système. Si on utilise les langages de programmation tels que Java, C#, etc., pour la composition on aura des solutions inflexibles, car il n'y a pas de séparation claire entre le flux de processus et la logique métier **[01]**, **[15]**.

3.3. Business Process Execution Language (BPEL)

Plusieurs initiatives ont vu le jour pour standardiser l'automatisation des processus métiers. BPEL représente le langage le plus accepté dans la communauté des concepteurs de processus métiers modernes **[15]**.

Microsoft, IBM, et BEA ont développé la première version de BPEL en Aout 2002. Depuis que SAP et Siebel les ont rejoints, il y eu beaucoup de modifications et d'améliorations. En Avril 2003, BPEL a été soumis à l'OASIS (Organisation for the Advancement of Structured Information Standards) pour des buts de standardisation **[01]**.

BPEL (connu aussi sous le nom BPEL4WS ou WSBPEL) représente la convergence de deux langages, WSFL (Web Services Flow Languages) et de XLANG. WSFL a été conçu par IBM et est basé sur le concept de graphe oriente. XLANG a été conçu par Microsoft ; BPEL combine les deux approches et fournit un vocabulaire riche pour la description des processus métiers.

BPEL utilise un vocabulaire basé sur XML pour spécifier et décrire les processus métiers. BPEL version 1.1 est basé sur les spécifications WSDL 1.1, XML Schema et XPath 1.0 **[01]**.

3.3.1. Fonctionnalités de BPEL

BPEL peut définir des processus métiers aussi bien simples que complexes. Pour certaines manipulations, BPEL est similaire aux langages de programmation structurels (comme Pascal et C). Il offre des constructions comme des variables, des assignements, des boucles, des branchements, etc. pour définir des processus métiers de manière algorithmique, par conséquent cette définition est relativement simplifiée, d'autre part il est moins compliqué que les langages de programmation traditionnels **[01]**.

En comparaison avec les langages de programmation, BPEL offre des constructions spécifiques aux web services: invocation des opérations des web services d'une façon synchrones ou asynchrones, de manière séquentielle ou parallèle, attendre les callbacks, et traitement des erreurs (*fault handling*). Ci-dessous sont décrites les fonctionnalités les plus importantes qu'offre BPEL **[01]**:

- Composition de services décrit la logique du processus métier.
- Composition de processus métiers complexe à partir de d'autres processus et de services plus simples.
- Maintenir des activités longues qui sont interruptibles.
- Reprendre des activités interrompues ou qui ont échoué pour minimiser le travail à refaire.
- Router les messages entrants à l'activité ou au processus destinataire.
- Organiser les activités en se basant sur leur temps d'exécution et définir leur ordre d'exécution.
- Exécuter les activités en parallèles.
- …

3.3.2. Les principales instructions de BPEL

Un processus BPEL est composé d'étapes appelées des activités. Il supporte les activités basiques et structurées **[01].**

Ici on ne va présenter que les activités essentielles de manière assez brève.

a- Les activités basiques

Les activités basiques sont généralement utilisées pour réaliser des tâches courantes **[01].**

- <**process**> : la racine de tout programme BPEL
- **<invoke>** : pour invoquer d'autres web services (externe par apport au processus central)
- **<receive>** : pour attendre et recevoir des messages (recevoir une requête, données,…)
- **<reply>** : pour générer une réponse pour les opérations synchrones.
- **<variable>** : pour définir les variables
- **<copy> et** <**assign>** : pour manipuler les données et assigner une valeur à une variable
- <**empty**> **:** pour insérer une activité vide

b- Les activités structurées

Pour définir des algorithmes complexes qui spécifient les étapes d'un processus, nous pouvons combiner des activités basiques par l'utilisation des activités structurées de BPEL. Les plus importantes sont **[01]**:

- **<sequence> :** pour définir un ensemble d'activités qui seront invoquées séquentiellement
- **<flow> :** pour définir un ensemble d'activités qui seront invoquées en parallèle.
- **<if> <else> :** pour définir les structures conditionnelles.
- **<While> et <RepeatUntil> :** pour créer des boucles
- **….**

Chapitre 3 : L'architecture Micro-Services (MSA)

1. Evolution des Architectures Logicielles

Chronologiquement, les architectures logicielles suivantes présentent les styles les plus utilisés pour développer des systèmes informatiques **[17]** :

a) Architecture Monolithique

•

Figure 3.1 : Architecture d'une application monolithique [17].

- Un gros code contenant toutes les fonctionnalités et les différentes couches logicielles
- Une seule grosse compilation et un seul livrable (un gros fichier WAR)
- Une seule pile logicielle (Linux, JVM, Tomcat et bibliothèques tierces)

b) SOA et ESB

Pour les détails de l'architecture SOA, voir chapitre 1.

Dans cette section, nous allons présenter un concept important pour mettre en place une architecture SOA dans le contexte de l'intégration d'applications=> **ESB : Enterprise Services Bus [17]**, **[18]**.

Le rôle d'un ESB est bien démontré dans la figure 3.2.

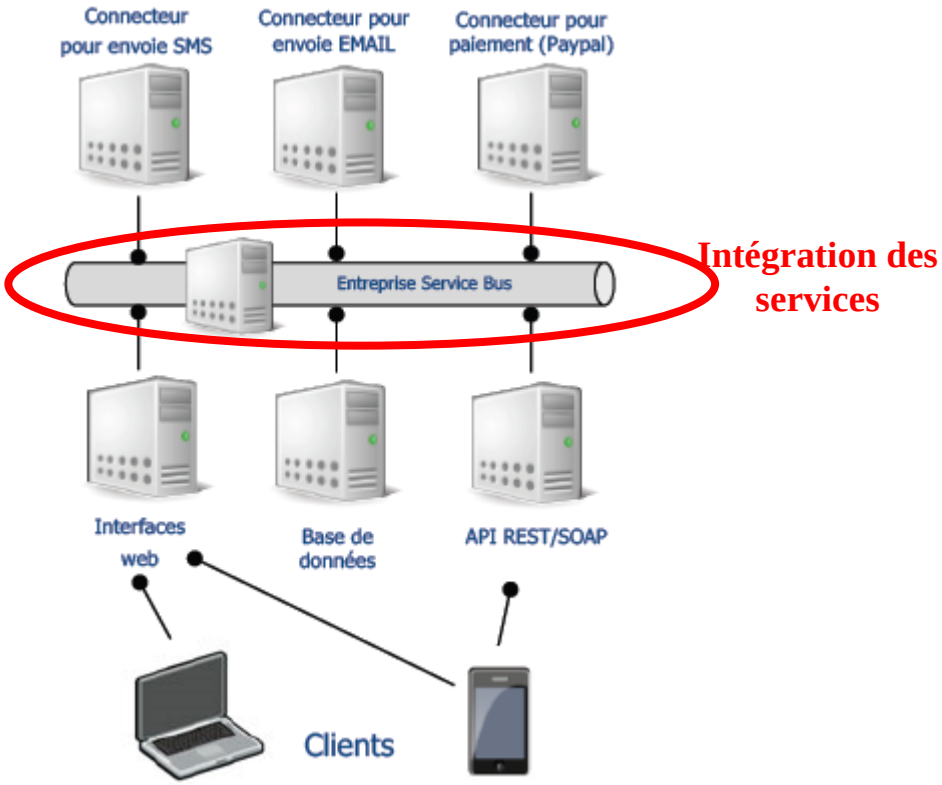


Figure 3.2 : rôle de l'ESB [17].

ESB est l'une des techniques utilisées pour l'intégration des applications et des services qui n'ont pas été conçues pour fonctionner ensemble. L'ESB est un ensemble d'outils qui permettent des échanges de données sécurisées entre plusieurs services et applications différentes.

L'ESB possède quatre composants **[18]:**

- Le message-oriented middleware (MOM) qui permet l'échange de messages de manière asynchrone. L'ESB utilise généralement une file d'attente pour sauvegarder les messages reçus avant d'être consommé par le destinataire.
- Les services web qui permettent d'exposer les fonctionnalités des applications existantes et d'assurer la communication avec le bus.
- Les transformations des messages.
- Le routage intelligent des messages.

Par exemple, l'ESB peut s'appuyer sur les standards suivants :

c) JMS (Java Message Services) pour implémenter MOM.

d) SOAP, WSDL, UDDI pour les services Web.

e) JBI (Java Business Integration) pour les conteneurs de services.

f) XML, XSLT, XPath pour la transformation et le routage.

g) JCA (Java Connector Architecture) pour la connexion aux applications.

h) BPEL pour l'orchestration des processus métier.

.

c) Microservices (MSA)

La figure ci-dessous présente un exemple d'une architecture microservices.

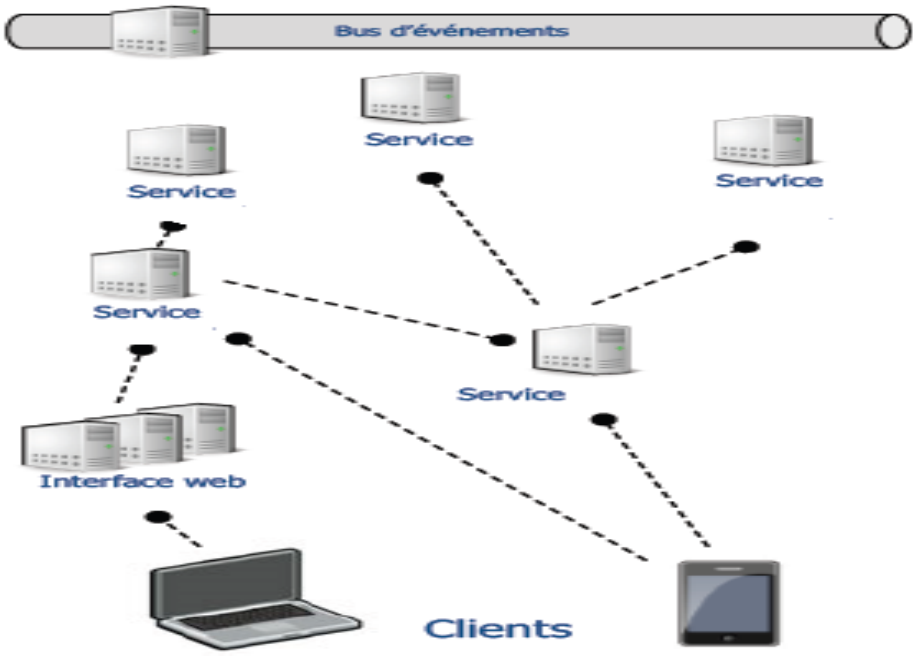


Figure 3.3 : Architecture microservices [17].

Les premières discussions autour du terme microservice été en 2011 lors d'un workshop sur les architectures logicielles. Initialement, sa relation avec l'architecture SOA peut être résumée comme suit **[17]**:

- Les architectures microservices sont une évolution, une spécialisation des architectures orientées services (SOA)
- Vu comme une bonne pratique du SOA (réutilisabilité)
- Une architecture microservice est donc un ensemble de microservices autonomes
- Pas de bus d'intégration (ESB) ou une orchestration BPEL par exemple (un genre de couplage même s'il contient très peu de la logique)=> Utilisation d'un Bus d'événements pour MSA => **Mais il faut équilibrer la charge** (Utilisation d'un Load Balancer)
- Bus d'événements basés sur le protocole AMQP (Advanded Message Queuing Protocol)

- La communication entre des microservices peut se faire aussi via des services web pour une communication synchrone (Web services SOAP ou REST mais REST sont plus utilisés –voir chapitre suivant-)

2. Caractéristiques de MSA

Les principales caractéristiques de la MSA sont les suivantes **[17]**:

2.1. Fonctionnalité unique

- ❖ Un microservice doit réaliser une seule fonctionnalité de l'application globale (une tache bien précise)
- ❖ Un microservice peut contenir toutes les couches logicielles (IHM [front end], middleware et base de données)
- ❖ Un microservice possède un contexte d'exécution séparé des autres (exemple : machine virtuelle ou conteneur)

2.2. Flexibilité technologique

Utiliser les bons langages et frameworks selon la fonctionnalité à réaliser.

2.3. Equipe de développement réduite

- ❖ Chaque microservice à sa propre équipe de développement
- ❖ L'équipe de développement orientée fonctionnalité (spécialisée)=> développement agile et rapide.
  - ✓ Développeurs backend
  - ✓ Développeurs web
  - ✓ Administrateurs bases de données
  - ✓ ……

2.4. Déploiement ciblé

- ❖ Evolution d'une certaine partie sans tout redéployer
- ❖ Un seul livrable à partir d'un seul code source
- ❖ Moins de coordination entre équipe quand il y a un seul déploiement
  - ✓ Déploiement plus fréquent
  - ✓ Moins de risque
  - ✓ Plus rapide

- ⇨ **Faciliter les mises à jour :** l'entreprise est capable de faire évoluer son application de façon très fréquente et de répondre rapidement aux nouvelles fonctionnalités que propose la concurrence=>Pas nécessaire de re-tester et de déployer l'application complète.
- ⇨ **Passer d'une technologie à l'autre sur des portions individuelles**

2.5. Montée en charge « scalability»

L'architecture microservices est très forte de point vue évolutivité et montée en charge. Pour cette raison,il est très crecommandé d'utiliser cette architecture, lorsque l'application ou le système à développer est destiné au grand public. MSA capable de s'étendre sur plus de ressources **[17]** :

- ❖ Comme chaque fonctionnalité est isolée, nous avons une possibilité de **multiplie**r le nombre d'instances d'un microservice fréquemment demandé (Forte sollicitation sur un microservice : page web d'Amazon en période de « Black Friday » par exemple).
- ❖ Il faut donc envisager, dès le départ, une autre approche dans la conception de l'application afin que celle-ci soit "**Cloud-native**", c'est-à-dire conçue pour fonctionner et profiter pleinement des avantages qu'offre le cloud **[19]**.
- ❖ Comme chaque fonctionnalité est isolée dans un microservice il est plus facile de les tester, déployer et re-déployer en cas de mise à jour.
  - ✓ Tester l'API exposée du microservice
  - ✓ Bout-en-bout : intégrer les autres microservices

3. SOA vs MSA

Le tableau ci-dessous présente une comparaison entre les deux architectures SOA et MSA **[17]**, **[19]** :

SOA	MSA
✓ Standards définis par OASIS SOAP, BPEL ✓ Couplage faible Mais: **dans une SOA, il est courant de trouver une base de données commune à plusieurs services (**modernisation des anciens systèmes**)**.	✓ Standards du web ✓ Couplage très faible : **chaque Microservice doit être très indépendant à tous les niveaux (**Chaque Microservice possède sa propre base de données**)**.

✓ Bus d'intégration ✓ Plusieurs fonctionnalités par service ✓ Déploiement monolithique ✓ La **SOA** accepte que ses composants (services et autres) communiquent avec des **protocoles différents**, (l'ESB s'occupant ensuite d'adapter et de transformer).	✓ Bus d'événement ✓ Une seule fonctionnalité par microservice ✓ Automatiquement déployable ✓ La **MSA** tend à obliger à utiliser **un seul protocole** et de s'y tenir.

Chapitre 4 : REST et Web Services RESTFUL

1. Style REST

1.1. Introduction

REST estl'acronyme de "REpresentationalState Transfer" inventé par Roy T. Fielding dans sa dissertation "an architecture style of networked systems". REST décrit les caractéristiques du web qui en ont fait son succès. L'explication de la signification de REST telle que donnée par Roy T. Fielding est la suivante : 'Representational State Transfer' évoque l'image du fonctionnement d'une application web bien construite : « *un réseau de pages web où l'utilisateur progresse dans l'application en cliquant sur des liens (transition entre états) ce qui provoque l'affichage de la page suivante (représentant le nouvel état de l'application) à l'utilisateur qui peut alors l'exploiter* » **[24]**.

1.2. Définition

Si nous voulons réaliser une comparaison entre les web services de type SOAP et REST, nous pouvons dire que REST n'est pas un standard c'est un style d'architecture. Donc, il n'existe pas de spécifications de REST. C'est un style d'architecture réseau pour web services où les ressources sont identifiées par des URI et les messages échangés entre le client et le serveur sont définis sémantiquement par une interface uniforme (les messages du protocole http).

Roy T. Fielding dit dans sa Thèse de doctorat : REST est un modèle hybride dérivé de plusieurs modèles basés sur les concepts réseau.

1.3. Les contraintes de REST défini par Roy T. Fielding

Voici les six (6) contrainte de REST sont défini par Roy T. Fielding **[25]**, **[27]** :

- **Client-Serveur :**

Cette contrainte est liée au modèle architectura de base du WWW. Elle vise à séparer la responsabilité entre le client et le serveur par la désunion entre l'interface utilisateur et le stockage des données. Cela permet aux deux d'évoluer indépendamment. De cette manière, nous améliorons la portabilité de l'interface utilisateur à travers les diverses plates-formes, et Nous augmentons la montée en charge en simplifiant les composants du serveur **[27]**.

- **Sans état (Stateless) :**

La communication entre le client et le serveur est sans état. Pour réaliser cet objectif, il est intéressant que chaque requête d'un client vers un serveur doit contenir toute l'information

nécessaire pour permettre au serveur de comprendre la requête, sans avoir à dépendre d'un contexte conservé sur le serveur. Cela libère de nombreuses interactions entre le client et le serveur. L'état de la session est donc entièrement détenu par le client **[27]**.
Ignorer le stockage de l'état entre les requêtes permet de libérer le serveur d'une bonne partie du traitement. Donc, sans d'être obliger de gérer l'utilisation des ressources à travers les différentes requêtes, le serveur renvoi les ressources demandées plus rapidement. Par conséquent, le client peut faire des requêtes pour ces ressources n'importe quel nombre des fois, dans un ordre quelconque. De cette façon, le serveur n'a jamais besoin de connaître l'état du client, il ne sait même pas où il est, il sait juste qu'une requête arrivant avec tels paramètres doit restituer telles données **[26]**, **[27]**. De cette façon, nous améliorons aussi la montée en charge.

- **Mise en cache :**

Cette contrainte signifie que les données d'une réponse à une requête soient marquées, implicitement ou explicitement, comme pouvant être mises ou non en cache. Le serveur envoie une réponse qui donne l'information sur la propension de cette réponse à être mise en cache, comme la fraîcheur, sa date de création, si elle doit être conservée dans le futur. Cela permet à des serveurs mandataires de décharger les contraintes sur le serveur et aux clients de ne pas faire de requêtes inutiles. Cela permet également d'améliorer l'extensibilité des serveurs.

- **Interface uniforme :**

Toutes les ressources sont accessibles par une interface générique (par exemple, HTTP GET, POST, PUT, DELETE).

- **Un système hiérarchisé par couche :**

Selon la traduction du chapitre 5 de la thèse de Roy T. Fielding, cette caractéristique est définie de cette manière : « *Afin d'améliorer le comportement de l'architecture face aux besoins de grande échelle (internet), nous ajoutons des contraintes de système en couches. Le modèle de système en couches permet à une architecture de se composer de couches hiérarchiques en contraignant le comportement des composants. Chaque composant ne peut pas «voir» au delà de la couche immédiate avec laquelle il interagit. En limitant la connaissance du système à une seule couche, nous mettons une limite sur la complexité du*

système global et favorisons l'indépendance des couches. Les couches peuvent être employées pour encapsuler des services déjà en place et protéger les nouveaux services des clients existants, en simplifiant les composants par le déplacement de fonctionnalités rarement utilisées dans un espace intermédiaire partagé. Des intermédiaires peuvent également être employés pour améliorer la montée en charge du système en offrant un équilibrage de charges pour les services et ce à travers de multiples réseaux et processeurs » **[27]**.

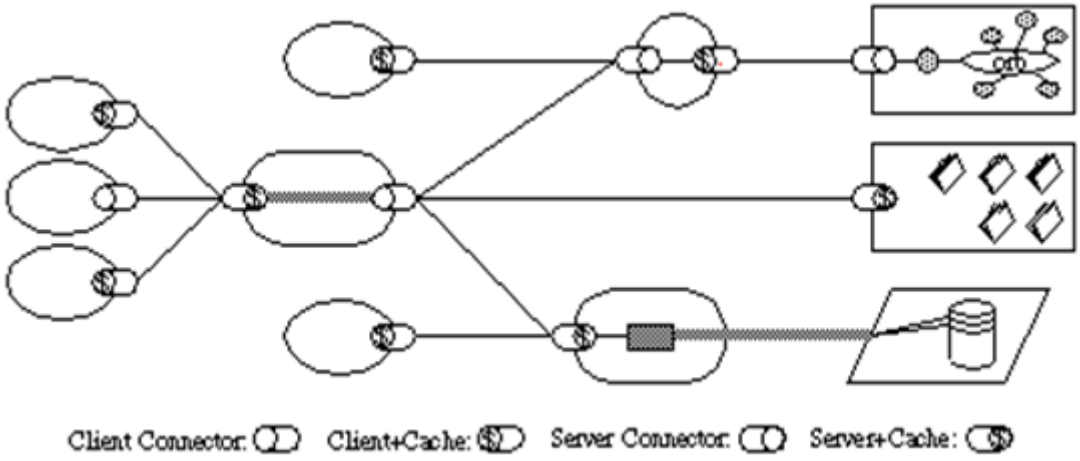


Figure 4.1 : Couche uniforme –Client-Cache-Stateless-Serveur [25]

- **Code à la demande (facultatif) :**

Exécution d'applet ou scripts côté client obtenus par le serveur. Cela permet de rendre le client plus léger et plus générique, comme il peut réduire le nombre de fonctionnalités qu'ils doivent mettre en œuvre par défaut.

1.4. Les éléments de REST

Roy T. Fielding définit 4 éléments architecturaux pour le style REST dans un système hypermédia distribué. Pour cela, Fielding a concentré sur les rôles des composants, l'interaction entre eux et leur interprétation à éléments de données significatifs. Dans sa définition des composants REST, Fielding ignore les détails de la mise en œuvre des composants et de la syntaxe du protocole. Il englobe les contraintes fondamentales sur les composants, les connecteurs et les données qui définissent la base de l'architecture Web.

1.4.1. Les éléments de données

Les éléments de données de REST sont résumés comme suit :

- Ressource
- Identifiant de ressource
- Représentation
- Métadonnées de représentation
- Métadonnées de ressource
- Données de contrôle

a) URI comme identifiant des ressources

Afin d'identifier une ressource, REST se base sur les URI (Uniform Resource Identifier). En tenant compte des contraintes REST, l'architecte d'une application doit construire les URIs de manière précise **[31]**. Il est nécessaire de définir les routes principales et les sous routes et de prendre en compte la hiérarchie des ressources et la sémantique des URIs pour les éditer.
En construisant correctement les URI, il est possible de les trier, de les hiérarchiser et donc d'améliorer la compréhension du système.

Exemples:

/biblio/livres→ tous les livres

/biblio/livres 39 → un livre (numéro 39)

/biblio/livres /39/commentaires→ tous les commentaires sur un livre

/biblio/livres /39/commentaires /12→commentaire N° 12 sur le livre N° 39

b) Représentation des ressources

Une ressource peut avoir une ou plusieurs représentations possibles. En pratique, dans l'interaction client/serveur il faut passer de la représentation utilisée en interne à la représentation utilisée dans le message http (requête ou réponse).
Pour les Web Services RESTful, les encodages les plus utilisés sont **XML** et **JSON** (JavaScript Object Notation).

Remarque JSON

C'est un format texte qui permet de représenter des données et de les échanger facilement à l'instar d'XML. Ce sous ensemble de JavaScript permet de décrire le modèle objet de JavaScript. Deux types de structures sont disponibles :

- Objet : une collection de paire nom/valeur, i.e. un tableau associatif.
- Tableau : une liste ordonnée de valeurs.

La syntaxe est celle de JavaScript, simpliste. Voici par exemple la description d'un étudiant avec format JSON :

```
[ {
        "id":17,
        "nom":"Bendahi",
        "prenom": "mohamed",
        "sexe": "homme",
        "date_naissanse": "10/10/1991",
        "telephone": "066666666",
        "adresse":"Biskra",
        "email":"bmohamed1991@gmail.com",
}]
```

c) Données de contrôle

« *Les données de contrôle définissent le but d'un message entre les composants, tels que l'action demandée ou la signification d'une réponse. Il sert également à paramétrer les requêtes et à remplacer le comportement par défaut de certains éléments de connexion. Par exemple, le comportement du cache peut être modifié par les données de contrôle incluses dans le message de la requête ou de la réponse. Selon les données de contrôle d'un message, une représentation donnée peut indiquer l'état actuel de la ressource demandée, l'état souhaité pour la ressource demandée ou la valeur d'une autre ressource, telle qu'une représentation des données d'entrée dans le formulaire d'une requête d'un client, ou une représentation d'une condition d'erreur pour une réponse*» **[27]**.

1.4.2. Relation entre ressources

Les liens entre les ressources indiquent la présence d'une relation. Il est cependant possible de décrire cette relation afin d'enrichir la sémantique du système et d'améliorer sa compréhension. Pour expliciter la nature de la relation, l'attribut *rel* doit être précisé sur tous les liens. Ainsi l'IANA (*Internet Assigned Numbers Authority*) donne une liste de relation (self, next, edit, last, payment, content…).

Exemple

```
<?xml>
<search>
  <link rel="self"title="self"href="http://mywebsite.com/
serviceEtud/etudiants?q=formation&nm=STIC&nv=M1"/>
  <link rel="next"title="next"href="http://mywebsite.com/
serviceEtud/etudiants?q=formation&nm=STIC&nv=M2"/>
```

```
  <Etudiants>
   //...
</search>
```

1.4.3. Les Connecteurs

REST utilise différents types de connecteurs, pour encapsuler les activités d'accès aux ressources et le transfert des représentations de ressources. Les connecteurs présentent une interface abstraite pour la communication des composants, améliorant la simplicité en fournissant une séparation propre des préoccupations et cachant l'implémentation sous-jacente des ressources et des mécanismes de communication. Les différents types de connecteurs sont:

- Client
- Serveur
- Cache : cache du navigateur, ….
- Résolveur : DNS
- Tunnel : SOCKS, SSL ..

1.4.4. Les Composants

Les Composants de REST sont :

- Serveur d'origine
- Passerelle
- Proxy
- Agent utilisateur

2. Principe des services web RESTful

Régulièrement, lire REST ou RESTful lorsque la technologie est abordée. En bref, il n'y a aucune différence, RESTful est simplement l'adjectif qui qualifie une architecture de type REST.

2.1. Utilisation des Méthodes HTTP

Pour assurer la contrainte « interface uniforme», le protocole HTTP est généralement utilisé pour communiquer avec un service Web REST (implémente une API REST). Ainsi,

généralement pour une ressource, il y a 4 opérations possibles et HTTP propose les verbes correspondant :

- Créer (Create) => POST
- Afficher (Read) => GET
- Mettre à jour (Update) => PUT
- Supprimer (Delete) => DELETE

Exemple d'URI pour une ressource donnée (une Actualité par exemple), pour ajouter ou modifier ou supprimer une ressource on peut utiliser la méthode GET de manière suivante par exemple:

a- GET 127.0.0.1:8080/biblio/livre/afficher?id=7

b- GET 127.0.0.1:8080/ biblio/livre/supprimer?id=7

c- GET 127.0.0.1:8080/biblio/livre/ajouter?**titre**=REST%20en%20deux%20jours&**ISBN**=345678907

Dans les cas « **a** » et « **b** » nous devons modifier les méthodes HTTP, pour respecter l'architecture REST et l'URI devenu descriptif :

GET127.0.0.1:8080/ biblio/livre /7- Afficher le livre numéro 7

DELETE127.0.0.1:8080/ biblio/livre /7- Supprimer le livre 7

Le cas « **c** », il faut ne pas faire circuler des informations clairement sur le réseau. La simple moyen pour éviter ce problème et de envoyer les noms et les valeurs des paramètres de la requête HTTP dans format JSON ou balise XML à partir du « Request body ».

Par exemple le cas « **c** » ajouter livre (utilisation de la méthode POST)

```
POST /biblio/livre  HTTP/1.1
Host :127.0.0.1 :8080
Content-Type: application/json

{
  "titre"="REST en deux jours"
  "ISBN"= "345678907"
}
```

Request Header

RequestBody

2.2. Résumé : règles pour développer une application REST

Pour développer une application REST, vous devez utiliser :

- L'URI comme identifiant des ressources (conception des routes)
- Les verbes HTTP comme identifiant des opérations
- Les réponses HTTP comme représentation des ressources

- Les liens comme relation entre ressources
- Pour assurer la contrainte « stateless », utiliser un paramètre comme jeton d'authentification (Token comme JSON Web Tokens (JWT)

En plus, vous devez prendre en considération le mécanisme de cache et l'exploitation des entêtes HTTP.

3. Microservices basée sur les services Web Restful

Dans le chapitre précédent, nous avons présenté le principe de l'architecture microservices (MSA) qui se base sur le **découpage** d'une application en **petits services**, appelés Microservices, **parfaitement autonomes**.

Les web services REST peuvent être utilisés de la manière suivante **[18]**:

- Chaque microservice expose une API *REST* que les autres Microservices pourront consommer.
- Dans l'absence d'un service de gestion centralisé, le système devient totalement et largement distribué.
- Les microservices sont indépendants l'un de l'autre et chaque microservice délivre une seule fonctionnalité (avec un couplage très faible et forte autonomie).
- La communication est légère via des messages HTTP (en utilisant un format d'échange simple comme JSON).

La figure suivante présente l'utilisation des API REST pour composer une architecture MSA.

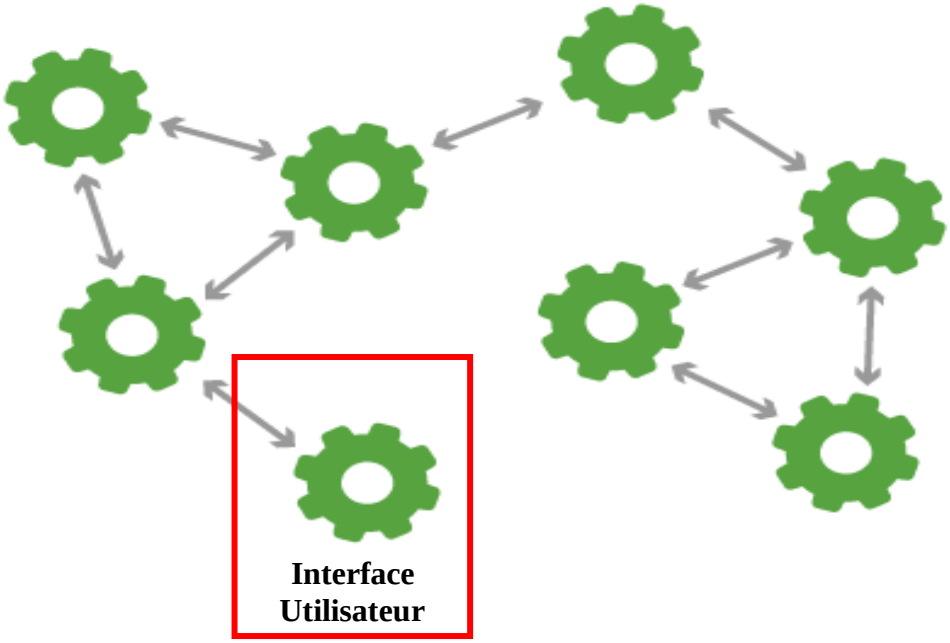


Figure 4.2 : Architecture MSA basée REST

Chapitre 5 : Qualité de Web Services (QoS) et SLA

1. Introduction

Avec le nombre énorme de services fournis sur Internet, la notion de QoS prend de plus en plus une grande importance pour les fournisseurs de service aussi bien que pour les clients de service. Il est important de signaler que ce concept est utilisé dans plusieurs domaines, mais ce qui nous intéresse est la qualité de services logiciels (web services).

2. Qualité de service (QoS) de web services

Dans le contexte des web service, QoS est l'ensemble des caractéristiques quantitatives et qualitatives d'un service, nécessaires pour répondre d'une manière adéquate à des exigences, exprimées ou implicites, qui visent à satisfaire ses usagers. Ces exigences peuvent être liées à plusieurs aspects d'un service **[20]**.

La QoS est très utile pour les raisons suivantes :

- Différencie les services fonctionnellement équivalents
- Les utilisateurs peuvent exprimer leurs besoins, comme ils peuvent aussi choisir le meilleur service à l'égard de leurs besoins.
- Les fournisseurs peuvent mieux faire connaître leurs services, les améliorés,…

3. Caractéristiques des QoS

Dans le cadre des services Web, un ensemble de caractéristiques de QoS pertinentes ont été définies pour le domaine des services Web **[20]**, **[22], [33]**:

3.1. QoS liée au Temps d'exécution

Les QoS liées au temps d'exécution sont les suivantes :

- **Performance** : représente la vitesse avec laquelle un service Web répond à une requête. Elle est mesurée par:
  - **Temps de réponse:** Le temps maximum garanti demandé pour compléter une requête du service.
  - **Latence:** Temps pris entre l'arrivée de la requête du service et la réponse émise par le service.
  - **Débit:** capacité d'exécution: Le nombre de requêtes accomplies par le service pendant une période de temps.

- **Fiabilité**: La capacité d'un service d'exécuter ses fonctions dans des conditions indiquées dans une période de temps spécifié. Elle peut être mesurée par:
  - **MTBF: "**Mean Time Between Failure" - Temps moyen entre pannes.
  - **MTF:** "Mean Time to Failure" - Temps moyen par panne.
  - **MTTT:** "Mean Time To Transition" – Temps moyen pour la transition. Il est très lié à la disponibilité.
- **Passage à l'échelle**: permet de quantifier le nombre de requêtes auxquelles le service peut faire face dans un intervalle de temps donné. Capacité: nombre de requêtes qu'il est possible de traiter simultanément.
- **Disponibilité**: Elle est la probabilité que le système soit actif. Elle est lie à la fiabilité. Elle peut être mesuré comme : **Disponibilité = Nombre de requêtes réussites/Nombre total de requêtes.**
- **Robustesse/Flexibilité**: elle est le degré dans lequel un service peut s'exécuter correctement dans la présence des entrées inadmissibles, inachevées ou contradictoires.
- **Traitement des exceptions** : Puisqu'il n'est pas possible pour le concepteur du service de spécifier tout les résultats possibles et alternatifs (particulièrement avec de divers cas spéciaux et possibilités imprévues), des exceptions peuvent être attendues (comment le service traite ces exceptions? Il peut être d'une manière brutale ou appropriée).
- **Exactitude**: définit le taux d'erreur produit par le service. Combien d'erreurs le service produit sur une période de temps ?

3.2. QoS liée aux transactions

- **Intégrité**: propriété garantissant que l'intégrité des données et des transactions est bien respectée, afin de ne pas aboutir à une situation inconsistante.
- C'est décrit par les propriétés **ACID**: l'Atomicité (exécute entièrement ou pas du tout), la consistance (maintient l'intégrité des données: chaque transaction amènera le système d'un état valide à un autre état valide), l'isolement (des transactions individuelles exécutées comme si aucune autre transaction n'est présente) et la durabilité (les résultats sont persistantes: lorsqu'une transaction a été confirmée, elle demeure enregistrée même à la suite d'une panne).

3.3. QoS liée a la gestion de la configuration et coût

- **Régulateur** : c'est une mesure qui spécifie si le service est aligné sur des règlements.
- **Norme Soutenue** : une mesure de si le service observe des normes (standards) (par exemple normes ou standards spécifiques). Cela peut affecter la portabilité du service et de l'interopérabilité du service avec d'autres.
- **Cycle de stabilité/changement: une** mesure de la fréquence de changement lié au service en termes de son interface et/ou mise en œuvre.
- **Coût** : C'est une mesure du coût impliqué dans la requête du service.
- **Etat complet: une** mesure de la différence entre le jeu indiqué de caractéristiques et le jeu mis en œuvre de caractéristiques.

3.4. QoS liée à la sécurité

Elle mesure la fiabilité et la sécurité de mécanismes mise en œuvre.

- **Authentification** : Comment le service authentifie-t-il des principaux (des utilisateurs ou d'autres services) qui peut avoir accès au service et des données ?
- **Autorisation** : Comment le service autorise-t-il des principaux pour que seulement eux puissent avoir accès aux services protégés ?
- **Confidentialité** : Comment le service traite-t-il les données, pour que seulement les principaux autorisés puissent avoir accès ou modifie les données ?
- **Responsabilité** : le fournisseur peut-il être responsable par leurs services ?
- **Traçabilité et vérifiabilité** : Est-il possible de tracer l'histoire d'un service lorsqu'une demande a été entretenue.
- **Cryptage des données** : Comment le service chiffre-t-il des données ?

4. SLA: Service Level Agreement

Définition: Selon JDN « *Le Service Level Agreement, ou **SLA** est un contrat ou la partie d'un contrat par lequel un prestataire informatique s'engage à fournir un ensemble de services à un ou plusieurs clients. Autrement dit, il s'agit d'une clause contractuelle qui définit les objectifs précis et le niveau de service qu'est en droit d'attendre un client de la part du prestataire signataire* » **[21]**.

Le **SLA** est intimement lié aux environnements Cloud. SLA doit contenir les parties suivantes **[23]**:

***Objectif**:* Ce champ décrit les raisons de la création du contrat.

Parties: décrit les parties impliquées dans le contrat leur rôles respectifs (fournisseur, client,…).

Période de validité: Ce champ définit la période de temps que couvrira le SLA (l'heure de début et de fin de la durée de l'accord).

Portée (Scope): Ce champ définit les services couverts par la convention.

Restrictions: Ce champ définit les étapes à suivre pour que les niveaux de service demandés soient fournis.

Objectifs de niveau de service: ce champ définit les niveaux de service sur lesquels les clients du service et les fournisseurs de services conviennent, et inclut généralement un ensemble d'indicateurs de niveau de service, tels que la disponibilité, les performances et la fiabilité.

Chacun de ces aspects du niveau de service aura un niveau cible à atteindre **(un ensemble de QoS Flexibles**).

Pénalités: Ce champ définit les sanctions à appliquer au cas où le prestataire de services sous-performerait et ne pourrait pas atteindre les objectifs spécifiés dans le SLA.

Services facultatifs: ce champ spécifie les services qui ne sont normalement pas requis par l'utilisateur, mais peut être requis en cas d'exception.

Termes d'exclusion: ils spécifient ce qui n'est pas couvert dans le SLA.

Administration: décrit les processus et les objectifs mesurables dans un SLA et définit le pouvoir organisationnel permettant de les superviser.

Les SLA peuvent être de nature statique ou dynamique.

- Un SLA statique reste généralement inchangé pour plusieurs intervalles de temps de service. Les intervalles de service peuvent être des mois, une transaction ou toute autre période mesurable et pertinente pour d'autres processus. Ils sont utilisés pour l'évaluation de la qualité de service et sont convenus entre un fournisseur de services et un consommateur de services.
- Un SLA dynamique est un accord de niveau de service qui passe généralement d’une période de service à l’autre en fonction des modifications apportées à la fourniture de services.

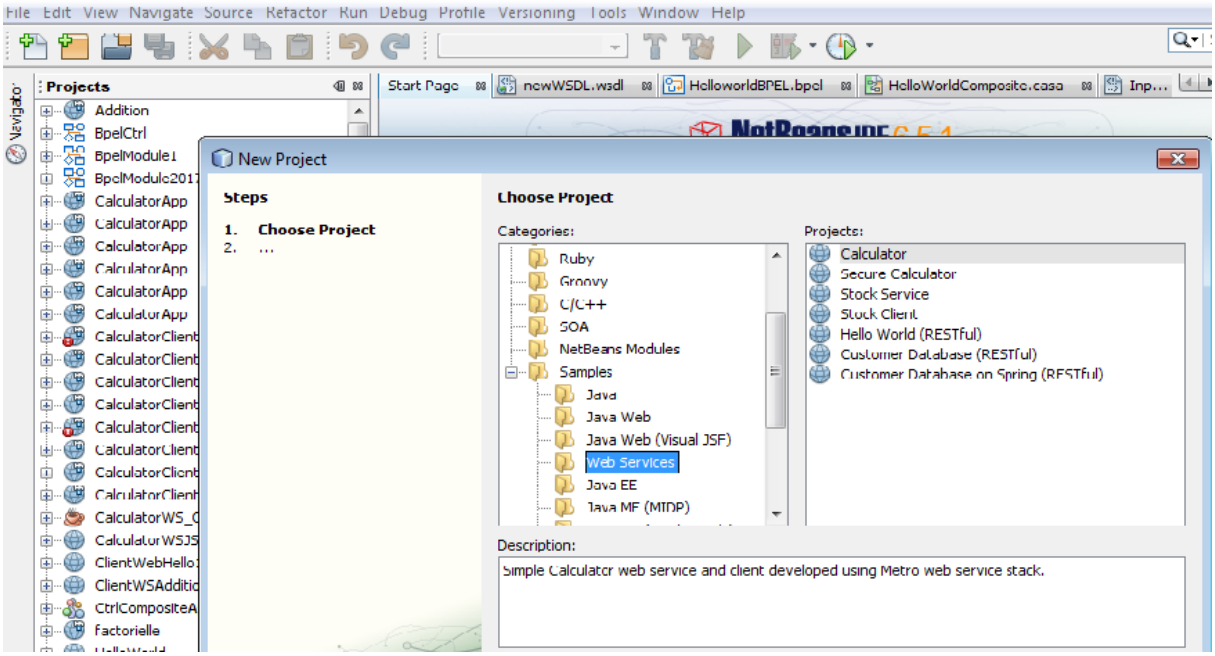
Travaux Pratiques

1. Avant de commencer

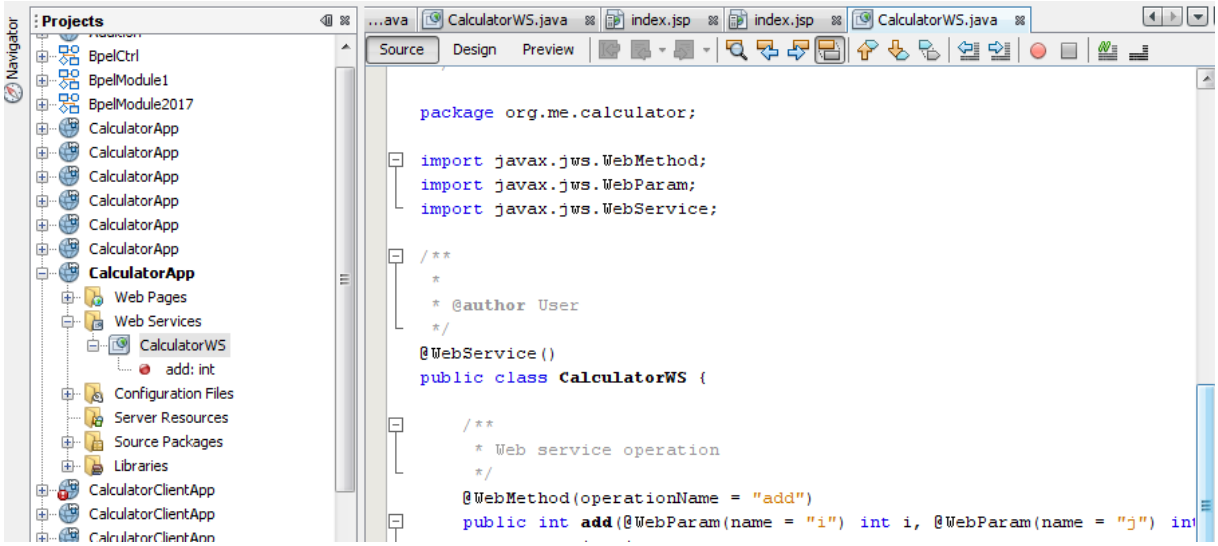
1. Télécharger Netbeans (V 07.x ou inférieure) ou bien OpenESB : https://www.open-esb.net/
2. Télécharger si nécessaire un serveur web comme Glassfish (version compatible avec celle de OpenESB). Ajouter Glassfish à OpenESB s'il n'existe pas (Add server).
3. Installer OpenESB https://www.pymma.com/images/documentation/OE%20EE%20Documentation/770-001%20Startup%20with%20OpenESB%20SE.pdf
4. Configurer OpenESB : bibliothèques et connexion JMS, voir les deux liens suivants : https://www.pymma.com/images/documentation/OE%20EE%20Documentation/770-003%20Administrative%20web%20console.pdf https://www.pymma.com/images/documentation/OE%20EE%20Documentation/770-012%20OE%20Standalone%20Edtion%20JMS%20Connection.pdf

2. TP1 : Développement d'un web service (Approche Bottom-Up)

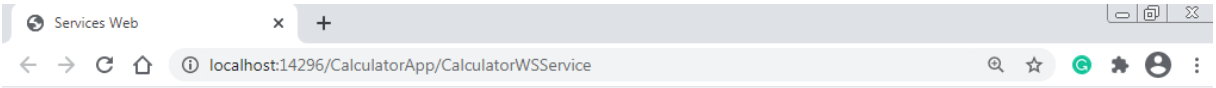
- Aller à File->New project->samples (ou web services dans OpenESB)->calculator-> Next->Finish



- Il vous propose deux Application : CalculatorApp et CalculatorClientApp. La deuxième représente l'application cliente correspondante (Voir TP 3). Maintenant, nous allons utiliser la première application qui est un web service simple implémente une opération add.



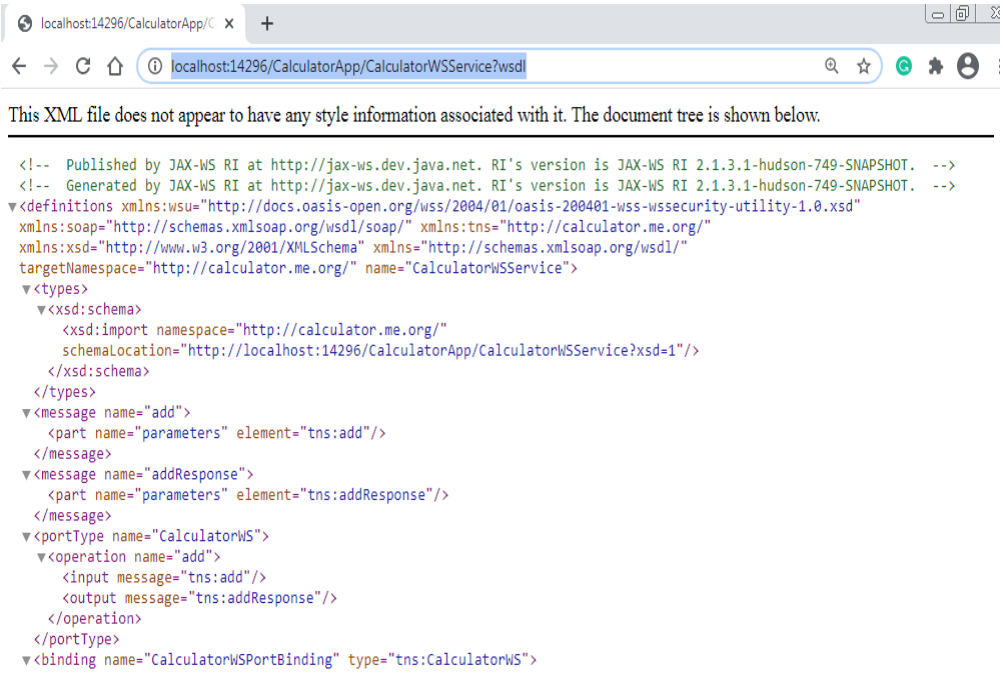
- Maintenait, déployer et exécuter le projet : Deploy et puis Run
- L'interface suivante sera affichée dans le navigateur



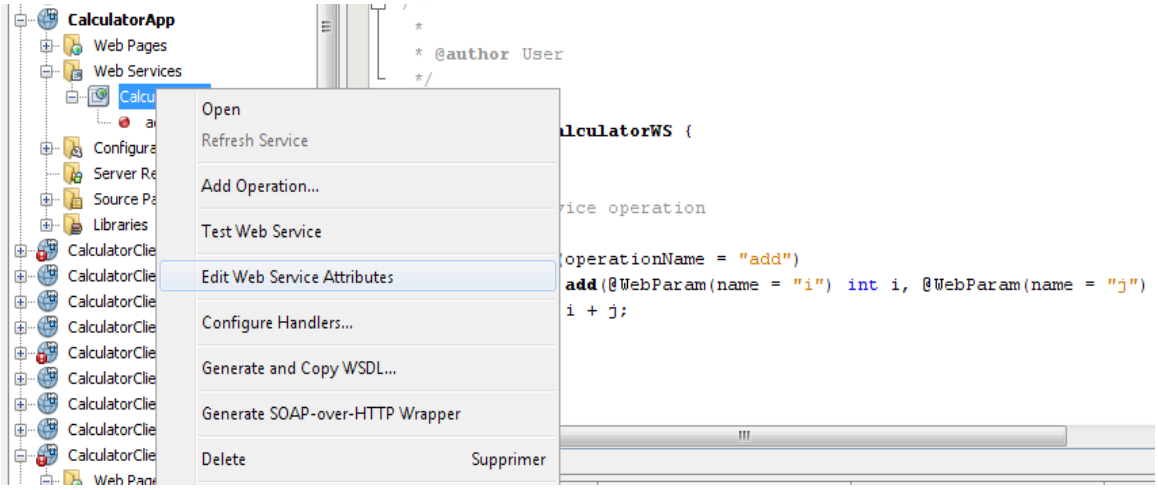
Services Web

Point dextrémité	Informations
Nom du service : {http://calculator.me.org/}CalculatorWSService Nom du port : {http://calculator.me.org/}CalculatorWSPort	Adresse : /CalculatorWSService WSDL : /CalculatorWSService?wsdl Classe d'implémentation : org.me.calculator.CalculatorWS

- Pour afficher la description WSDL, ajouter à l'URL de la page ?wsdl



- Pour tester le Web service, clic-droit sur le fichier CalculatorWS du projet CalculatorApp et choisir Test Web Service. Le Tester vous propose une interface de test.



- L'interface de test est la suivante :

- Tester maintenant le web service en saisissant deux entiers dans les zones de saisie et puis cliquer sur le bouton add (add est le non de l'opération dans le code source du web service).

- Le résultat est comme suit :

localhost:14296/CalculatorApp/C ✕ Suivi d'appel de méthode ✕ +

localhost:14296/CalculatorApp/CalculatorWSService?Tester

add Appel de méthode

Method parameter(s)

Type	Value
int	4
int	9

- En plus, la page contient la requête SOAP envoyer au Web service et la réponse SOAP reçue (En XML : Voir chapitre 1)

Demande SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:add xmlns:ns2="http://calculator.me.org/">
            <i>4</i>
            <j>9</j>
        </ns2:add>
    </S:Body>
</S:Envelope>
```

Réponse SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:addResponse xmlns:ns2="http://calculator.me.org/">
            <return>13</return>
        </ns2:addResponse>
    </S:Body>
</S:Envelope>
```

- Pour tester le Web service, vous pouvez aussi utiliser une logiciel de test comme SOAPUI ou ou ReadyAPI (nécessite téléchargement et installation). La figure suivante présente le tester SOAPUI

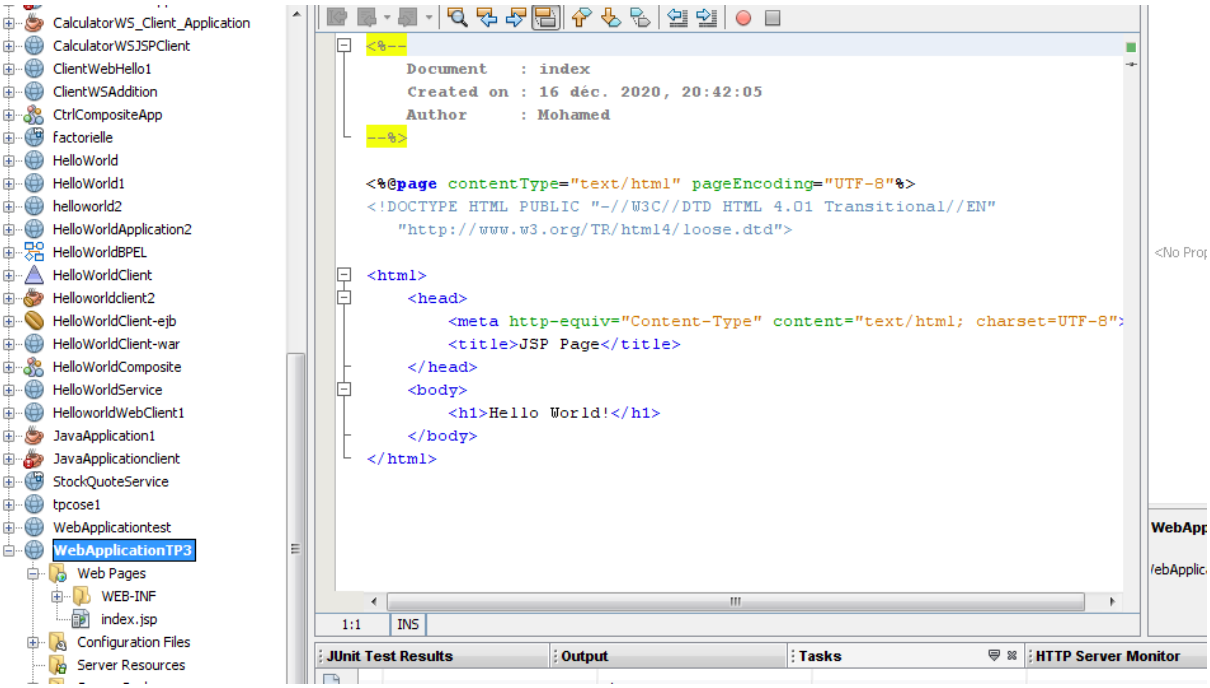
- Pour tester votre service, créer un nouveau projet en tapant : File -> New soapUI
- Project. Taper comme nom de projet HelloWorldTest et donner l'URL du fichier WSDL : http://localhost:14296/CalculatorApp/CalculatorWSService?WSDL
- Remplacer dans le fichier entrant (la requête SOAP) les mots « int » dans les deux balises <a> et <b> (les noms de paramètres de la méthode add dans le code source) par deux entiers, et puis cliquer sur le bouton executer et observer le résultat.

3. TP2 : Développement d'un client web service

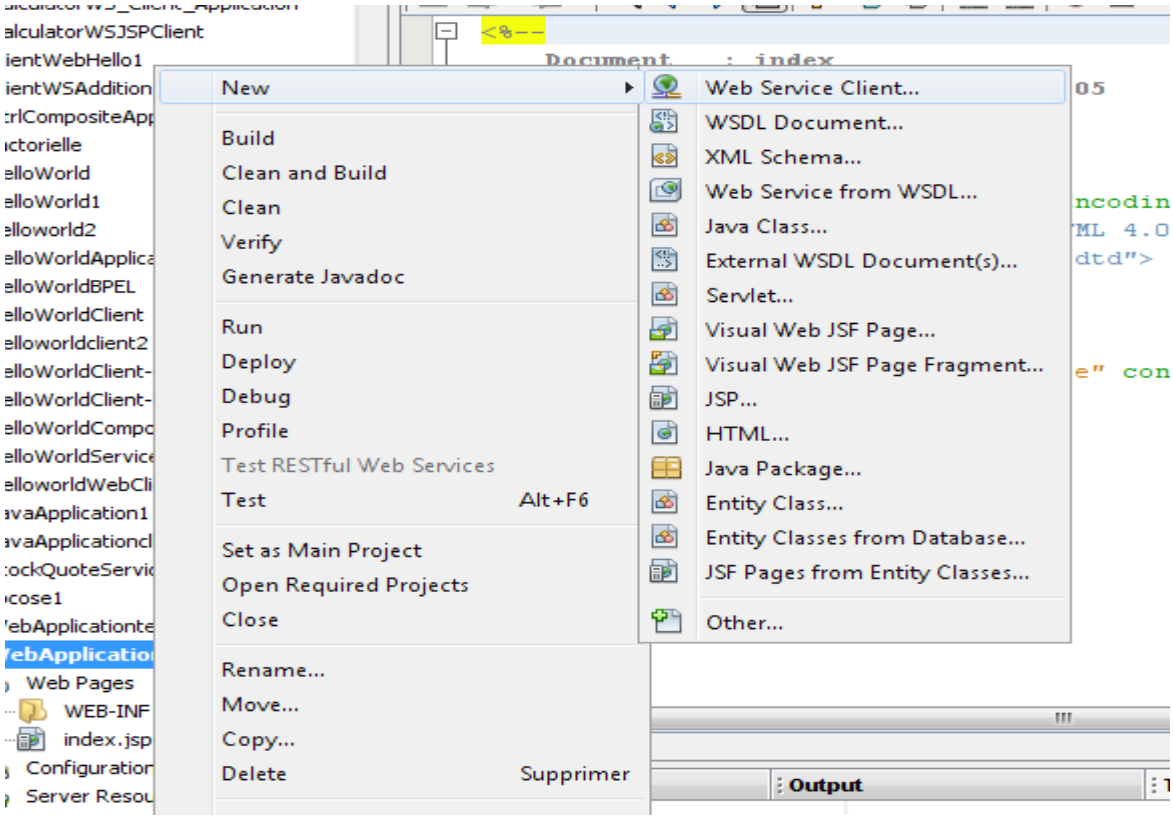
- Pour développer une Application Cliente pour le web service précédent, vous pouvez utiliser l'application CalculatorClientApp ou bien vous pouvez créer une autre nouvelle application (Web ou autre).
- Le principe est de développer une application qui représente une interface personnalisée pour consommer un Web service c à d vous pouvez même

développer une application cliente pour un web service fourni par un fournisseur externe (sur internet).

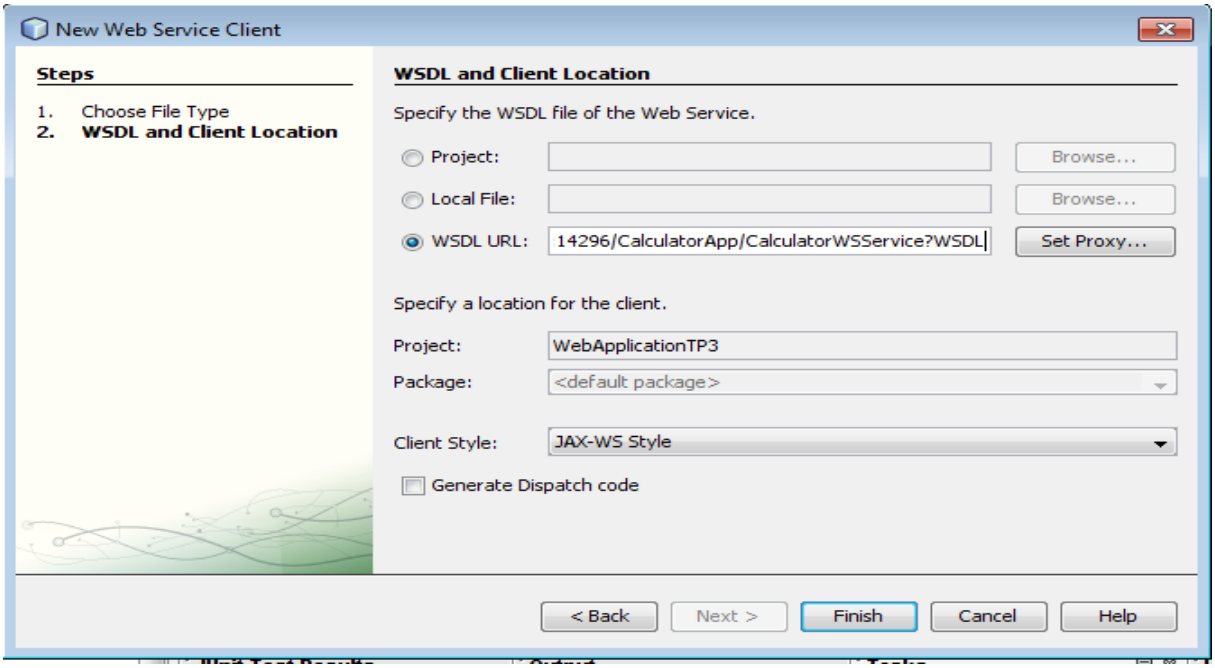
- Nous utilisons la description WSDL pour créer une liaison entre l'application et le web service (voir chapitre 1).
- Nous allons créer maintenant une application cliente sous frome d'une application Web (JSP).
- Aller à New project->java Web-> Application Web-> donner un nom->choisir le serveur Web pour le déploiement-> vous pouvez utiliser un Framework ou non (dans cette exemple non).
- Dans notre exemple le nom de l'application est WebApplicationTP3



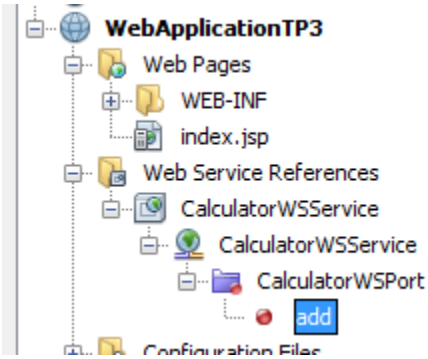
- Si vous trouver une page HTML au lieu d'une page JSP, supprimer la page HTML, Clic-droit sur l'application et ajouter une JSP
- Clic-droit sur l'application et ajouter ->New-> Web service Client



- Choisir WSDL URL et saisir l'URL du WSDL du Web service CalculatorApp-> Finish



- Un dossier « Web service References » sera ajouter à l'application « WebApplicationTP3 »
- Aller à la page JSP, puis entrer dans le dossier « Web service References » jusqu'à l'opération référenciée « add ».



- Faire glisser maintenant « add » dans le Body de la page JSP. Le code suivant sera généré automatiquement.

```
<%
try {
org.me.calculator.CalculatorWSService service = new org.me.calculato
org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
 // TODO initialize WS operation arguments here
int i = 0;
int j = 0;
// TODO process result here
int result = port.add(i, j);
out.println("Result = "+result);
} catch (Exception ex) {
// TODO handle custom exceptions here
}
%>
<%-- end web service invocation --%><hr/>
```

- Faire maintenant les modifications suivantes dans la page JSP :
  - Dans la partie HTML ajouter un formulaire qui contient deux zones de saisie (pour saisir les paramètres prochainement).
  - Dans la partie JSP, récupérer le contenu du formulaire
- Le code final sera comme suit :

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Client web service</h1>
      <form action="index.jsp" method="post">
            <input type="text" name="op1"/>
            <input type="text" name="op2"/>
            <input type="submit" value="addition"/>
        </form>
            <%-- start web service invocation --%><hr/>
    <%
```

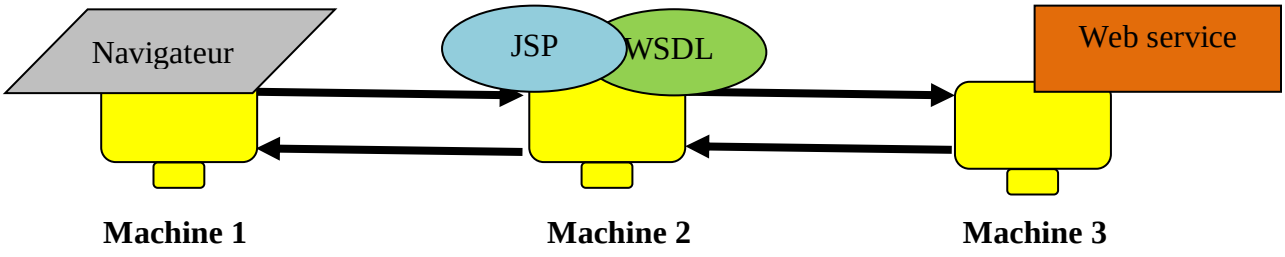
```
    <%
    try {
    org.me.calculator.AdditionService service = new org.me.calculator.Add
    org.me.calculator.Addition port = service.getAdditionPort();
     // TODO initialize WS operation arguments here
    String opr1 = request.getParameter("op1");
    String opr2 = request.getParameter("op2");
    int i = Integer.parseInt(opr1);
    int j = Integer.parseInt(opr2);
    // TODO process result here
    int result = port.add(i, j);
    out.println("Result = "+result);
    } catch (Exception ex) {
    // TODO handle custom exceptions here
    }
    %>
    <%-- end web service invocation --%><hr/>


    </body>
```

- Vérifier que le Web service est bien déployer, et exécuter maintenant l'application cliente WebApplicationTP3

Remarque

Le déploiement du web service et l'exécution de l'application cliente simule 3 machines : une machine cliente, un serveur web exécute l'application web cliente (la page JSP) et le dernier serveur exécute le web service. Pour bien observer ce système distribué, tester ce TP sur Trois machines. Dans la première ouvrir le navigateur et saisir l'URL de la page JSP en remplaçant « localhost » par l'adresse IP de la deuxième machine. Aller à l'application cliente dans la deuxième machine et remplacer « localhost » dans l'URL de la référence WSDL par l'adresse IP de la troisième machine. Le schéma suivant

démontre ce système distribué et la communication via l'interface WSDL entre l'application cliente et le web service.



4. TP 3 : Développement d'un web service (Top-Down)

Pour réaliser un service Web à partir d'un fichier WSDL existant (voir chapitre 1), suivre les étapes suivantes :

- Créer un nouveau projet de type Web Application
- Faire un clic-droit sur le nom du projet créé, et choisir : New -> Web Service from WSDL.
- Nommer le web service et le package et taper dans la case « *Select Local WSDL File or Enter WSDL URL* » l'URL vers le WSDL du Web service développer dans le TP 1 http://localhost:14296/CalculatorApp/CalculatorWSService?WSDL
- Cliquer sur *Finish*.
- Modifier le contenu de la classe de manière à implémenter un nouveau web service (l'implémentation est identique à celle du premier web service)
- Clic-droit sur votre projet, et choisir *Clean and Build*.
- Exécuter le Web service maintenant

5. Composition de web services avec BPEL

5.1. Concepts de base

Avant de commencer, il faut définir un ensemble de concepts pour développer un processus BPEL:

- **Les liens des partenaires** (*Partner Link*) : un partenaire est tout service externe ou client qui interagit avec le processus BPEL. Les partenaires externes peuvent être des services web, des bases de données, ou d'autres processus BPEL.
- **Les activités** : ce concept est bien défini dans le deuxième chapitre. Pratiquement, une activité est une tâche métier individuelle dans le processus. L'assemblage et le regroupement des différentes tâches permettant de réaliser un objectif final ou partiel du processus BPEL. Il reçoit au début une requête (généralement à partir d'un client), un processus BPEL commence toujours avec une activité de réception (*receive*), puis invoque des services externes (*invoke*) et enfin renvoient le résultat au client (*reply*).
- **Les variables** : Pour assurer la communication entre les activités du processus ou bien entre le processus et les partenaires, plusieurs variables sont généralement utilisées. L'activité *Assign* par exemple est toujours présente entre entre les activités *Receive* et *Invoke*, pour faire le mapping Input/output. Cela peut être réalisé par une simple opération de copie, ou par un traitement sur les entrées (concaténation, somme…).

5.2. Hello World avec BPEL

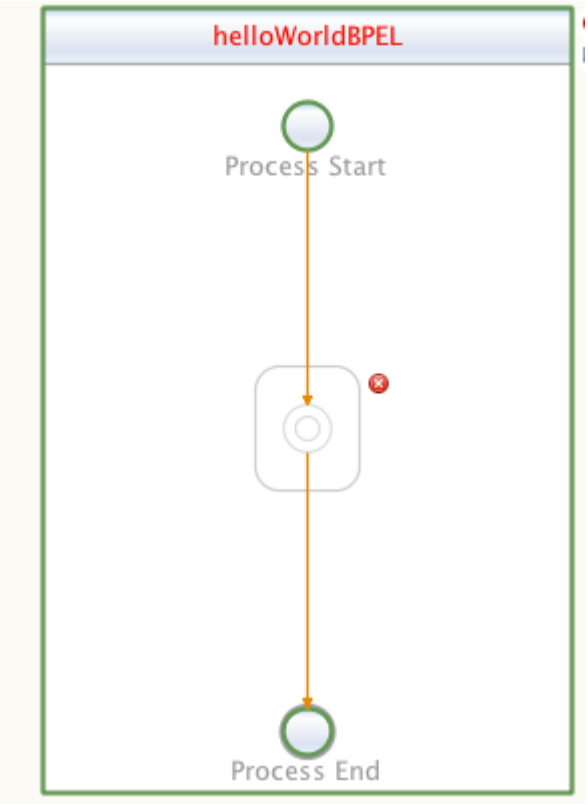
L'objectif de ce processus BPEL est d'implémenter l'exemple « Hello X ». Pour développer cet exemple, le processus permettant la lecture en entrée d'une chaine de caractères (un nom X), et de la concaténer avec une autre chaine (Hello) et l'envoi du résultat en sortie.

Les étapes à suivre sont les suivantes :

- Création du processus BPEL
- Définir un nouveau type de données complexe en utilisant XSD (cette étape est facultative pour cet exemple, car nous pouvons utiliser directement le type simple prédéfini String, mais cette étape peut être utile pour d'autres exemples).
- Définir une application composite qui appelle de processus BPEL

Pour commencer

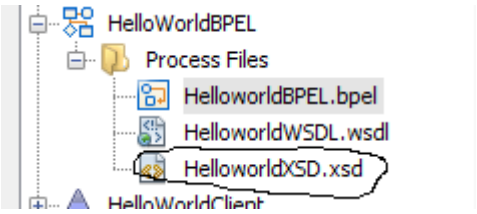
- New Project, SOA, BPEL Module.
- Nommer le projet (par exemple HelloWorldBPEL).



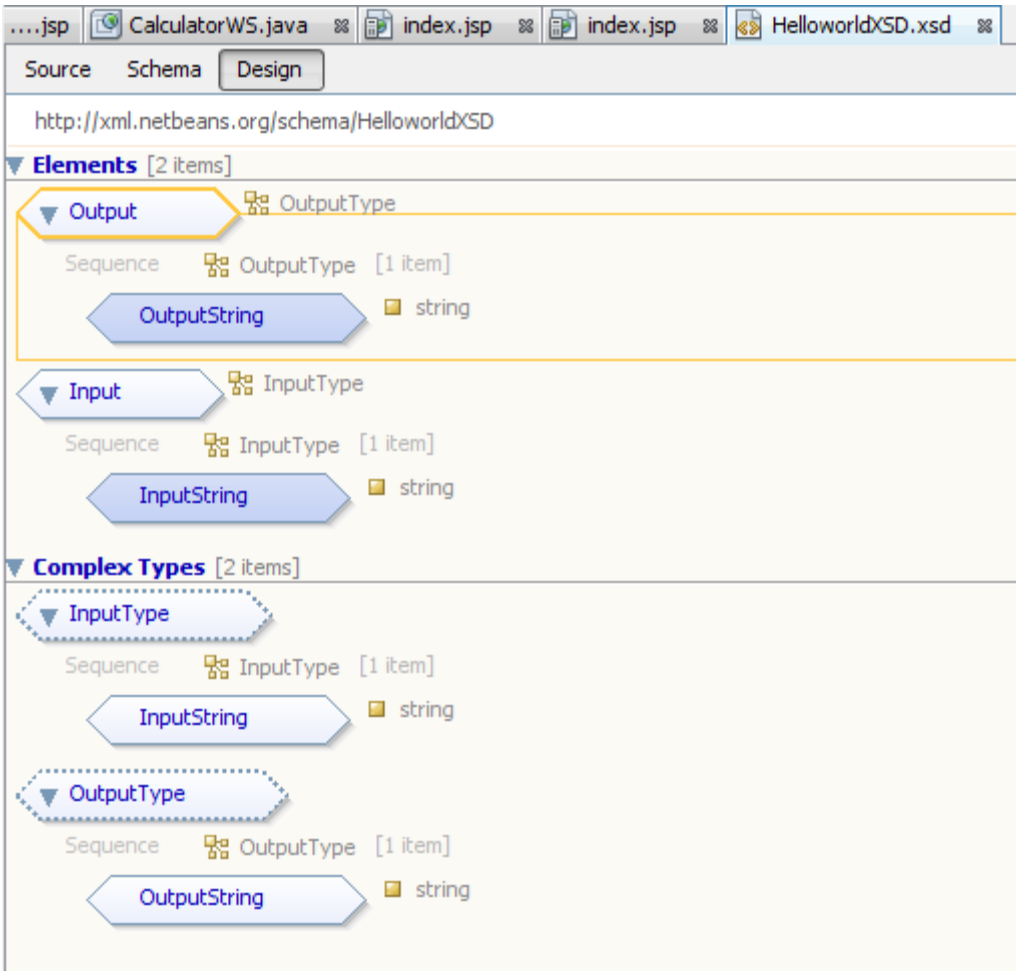
Avant de définir le partenaire client qui va fournir les entrées au processus et qui sera décrit pas un fichier WSDL, nous allons d'abord réaliser un fichier XSD (XML Schéma Definition) pour proposer un nouveau type de données complexe permettant la définition des types de messages qui seront échangés dans ce processus (une chaîne de caractères en entrée et une autre en sortie).

Pour créer un nouveau fichier XSD:

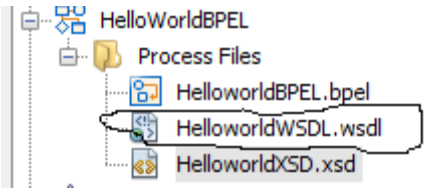
- Clic-droit sur le répertoire *Process Files* du projet, *New, XML Schema*
- Donner un nom (par exemple *HelloworldXSD*)



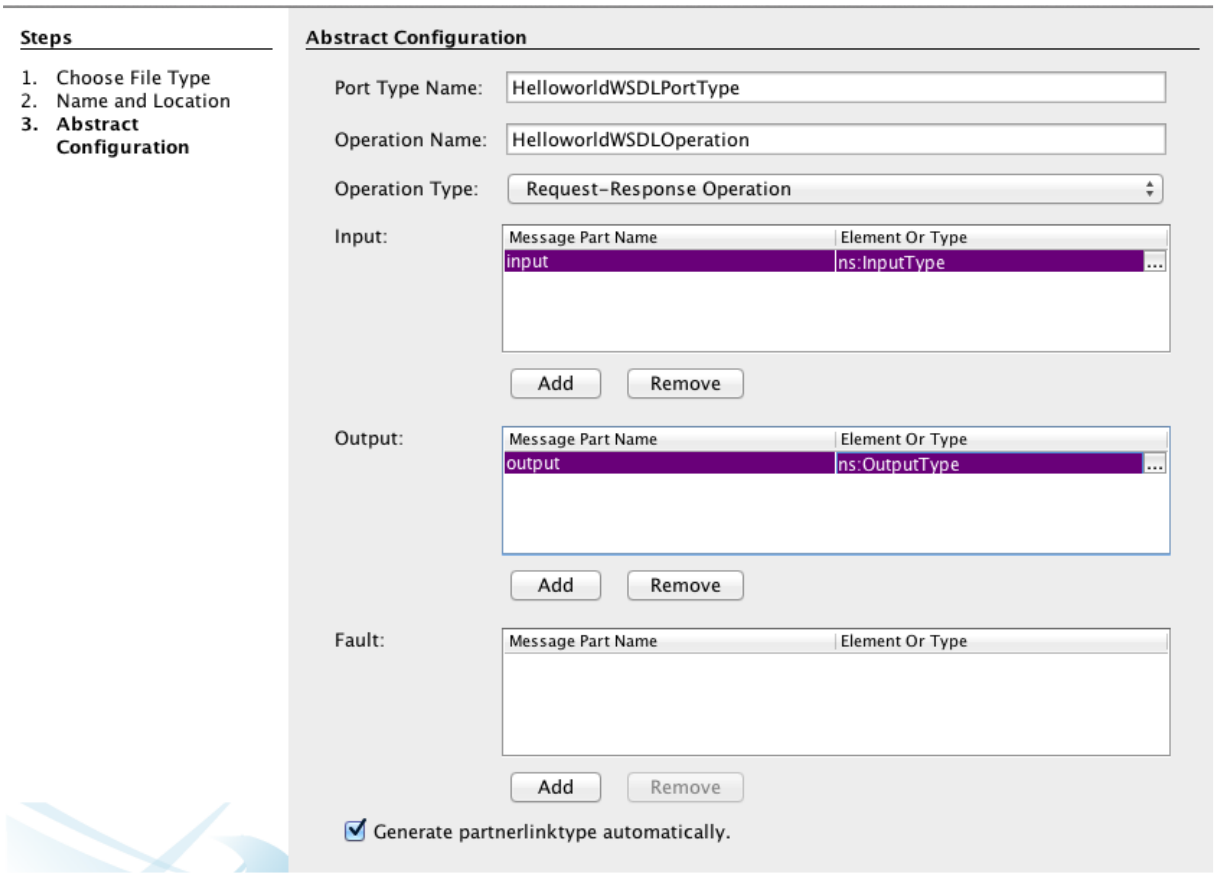
- Aller à l'onglet *Design* pour une représentation graphique du schéma.
- Faire glisser *Complex Type* à partir de la palette (située à droite) vers le champ *Complex Types*.
- Définir deux types complexes *InputType* et *OutputType*, avec chacun un élément simple, respectivement *inputString* et *outputString* de type *string* (voir **[30]** pour plus de détails sur ce point).



Pour créer un partenaire qui représente le client, définir le fichier WSDL correspondant:



- Clic-droit sur *Process Files, New, WSDL Document*
- Nommer le fichier (*HelloworldWSDL)*
- Dans la partie *Abstract Configuration*, définir le type des entrées et des sorties respectivement dans les cases Input et Output, comme suit:

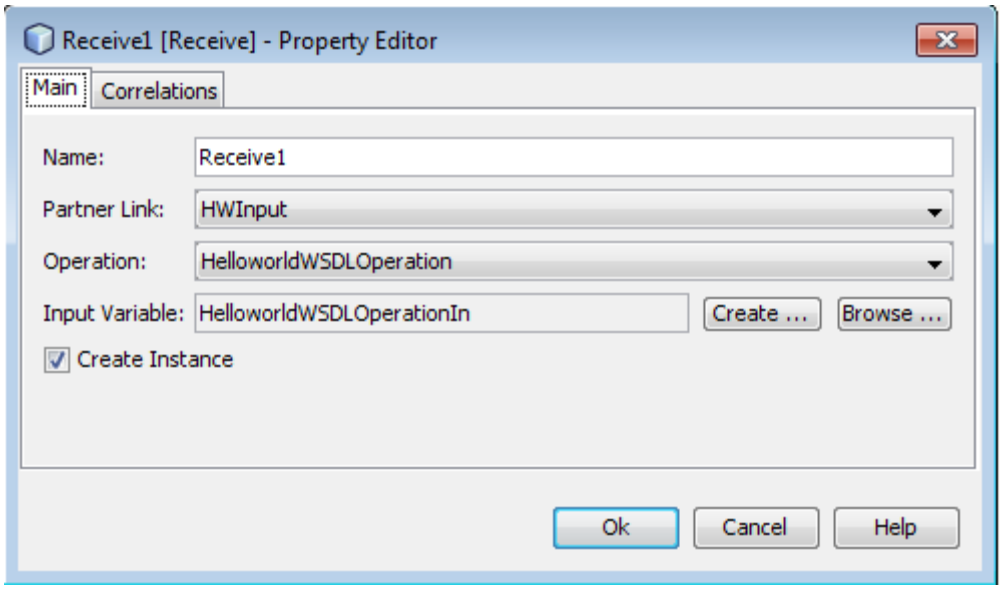


Pour compléter l'implémentation du processus BPEL:

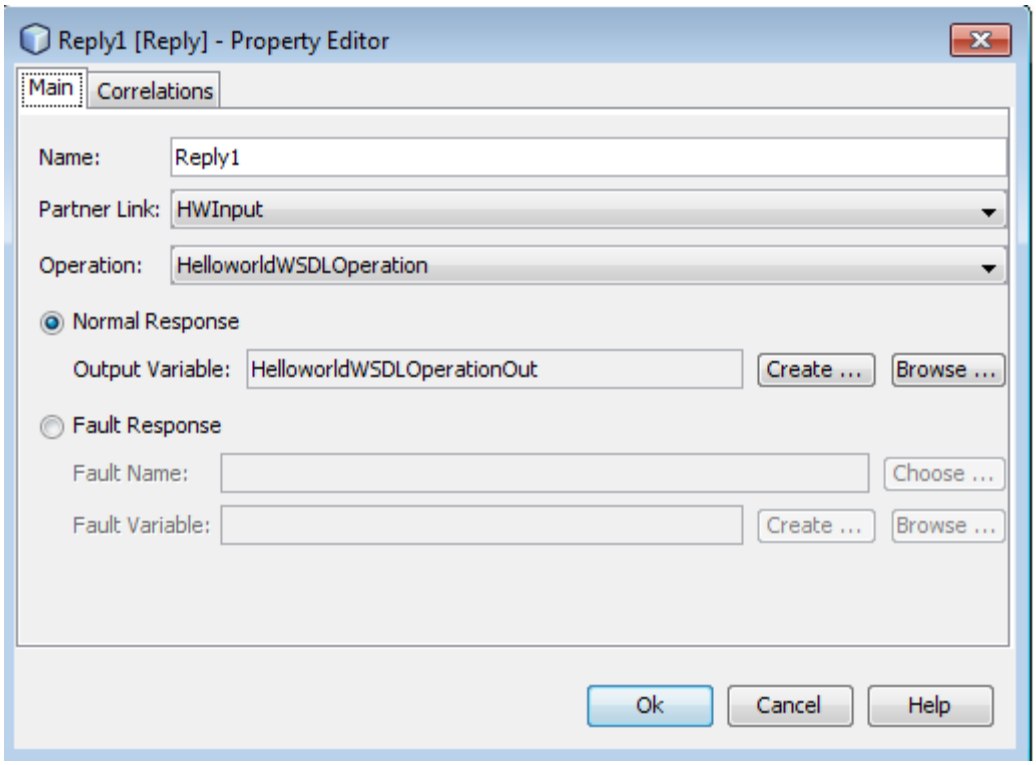
- Ouvrir le fichier *HelloworldBPEL* et choisir l'onglet *Design*.
- Faire glisser le fichier WSDL (le partenaire client) créé dans la fenêtre principale (des cercles oranges apparaitront aux endroits où il est possible de placer le fichier). Placer le fichier WSDL à gauche du processus *helloworldBPEL*.
- Renommer le nouveau *Partner Link* (par exemple *HWInput*).

A partir de la palette de droite, faire glisser respectivement les activités suivantes entre les activités *Process Start* et *Process Stop*:

- **Receive** : pour recevoir la chaîne en entrée à partir du service décrit par HelloWorldWSDL (le partenaire client). Double-clique sur cette activité pour la configurer. Préciser le Partner Link à partir duquel les données seront reçues (HWInput), l'opération visée (HelloworldWSDLOperation) et générer la variable en entrée en cliquant sur le bouton *Create*.



- **Reply** : pour envoyer le résultat du processus au service client. De la même manière que l'activité Receive, configurer cette activité.

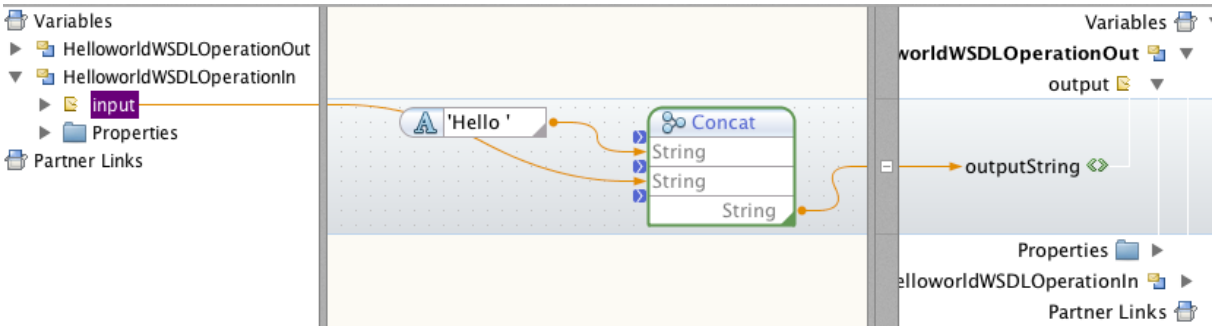


- **Assign** : toujours à partir de la palette, insérer cette activité entre les deux activités *receive* et *reply*. Elle permet d'affecter les variables en entrée aux variables en sortie du processus.

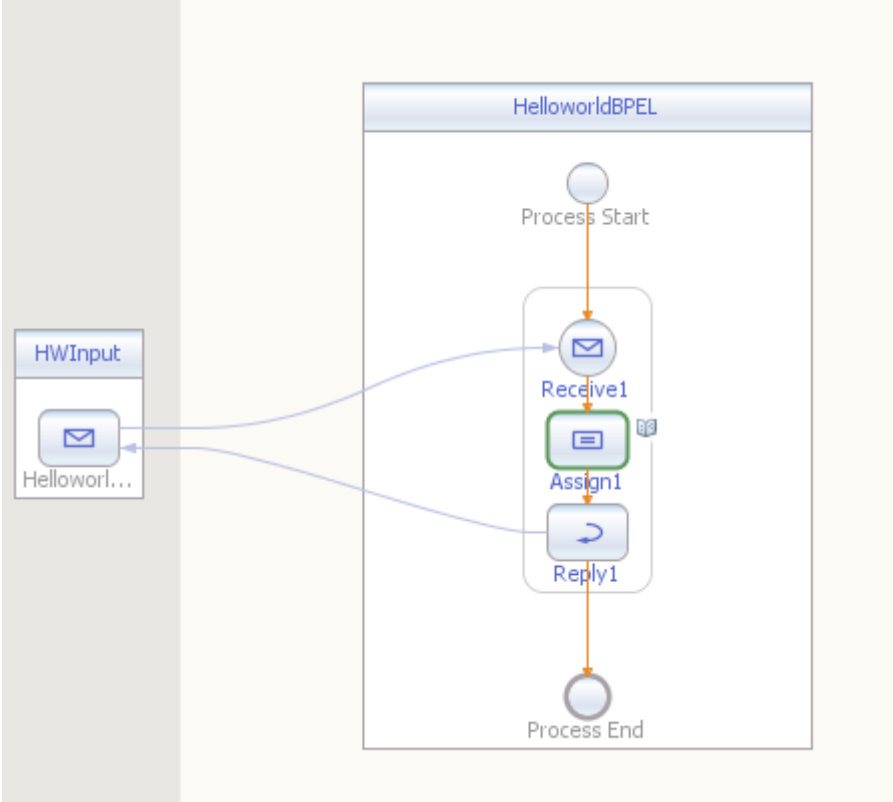
Pour configurer cette activité, double-clique dessus. Une fenêtre BPEL Mapper s'ouvrira.

- Cliquer sur le bouton String sur le menu supérieur, et choisir String Litteral.

- Faire glisser dans la fenêtre principale, et écrire la chaîne « Hello ».
- Faire glisser de la même manière String->Concat dans la fenêtre principale.
- Relier les différents éléments de manière à obtenir le mapping suivant:



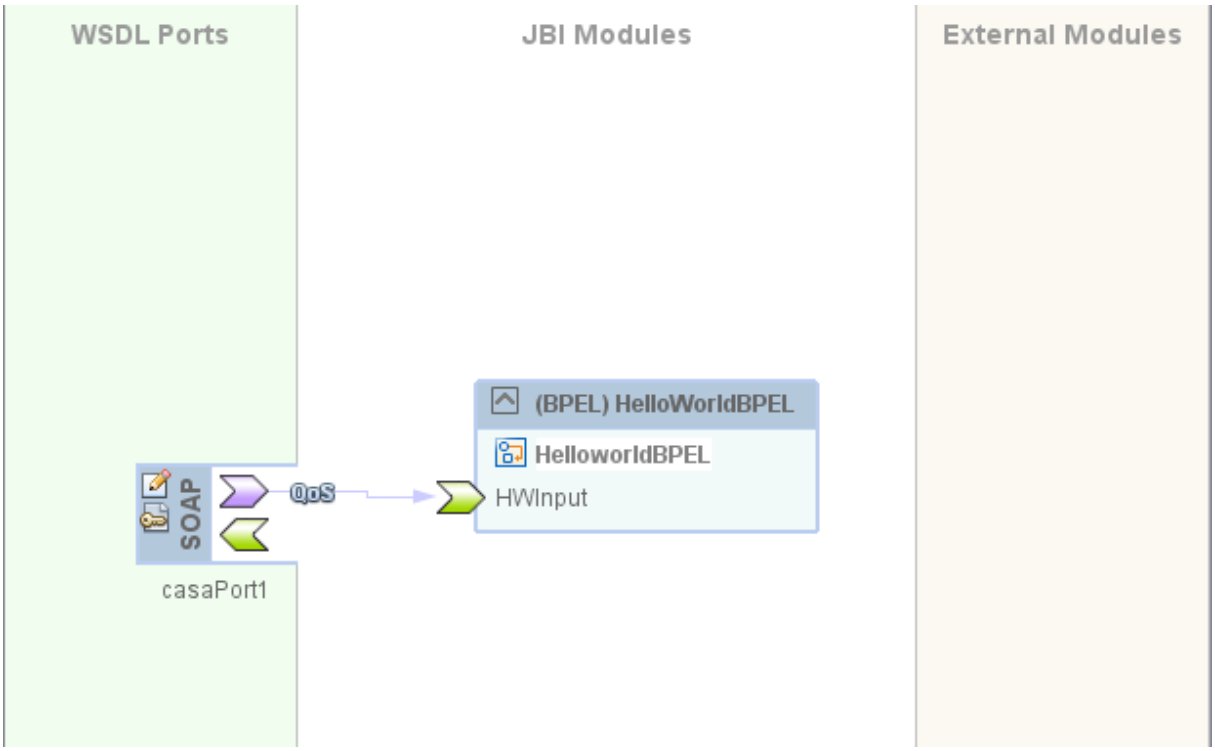
Le résultat final du processus est donc sera :



Pour créer l'application composite:

- *New Project, SOA, Composite Application*
- Nommer l'application (*HelloWorldComposite* par exemple)
- Double-clique sur *Service Assembly*. La fenêtre est divisée en 3 parties: ports WSDL, modules JBI, et une pour les modules externes.
- Faire glisser le processus *HelloworldBPEL* dans la partie JBI Modules.
- Cliquer sur Build pour voir son contenu.

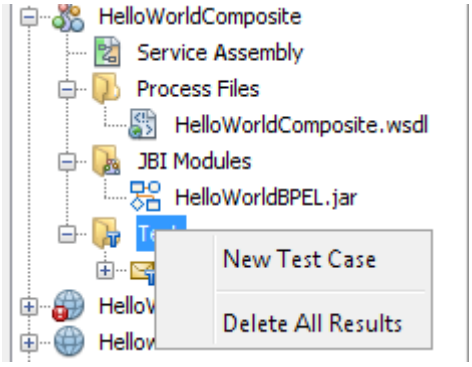
- De la palette à droite, faire glisser un binding SOAP dans la partie WSDL Ports.
- Relier le binding SOAP au module BPEL comme suit :



- Enregistrer et Déployer l'application.

Pour tester l'application composite :

- Clic-droit sur le répertoire Test



- Choisir *New Test Case*
- Donner un nom (*HelloworldTest*).
- Choisir le fichier WSDL et l'opération correspondente
- Clic-droit sur le fichier *HelloworldTest*, et lancer le test
- Au début un message d'erreur s'affiche. Cliquer sur Yes.
- Effectuer une deuxième exécution et consulter le résultat.

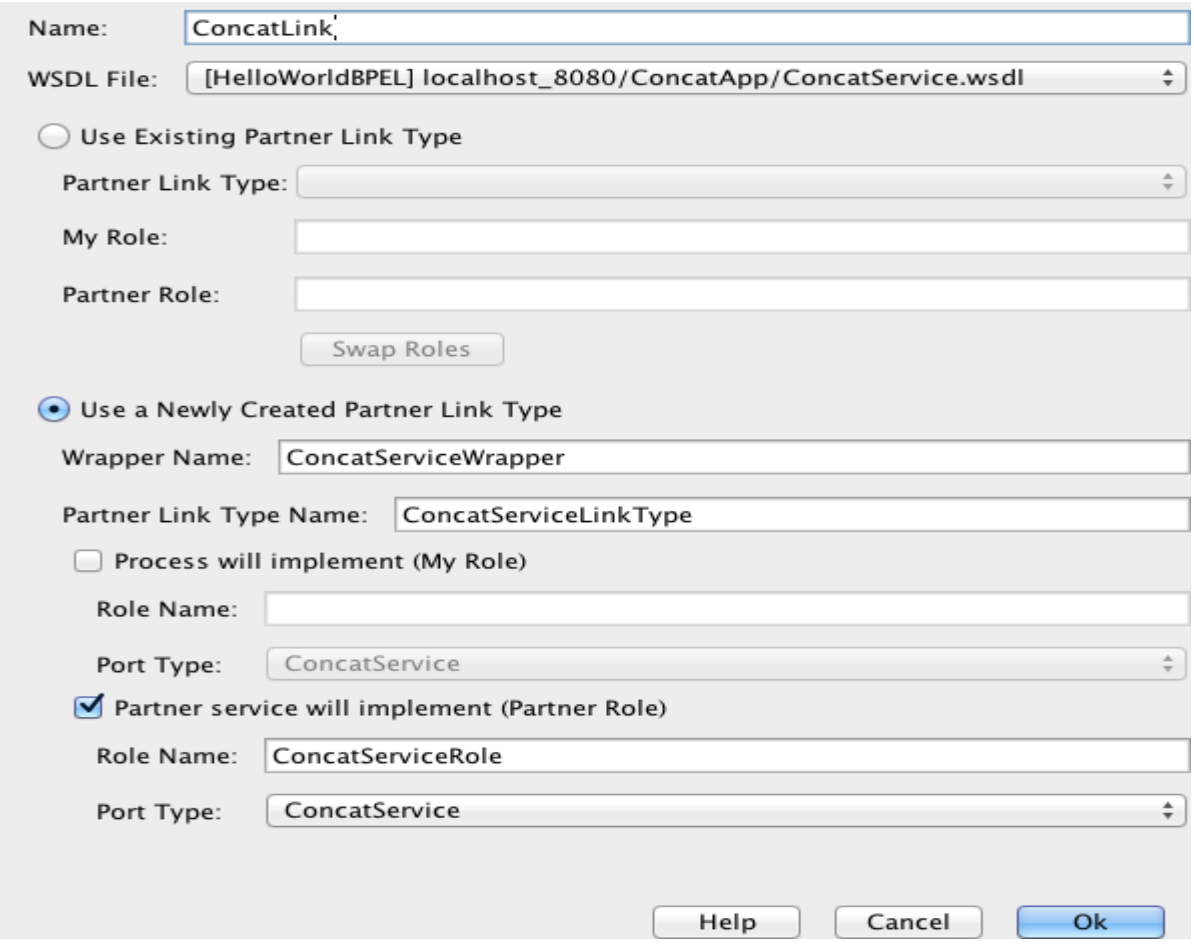
5.3. Appel d'un service web externe

Nous reprendrons le même exemple précedent, mais cette fois l'opération de concaténation est réalisée par un web service indépendant (externe) au lieu de l'implémenter directement dans le processus BPEL.

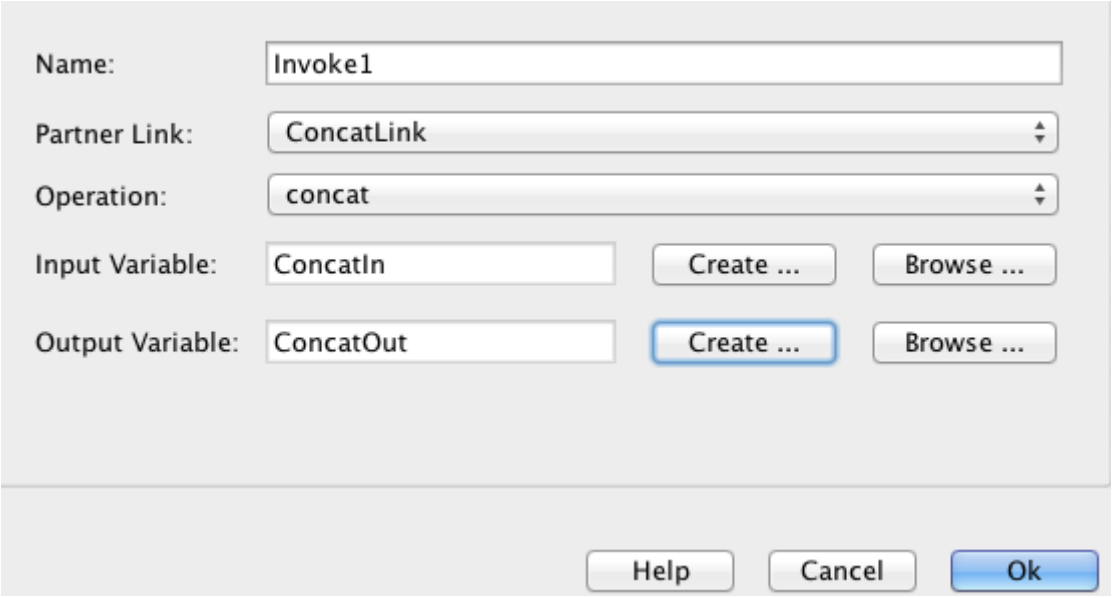
- Suivre les étapes présentées dans le premier TP (TP1) pour créer un nouveau service web intitulé (*Concat* par exemple).
- Le service expose une seule opération qui saisit deux chaînes de caractères, et retourne leur concaténation.
- Enregistrer et Déployer le nouveau service d'abord pour obtenir sa description WSDL.

Ajouter le service externe comme partenaire au processus BPEL

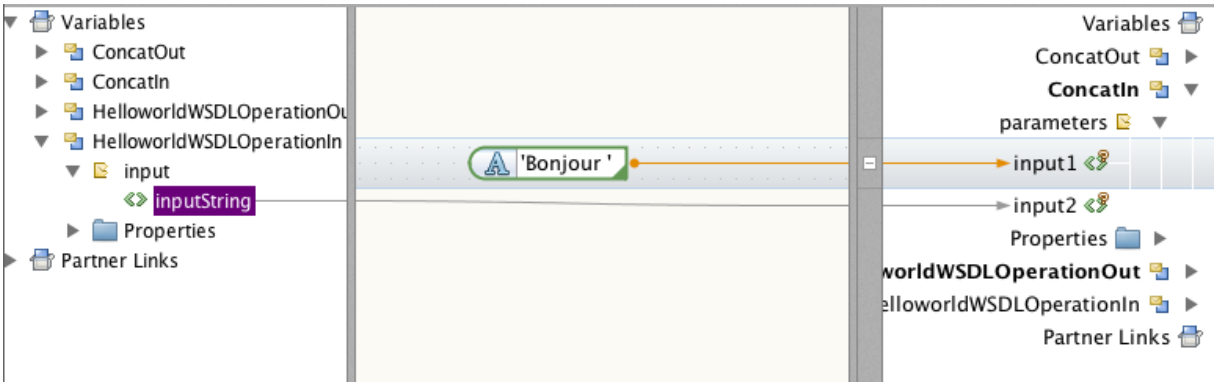
- Clic-droit sur *Process Files* de l'application *HelloworldBPEL*, et choisir *New* , *External WSDL Document(s)*
- Spécifier l'URL du fichier WSDL du service externe.
- Faire glisser le fichier WSDL importé vers la partie droite de la fenêtre principale.
- Nommer le partner Link créé (*ConcatLink* par exemple).



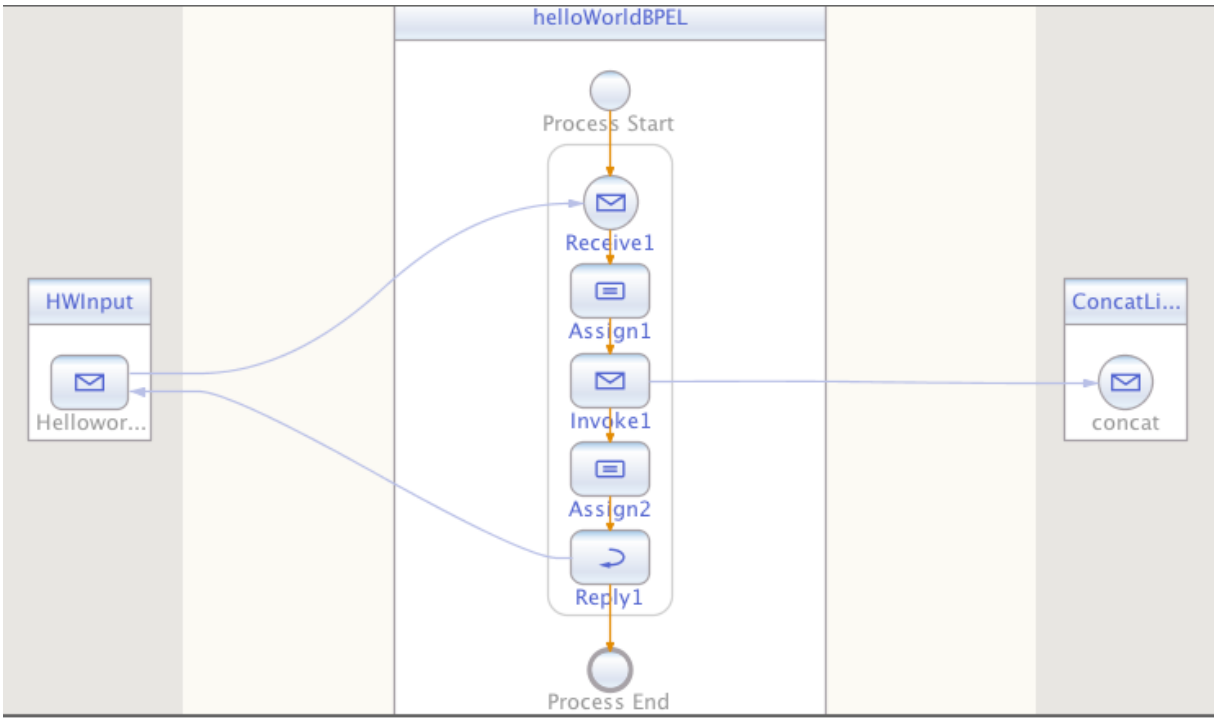
- Dans le processus BPEL, supprimer l'activité Assign.
- Ajouter l'activité **Invoke** entre les activités *Receive* et *Reply* (pour faire appel au service externe).
- Configurer l'activité *Invoke* comme suit:



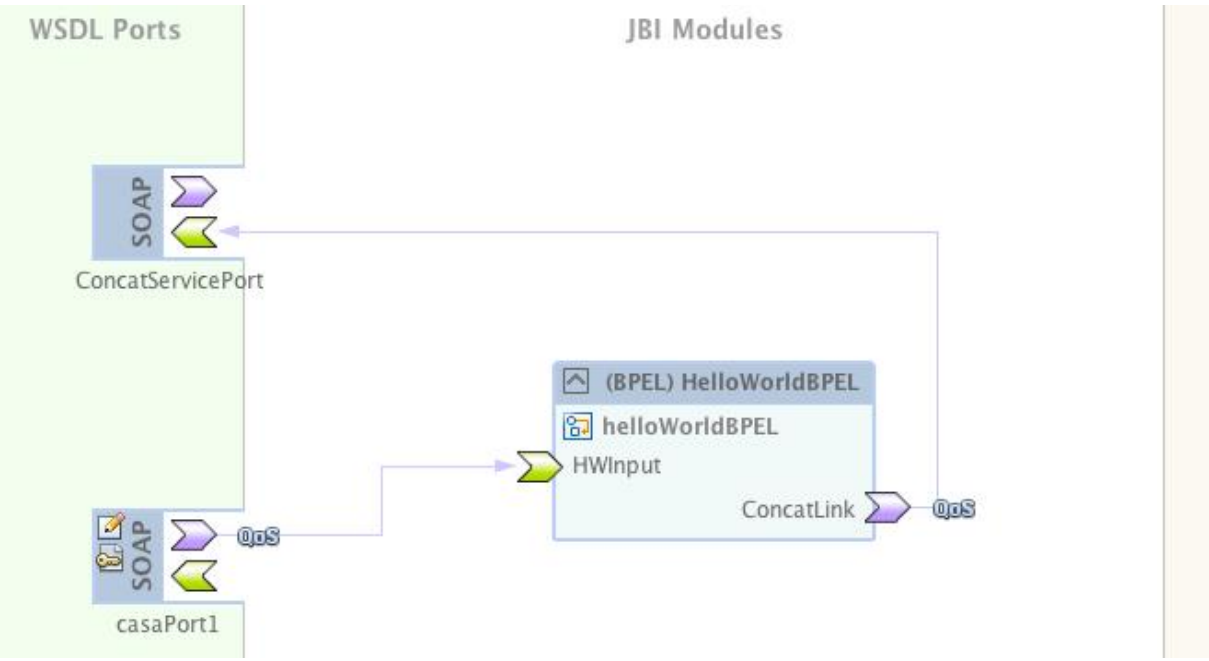
- Insérer deux activités *Assign*, respectivement entre *Receive et Invoke*, et entre *Invoke et Reply*.
- Configurer le premier *Assign* comme suit:



- Configurer le deuxième *Assign* en associant les deux variables de retour.



– Revenir à l'application composite, et la mettre à jour.



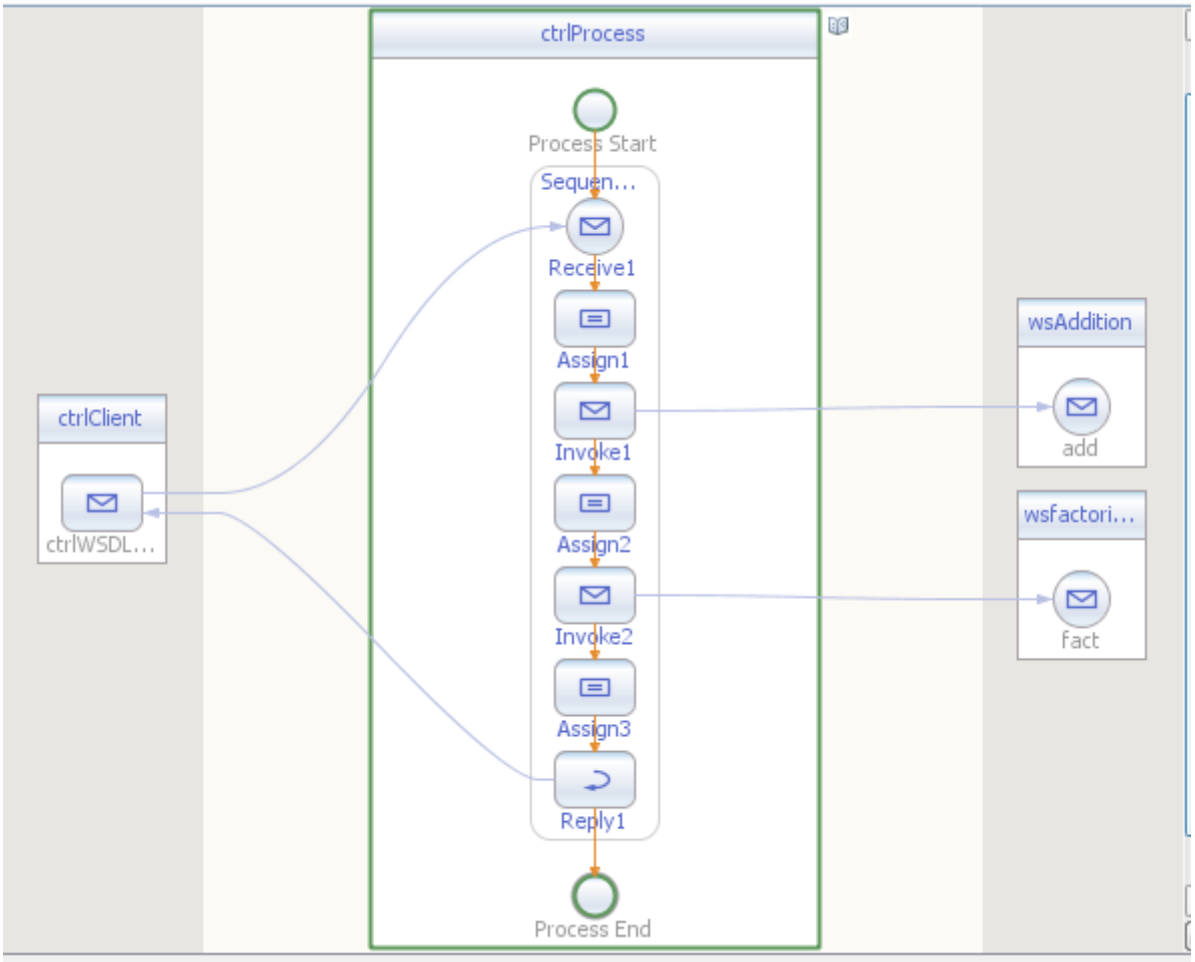
– Tester l'application composite.

5.4. Exercice 1

Réaliser le processus ci-dessous.

– Il reçoit deux entiers en entrée.

- Le premier service web (wsAddition) calcul la somme des deux entiers
- Le service web wsfactoriel calcul le factoriel de la somme résultante.
- Toutes les activités sont réalisées en séquentielle (une activité sequence englobe les activités)



5.5. Exercice 2

Essayer de développer l'exercice précédent sous forme d'un système distribué :

- Déployer le service web « wsAddition » sur une machine
- Déployer le service web « wsfactoriel » sur une deuxième machine
- Déployer l'application composite du processus BPEL sur une troisième machine

Remarque : remplacer « localhost :port » par les adresses IP correspondantes

- Développer une application cliente pour le processus BPEL sur un 4eme machine (voir TP 2 pour plus de détails).

6. Enoncé TP BPEL

Développer une application BPEL qui compose trois services web externes:

- Un service web « searchISBN » qui a comme entrée le titre d'un ouvrage et le nom de son auteur (en cas de plusieurs auteurs le nom sera: « premeir-auteur et al »), et en sortie il donne l'ISBN de l'ouvrage correspondant.
- Un deuxième service web «GiveBookPrice » qui a comme entrée l'ISBN d'un ouvrage et en sortie son prix en dollar.
- Le troisième service web est un convertisseur « Dollar-Dinar Algérien » qui comme entrée un nombre réel qui représente le prix en dollar (USD) et en sortie son équivalence en Dinar Algérien (DZD).

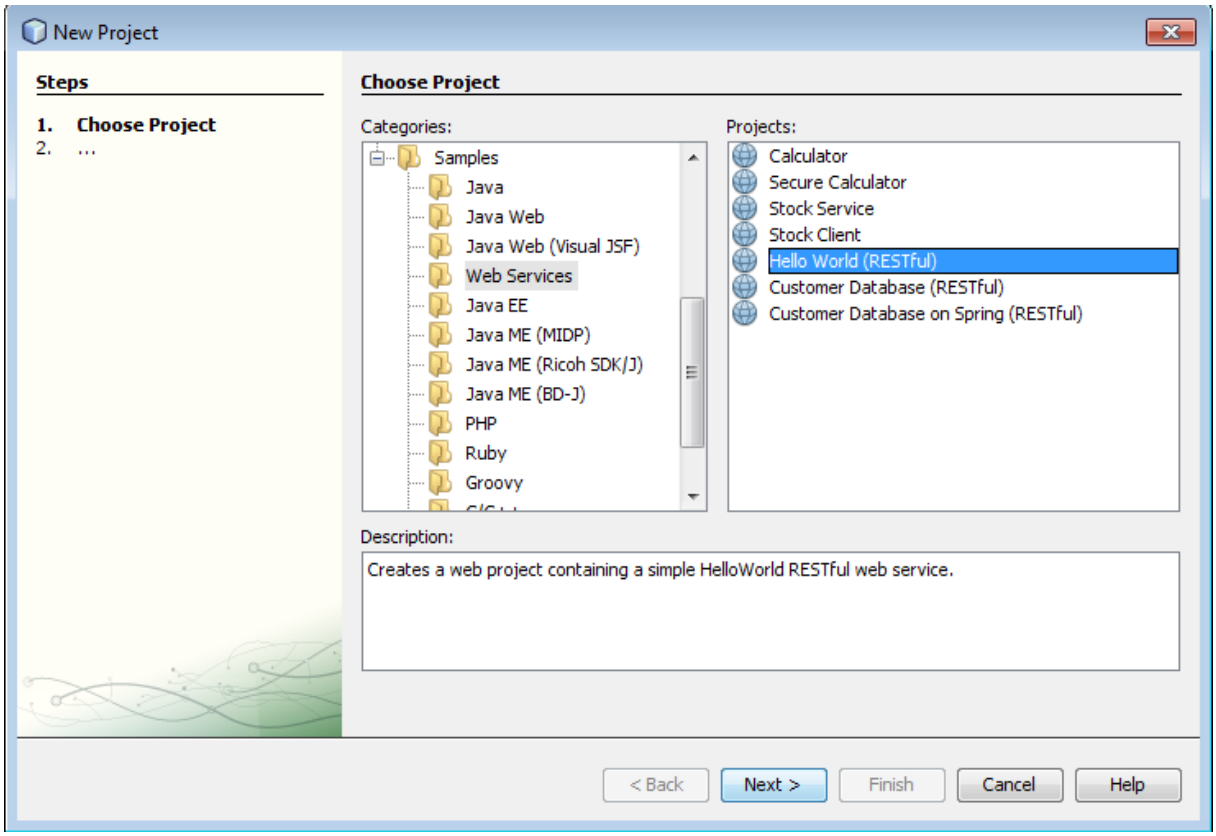
Remarque : chaque service web utilise sa propre base de données

7. Développement d'un service web RESTful

7.1. Création du service web

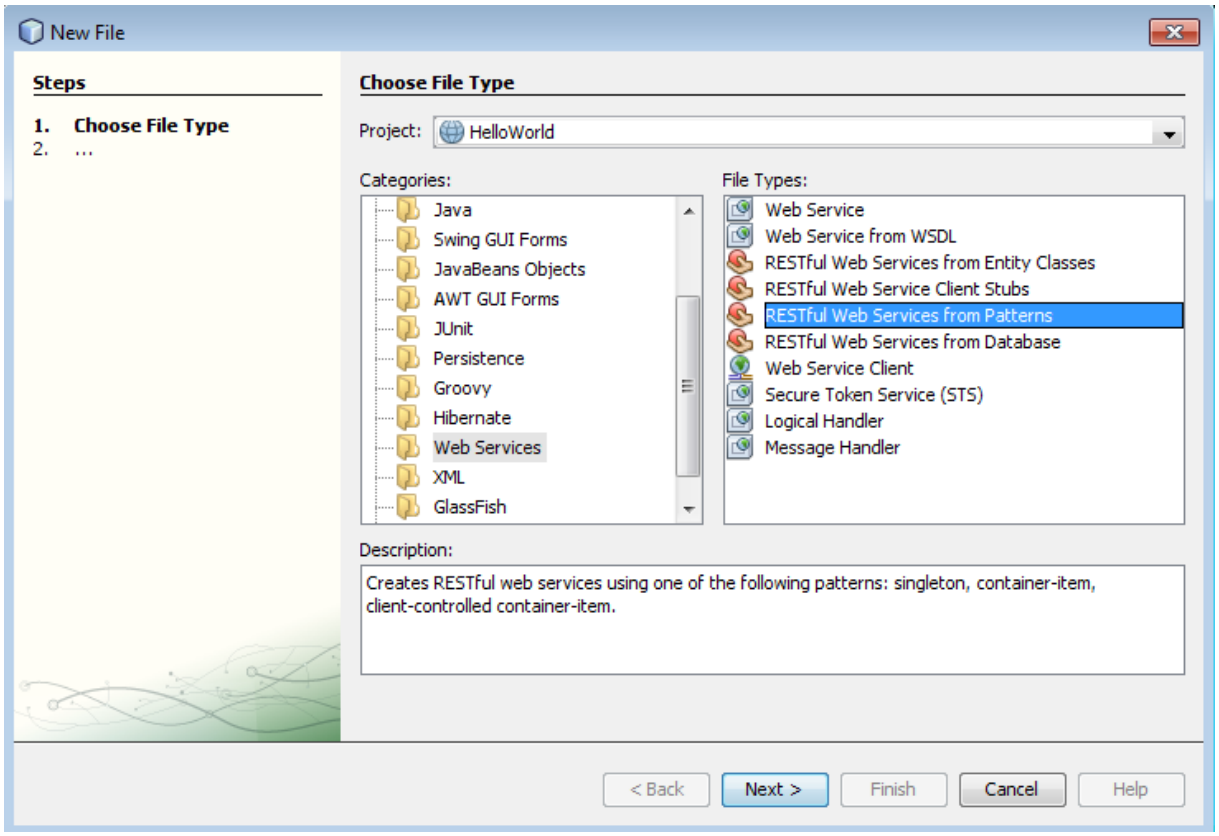
Nous allons utiliser l'IDE Netbeans pour créer ce service web. Sachant que le serveur web Glassfish est bien installé et configurer.

- Aller à New Project, Samples, Web services, Hello World (RESTful)

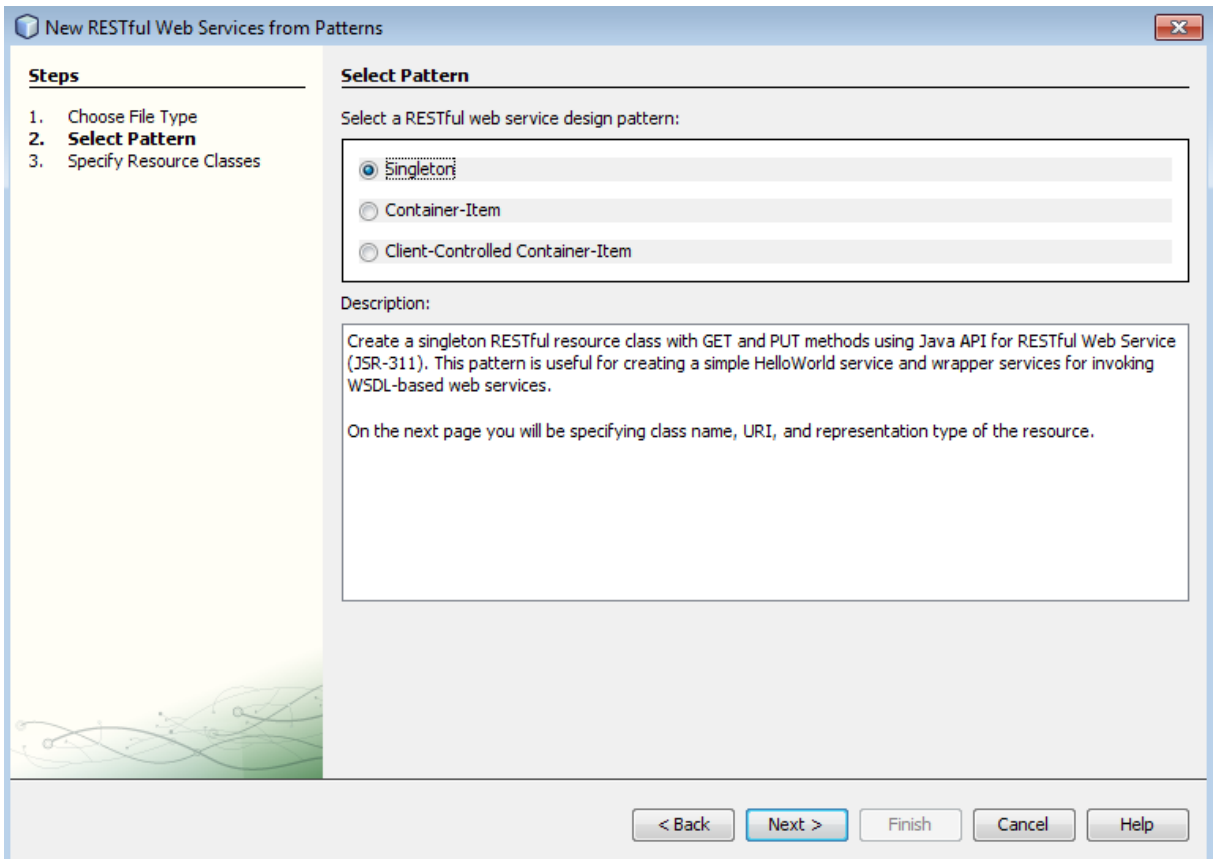


Ou bien, vous pouvez suivre les étapes suivantes :

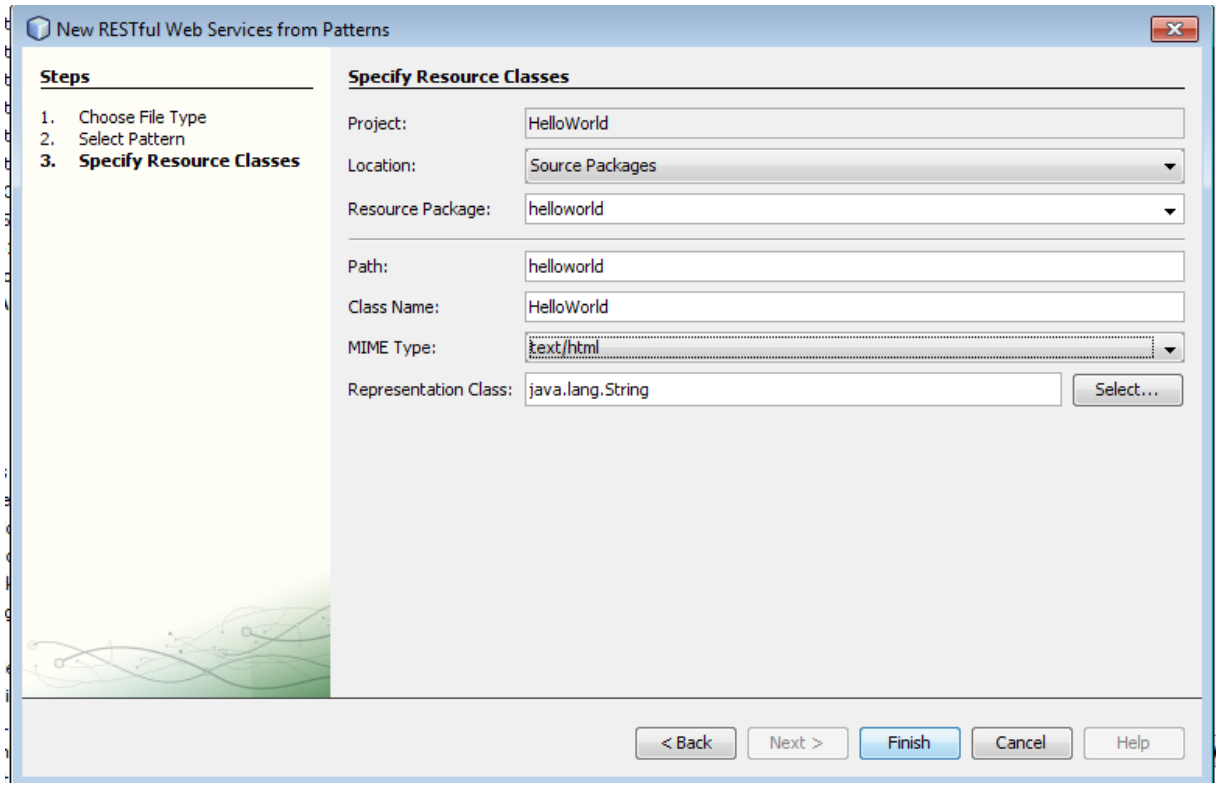
- Créer une application web simple: New project, java web, Web Application
- Nommer l'application
- Faire un clic-droit sur le projet, et sélectionner *New-> RESTful Web Services from Patterns*. Ou bien, *New->Other->Web service-> RESTful Web Services from Patterns*



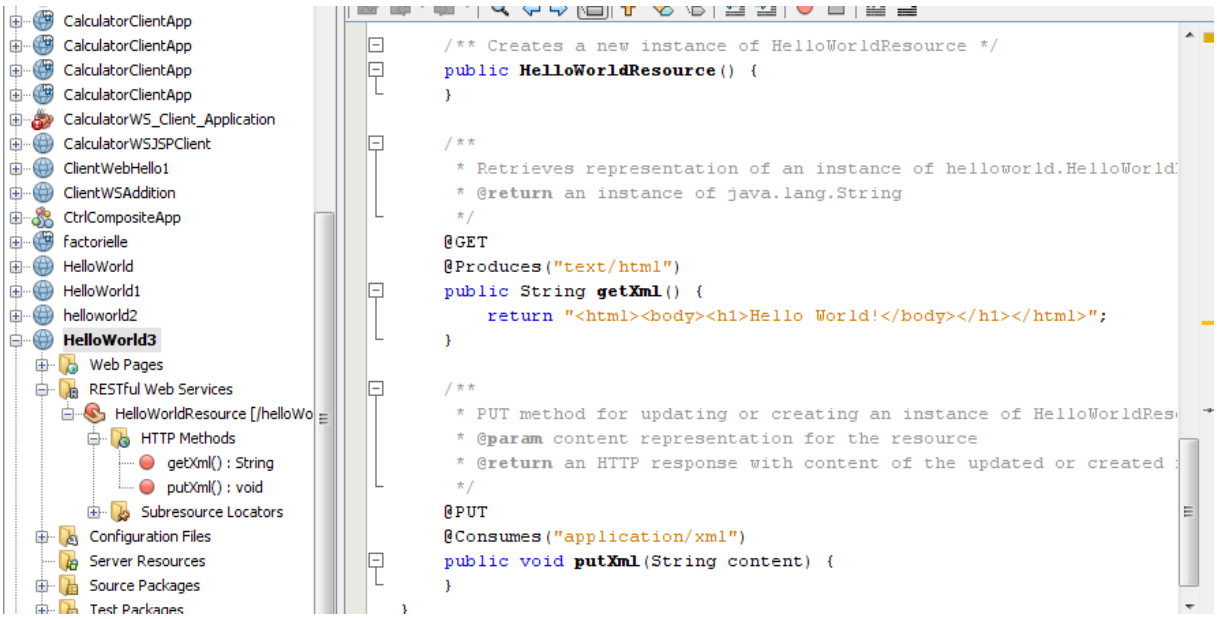
- Choisir Sigleton pour créer service web simple (hello world) avec deux méthodes HTTP GET et PUT



- Spécifier comme design pattern *Simple Root Resource*
- Spécifier un nom de package
- Nommer le Path
- Nommer la classe
- Utiliser un type MIME (text/html par exemple)



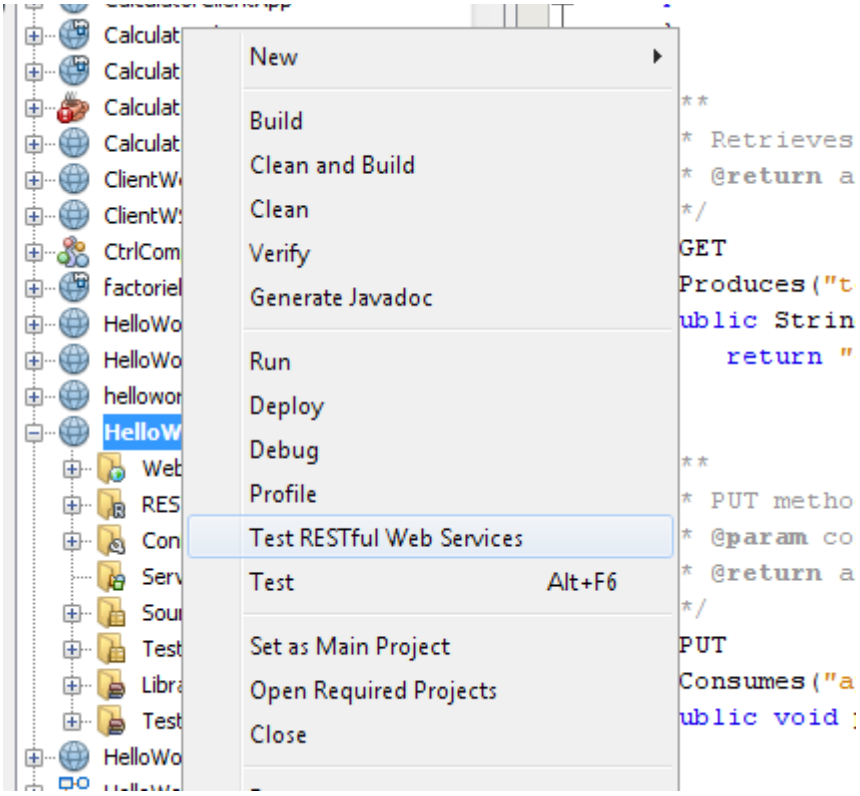
Nous allons continuer en utilisant la première méthode où le nom de la classe est « HelloWorldResource », le code aura l'apparence suivante :



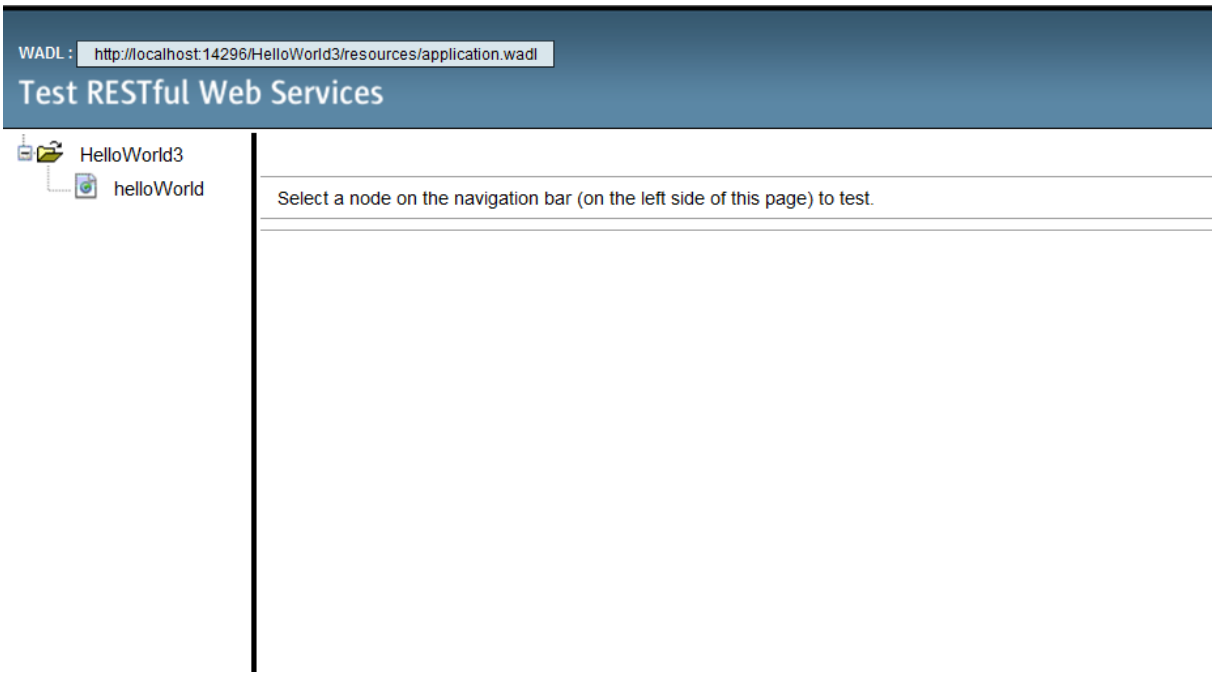
- La méthode *getHTML()* représente une requête de type GET pour afficher le contenu d'une ressource (dans cet exemple c'est le texte « Hello World »).

7.2. Test du service web

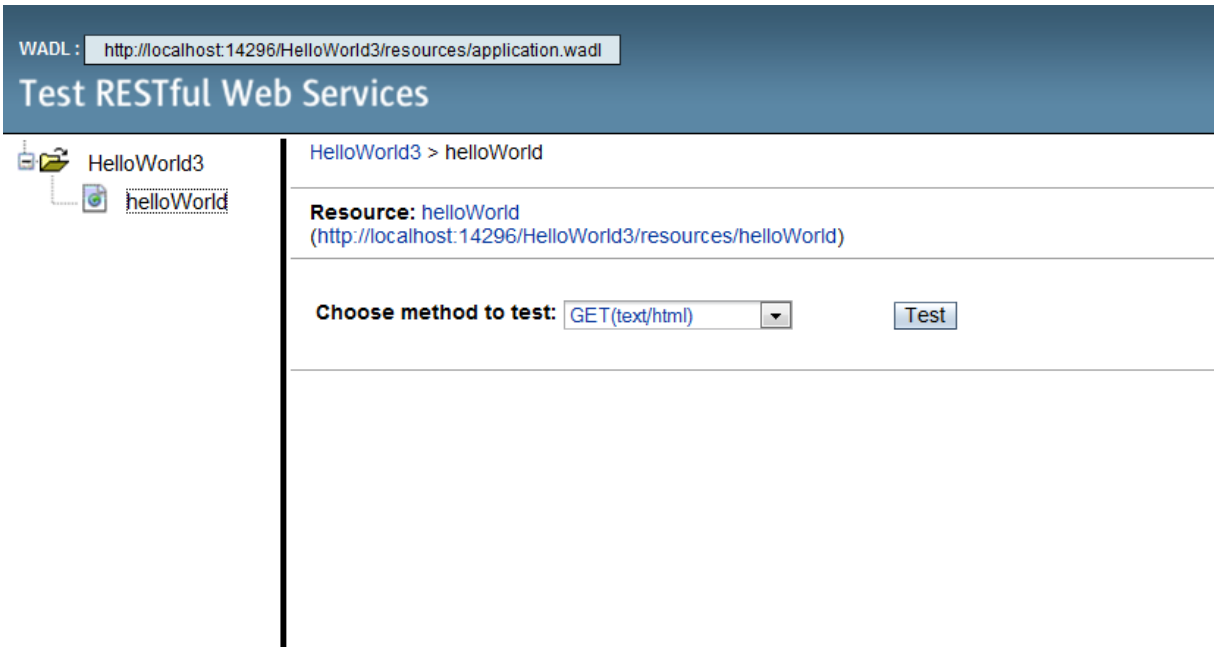
- Clic-droit sur le projet et choisir *Test Restful Web Services*.



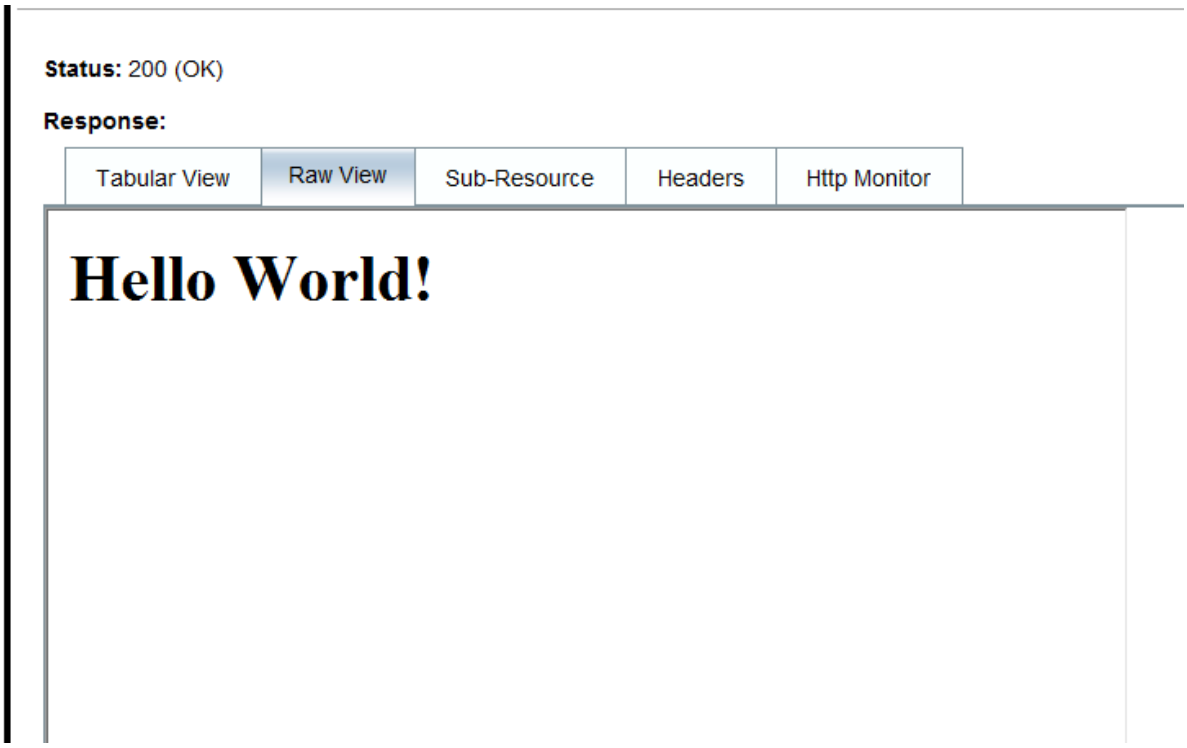
- L'interface suivante s'affiche dans un navigateur



- Choisir la ressource *helloworld* dans le panneau gauche
- Choisir la méthode à tester (dans cet exemple, c'est la méthode GET)



- Cliquer sur le bouton *Test* pour afficher le résultat

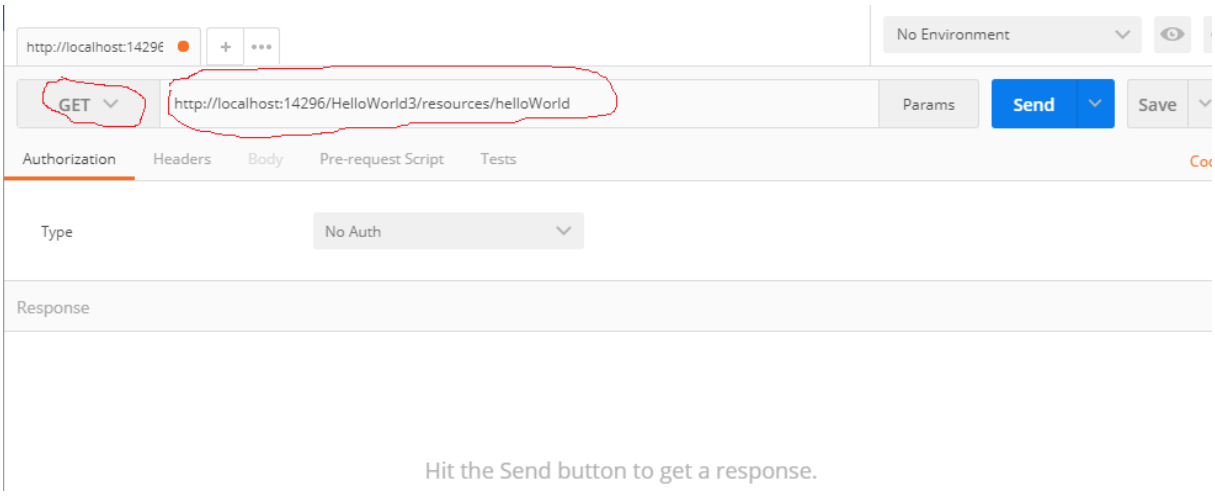


- La valeur du Relative URL dans cet exemple celui en haut du navigateur

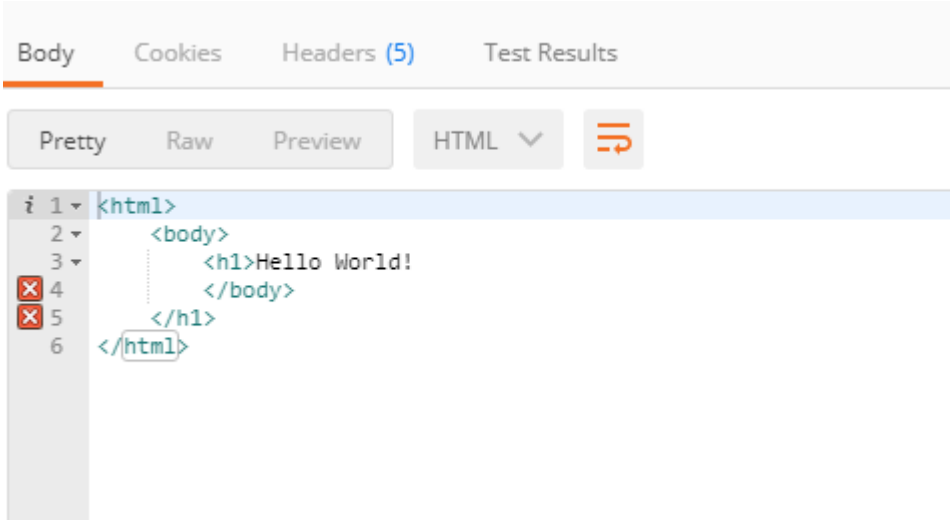
- Cliquer sur ce lien pour voir l'affichage final sur le navigateur

7.3. Test du service web avec Postman

- Installer Postman
- Dans l'interface pricipale, spécifier l'URL de la ressource (de l'exemple précédent) et la méthode http utilisée (GET)



- Cliquer maintenant sur Send
- Parce que le code initialement contient une erreur (fausse imbrication de deux balises <body> et <h1>, le résultat aura comme suit (avec indication des erreurs).



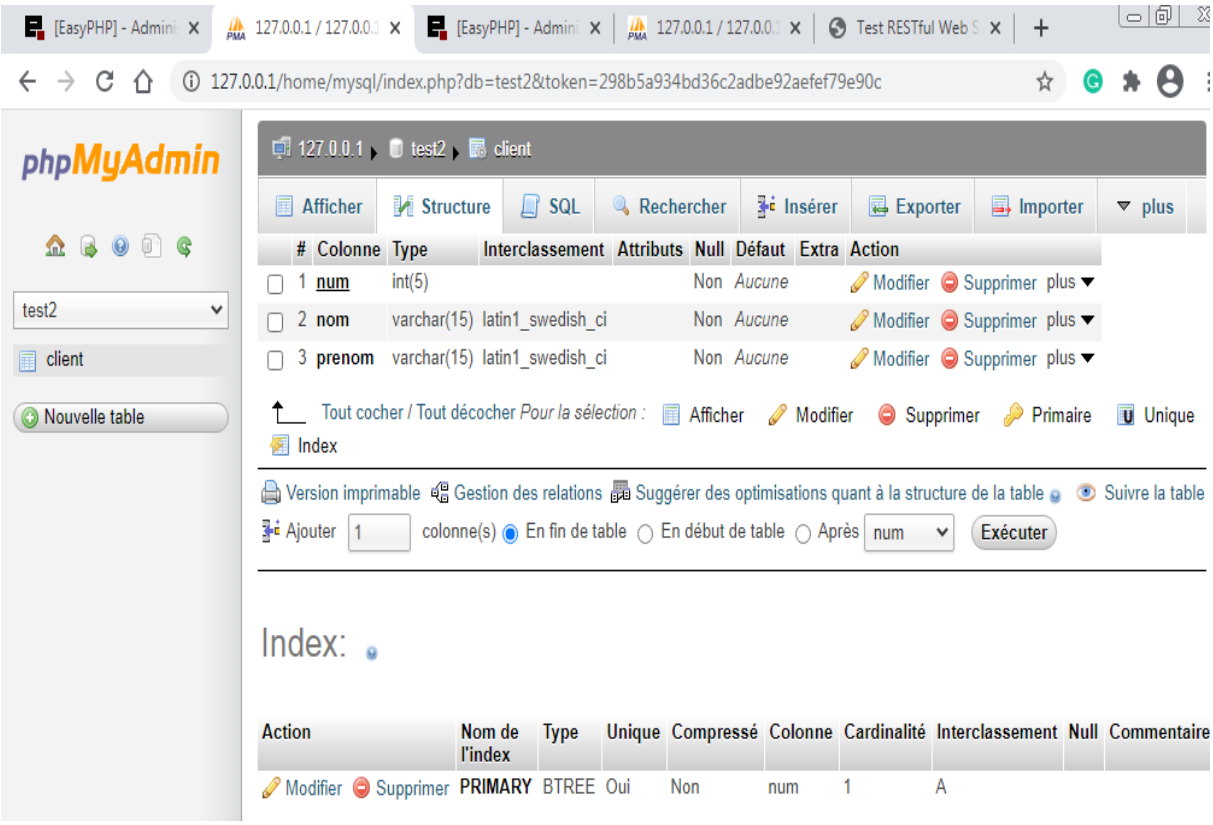
8. Service Web RRESTful à partir d'une base de données

8.1. Création de la base de données

Le principe de ce genre de services est de générer un web service RESTful à partir d'une base de données existante. Pour cette raison, avant de commencer le codage, il faut d'abord créer une base de données ou bien utiliser une déjà créée.

Dans cet exemple, je vais réutiliser une base de données qui existe déjà, où j'ai utilisé le Kit EasyPHP qui contient le MySQL. Cette base est nommée « test2 » et elle contient une seule table nommée « client ».

La table client contient 3 attributs : « num », « nom » et « prenom ».



Remarque :

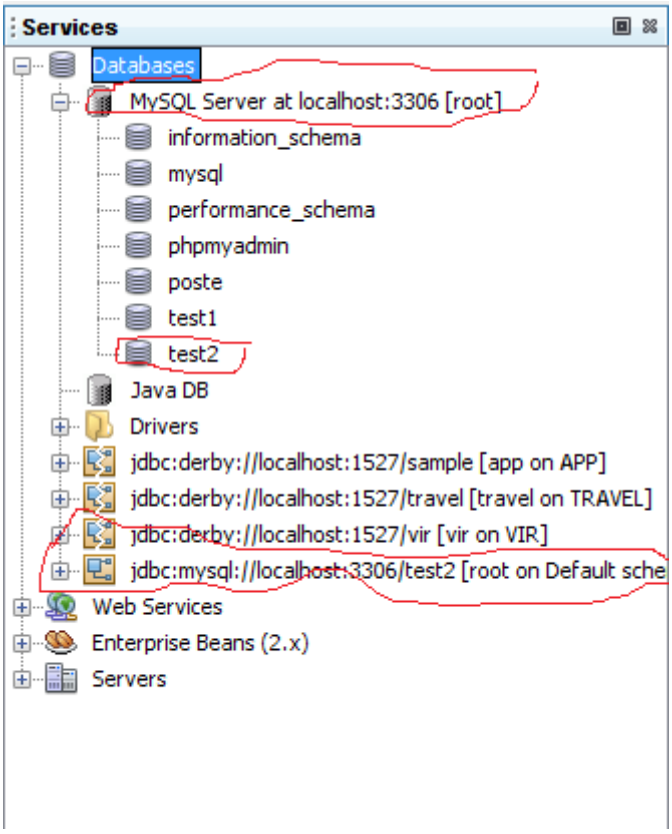
Pour que vous puissiez ajouter la table « client » plus tard après la configuration de la base dans NetBeans, il faut vérifier que cette table contient un attribut déclaré comme clé primaire (dans cet exemple num est spécifié comme clé primaire).

8.2. Configuration de la base de données dans NetBeans

La deuxième étape maintenant est de configurer la base au niveau de NetBeans :

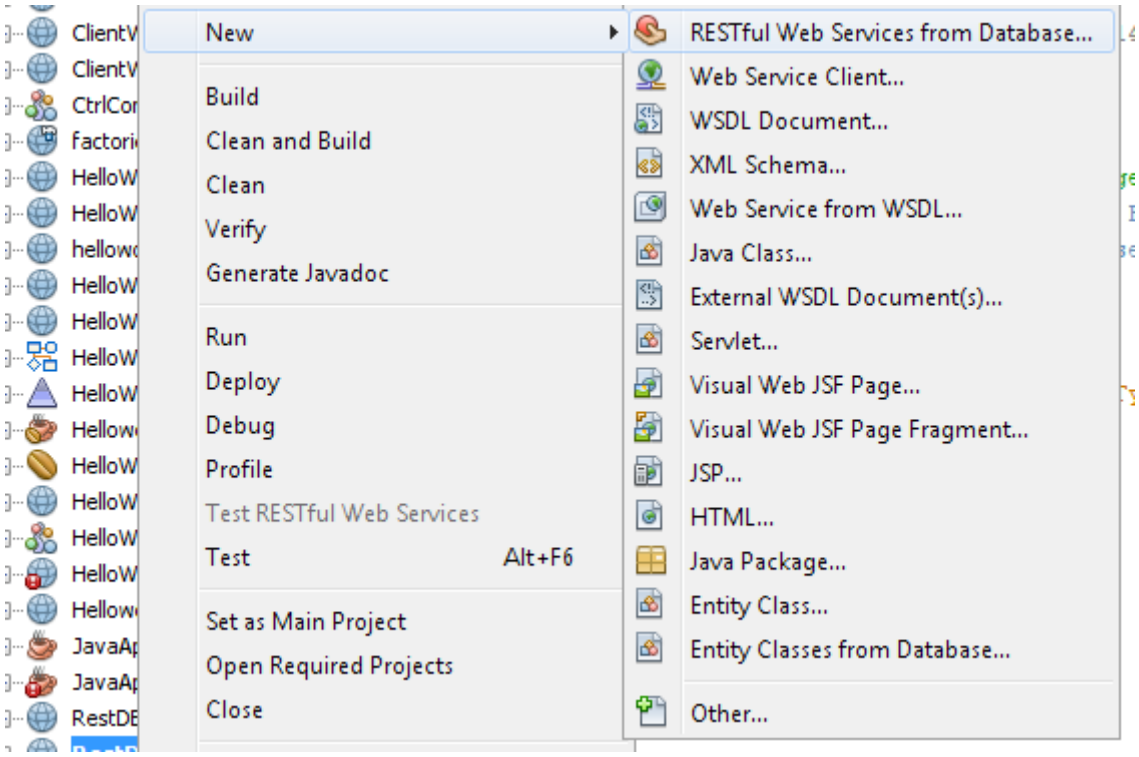
- Sous NetBeans, aller sous l'onglet *Services*
- Clic-droit sur *Databases*
- Choisir *Register MySQL Server*
- Garder les paramètres et valider
- Un nouveau serveur MySQL est apparu
- Clic-droit et choisir *Connect.*
- La liste des bases de données disponibles dans le serveur apparaît, dont la base test2.

- Clic-droit sur test2 et choisir *Connect*.
- Une connexion à la base de données sera créée
- Le résultat aura comme suit :

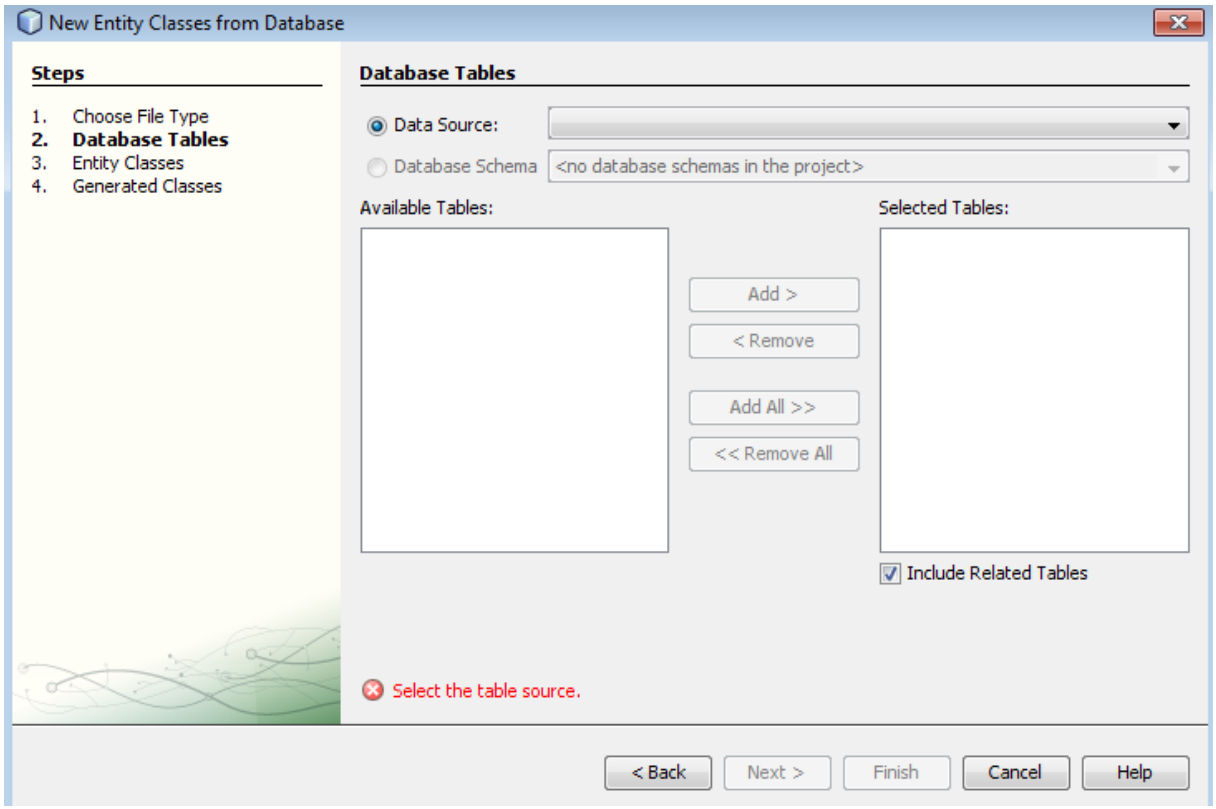


8.3. Génération du service web

- Créer un nouveau projet de type Application Web, qu'on nommera *RestDB*
- Clic-droit sur ce projet, et faire *New -> Restful Web Services from Database* (ou *Other->Web services-> Restful Web Services from Database*)

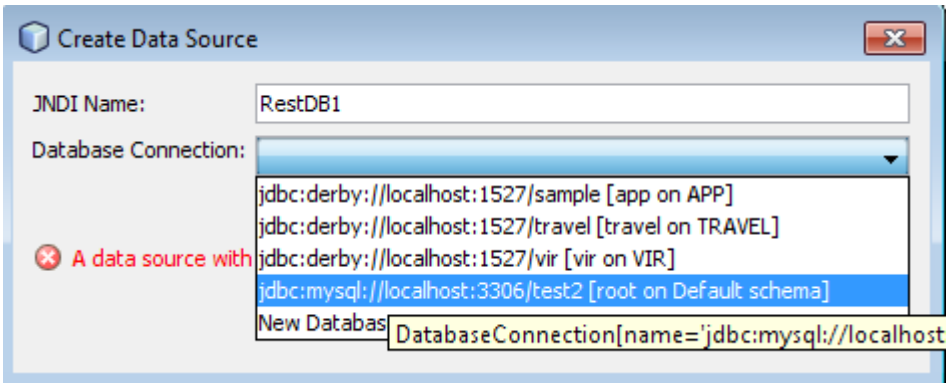


- Créer une nouvelle source de données relative à la base *test2*
- Dans le champ Data source, choisir New Data Source

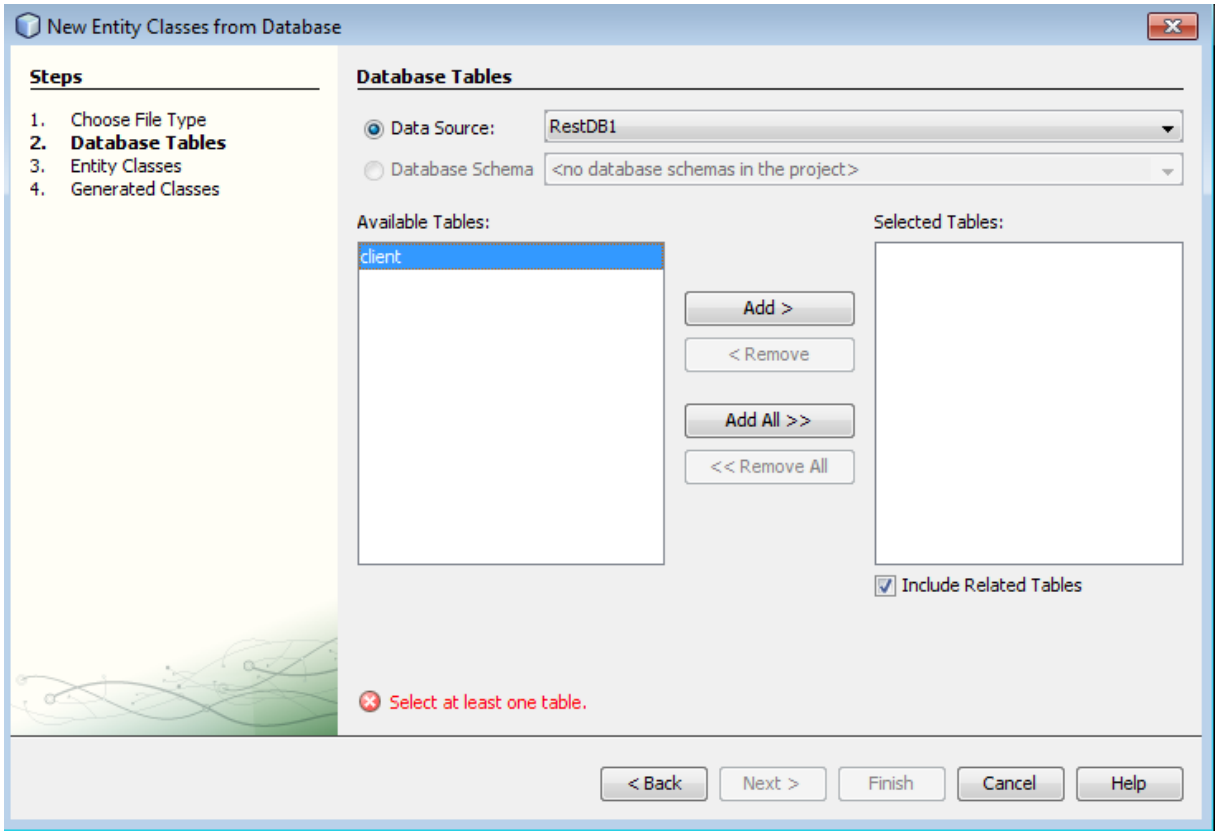


- Donner un nom JNDI (RestDB1 par exemple)

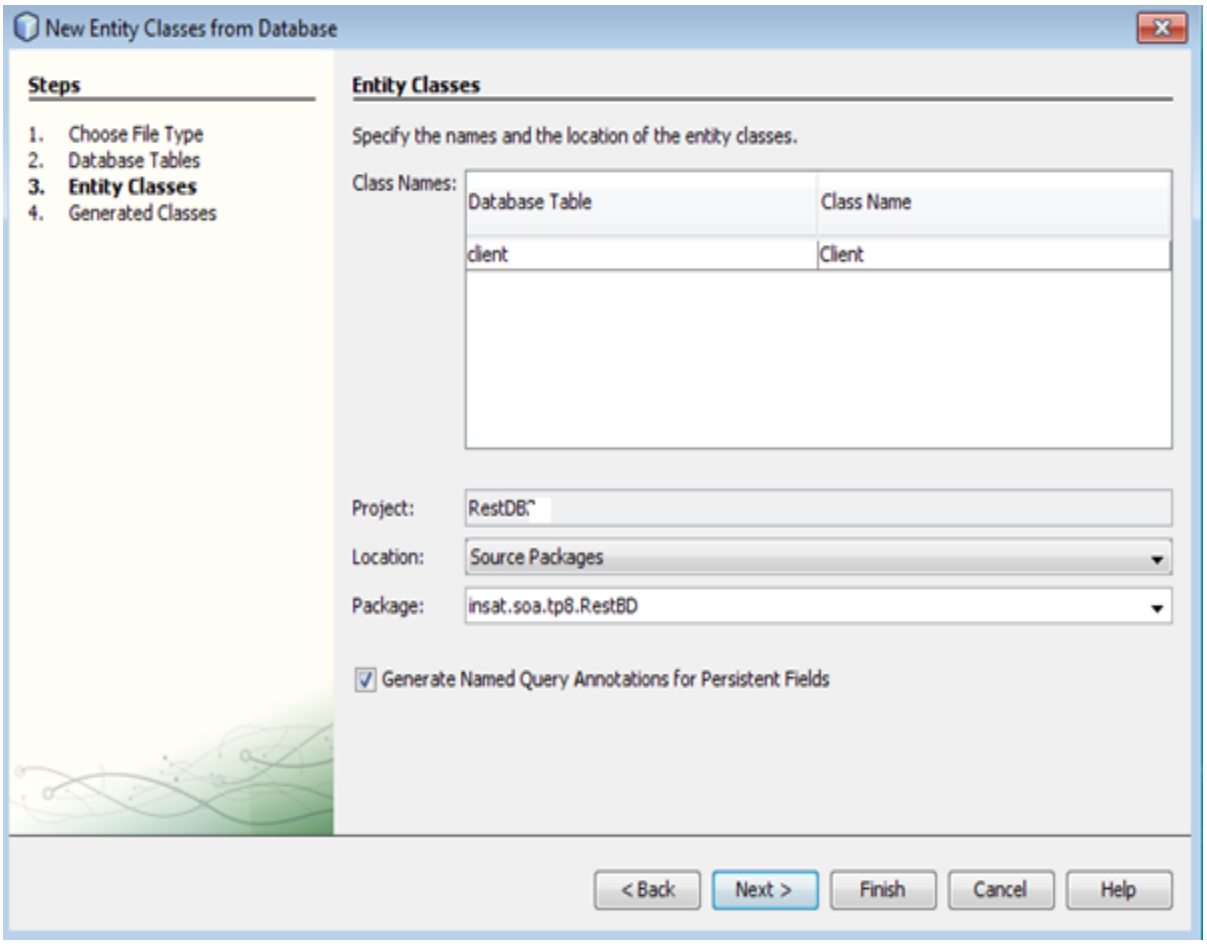
- Choisir la connexion correspondante à la base test2



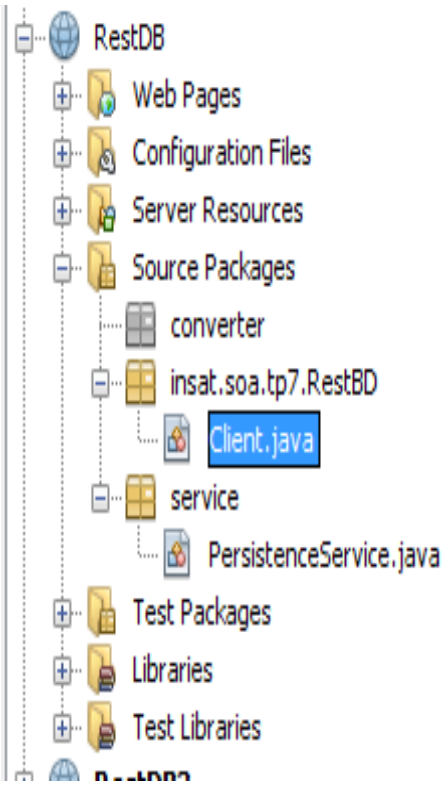
- La liste des tables dans la base apparaît dans le champ *Available*
- Ajouter la table client à la liste des *Selected Tables* (si la table client ne contient pas une clé primaire, un message d'erreur sera affiché).



- Choisir comme nom de package *insat.soa.tp8.RestBD*
- Générer une unité persistante



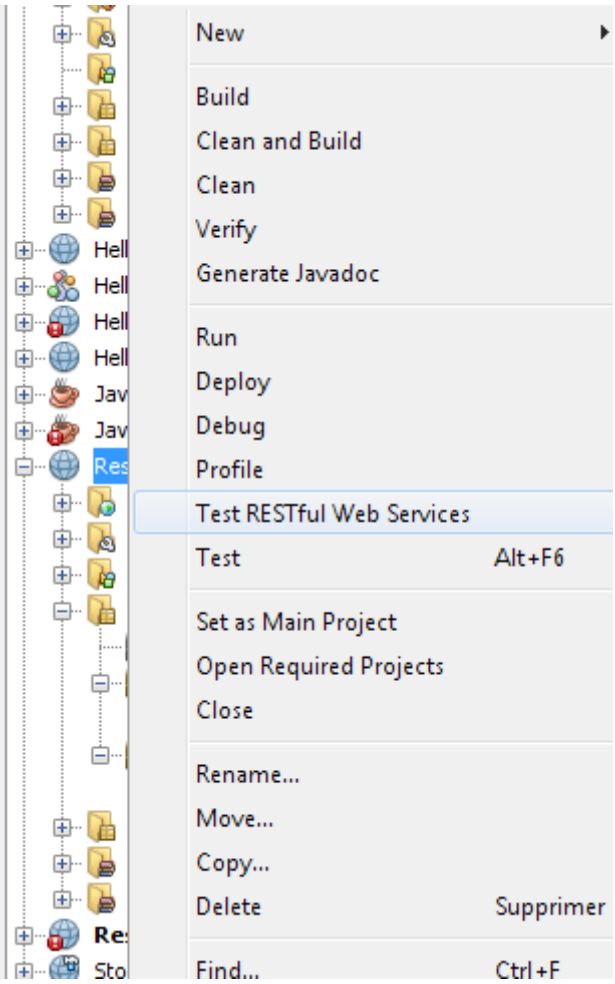
- Valider et le projet aura comme suit :

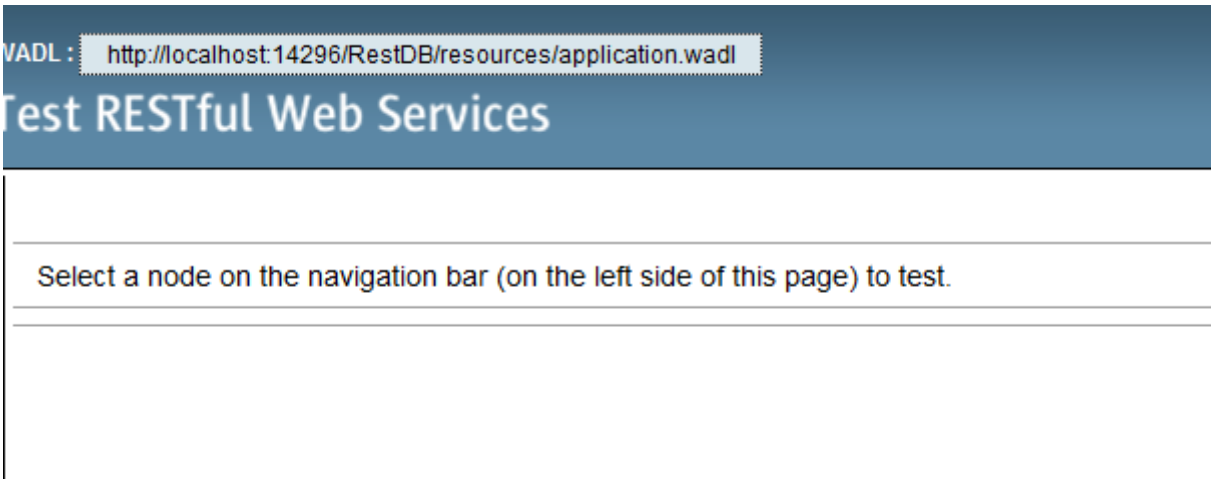


- Afficher le code de « client.java »
- Essayer de comprendre les différentes fonctionnalités proposées (Vous pouvez ajouter d'autres en exploitant d'autres méthodes HTTP).

```
private String nom;
@Basic(optional = false)
@Column(name = "prenom")
private String prenom;

public Client() {
}

public Client(Integer num) {
    this.num = num;
}

public Client(Integer num, String nom, String prenom) {
    this.num = num;
    this.nom = nom;
    this.prenom = prenom;
}

public Integer getNum() {
    return num;
}

public void setNum(Integer num) {
    this.num = num;
```

- Clic-droit sur le projet et lancer un Test





Remarque :

Pour éviter le conflit de l'exclusion mutuelle sur le Port 80 entre le serveur web Apache de EasyPHP (dans cet exemple) et le serveur Glassfish de NetBeans, essayer de changer le Port dans l'un des deux serveurs avant de commencer ce TP.

9. Enoncé TP Microservices avec REST

Développer un système basé sur l'architecture microservices qui compose trois services web RESTful:

- Un service web « searchISBN » qui a comme entrée le titre d'un ouvrage et le nom de son auteur (en cas de plusieurs auteurs le nom sera: « premeir-auteur et al »), et en sortie il donne l'ISBN de l'ouvrage correspondant.
- Un deuxième service web «GiveBookPrice » qui a comme entrée l'ISBN d'un ouvrage et en sortie son prix en dollar.
- Le troisième service web est un convertisseur « Dollar-Dinar Algérien » qui comme entrée un nombre réel qui représente le prix en dollar (USD) et en sortie son équivalence en Dinar Algérien (DZD). Pour ce service, l'utilisateur peut changer le tau d'échange.

Remarque :

- Chaque service web utilise sa propre base de données
- Essayer de créer d'abord les trois bases de données et ensuite générer les services Web RESTful correspondants (en utilisant le TP précédent).

Références

[01] Matjaz B. Juric, Poornachandra Sarang, and Benny Mathew. “Business Process Execution Language for Web Services”, Second Edition Packet Publishing, 2006.

[02] Michael Stevens, “Service-Oriented Architecture Introduction”, http://www.developer.com/services/article.php/1010451

[03] Michael Rosen, Boris Lublinsky, Kevin T. Smith, Marc J. Balcer, “Applied SOA Service-Oriented Architecture and Design Strategies”, Wiley Publishing, 2008.

[04] fr.wikipedia.org/wiki/Architecture_orientée_services

[05] Xavier Fournier-Morel, Pascal Grojean, Guillaume Plouin, Cyril Rognon, “SOA le guide de l'architecte”, Dunod, 2006

[06] Stephen Potts and Mike Kopak. “Teach yourself we services in 24 hours”, Sams Publishing, 2003

[07] http://www.w3schools.com/

[08] Aly Wane Diene. « Introduction aux Services Web », 15 septembre 2003.

[09] Yassmina Rahmoune « Une approche Agent integrant les services web pour les gestion de l’intéropérabilité des processus métiers ».Mémoire présente en vue de l’obtention du titre magistère en informatique.2008.

[10] Jean-Marie Chauvet. “Service web avec SOAP, WSDL, UDDI, ebXML“, Eyrolles, 2002

[11] SOAP Specifications https://www.w3.org/TR/soap/

[12] WSDL Specifications https://www.w3.org/TR/2007/REC-wsdl20-20070626/

[13] Jean-François Pillo. “ Introduction au WorkFlow”, 2005.http://www.net-Training.fr

[14] Bernard Gutierrez, Patrick Esquirol, Jacques Erschler «Affectation des activités aux acteurs dans les processus socio-techniques » 4ème Conférence Francophone de Modélisation et SIMulation “ Organisation et conduite d’activités dans l’industrie et les services ” MOSIM’03 - du 23 au 25 avril- Toulouse (France).

[15] Mohamed Gharzouli, «Composition des web services sémantiques dans les systèmes Peer-to-Peer», thèse de doctorat en sciences, spécialité informatique, université Mentouri – Constantine, 2011.

[16] Mohamed Gharzouli «Proposition d’un processus de développement des portails d’entreprises basé sur les services web», thèse de magister, spécialité Informatique, Université Larbi Tebessi, Tebessa, 2004.

[17] Mickaël BARON « SOA-Microservices : Introduction générale », https://mbaron.developpez.com/cours/microservices/introduction-generalites/ , 2016.

[18] Soufiane Amar «Appréhendez l'architecture Microservices », https://openclassrooms.com/fr/courses/4668056-construisez-des-microservices/5122300-apprehendez-larchitecture-microservices , 2020.

[19] Rade Despodovski «Microservices vs. SOA – Is There Any Difference at All? », https://dzone.com/articles/microservices-vs-soa-is-there-any-difference-at-al , 2017.

[20] Ricardo DE LA ROSA-ROSERO « Découverte et Sélection de Services Web pour une application Mélusine», mémoire de Master Mathématiques Informatique, Spécialité Systèmes d'Information, Université Joseph Fourier, 2004.

[21] https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203473-sla-service-level-agreement-definition-traduction/ , 2019.

[22] Kuyoro Shade O. et al «Quality of Service (Qos) Issues in Web Services», International Journal of Computer Science and Network Security, VOL.12 No.1, January 2012.

[23] Chrysostomos Zeginis, Dimitris Plexousakis « Monitoring the QoS of Web Services using SLAs », Institute of Computer Science, FORTH-ICS, Crete, Greece, 2010.

[24] Jean-Paul Figer, «REST un style d'architecture universel», Http://www.figer.com/Publications/REST.htm , 2005.

[25] Roy Thomas Fielding, Thèse de doctorat: « *Architectural Styles and the Design of Network-based Software Architectures, Representational State Transfer REST* », Université de California, Irvine, 2000.

[26] Leonard Richardson et SAM Ruby, « RESTful Web Services », O'Reilly Media, 2007.

[27] Frédéric Laurent «traduction du chapitre 5 de la thèse de doctorat de Roy Thomas Fielding», http://opikanoba.org/tr/fielding/rest/ , 2006.

[28] P. Brittenham «Web Services Development Concepts (WSDC 1.0) » 2001. http://www-306.ibm.com/software/solutions/ webservices/pdf/WSDC.pdf

[29] H. Kreger «Web Services Conceptual Architecture (WSCA 1.0) » IBM Software Group 2001

[30] Lilia Sfaxi, «OpenESB et BPEL», https://fr.slideshare.net/LiliaSfaxi/tp2-soa-avec-bpel-et-open-esb.

[31] Nicolas Hachet, «L'architecture REST expliquée en 5 règles», https://blog.nicolashachet.com/developpement-php/larchitecture-rest-expliquee-en-5-regles/ .

[32] Tarak MELLITI « Interopérabilité des services Web complexes. Application aux systèmes multi-agents », thèse de doctorat, Université Paris IX Dauphine 2004.

[33] Telitel Abdelmadjid, Tir chabane « Qualité de Service des Services Web», https://slideplayer.fr/slide/174419/

Bibliographie

- Mike Papazoglou «Web Services and SOA: Principles and Technology», Elsevier, 2nd Edition.
- Munindar Singh and Michael Huhns «Service-Oriented Computing: Semantics, Processes, Agents», John Whiley and Sons.