Intégration de Services Web dans Xweb

Dans cette partie nous étudions le concept de services web et leur intégration dans Xweb (système de gestion de site web dynamique basé sur XML).

I. Les services web

1. Généralités

Le concept de services web propose simplement de faire communiquer des composants logiciels distants. Certains modèles (Corba, EJB) devaient, en théorie, apporter une solution fiable à cette problématique[31].

Mais deux difficultés se posent dans la pratique. D'une part, il est difficile de faire communiquer un objet d'un type avec un objet d'un autre type ; d'autre part, les coupe-feu des entreprises bloquent les protocoles de communication utilisés par ces objets, compliquant terriblement leur déploiement. Les services web permettent de résoudre ces problèmes.

Leur vocation est de permettre à des applications de dialoguer entre elles à distance via Internet, sans barrières techniques, c'est-à-dire sans même avoir besoin de savoir à priori comment elles ont été conçues et sur quelle plate-forme elles opèrent. Pour ce faire, les services web s'appuient sur des protocoles standardisant les modes d'invocation mutuels de composants applicatifs. Les standards sous-jacents doivent permettre notamment :

- La définition des protocoles de communication et des structures de messages échangés,
- La description et la publication des services pertinents dans un annuaire accessible à toute application susceptible de les invoquer,
- La transformation des formats de données s'ils sont différents entre l'émetteur et le récepteur du service,
- La description et la codification des objets métiers comme le produit, le fournisseur, le bon de commande, etc.

Le principe simple du service web permet de créer un modèle de type fournisseur/consommateur, où fournisseur et consommateur sont à même de se mettre en relation de manière souple, standardisée et dynamique.

2. Etude du fonctionnement des services web

2.1Fonctionnement général [31]

Un service web est un composant logiciel qui interagit avec d'autres composants logiciels autonomes au moyen de protocoles universels.

Les services web permettent l'appel d'une méthode d'un objet distant en utilisant un protocole web pour le transport (HTTP en général) et XML pour formater les échanges.

Les services web fonctionnent sur le principe du client/serveur :

- Un client appelle le service web
- Le serveur traite la demande et renvoie le résultat au client
- Le client utilise le résultat

L'appel de méthodes distantes n'est pas une nouveauté mais la grande force des services web est d'utiliser des standards ouverts et reconnus : HTTP et XML. L'utilisation de ces standards permet d'écrire des services web dans plusieurs langages et de les utiliser sur des systèmes d'exploitation

différents.

Les services web utilisent des échanges de messages au format XML.

Il existe deux types de services web :

- synchrone : appel de méthodes (SOAP)
- asynchrone : échange de messages (SOAP, ebXML)

Les services web ne sont pas encore complètement mature à cause de la jeunesse des technologies utilisées pour les mettre en oeuvre. Il reste encore de nombreux domaines à enrichir (sécurité, gestion des transactions, workflow, ...). Des technologies pour répondre à ces besoins sont en cours de développement.

Le schéma suivant montre le fonctionnement global des services web :

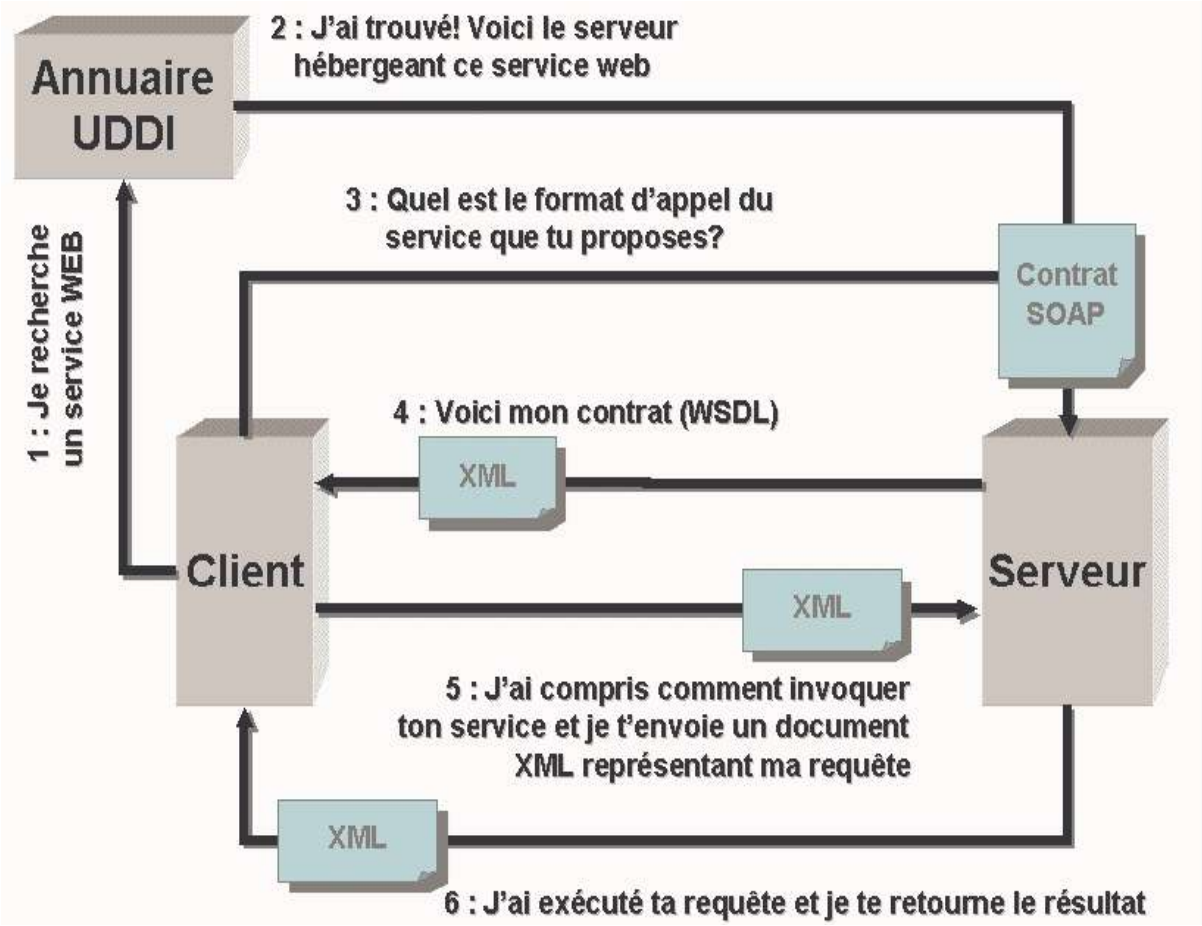

Figure7 : Architecture fonctionnelle des services web

Il est possible de détailler ces différentes étapes :

• Un service est créé en fonction d'un besoin lié à l'activité d'une entreprise à l'aide des outils et des langages appropriés

• Une fois créé, le service est publié dans un annuaire d'activités (UDDI) qui va contenir des informations complètes sur le service et son créateur (références de l'entreprise + références du service (adresse du fichier WSDL)).

• Le client fait une recherche dans un annuaire d'activités. Il récupère les références correspondant au service qu'il a choisi. Cette référence lui servira à utiliser le service

• Le service est invoqué au moyen de technologies standardisées, la transmission des données s'accomplit avec SOAP en combinant XML et HTTP.
Le client récupère le résultat voulu et le traite. L'application du client peut être développée dans n'importe quel langage, il faut juste qu'il utilise SOAP pour l'invocation du service

2.2 Technologies utilisées

Les services web utilisent trois technologies :

- SOAP (Simple Object Access Protocol) pour le service d'invocation : il permet l'échange de messages dans un format particulier
- WSDL (Web Services Description Language) pour le service de description : il permet de décrire les services web
- UDDI (Universal Description Discovery and Integration) pour le service de publication : il permet de référencer les services web

2.2.1SOAP

SOAP signifie *Simple Object Access Protocol*. Ce protocole est celui utilisé pour la communication entre le client et le serveur qui héberge le service. Ce protocole est un peu spécial car il ne définit aucun protocole de transport. Les écrivains traitant de ce sujet aiment à dire que SOAP peut-être véhiculé par HTTP, SMTP ou même par pigeon voyageur [24].
SOAP est une norme de communication qui standardise l'échange de messages en utilisant un protocole de communication (le plus utilisé est HTTP) et XML pour formater les données [32].

SOAP se veut simple à utiliser et extensible.

SOAP peut être utilisé :

- pour l'appel de méthodes (SOAP RPC)
- pour l'échange de message (SOAP Messaging)

SOAP définit la structure principale du message, dite « enveloppe » qui contient deux parties :

- l'en-tête (Header) : facultatif
- le corps (Body) : obligatoire

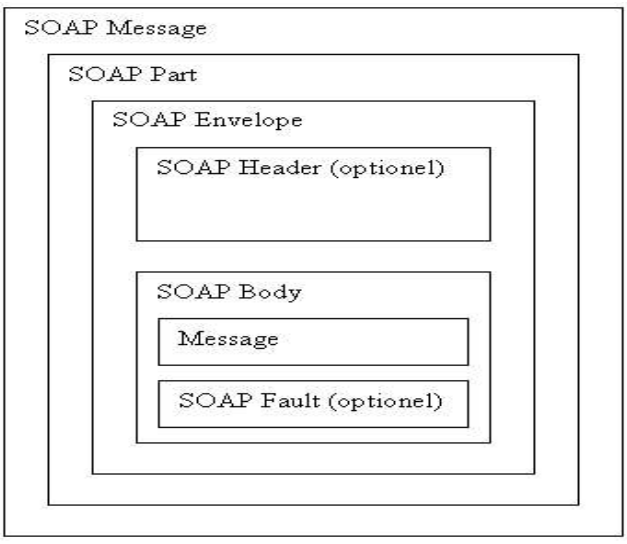

Figure 8: Structure principale d'un message défini par SOAP

Le corps est composé d'un ou plusieurs blocs. Un bloc contient des données ou un appel de méthode avec ses paramètres.

Un message SOAP peut aussi contenir des pièces jointes contenues chacune dans une partie optionnelle nommée AttachmentPart. Ces parties sont au même niveau que la partie SOAP Part.

SOAP définit aussi l'encodage pour les différents types de données qui est basé sur la technologie schéma XML du W3C. Les données peuvent être de type simple (chaine, entier, flottant, ...) ou de type composé.

Les types simples peuvent être

- un type de base : string, int, float, ...
- une énumeration
- un tableau d'octets (array of byte)

Les types composés

- une structure (Struct)
- un tableau (Array)

La partie SOAP Fault permet d'indiquer qu'une erreur est survenue lors des traitements du service web. Cette partie peut être composée de 4 éléments :

- faultcode : indique le type de l'erreur (VersionMismatch en cas d'incompatibilité avec la version de SOAP utilisée, MustUnderstand en cas de problème dans le header du message, Client en cas de manque d'informations de la part du client, Server en cas de problème d'éxécution des traitements par le serveur)
- faultstring : message décrivant l'erreur
- faultactor : URI de l'élement ayant déclenché l'erreur
- faultdetail : Détail de l'erreur non lié au Body du message

Pour l'appel de méthodes, plusieurs informations sont nécessaires :

- l'URI de l'objet à utiliser
- le nom de la méthode
- éventuellement le ou les paramètres

La sécurité est un sujet souvent abordé lors de conversation au sujet de SOAP. SOAP ne gère aucun mécanisme de sécurité. Cependant, rappelons que celui-ci n'est pas lié à un protocole de transport défini. Ainsi, il est possible d'augmenter le niveau de sécurité des messages en utilisant HTTPS en lieu et place d'HTTP. Le service se reposant sur le serveur, Web par exemple, pour les tâches de transport, ceci n'est pas forcément très compliqué à mettre en place.

Les trois exemples suivants représentent les messages SOAP envoyés, respectivement reçus par le client [33].

L'envoi :

Demande de calcul à un serveur

```
POST /path/foo.pl HTTP/1.1
Content-Type: text/xml
SOAPAction: interfaceURI#Add
Content-Length: nnnn
<soap:Envelope xmlns:soap='uri for soap'>
<soap:Body>
<Add xmlns='interfaceURI'>
<arg1>24</arg1>
<arg2>53.2</arg2>
</Add>
</soap:Body>
</soap:Envelope>
```

Et le retour s'il n y a pas d'erreur:

```
200 OK
Content-Type: text/xml
Content-Length: nnnn
<soap:Envelope
xmlns:soap='uri for soap'>
<soap:Body>
<AddResponse xmlns='interfaceURI' >
<sum>77.2</sum>
</AddResponse>
</soap:Body>
</soap:Envelope>
```

En vert se trouve la valeur souhaitée. On voit ici que, par rapport à d'autres solutions comme CORBA ou RMI, il y a une quantité non négligeable de données qui transitent pour une quantité utile relativement faible. Notons également l'espace de nom interfaceURI qui permet de s'assurer que le message soit bien celui attendu .

Et le retour s'il y a erreur:

```
<SOAP:Envelope
xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1>
<SOAP:Body>
<SOAP:Fault>
<faultcode>200</faultcode>
<faultstring>
SOAP Must Understand Error
</faultstring>
</SOAP:Fault>
<SOAP:Body>
</SOAP:Envelope>
```

2.2.2WSDL

WSDL signifie *Web Services Description Language*. Il s'agit d'un langage, lui aussi basé sur XML, permettant de définir un service Web. Ce fichier contient le chemin d'accès au serveur, mais aussi les prototypes des méthodes qui servent de points d'entrées et la description des types complexes [24].

WSDL peut etre vu comme un complément de SOAP car il facilite l'interopérabilité des Web Services. Tout comme IDL (Interface Definition Language) qui joue le rôle de descripteur de

services avec CORBA, WSDL est une syntaxe XML pour décrire les Web Services. Grace à WSDL, les applications seront capables d'auto-configurer les échanges entre Web Services, tout en masquant la plupart des détails techniques de bas niveau [25].

Concrètement, les services sont définis dans un document WSDL au moyen de six éléments principaux :

- types : fournissent des définitions de types de données afin de décrire les messages échangés.
- message : sert à représenter une définition abstraite des données transmises ; un message est constitué de parties logiques, chacune étant associée à une définition de type.
- type de port (*portType*) : ensemble d'opérations abstraites. Chaque opération se réfère à un message entrant et à des messages sortants.

Une opération est composée d'un message de type input et d'un message de type output. Les deux types de message sont optionnels et leur ordre désigne le type de l'opération:

- One-way: input sans output
- Request-response: input suivi de output
- Solicit-response: output suivi de input
- Notification: output sans input

- rattachement ou liason WSDL (*binding*) : ce qui spécifie les aspects concrets de protocole de communication et le format des données pour les opérations et messages définis par un type de port particulier.
- port : ce qui spécifie une adresse pour un rattachement, déterminant ainsi un seul noeud branché, ou point terminal (*endpoint*).
- service : ce qui sert à regrouper un ensemble de ports.

Dans [24], le client possède généralement des outils permettant de créer des proxy de manière tout aussi automatique. Une classe proxy est une sorte d'interface (pas au sens Java) permettant d'accéder à un service Web. Elle encapsule le code nécessaire à la communication distante, ce qui permet, lors de l'implémentation du client, d'accéder au service Web comme s'il s'agissait d'un objet local.

Notons que la classe proxy permet donc d'implémenter tout type de client imaginable application en ligne de commande, en mode fenêtré, applet, HTML dynamique, etc. Les trois premiers sont considérés comme des clients lourds car ils s'exécutent sur la machine de l'utilisateur, alors que le dernier est nommé un client léger car il s'exécute sur un serveur Web et ne nécessite donc aucune installation sur la machine du client.

Lors de la réalisation de projets plus complexes que de simples exemples, il apparaît assez vite que la génération automatique du schéma WSDL n'est pas suffisante. Il est souvent nécessaire de modifier le fichier WSDL ou la classe *proxy* générée. Pour le premier, une partie du travail peut souvent être réalisée dans le code, par exemple à l'aide de scripts (chez BEA) ou d'attributs (.NET).

Voici un exemple de code WSDL. Celui-ci définit un service Web possédant une méthode HelloWorld, sans paramètre et retournant une chaîne de caractères. Celui-ci a été simplifié de manière à ne permettre que les communications via SOAP.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:s0="http://eivd.ch/helloworld"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encodi
ng/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching
/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 targetNamespace="http://eivd.ch/helloworld"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
 <types>
```

```
   <s:schema elementFormDefault="qualified"
   targetNamespace="http://eivd.ch/helloworld">
    <s:element name="HelloWorld">
     <s:complexType />
    </s:element>
    <s:element name="HelloWorldResponse">
     <s:complexType>
      <s:sequence>
       <s:element minOccurs="0" maxOccurs="1"
        name="HelloWorldResult" type="s:string" />
      </s:sequence>
     </s:complexType>
    </s:element>
    <s:element name="string" nillable="true" type="s:string"
/>
   </s:schema>
  </types>
  <message name="HelloWorldSoapIn">
   <part name="parameters" element="s0:HelloWorld" />
  </message>
  <message name="HelloWorldSoapOut">
   <part name="parameters"
element="s0:HelloWorldResponse" />
  </message>
  <portType name="HelloSoap">
   <operation name="HelloWorld">
    <documentation>Renvoie un
HelloWorld</documentation>
    <input message="s0:HelloWorldSoapIn" />
    <output message="s0:HelloWorldSoapOut" />
   </operation>
  </portType>
  <binding name="HelloSoap" type="s0:HelloSoap">
   <soap:binding
transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
   <operation name="HelloWorld">
    <soap:operation soapAction=
     "http://eivd.ch/helloworld/HelloWorld"
style="document" />
    <input>
     <soap:body use="literal" />
    </input>
    <output>
     <soap:body use="literal" />
    </output>
   </operation>
  </binding>
  <service name="Hello">
   <documentation>Premier service
Web</documentation>
   <port name="HelloSoap" binding="s0:HelloSoap">
    <soap:address location=
     "http://ap0501/HelloWorld/HelloWorld.asmx" />
   </port>
   <port name="HelloHttpGet"
binding="s0:HelloHttpGet">
    <http:address location=
     "http://ap0501/HelloWorld/HelloWorld.asmx" />
   </port>
  </service>
</definitions>
```

2.2.3UDDI

UDDI (Universal Description, Discovery and Integration) est un protocole et un ensemble de services pour utiliser un annuaire afin de stocker les informations concernant les services web et de permettre à un client de les retrouver. Il existe plusieurs serveurs, tous compatibles avec le protocole UDDI qui permettent d'automatiser la recherche de services. Nous noterons par exemple uddi.ibm.com et uddi.microsoft.com. Ces serveurs sont atteignables via un navigateur Web standard.

L'interface Web des serveurs UDDI permet généralement d'inscrire un nouveau service Web ou de retrouver des services existants. Il est parfois possible de les atteindre avec des services Web pour, par exemple, automatiser la recherche ou de permettre aux clients d'être plus indépendants de la localisation physique du service [24].

L'annuaire [22] inclut des informations sur les fournisseurs de services, les services hébergés et les protocoles mis en oeuvre par ces services. L'annuaire propose également des mécanismes pour ajouter des métadonnées à toute information enregistrée.

Cette approche par annuaire est intéressante lorsque les informations sont stockées dans les emplacements bien connus. Une fois que l'annuaire est localisé, il reste à l'interroger pour obtenir les informations souhaitées. L'emplacement de l'annuaire UDDI est généralement défini par l'administrateur du système.

Les fournisseurs de services web présentent diverses options pour déployer les registres UDDI. Les scénarios de déploiement se répartissent en trois grandes catégories: déploiement public, inter-entreprise et intra-entreprise. Pour les déploiements publics, un groupe d'entreprises regroupant notamment Microsoft, IBM et SAP, soutiennent UBR (UDDI Business Registry)[23]. UBR est un registre public repliqué dans de nombreuses entreprises. Il sert à la fois de ressources pour les services web présents sur Internet et de test pour les développeurs de services web. Bien que l'implémentation publique d'UDDI ait reçu le plus d'attention jusqu'à présent, les concepteurs utilisent le plus souvent l'approche inter- et intra-entreprise.
Dans ces deux scénarios de déploiement, un registre privé est déployé par une entreprise; un contrôle plus étroit sur les types d'informations enregistrées est alors possible. Ces registres privés peuvent être dédiés à une seule entreprise ou à des groupes de partenaires commerciaux.

Les annuaires UDDI contiennent des informations détaillées concernant les services web et leurs emplacements. Une entrée d'annuaire UDDI se compose de trois parties principales: le fournisseur du service, les services web offerts et les liens vers les implémentations. Chacune de ces parties donne progressivement davantage d'informations sur le service web.

L'information la plus générale décrit le fournisseur du service. Elle n'est pas destinée à un logiciel mais à un développeur ou à un installateur qui peut avoir besoin de contacter directement quelqu'un responsable du service. Cette information inclut des noms, des adresses, des contacts et d'autres détails administratifs.

La liste des services web disponibles est stockée dans une entrée de type fournisseur de service. Les services peuvent être organisés en fonction de leur utilisation prévue: ils peuvent être groupés en domaines d'application, par régions géographiques, ou tout autre schéma approprié. Une information de service stockée dans un registre UDDI inclut simplement une description d'un service et un pointeur vers les implémentations de service web qu'il contient. Il est aussi possible d'enregistrer des liens vers des services hébergés par d'autres fournisseurs. Ces liens se nomment « projection de service ».

La dernière partie d'une entrée de fournisseur de service UDDI est la liaison vers une implémentation. La liaison associe l'entrée de service web à l'URI exact identifiant l'emplacement du service. Cette liaison spécifie aussi le protocole à utiliser pour accéder au service. Enfin, elle contient des références aux protocoles exacts qui sont mis en œuvre.

Ces détails sont suffisants pour qu'un développeur puisse écrire une application qui fasse appel au service web. La définition détaillée du protocole est fournie par une entité UDDI nommée Type Model (ou tModel). Dans de nombreux cas, le tModel référence un fichier WSDL qui décrit l'interface SOAP du service web mais les entités tModel sont suffisamment flexibles qu'elles peuvent servir à décrire pratiquement n'importe quel type de ressource.

3Implémentation

3.1JAX-RPC

JAX-RPC [26] (Java API for XML based RPC) est une API permettant l'appel de méthodes distantes et la réception de leur réponse en utilisant SOAP 1.1 et HTTP 1.1. Elle facilite la communication via Internet en permettant à des paramètres au format XML d'être passés à des services distants et en permettant à des valeurs au format XML d'être renvoyées.

JAX-RPC inclut un modèle extensible de mapping de types de données (mécanisme de

correspondance entre types de données) associé à des mécanismes de sérialisation/désérialisation entre objets Java et types de données XML-Schema. D'autre part, JAX-RPC définit des fonctionnalités de mapping à double sens entre Java (classes interfaces, méthodes), et descriptions XML/WSDL d'interfaces de services web.

Le grand avantage de cette API est de masquer un grand nombre de détails de l'utilisation de SOAP et ainsi de la rendre facile à utiliser.

Le schéma ci-dessous est une vue simplifiée de ce qui se passe lors de l'appel d'un service web JAX-RPC appelé Calculator par un client.

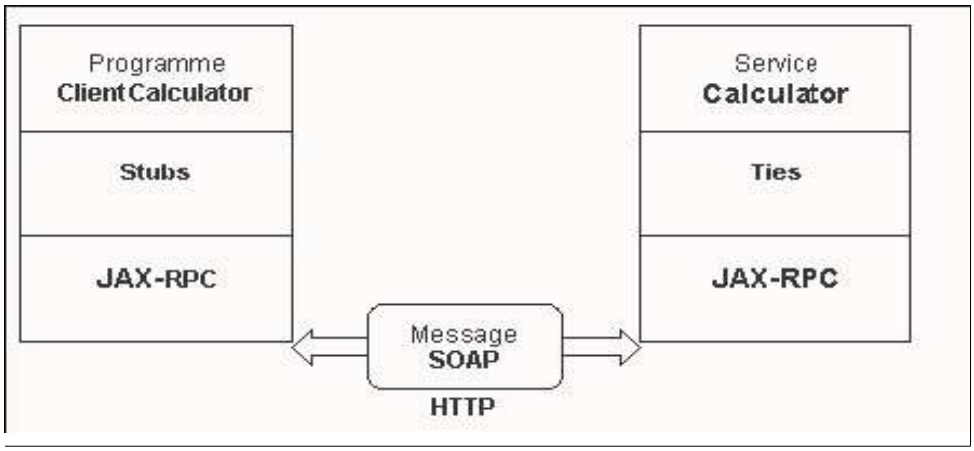

Figure 9: Schéma d'invocation d'un service web JAX-RPC

Voici une description plus détaillée de ce qui se passe :

· Pour appeler une procédure distante, le programme CalculatorClient invoque une méthode d'un stub, un objet local qui représente le service
· Le stub invoque des méthodes de JAX-RPC
· JAX-RPC convertit les appels à distance en un message SOAP et le transmet comme une requête HTTP
· Le serveur reçoit la requête, JAX-RPC extrait le message SOAP et le transforme en un appel de méthode
· JAX-RPC invoque la méthode de l'objet tie
· L'objet tie invoque la méthode présente dans l'implémentation du service Calculator
· Le serveur convertit la réponse en un message SOAP et le transmet au client comme une requête HTTP
· Chez le client, JAX-RPC extrait le message SOAP de la requête et le transforme en réponse pour le programme CalculatorClient

Lors de la programmation du service ou du client, le programmeur ne s'occupe que des couches supérieures : les implémentations.
L'implémentation du service se passe en plusieurs phases :
· Ecriture des classes JAVA représentant le service (interface + implémentation)
· Utilisation de wsdeploy pour générer toutes les classes nécessaires au service et le fichier WSDL.

· Déploiement sur le serveur

L'implémentation du client se passe en plusieurs phases :

· Ecriture du client JAVA

· Utilisation de wscompile pour générer les classes nécessaires au client à partir du fichier WSDL décrivant le service

· Exécution du client

Le client du service web peut être codé de différentes façons en fonction du type de stub utilisé :

· Stub statique : le stub est créé avant l'exécution par wscompile.

· Stub dynamique : le stub est créé pendant l'exécution. Avant de le créer, le client récupère les informations concernant le service dans le fichier WSDL.

· DII (Dynamic Invocation Interface) : le client peut appeler une procédure distante même si le nom de la procédure ou du service ne sont pas connus avant l'exécution.

Ce dernier type est le plus flexible, il peut être utilisé dans le cas d'une application recherchant les services dans un annuaire, configure et exécute les appels aux méthodes distantes.

3.2JAXM

JAXM [27] (Java API for XML Messaging) permet de créer, d'envoyer et de recevoir des documents XML. Le grand avantage de cette API est qu'elle permet de faire abstraction de la syntaxe SOAP, un peu plus laborieuse et compliquée. Les services web basés sur JAXM utilisent les échanges de documents contrairement à ceux basés sur JAX-RPC qui font appel à des méthodes. Les services web basés sur JAXM peuvent être utilisés dans les cas suivants :

· Quand les interactions entre client et service sont asynchrones. JAXM propose un support pour les échanges synchrones et asynchrones de documents XML

· Quand client et service veulent échanger des données à l'intérieur de documents XML utilisant des schémas XML bien définis au lieu d'appeler des méthodes

L'API JAXM se conforme à la spécification SOAP 1.1 ainsi qu'à la spécification SOAP with Attachments. Cette API se compose de deux packages principaux :

· javax.xml.soap : le package défini dans la spécification SOAP with Attachments API for JAVA (SAAJ) et qui constitue le package de base de l'échange de messages SOAP, contenant ainsi l'API destinée à créer et à remplir un message SOAP. Ce package contient toutes les API nécessaires pour envoyer des messages requêtes/réponses (communication synchrone).

· javax.xml.messaging : le package défini dans la spécification JAXM 1.1. Ce package contient l'API nécessaire afin d'utiliser un fournisseur de messages (messaging provider), et ainsi d'envoyer des messages one-way (communication asynchrone). Le second package est dépendant du premier, l'inverse n'étant pas vrai. Ainsi, un client envoyant des messages requêtes/réponse peut tout à fait se contenter de l'API javax.xml.soap.

Le service JAXM traite les messages JAXM envoyés par le client JAXM. Pour développer le service, il faut créer une servlet héritant de javax.xml.messaging.JAXMServlet.
Elle doit implémenter soit l'interface OneWayListener soit l'interface ReqRespListener selon que la communication entre le client et le service sera asynchrone ou synchrone.

Le client JAXM échange des messages avec le service JAXM. Il peut interagir soit directement avec le service soit à travers un provider selon le type de communication choisi.

Un client qui utilise un provider peut participer à des interactions synchrones ou asynchrones alors qu'un client n'utilisant pas de provider peut seulement participer à des interactions synchrones.

Un provider de messages est un service qui permet la transmission et le routage des messages.

L'utilisation d'un provider permet d'avoir une séparation entre le client et le service. Elle permet aussi d'avoir accès à des services supplémentaires tels que la sécurité ou la gestion de protocoles (eb-XML par exemple).

3.3.NET

Le *framework* .NET [29] est très récent. Son développement a été parallèle à celui des services Web. Il contient déjà toutes les librairies et tous les outils nécessaires pour la réalisation de services Web. Ainsi, les espaces de nom System.Web.Services ou System.Web.Protocols.Soap contiennent les objets de base pour la sérialisation ou l'implémentation des services, de la sérialisation, etc.

Tout service Web réalisé en .NET doit hériter de l'objet System.Web.Services.WebService.

Le code permet de spécifier des attributs pour les méthodes, les classes, etc. Ces attributs contiennent des informations pour les méthodes de sérialisation. Ainsi, un attribut *WebMethod* d'une méthode définit celle-ci comme méthode web, qui peut alors être appelée par un client. Si une méthode n'est pas précédée par un attribut *WebMethod*, elle ne sera pas disponible pour le client. L'attribut *WebMethod* permet de modifier l'espace de nom XML de la méthode Web, son nom, s'il diffère de celui de la méthode de l'objet ou encore le fait que la méthode utilise les options de sessions du serveur Web.

L'attribut *WebService* quant à lui n'est pas obligatoire. Il permet principalement de modifier l'espace de nom du service Web. Si celui-ci n'est pas redéfini pour les méthodes ou les types, ceux-ci récupèrent celui du service.

Un autre avantage du *framework* .NET est Visual Studio. Certes, l'acquisition de celui-ci est onéreuse, mais le temps de développement est fortement diminué. De même, des outils gratuits tel Asp.NET Web Matrix ou SharpDevelop permettent également d'accélérer le développement de services Web.

On le voit, le développement de service Web est bien intégré dans la technologie .NET. Ceci devrait amener de nombreux développeurs à se rapprocher de celle-ci.

3.4BEA Weblogic

BEA [30] a développé un ensemble d'outils et de librairies pour le développement de service Web à l'aide du langage Java. Le principal problème de cette solution est son prix. En effet, contrairement aux deux autres solutions proposées dans ce document, celle-ci est payante ; Il est cependant possible d'obtenir une version d'évaluation d'une durée de nonante jours.

L'outil BEA Weblogic workshop s'occupe de tout pour nous. Le résultat est que nous développons des services Web qui fonctionnent, mais le développeur moyen ne sait pas par quelle magie cela fonctionne.

Le Workshop de BEA permet de développer rapidement des services Web servant d'interface avec des EJB ou d'autres objets déjà existants. Leur développement en soi n'est donc pas très complexe. Cependant, il est nécessaire de garder à l'esprit que, même si le temps de développement d'un service Web est fortement diminué, il reste très important de respecter certaines règles.

Notons encore que l'outil de BEA est livré avec un serveur d'application faisant office de serveur Web. Il est possible d'utiliser ce serveur pour faire fonctionner des EJB ou *Enterprise Java Beans*.

Il n'est pas nécessaire d'utiliser ceux-ci pour réaliser des services Web ; ils permettent cependant, à condition d'être utilisés correctement, de gérer les transactions, ce qui n'est pas possible si nous utilisons uniquement les services Web. Dans ce cas, les services Web sont une sorte d'interface d'accès aux EJB.

3.5Apache Axis

Apache Axis [28] est une implémentation open-source sur la plate-forme Java des spécifications SOAP version 1.1 et SOAP Messages with Attachments (messages SOAP avec pièces jointes), et inclut certaines fonctionnalités de la nouvelle version de la norme, SOAP 1.2. Axis est développé par la fondation Apache.

Apache Axis est à la fois un environnement d'hébergement de services web, et un toolkit complet de développement pour la création de services ainsi que l'accès client à des services tiers. Ce toolkit comporte des fonctionnalités avancées de génération de code automatique basées sur l'analyse de descriptions WSDL de services web et le mapping automatique à double sens entre classes Java et descriptions WSDL de services.

Axis est un projet qui est issu du projet SOAP du projet Apache. Pour plus de renseignements, le site xml.apache.org contient toutes les informations nécessaires.

Le projet Apache fournit un serveur Web efficace (très connu dans le monde Linux) ; cependant, bon nombre d'autres projets gravitent autour de celui-ci. Le projet AXIS en est un exemple. Pour le moment encore en version Beta 3, il s'agit en fait de la version 3 de Apache SOAP. Cette API est gratuite mais nécessite un serveur d'application pour le support des EJB.

La librairie permet cependant les manipulations habituelles, telles que le développement de service ou de client. Le principal problème est que la version actuelle est une Beta. Ceci implique qu'il faut, comme bien souvent dans ce cas, se contenter d'une version pas toujours très stable, mais surtout plus complexe à mettre en oeuvre. Enfin, notons que la librairie d'Apache ne fournit pas d'outils de développement aussi efficace que le Weblogic Workshop de BEA.

Axis est un package qui fournit :

· Un environnement pouvant soit fonctionner comme un serveur SOAP indépendant soit comme un plug-in de moteurs de servlet (en particulier TOMCAT)
· Une API pour développer des services web SOAP RPC ou à base de messages SOAP
· Le support de différentes couches de transport : HTTP, FTP...
· La sérialisation/désérialisation automatique d'objets Java dans des messages SOAP
· Des outils pour créer automatiquement les WSDL correspondant à des classes Java ou inversement pour créer les classes Java sur la base d'un WSDL (classe proxy en quelque sorte, qui fait le lien entre l'application Java cliente et le service distant)
· Des outils pour déployer, tester et monitorer des web-services

II. Intégration de services web dans Xweb

Dans cette section nous étudions l'intégration de services web dans xweb. Cette intégration doit permettre la conception de sites web permettant l'accés à des services web.

1.Unité médiatique de type service web

Nous rappelons qu'une unité médiatique (U.M.) est une « unité informative qui a une certaine autonomie du point de vue d'un utilisateur (i.e qui a un sens en soi et donc présente une idée ou un concept cohérent) et qui mérite d'être sollicitée dans plusieurs démarches de consultation »[5].

Il existait huit types d'unités médiatiques dans xweb et nous en avons ajouté une de plus de type service web.
Une unité médiatique de type service web décrit un service afin de le rendre accessible à partir d'une autre U.M. . Elle fournit le nom, le fournisseur, les fonctionnalités et l'adresse WSDL du service.

Voici un exemple d'unité médiatique de type service web qui permet de décrire le service nommé Calcul1Service.

```
<um id="sw" type="service_web">
  <nom>CalculService</nom>
  <fournisseur>LANI</fournisseur>
  <fonctionnalites>
    <fonction>somme</fonction>
    <fonction>produit</fonction>
  </fonctionnalites>
  <adresse_wsdl>http://localhost:8080/axis/calcul.jws?wsdl</adresse_wsdl>
</um>
```

2.Extension de l'algorithme de génération d'U.M.

Xweb permet de générer un (ou plusieurs) site(s) web à partir des données XML d'une base de documents XML.
Xweb se compose de trois couches: Données, Application et Présentation.

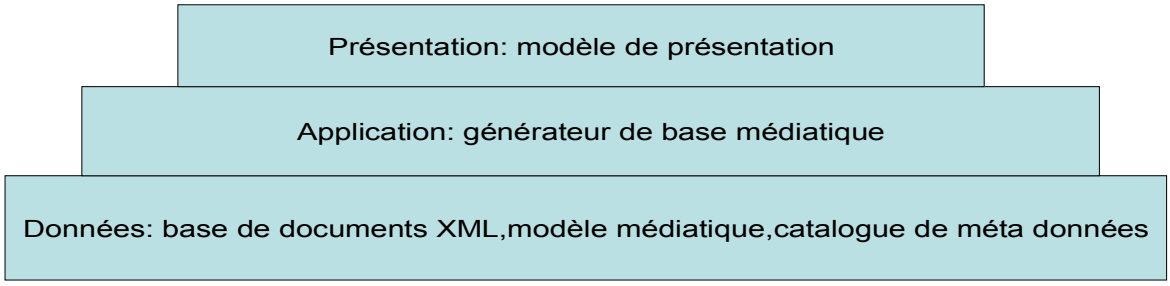

Figure 10: Schématisation du système Xweb

La couche Données offre à l'administrateur du système Xweb la possibilité de définir et de mettre à jour un modèle médiatique ou un catalogue de méta données.
La couche Application permet de générer automatiquement une base médiatique (matérialisant le contenu d'un site web) à partir d'un modèle médiatique. Elle utilise l'algorithme de génération d'unités médiatiques.
La couche Présentation permet d'associer à chaque document XML d'une base médiatique une forme médiatique (feuille de style XSL ou CSS) définie par l'administrateur afin de permettre une visualisation conviviale du côté client.

Nous avons étendu l'algorithme de génération d'unités médiatiques en lui permettant de pouvoir traiter les unités médiatiques de type service web.

```
Procédure Générer_Page ( Racine : Noeud )
 Début
    Ouvrir en écriture le fichier (nommé F) dans lequel sera créé l'UM de type «unité page»
       / * le nom physique est déterminé à partir de la balise id et du catalogue */
       Pour chaque noeud fils (nommé nd_fils) de la racine de l'arbre Faire
              Traiter_Noeud (nd_fils, F)
       Fin Pour
 Fin
```

```
Procédure Traiter_Noeud ( nd : Noeud, F : Fichier )
      Debut
        Selon le type de nd
         Cas "Texte" :
                insérer le texte dans le fichier F
        Cas "Image" :
                trouver le nom physique du fichier image à partir du catalogue
                l'insérer dans le fichier F
         Cas "Lien" :
                 trouver le nom physique de l'URL à partir du catalogue
                 créer un lien référentiel vers cette URL dans le fichier F
         Cas "Xobjet" :
                 exécuter la vue
                 insérer le résultat dans le fichier F
         Cas "contexte simple" :
                 exécuter la vue
                 en déduire les Xobjets et un index
                 insérer l'index dans le fichier F
                 créer un fichier pour chaque Xobjet
         Cas "contexte composé" : /* version simplifiée : seules 2 vues sont considérées ! */
                 exécuter la 1ère vue
                 créer un index principal à partir de la 1ère vue et l'insérer dans le fichier F
                 Pour chaque élément de l'index principal
                        exécuter la 2ième vue conditionnellement à cet élément
                        en déduire les Xobjets et un index secondaire
                        créer un fichier pour l'index secondaire
                        créer un fichier pour chaque Xobjet
                 Fin Pour
          Cas "service web" :
                  /* un fichier vide nommé uddi.xml ayant un seul et unique noeud (<racine/>)
                     sera créé au préalable par l'administrateur du système xweb */
                  insérer le nom du service web dans le fichier uddi.xml
                  insérer et le nom et les autres informations relatives au service dans le fichier F
        Fin Selon
      Fin
```

Algorithme 2 : Algorithme de génération d'unités médiatiques et d'intégration de services web dans Xweb

3.Les différentes étapes de la procédure d'intégration de services web dans Xweb

a) Créer en tant que administrateur du système Xweb un fichier XML vide nommé uddi.xml (i.e ce document XML ne comporte qu'un seul noeud qui est le noeud racine).

b) Décrire formellement en XML les unités médiatiques de type « service web » à intégrer dans le système Xweb.

L'algorithme de génération d'une page étant maintenant en mesure de traiter des unités médiatiques de type « service web » s'opére en recevant en entrée une U.M. de ce type comme suit :

- il ouvre d'abord en écriture le fichier uddi.xml et y insére le nom du service web émanant de l'U.M. reçue en entrée (i.e dans ce cas la liste des services web disponibles sera stockée dans une entrée de type nom de service).
- il génére ensuite une U.M. de type «unité page » à partir de la description de l'unité médiatique reçue comme paramètre d'entrée.