Institut **U**niversitaire **P**rofessionnel

UNIVERSITÉ D'EVRY
VAL D'ESSONNE

Méthodes **I**nformatiques **A**ppliquées à la **G**estion des **E**ntreprises

Projet de Documents Structurés

A l'intention de M. Henry Boccon-Gibod

Promotion 2006-2007

Par :

Benjamin ANTOINE

Julien HERON

Suivie du document

Destinataires	Contact	Rôle
Henry BOCCON-GIBOD	Henry.Boccon-Gibod@edf.fr	Ingénieur senior en recherche appliquée
Antoine RIZK	arizk@valoris.com	Ingénieur

Identification du document	
Référence	RapportWebServices V3.3WebServices-Heron-Antoine_2007_Rapport.doc.doc
Auteurs	Benjamin ANTOINE, Julien HERON
Date de création	1 décembre 2006
Version	V3.2
Date dernière mise à jour	20 février 2007

Historique du document	
Référence	RapportWebServices V3.2.doc
Date de rendu	20 février 2007
Référence	CDC WS2version.doc
Date de rendu	18 février 2007
Référence	CDC-Spécification.doc
Date de rendu	25 janvier 2007
Référence	CDCmodifiéWebServices-Heron-Antoine_2007_Rapport.doc.doc
Date de rendu	13 janvier 2007
Référence	CDCvWebServices-Heron-Antoine_2007_Rapport.doc1.doc
Date de création	1 décembre 2006
Date du rendu	5 décembre 2006
Avis du professeur	Validé

SOMMAIRE


WEB SERVICES

Introduction

Pour l'obtention de notre diplôme de Méthodes Informatiques Appliquées à la Gestion des Entreprises, au cours de l'IUP Master 2, nous réalisons un projet dont l'objectif est d'écrire une application de service Web et la rendre accessible via SOAP/WSDL.
La connaissance des ressources intra-entreprise est devenue depuis une dizaine d'année, un enjeu stratégique car une connaissance non exploitée entraîne généralement un manque à gagner pour l'entreprise, voir un risque.
Le service Web se propose de répondre à cette problématique.

Tout au long de ce projet, il nous a été demandé en parallèle de :

- Mener une étude afin de cibler des besoins fonctionnels de notre application
- Rédiger et définir le Cahier des Charges (CdC)
- Concevoir différentes solutions répondant au Cahier Des Charges

Ce rapport est constitué de plusieurs parties qui relatent les différentes recherches, études, conceptualisations techniques menées par chaque membre de notre équipe dans le but de mener à bien ce projet.

I. Présentation du sujet

« EXTERNALIS » est une entreprise SSII regroupant des experts dans des domaines informatiques.

L'entreprise Web@SERVICE auquel nous appartenons a répondu à l'appel d'offre de l'entreprise EXTERNALIS.

L'entreprise EXTERNALIS souhaite mettre en place au sein de son système d'information, un service Web permettant la constitution d'une équipe à partir de la consultation d'une base de Curriculum Vitae.

L'enjeu pour l'entreprise EXTERNALIS sera de sélectionner les personnes de leur entreprise ayant les qualifications appropriées pour telle ou telle mission.

L'entreprise EXTERNALIS souhaite par cet intermédiaire répondre plus rapidement aux offres de leurs clients en termes d'effectif.

II. Equipe projet

L'équipe projet est composée de deux étudiants issus de la promotion 2006-2007 en Master 2 MIAGE Alternance :

Benjamin ANTOINE
25 ans
(MASTER2 MIAGE ALTERNANCE)

En apprentissage au sein de l'entreprise DASSAULT SYSTEMES :
Il participe à la qualité de l'infrastructure d'intégration des logiciels dans le respect du processus de certification.

Julien HERON
23 ans
(MASTER2 MIAGE ALTERNANCE)

En apprentissage au sein de l'entreprise AIR FRANCE :
Il participe au rapprochement Air France-KLM au travers du rapprochement des logiciels de calcul de recette.

III. Cahier des charges

III.1 Analyse du besoin

But du service Web

Le but du service Web est de constituer une équipe par l'intermédiaire de données stockées dans des fichiers intégrant les compétences des collaborateurs de l'entreprise « EXTERNALIS »

Utilisateurs du service Web

Les utilisateurs de ce service seront les managers désireux d'obtenir une équipe d'experts qualifiés dans les domaines sélectionnés précédemment.

Personnes et organismes impliquées dans les enjeux du projet

Les personnes impactées dans ce projet sont :

-les employés de l'entreprise « EXTERNALIS » qui devront émettre leurs CV afin qu'ils soient stockés dans la base,
-les managers en Ressources Humaines qui ont une grande attente de ce projet,
-l'equipe projet qui devra mettre en place un outil répondant aux attentes du client,
-l'équipe projet « E-Miage » travaillant sur la base de données HR-XML.
(Intervenants du projet : Mme DAOUDI et M. REGNIER)

III.2 Contraintes sur le projet

Contraintes non négociables

Un système d'exploitation qui puisse supporter l'application et un autre pour la prise en compte d'un serveur.
Le projet prend en compte le modèle client/serveur.
Donc, il est recommandé de mettre en place un réseau nécessitant un grand niveau de fiabilité.

Faits et hypothèse utilisés

Il faudra mettre en place une base de données intégrant des fichiers CV sous format XML.
Les souhaits du client sur la partie service web :

-Toutes les données à échanger sont exprimées en XML.
Le choix du client s'est porté sur SOAP pour le codage d'échange et non par XML-RPC.

-Le Protocole commun utilisé durant notre projet pour permettre de transiter des données XML est HTTP.

-L'interface publique au service Web est décrite par le protocole WSDL qui est en cours de normalisation. C'est une description XML qui décrit la façon de communiquer en utilisant le service Web.

-Le service Web peut être connu sur le réseau au moyen du protocole UDDI. Il permet à des applications de rechercher le service web dont elles ont besoin.

Glossaire et conventions de nomenclature

XML : Extensible Markup Language
SOAP : Simple Object Access Protocol
XML-RPC : Extensible Markup Language Remote procedure call
HTTP : Hypertext Transfer Protocol
WSDL : Web Services Description Language
UDDI : Universal Description Discovery and Integration
BD : Base de données
Interface : Liaison entre l'utilisateur et le système.

Contraintes de déploiement

L'entreprise « EXTERNALIS » dispose de machines permettant l'hébergement de l'application et son accès Internet via un réseau.
Nous proposons à l'entreprise « EXTERNALIS » de mettre en place l'architecture du service Web n'entrainant aucune installation par le client. Ceci à pour conséquence la mise en place des pré-requis au niveau du client, du serveur et de la base de données.

Lors du déploiement, l'entreprise Web@SERVICE doit fournir :

Pour la partie cliente :
- L'installation d'un browser

Pour la partie serveur :
- L'installation et l'administration d'Apache «TOMCAT »
- et l'installation du module de Web-service AXIS 2.0

Pour la partie base de données :
Cette partie est sous-traité par l'entreprise « E-MIAGE » spécialiste du module HR-XML.

III.3 Exigences fonctionnelles

Contexte du travail : étude du domaine et de l'existant

A ce jour, l'entreprise « EXTERNALIS » n'a mis en place aucun système de récupération d'informations sur les compétences de ses employés et encore moins de la constitution d'une équipe suivant les compétences de chacun.

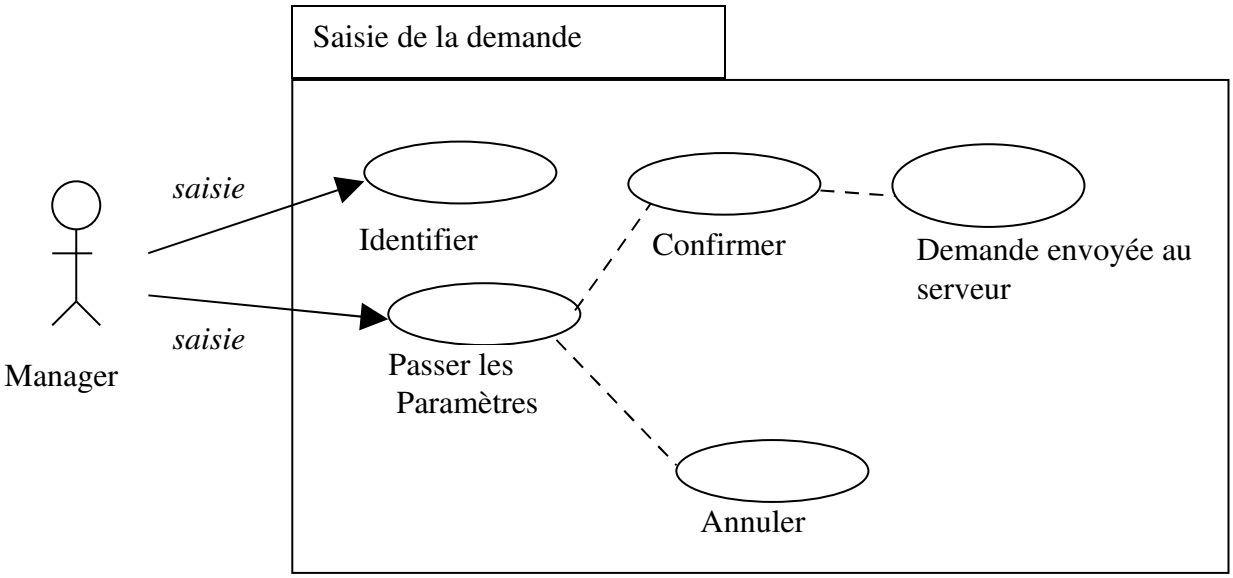
Limite du produit (défini par le cas d'utilisation)

En premier lieu, l'entreprise « EXTERNALIS » souhaite voir une première version de l'application fin 2007.
Cette première version devra comporter, l'accès à un profil correspondant aux compétences sélectionnées auparavant sur le poste client.

Scénario éventuel :
Le manager ou le responsable RH de l'entreprise « EXTERNALIS » formule sa demande via une IHM sur sa machine (machine cliente).

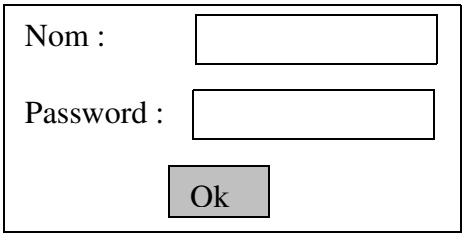
Après vérification des différents paramètres, le manager valide son formulaire et donc émet une requête vers le serveur qui va sélectionner son indice et va exécuter une procédure de recherche sur la base pour ensuite renvoyer la réponse au client.

Cas d'utilisation

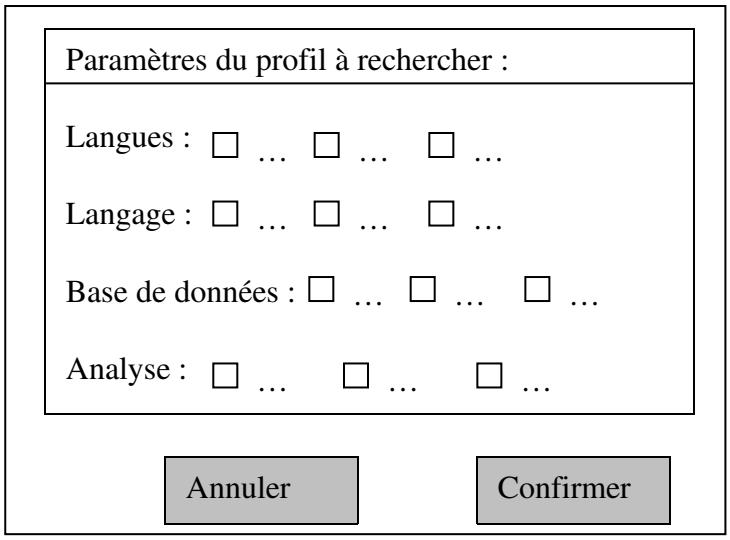


Au départ, le manager ou personnel ayant accès à la saisie de la constitution de l'équipe se doit de s'authentifier.

Interface 1:



Tant que le manager n'est pas identifié, il n'a aucune possibilité d'accéder à l'interface 2.
Ensuite l'utilisateur accède à l'interface de saisie des paramètres et des résultats.
Interface 2



Face à cette interface de saisie, le manager va renseigner les informations par exemple via des boites à cocher.
Après avoir soumis ses attentes, le manager souhaite obtenir la réponse. Pour ce faire, il doit confirmer.
La confirmation a pour effet, un envoi d'un message qui déclenche un processus de traitement au niveau du service se trouvant sur le serveur et la récupération des données au niveau de la base.
Les limites du projet sont que le manager ne peut réaliser deux demandes au même instant.

Résultat de la recherche (Infos Collaborateurs) :

Réponse de la requête

Id Employé	Nom	Prénom	Service	Tel	Email	Compétences	Statut act.

L'interface précédente montre les informations sur les collaborateurs susceptibles d'intéresser le manager.
Les informations concernent des données liées à la personne, à ses compétences et à son statut (libre ou en mission).
Concernant ce type de récupération, il serait intéressant dans un futur que le manager puisse cliquer sur l'adresse mail du collaborateur pour pouvoir réaliser l'envoi de mail.

Cas d'utilisation point de vue Serveur : (Traitement de la demande)

Le serveur reçoit la demande, trouve la méthode et fait ensuite une requête à la base de données pour chercher dans les fichiers XML, les personnes susceptibles de répondre aux attentes du manager.

III.4 Exigences non fonctionnelles

Ergonomie et convivialité du produit

Les utilisateurs du produit (utilisateurs finaux) ne seront pas des informaticiens, donc l'interface doit être simple et compréhensible par tous.
Les interfaces sont simples d'utilisation, et intuitives.
Une aide en ligne et une documentation permettent à l'utilisateur de naviguer simplement sans aide extérieure.

Facilité d'utilisation

L'interface graphique permet au manager de se guider dans la sélection des choix des paramètres de la constitution d'équipe sans difficulté à l'aide d'un menu et d'une aide.
Une formation sur le produit auprès du client final ne doit pas être indispensable.

Performance

Pour que notre logiciel réponde au critère de performance, le temps de réponse maximal ne doit pas dépasser 30 secondes, et ce, quelque soit la recette.

Conditionnement de fonctionnement

Pré requis :
Un système d'exploitation pour permettre la mise en place d'un serveur et un autre système d'exploitation pour la mise en place d'un client.
L'annuaire UDDI pour la recherche du service n'est pas pris en compte dans l'appel d'offre.

Maintenance et support, portabilité

L'offre comprend une assistance aux utilisateurs d'une durée de 2 ans.

L'entreprise « EXTERNALIS » a comme choix de reconduire ou non du contrat de maintenance au-delà d'un an.

Sécurité

Des moyens devront être mis en place pour bloquer les autres applications conflictuelles avec notre interface.

Essayer de sécuriser au maximum la machine pour éviter tout type de virus ou de programme qui pourrait récupérer des données confidentielles.

III.5 Chiffrage

Les différentes interfaces et les Bases de données peuvent être faites indépendamment. Les modules peuvent être regroupés toutes au long du projet.

Chiffrage Jour / Homme				
Désignations	**Modules**	**Descriptifs**	**Temps**	**Hommes**
Développement	Interfaces	Interface d'authentification	1 semaine	1
		Interface de sélection	1 semaine	1
		Interface résultat de l'équipe	2 semaines	1
	Traitements		2 semaines	2
	SOAP		1 semaine	2
	WSDL		1 semaine	2

IV. Spécification

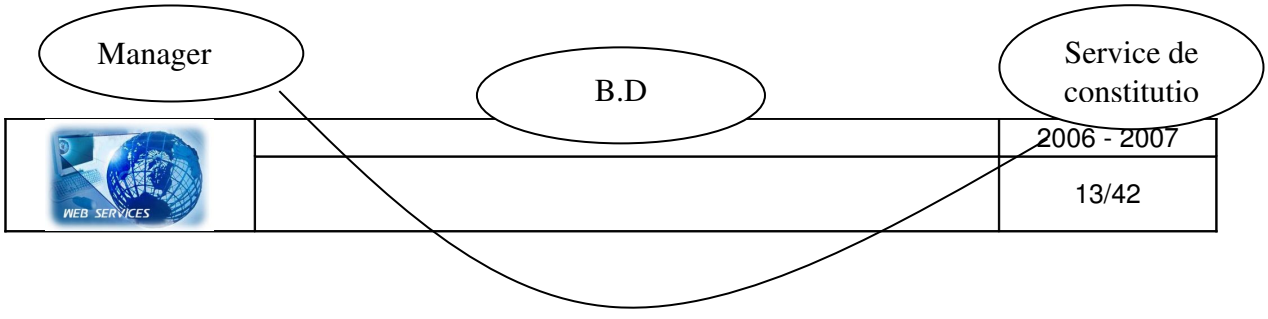
Dans la spécification, nous présentons comment nous allons mettre en place l'architecture pour arriver au résultat final qui est la création d'une application de service Web permettant la constitution d'une équipe.

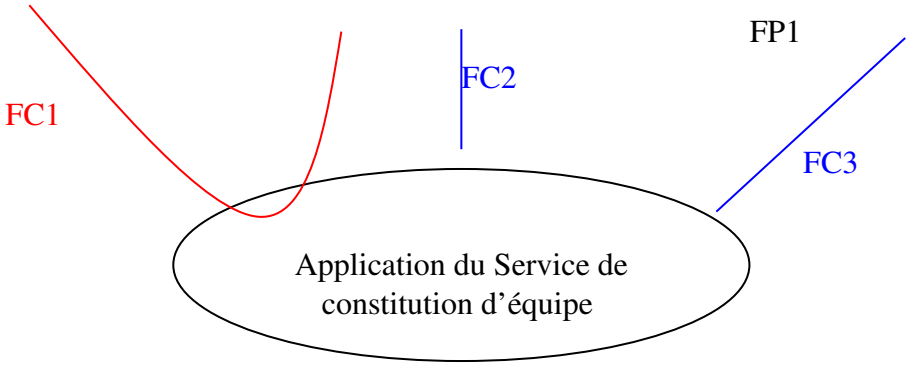
IV.1 Exigences fonctionnelles

Description des fonctions :

Fonctions de Contraintes = FC
Fonctions Principale = FP.

Diagramme d'interaction





FP1 : Permettre au manager (RH) de réaliser la constitution d'équipe.
FC1 : Vérifier si le manager est bien référencé dans la base
FC2 : Consulter la base de données
FC3 : Consulter le service

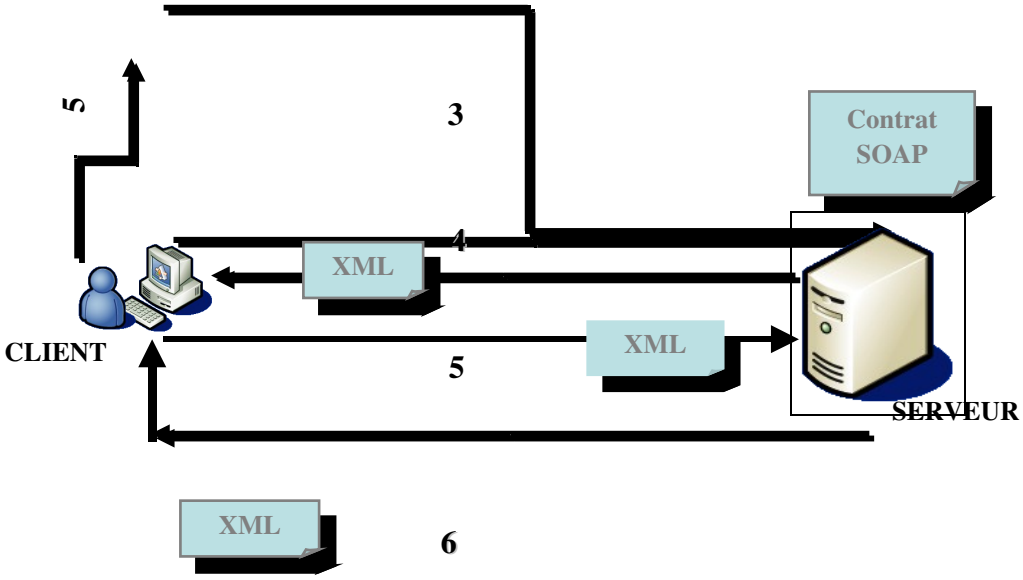
IV.2 Présentation de l'architecture

Architecture d'un service Web

La définition de l'architecture des services Web se réalise par la réponse aux questions suivantes :

- Qu'essaie-t-on de faire avec ce service Web ?
- De quelles technologies a-t-on besoin ?
- Quelles sont les limites d'application de chaque technologie ?
- Quelles sont les relations entre chacune des technologies identifiées ?

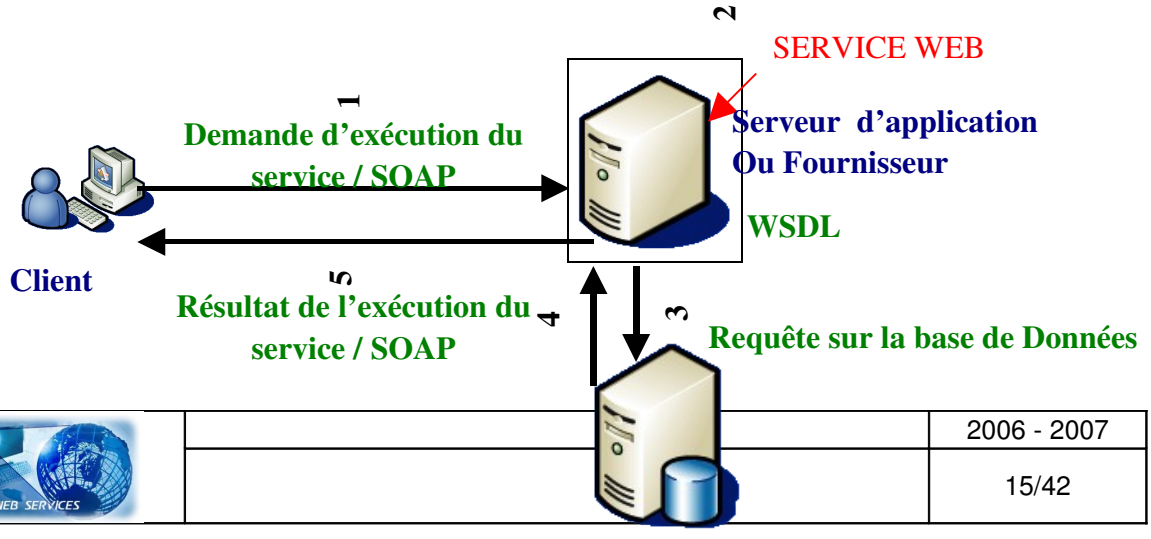
Le cycle de vie de l'utilisation d'un service Web :



Les différentes étapes vues sur le schéma de la page précédente:

1 : Je recherche un service WEB
2 : J'ai trouvé! Voici le serveur
3 : Quel est le format d'appel du service que tu proposes?
4 : Voici mon contrat (WSDL)
5 : J'ai compris comment invoquer ton service et je t'envoie un document XML représentant ma requête
6 : J'ai exécuté ta requête et je te retourne le résultat

Architecture de notre projet



Base de données HR-XML

1 : J'envoie les paramètres de mon formulaire via une requête HTTP-SOAP.
2 : En suite, après avoir trouvé mon service web, j'appelle les méthodes via le descripteur du service (WSDL).
3 : La méthode interroge la base de données HR-XML via des requêtes XPath.
4 : Retourne le résultat des requêtes
5 : Retour du résultat via une Réponse HTTP-SOAP.
Dans notre spécification, nous avons réfléchis sur l'éventualité d'utiliser DOM pour parser les documents XML.
Conceptuellement, l'architecture ci-dessus orientée service met en place deux rôles fondamentaux qui interagissent : le fournisseur de service et le requêteur du service (client).

Fournisseur :
Dans l'architecture, un fournisseur est un composant qui est vu par le monde extérieur comme le cœur d'un service. Celui-ci contient en général la description du service web au format XML (WSDL), ainsi que son implémentation concrète qui encapsule la logique métier du service.
Dans le cadre des services, ce service sera accessible via le protocole SOAP et devra fournir la gestion de protocole requête/réponse. Pour notre projet, cette implémentation est assurée par un serveur dédié comme AXIS de la communauté Apache.

Client:
Ce terme désigne toute partie logicielle qui recherche, trouve et invoque le fournisseur de services.

Description des différentes parties de l'architecture

Pour notre projet, il n'est pas encore définit la partie de la recherche du service Web (voir dans les prochaines réunions)

Partie Serveur
Le serveur aura pour fonction de stocker et de mettre à disposition les services appelés par le client. Pour que le serveur soit accessible, le client devra passer par une interface.

Partie Base de données

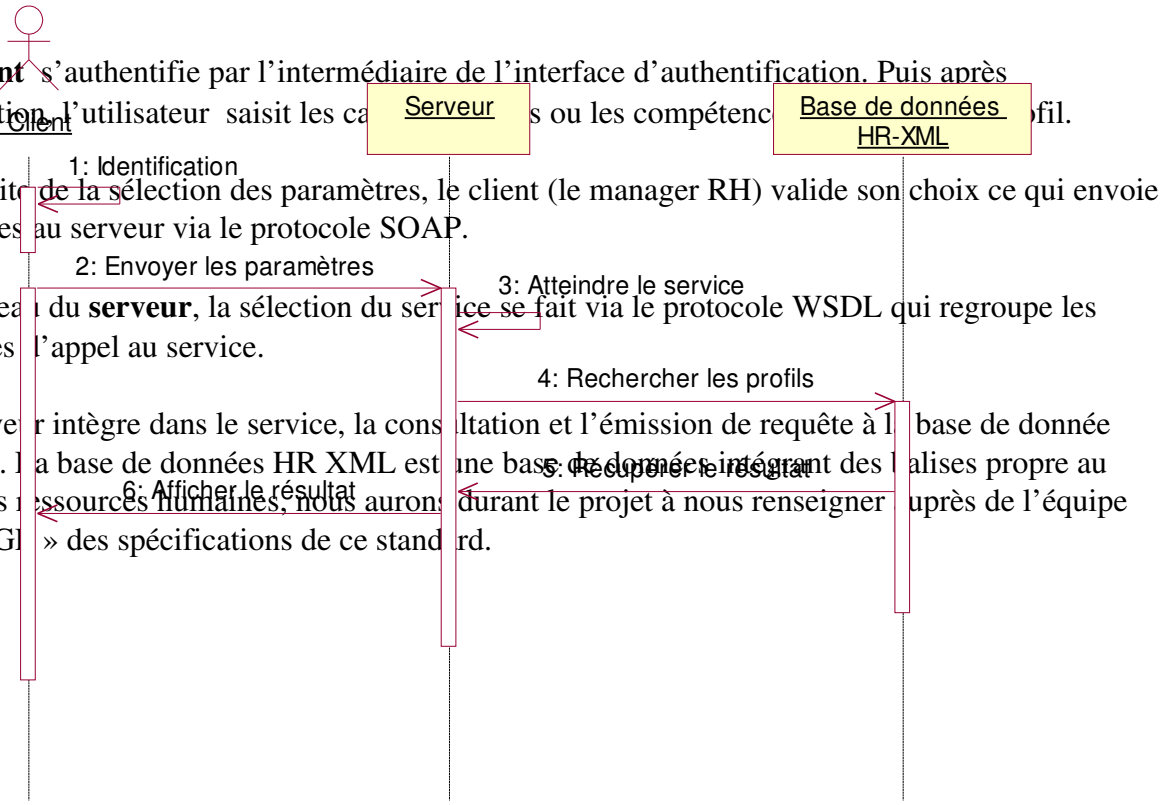
Dans la base de données, on retrouve les CV des différents employés exprimés en XML.
Il n'a pas été indiqué durant la rencontre avec les clients de la spécificité du stockage du fichier.
La base de données doit comporter l'authentification du manager (login, password).

Partie Interface cliente

Le client doit être convivial et être accessible par les utilisateurs sans besoin de formation.
Le client comporte les éléments graphiques nécessaires pour :

- s'authentifier (champs de saisie, boutons),
- sélectionner les paramètres de la constitution d'équipe, (Boites à cocher)
- afficher le résultat de l'équipe.

Diagramme de séquence :



1/ Le **client** s'authentifie par l'intermédiaire de l'interface d'authentification. Puis après identification, l'utilisateur saisit les ca s ou les compétenc fil.

2/ A la suite de la sélection des paramètres, le client (le manager RH) valide son choix ce qui envoie les données au serveur via le protocole SOAP.

3/ Au niveau du **serveur**, la sélection du service se fait via le protocole WSDL qui regroupe les paramètres l'appel au service.

4/ Le serveur intègre dans le service, la consultation et l'émission de requête à la base de donnée HR XML. La base de données HR XML est une base de données intégrant des balises propre au métier des ressources humaines, nous aurons durant le projet à nous renseigner auprès de l'équipe « E-MIAGE » des spécifications de ce standard.

Description des deux composants formant le service web

SOAP, Simple Object Access Protocol

SOAP est le langage d'invocation des services web. Le plus souvent, SOAP est utilisé au-dessus du protocole http. La spécification de SOAP définit la structure de l'enveloppe et les règles d'encodage utilisées. On utilise SOAP pour transporter un document XML ou une invocation de services.
Son utilisation dans notre service web sert principalement aux appels et aux retours de méthodes ou procédures encodées au format XML.

Dans le serveur SOAP, on retrouve :

- l'analyse de messages,
- la sérialisation/.désérialisation de données,
- et le formatage de message.

Le protocole SOAP s'organise autour de la notion d'enveloppe qui permet de spécifier le contenu du message, les règles d'encodage des données transportées.

WSDL, Web Service Description Language

Dans notre architecture vue précédemment, le WSDL correspond à la description du service web. Cette interface contient des informations de deux types : une description abstraite du service et les liens avec l'implémentation.

Structure d'un document WSDL :
<types> : définit les types de données utilisés pour les paramètres du service appelé.
<message> : décrit les noms et les types de l'ensemble des paramètres d'entrée/sortie pour chaque opération.
<porttype> : correspond à la description de l'ensemble des méthodes.
<binding> : spécifie la liaison de chaque opération définie dans la balise <porttype> à un protocole.
Les protocoles principaux sont :
SOAP : choix pour le protocole de transport,
HTTP : choix de la méthode uniquement (GET ou POST),
MIME : utilisation du protocole SOAP ou http avec un fichier attaché.

Nous utiliserons pour notre service, le protocole SOAP.

Dans le cadre de notre projet, AXIS nous offrira deux outils complémentaires que nous utiliserons. Le premier représenté par la classe **org.apache.axis.wsdl.Java2WSDL** permet, à partir d'une classe Java, de générer le flux WSDL adéquat le représentant. Nous avons au départ de notre implémentation écrit un fichier WSDL que nous avons mis de côté car celui-ci peut devenir vite lourd et fastidieux d'écrire l'interface WSDL.

A partir d'un flux WSDL, grâce à l'outil **org.apache.axis.wsdl.Java2WSDL**, le « stub » et « skeleton » du service web seront générés automatiquement.

IV.3 Présentation des outils utilisées

Nous avons choisit la plate-forme J2EE car celle-ci à le luxe d'être gratuite.

IV.3.1 Serveur d'application : Tomcat

Tomcat est un serveur d'applications Java. Il permet de générer une réponse HTML à une requête après avoir effectué un certain nombre d'opérations (connexion à une base de données).
Pour le client (un navigateur Web en général), il n'y a pas de différence avec une page Web statique : il reçoit toujours du HTML, seul langage qu'il comprend. Seule la manière dont la réponse est formée côté serveur change.
Les requêtes, pour le client, ne diffèrent pas non plus. Qu'il souhaite accéder à une ressource statique ou à une application Web, il utilise toujours une URL au même format (standard HTTP).

IV.3.2 Utilisation d'Axis

AXIS (Apache eXtensible Interaction System) est un framework de gestion de message SOAP de la communauté Apache. AXIS est principalement un moteur SOAP. AXIS se place autant comme serveur de messages SOAP que client.
AXIS se présente principalement sous deux types différents de serveurs. (mode « Standalone » et mode « servlet »)
Il est accompagné d'outils : - génération de flux WSDL à partir de classes métiers,
- génération de classes de communication basées sur SOAP (client/serveur) à partir d'un document WSDL.

Nous avons optez pour notre projet, à utiliser AXIS car les outils d'administration permettent de déployer, supprimer, lister des services web, ainsi que d'auditer les flux d'entrée/sortie.
L'utilisation d'AXIS a été un choix afin de nous renseigner et travailler sur ce qui est utilisé dans les entreprises qui créée des services web.

IV.3.3 Base de données XIndice

Avec l'apparition du langage XML sont également apparues des bases de données permettant de stocker directement des informations sous format XML. A la place de requête SQL, on effectuera des requêtes sous formats XPath.
La base de données XIndice permettra d'intégrer nos fichiers HR-XML afin de constituer l'ensemble des compétences et donc d'obtenir des résultats aux requêtes.
De plus, la base de données XIndice fait parti du projet Tomcat Jakarta.

IV.3.3 Outil de développement : Eclipse

L'outil de développement ou l'IDE que nous avons choisit est Eclipse car Eclipse a un Plug-in permettant de le lier au serveur d'application Tomcat.

V. Implémentation du service web

Pré-requis

Nous utiliserons pour les tests les outils suivants :

A. le serveur web TOMCAT (http://jakarta.apache.org/tomcat/),
B. la dernière version du JDK Java J2SE (http://java.sun.com/j2se/1.5.0/download.jsp)
C. un navigateur (IE, NETSCAPE, MOZILLA, OPERA, ...)
D. l'outil de développement java ECLIPSE (http://www.eclipse.org/) avec les plugins suivants :
   - XmlBuddy pour gérer les documents XML (http://xmlbuddy.com/)
   - Tomcat de Sysdeo (http://www.sysdeo.com/eclipse/tomcatPlugin.html) pour gérer Tomcat à partir d'Eclipse
   - Lomboz pour la gestion des documents JSP (http://www.objectlearn.com/)
E. Le serveur de déploiement « Axis »,
F. Un serveur de base de données XIndice.

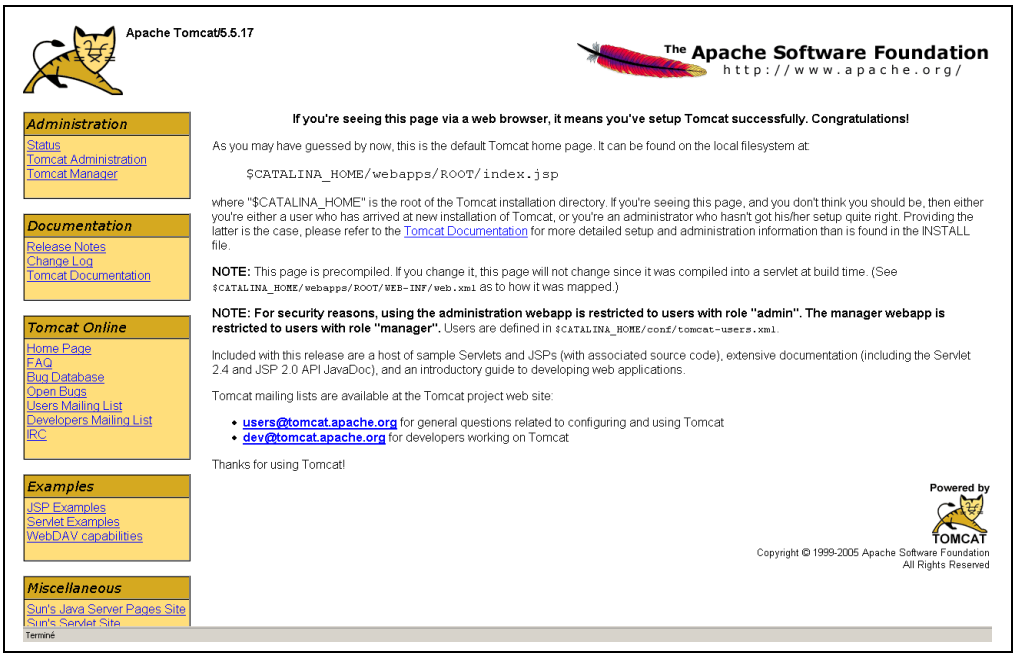
Ce sont des outils gratuits. De nombreux outils libres peuvent être utilisés pour créer des services web.

V.1 Installation de Tomcat

Une fois l'exécutable récupéré sur http://jakarta.apache.org/tomcat/ , nous l'avons installé avec toutes les options cochées, ce qui installe des exemples (JSP et Servlet), nous permettant de vérifier notre bonne configuration tant matériel que logiciel.

Une fois Tomcat lancé, en tapant **http://localhost :8080**, vous devez arriver sur une page du type :

Figure 1: page d'accueil de TomCat



Arborescence de Tomcat :

Une fois Tomcat installé, l'arborescence suivante est disponible dans le répertoire TOMCAT_HOME (répertoire d'installation) :

- **bin/** contient les fichiers exécutables de Tomcat. En l'occurrence, principalement les scripts Linux ou fichiers batchs Windows permettant le démarrage et l'arrêt du serveur. Mais également l'archive bootstrap.jar, contenant les classes permettant le démarrage et l'initialisation du serveur.
- **common/** contient les classes et librairies partagées par les classes internes du serveur et toutes les applications web.
- **conf/** contient les fichiers de configuration de Tomcat.
- **logs/** contient tous les logs, à moins que, pour certains logeurs, ont était spécifié d'autres chemins d'enregistrement.
- **server/** contient les librairies et les applications web du serveur en lui-même. L'application d'administration ("Admin") par exemple.
- **shared/** contient les classes et librairies partagées par toutes les applications web hébergées sur le serveur.
- **temp/** est un simple répertoire temporaire.
- **webapps/** contiendra les applications web. C'est en tout cas le répertoire par défaut pour ce

faire. Car, comme nous l'expliquons dans la configuration, il est tout à fait possible de mettre une application ailleurs, en spécifiant un chemin absolu.

- **work/** est le répertoire workDir par défaut, qui contiendra notamment les JSP compilées.

NB : Lors de notre installation de Tomcat, nous avons également installé les paquets 2 et 3, correspondant respectivement au module de déploiement et au module d'administration web de Tomcat (cf. figure 2)

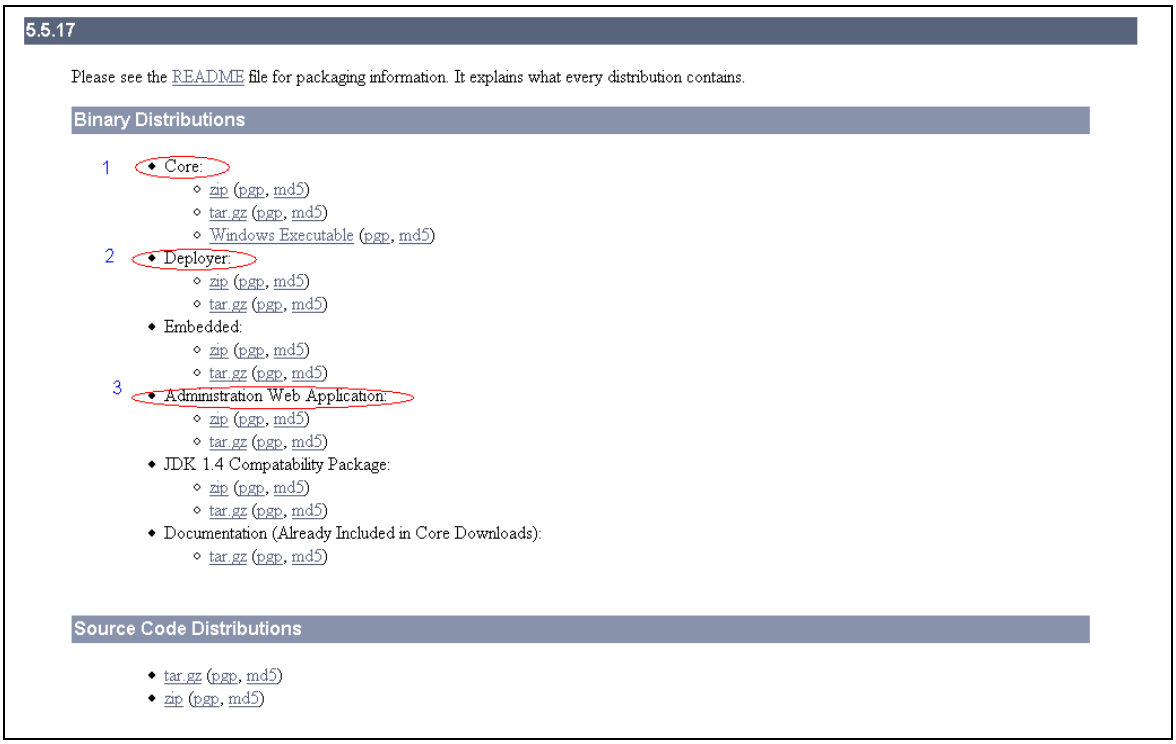


Figure 2 : Les archives à installer

Le module d'administration nous offre la possibilité de configurer les ressources mises par Tomcat à la disposition des applications web déployée en son sein. Un exemple classique est un pool de connexions à une base de données. Suivons le lien « Tomcat administration.
Nous obtenons une page d'identification :

Figure 3 : page d'accueil du module d'administration web de Tomcat

Pour s'y connecter, il suffit de créer un utilisateur dans <rep tomcat>\conf\tomcat-users.xml ayant le rôle « admin » ou « manager ».

Figure 4 : page principale du module d'administration

Le lien « Tomcat Manager » de http://localhost :8080 nous permet de gérer les applications chargées dans Tomcat, dynamiquement.
Passons maintenant à la configuration d'Eclipse, nous permettant ainsi de pouvoir écrire nos premiers Servlet/JSP.

V.2 Installation d'Eclipse

Nous avons ici utilisé la version d'Eclipse fournit par Lomboz (http://download.fr2.forge.objectweb.org/lomboz/lomboz-wtp-emf-gef-jem-eclipse-SDK-3.1.2-win32.zip), et comprenant tout le nécessaire au niveau des plugins pour traiter des pages JSP (ce qui nous intéresse) et bien d'autre chose encore.

A cette installation, nous ajoutons le plugin de Sysdeo pour nous permettre d'écrire plus facilement un projet utilisant Tomcat.

A) Configuration du plugin de Sysdeo

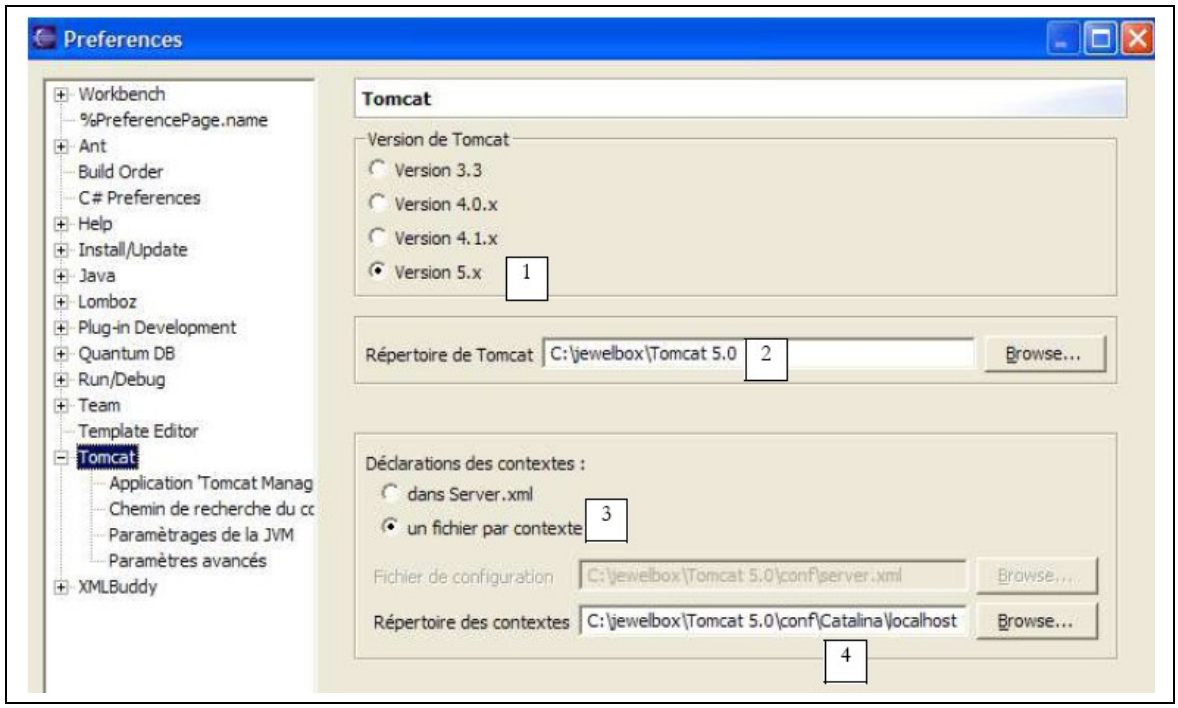
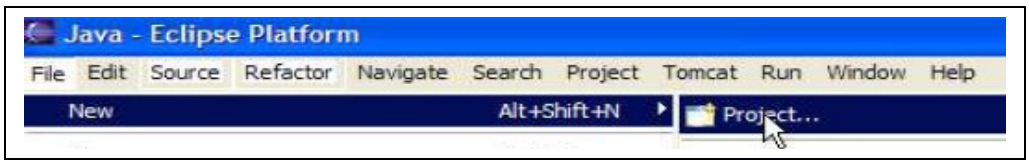
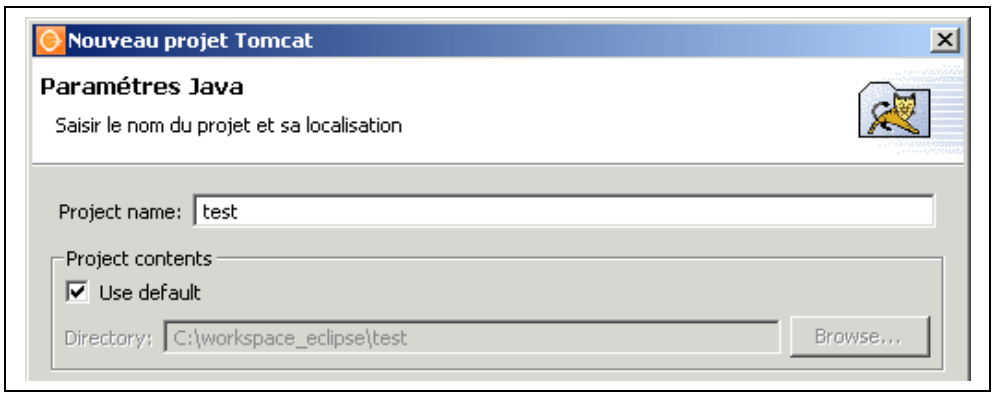


Figure 5 : configuration du plugin Tomcat dans Eclipse

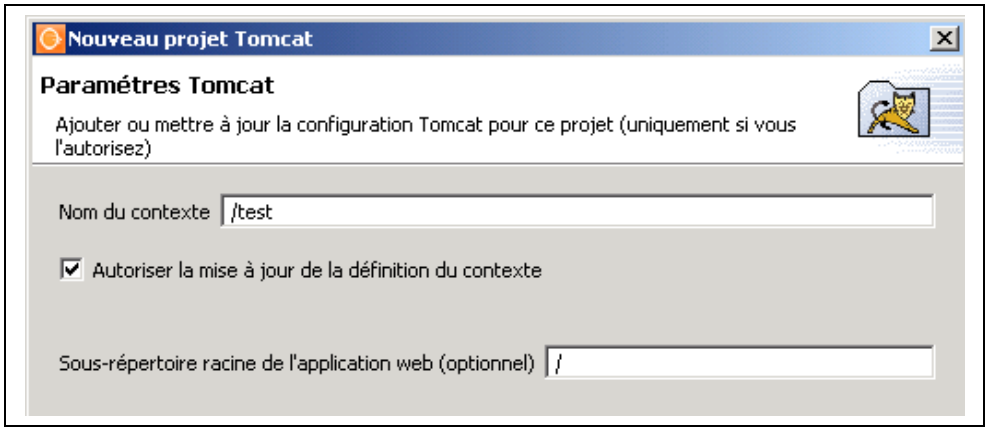
1 : définition de la version du serveur Tomcat. Ici, nous utilisons une version 5.x
2 : définition du répertoire d'installation de Tomcat
3 : le mode de déclaration des contextes
4 : le répertoire des contextes, de la forme <rep installation Tomcat>\conf\catalina\localhost

B) Création d'un projet Tomcat





On nous demande, ici, de définir le nom du projet et son emplacement



La seconde demande, quand à elle, spécifie la définition du contexte de l'application.

V.3 Ecriture d'une servlet

Définition

Une servlet est un programme Java, similaire aux CGI. Il se déclenche sur demande d'un navigateur et peut interagir avec des bases de données ou d'autres programmes pour fournir une réponse http au client.
Toutes les fonctionnalités offertes par les CGI sont disponibles pour les servlet (accès à une base de données, récupération de données d'un formulaire HTML, …)
Mais, contrairement aux CGI, un servlet n'est chargé en mémoire qu'une seule fois.

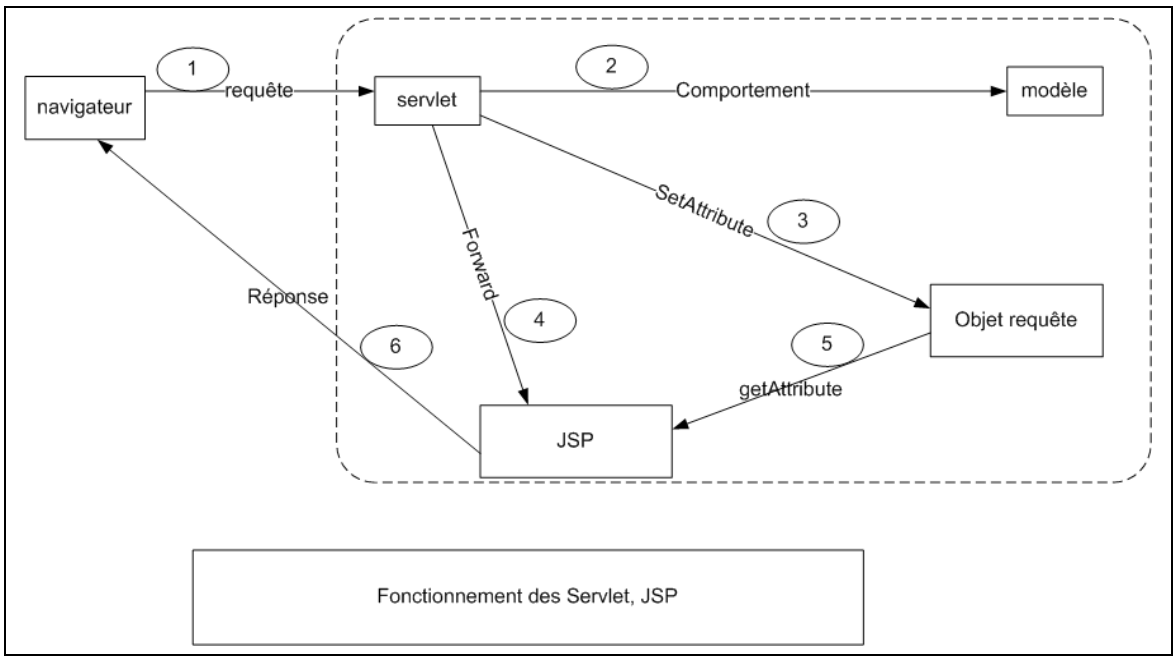
Une servlet est un programme qui offre de bonne garantie en matière de sécurité :
- Un servlet est écrit en java, il ne peut pas faire planter le serveur d'applications web par des erreurs de manipulation mémoire,
- Le code qui s'exécute sur le serveur est compilé, il n'est donc pas possible, en utilisant des failles de sécurité du serveur d'applications web d'accéder à des informations sensibles hébergées dans le code (contrairement aux CGI perl, aux scripts ASP, JSP ou PHP).
- Un servlet ne souffre pas de l'interprétation de metacaractères (comme les « |, >, >>, … » de perl) qui permettent aux « bidouilleurs » d'interagir avec le gestionnaire de fichiers.

Un servlet est la spécialisation de la classe java **javax.servlet.HttpServlet**. Afin d'initialiser un servlet un serveur d'applications web charge la classe du serveur, il crée une instance de cette classe en appelant le constructeur à zéro argument puis il déclenche la méthode **init(ServletConfig).** Les spécifications servlet garantissent aux développeurs que cette méthode ne sera exécutée qu'une seule fois par le serveur d'applications web. Lorsque le servlet est initialisé, il peut traiter les requêtes en provenance des navigateurs par la méthode **service (HttpServletRequest, HttpServletResponse)**.
Lorsque le servlet doit être déchargé la méthode **destroy()** est déclenchée.

Plusieurs types de requêtes HTTP peuvent être émis par un navigateur (GET, POST,…). La méthode service de **HttpServlet** achemine le traitement d'une requête HTTP vers la méthode adaptée. Une requête XXXX de HTTP est acheminée vers la méthode doXXXX(HttpServletRequest, HttpServletResponse) de java (i.e., le traitement de la requête HTTP GET est assurée par la méthode **doGet**). Les informations en provenance du navigateur sont accessibles via la classe **HttpServletRequest** (paramètres des formulaires, type de navigateur, referer,…). Les informations renvoyées aux navigateurs doivent l'être par l'intermédiaire de la classe **HttpServletResponse** (type MIME de la réponse, entête de la réponse, corps de la réponse, cookies,…).

Il est possible pour un servlet de récupérer une requête HTTP en provenance d'un navigateur, de générer une requête dynamiquement, en interrogeant le système d'informations si nécessaire, et de renvoyer une réponse contenant du HTML.
Le problème avec cette approche est que la construction de la page doit être faite par le servlet, ce qui veut dire qu'un infographiste qui voudrait modifier l'apparence de la page renvoyée devrait

modifier le code java de la servlet et le recompiler. Avec cette approche la génération de contenu dynamique nécessite des connaissances en programmation.
On comprend alors la nécessité de séparer la partie traitement de la requête HTTP de l'utilisateur (logique applicative, logique transactionnelle, règles métiers) de la partie renvoie de la page à l'utilisateur.
Pour cela, la plate-forme J2EE conseille plutôt le schéma suivant.



1. Un navigateur lance une requête sur un serveur d'applications web. La réception de cette requête est assurée par un servlet.

2. Le servlet soustraite la partie traitement de cette requête au modèle. Le modèle est assuré par un ensemble de classes java. Ces classes java accèdent au système d'information afin de récupérer les données requises par le traitement de la requête. La plateforme J2EE prévoit que le modèle respecte les spécifications EJB.

3. Le servlet récupère du modèle les données qui devront être renvoyées à l'utilisateur et les dépose dans un objet accessible par le JSP.

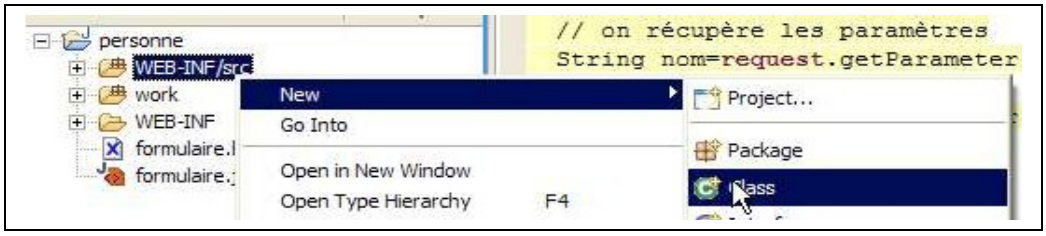
4. Le servlet soustraite l'affichage des données en provenance du modèle à un JSP.

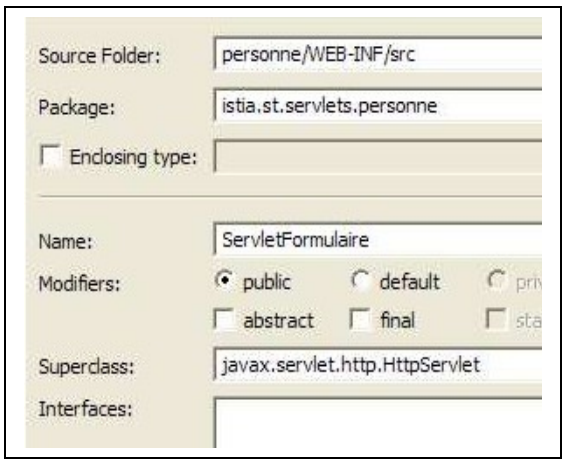
5. Le JSP récupère les données en provenance du servlet dans un objet.

6. Le HTML généré par le servlet est renvoyé au navigateur.

Notre premier servlet, avec Eclipse et Tomcat

Maintenant nous allons vous présenter la premier servlet que nous avons crée.
Pour cela nous avons crée une nouvelle classe et choisis java **javax.servlet.HttpServlet** comme classe mère.





Voici un exemple simple de servlet dont le seul but est d'afficher du texte sur le navigateur du client :

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class PremiereServlet extends HttpServlet {

 public void init() {
 }
 public void doGet(HttpServletRequest req, HttpServletResponse res)
              throws ServletException, IOException {

              res.setContentType("text/html");
              PrintWriter out = res.getWriter();
              out.println("<HTML>");
              out.println("<HEAD><TITLE> Titre </TITLE></HEAD>");
              out.println("<BODY>");
              out.println("Ma première servlet");
              out.println("</BODY>");
              out.println("</HTML>");
              out.close();
 }
}
```

La première étape consiste à importer les packages nécessaires à la création de la servlet, nous avons donc importé les paquetages *javax.servlet*, *javax.servlet.http* et *javax.io*

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*
```

Afin de mettre en place l'interface *Servlet* nécessaire au conteneur de servlet, il existe plusieurs possibilités :

- Définir manuellement chaque méthode
- Dériver la classe *GenericServlet* et redéfinir les méthodes dont on a besoin
- Dériver la classe *HttpServlet* et redéfinir les méthodes dont on a besoin

Dans la servlet ci-dessus, la classe *HttpServlet* a été étendue

```
public class PremiereServlet extends HttpServlet {
}
```

Lorsque la servlet est instanciée, il peut être intéressant d'effectuer des opérations qui seront utiles tout au long du cycle de vie de la servlet (se connecter à une base de données, ouvrir un fichier, ...). Pour ce faire, il s'agit de surcharger la méthode *init()* de la servlet.

```
public void init() {}
```

A chaque requête, la méthode *service()* est invoquée. Celle-ci détermine le type de requête dont il s'agit, puis transmet la requête et la réponse à la méthode adéquate (*doGet()* ou *doPost*). dans notre cas, on ne s'intéresse qu'à la méthode GET, c'est la raison pour laquelle la méthode *doGet()* a été surchargée

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
}
```

L'objet *HttpServletRequest* permet de connaître les éventuels paramètres passés à la servlet (dans le cas d'un formulaire HTML par exemple), mais l'exemple ci-dessus n'en a pas l'utilité. Par contre l'objet *HttpServletResponse* permet de renvoyer une page à l'utilisateur. La première étape consiste à définir le type de données qui vont être envoyées au client. Généralement il s'agit d'une page HTML, la méthode *setContentType()* de l'objet *HttpServletResponse* doit donc prendre comme paramètre le type MIME associé au format HTML (*text/html*) :

```
res.setContentType("text/html");
```

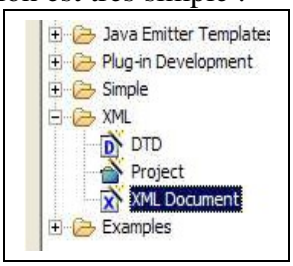
Ensuite la création d'un objet *PrintWriter* grâce à la méthode *getWriter()* de l'objet *HttpServletResponse* permet d'envoyer du texte formatté au navigateur (pour envoyer un flot de données, il faudrait utiliser la méthode *getOutputStream()*)

```
PrintWriter out = res.getWriter();
```

Enfin il faut utiliser la méthode *println()* de l'objet *PrintWriter* afin d'envoyer les données textuelles au navigateur, puis fermer l'objet *PrintWriter* lorsqu'il n'est plus utile avec sa méthode *close()*

```
        out.println("<HTML>");
        out.println("<HEAD><TITLE> Titre </TITLE></HEAD>");
        out.println("<BODY>");
        out.println("Ma première servlet");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
```

Déploiement d'un servlet dans Tomcat
Une fois la classe écrite et compilée, il nous faut faire prendre en compte cette dernière par Tomcat. Pour cela, il nous créer un fichier, nommé « web.xml ».
Le fichier web.wml
Pour que cette application soit reconnue par Tomcat, il nous faut donc écrire un fichier web.xml, situé dans le répertoire <nom du projet>\WEB-INF\web.xml.
Grâce au plugin xmlbuddy, sa création est très simple :



V.4 Lancement et configuration d'Axis

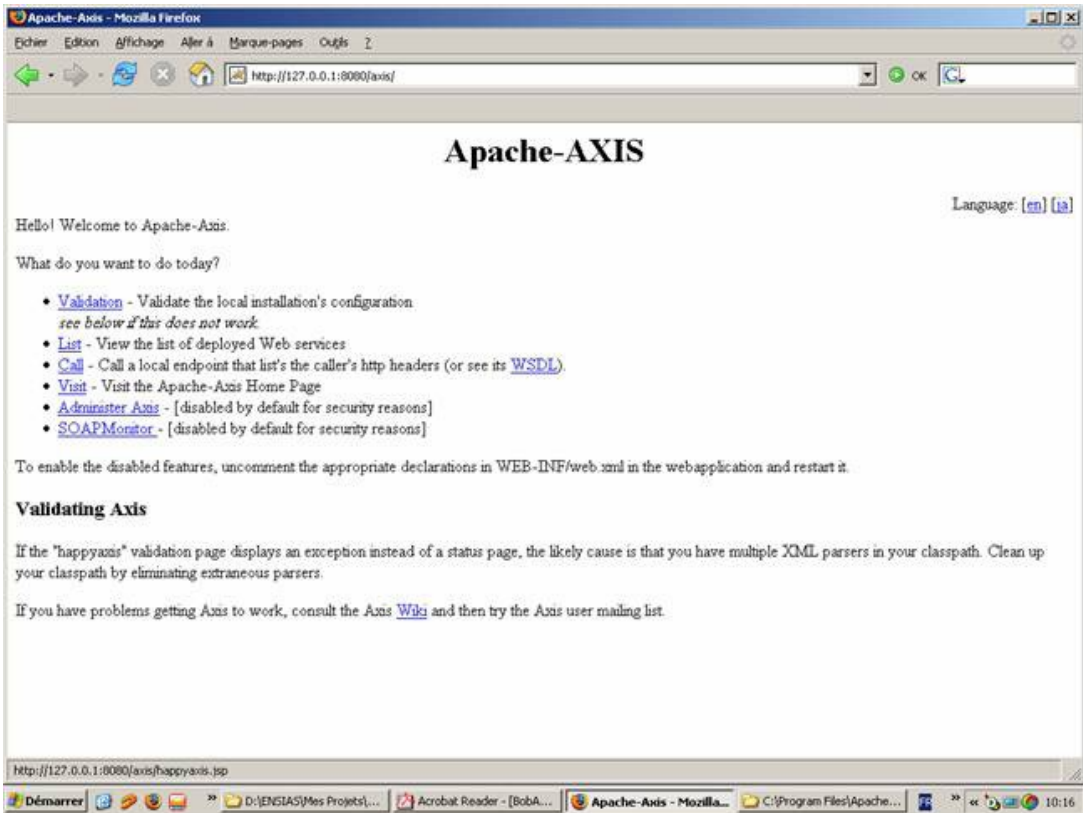
Démarrage de Tomcat

Nous devons, pour accéder aux possibilités de « Axis », activer préalablement le serveur « Apache Tomcat »
Nous avons utilisé la version 5.5.17 de Tomcat. (Sous Tomcat 5.5.x, double-cliquez sur : «dossierInstallation\Tomcat 5.5\bin\tomcat5.exe». Ou tout simplement, si le service Tomcat est installé, utiliser «Monitor Tomcat» puis le menu «Start Service».)

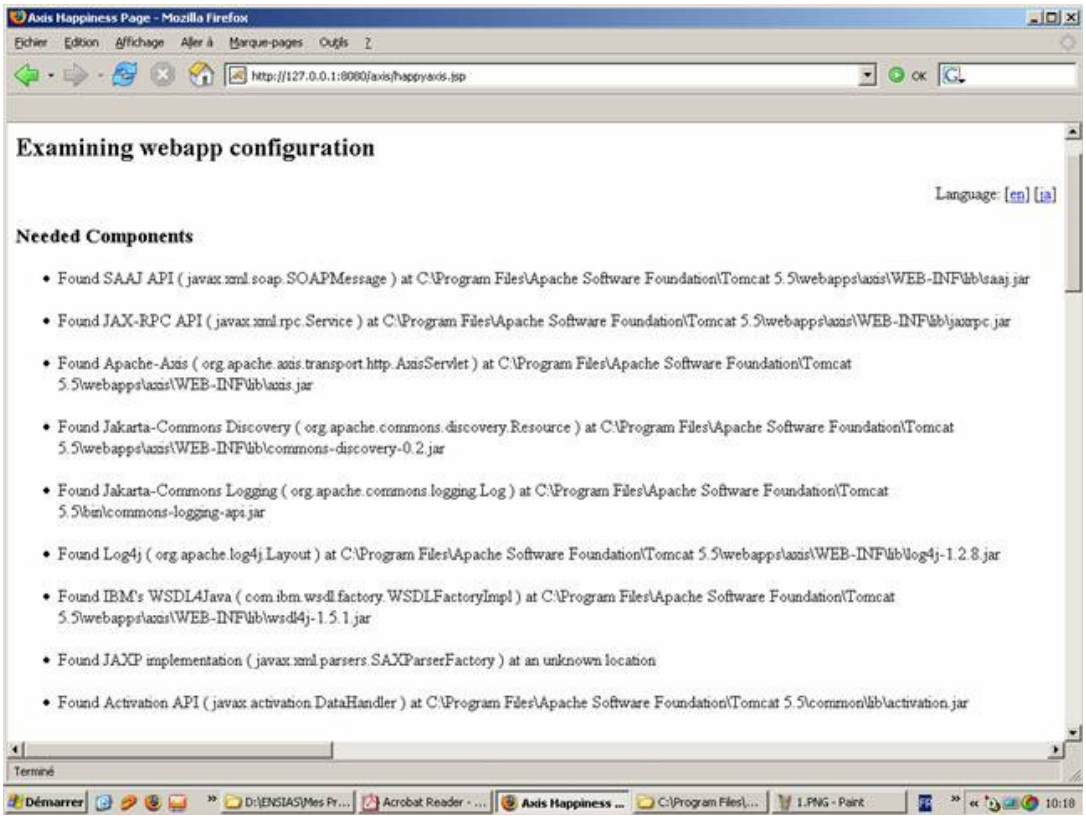
Démarrage de Axis

Pour installer Axis (nous utilisons la version 1.4), nous l'avons tout d'abord téléchargé sur http://ws.apache.org/axis, décompressé le fichier «axis-bin-1_4.zip», puis copié le dossier «axis-1_4\webapps\axis» dans «dossierInstallation\Tomcat 5.5\webapps\». De cette façon, Axis devient une application web déployée sous Tomcat.
Il nous manquait juste la librairie «activation.jar» et d'autres optionnelles: «soap.jar» et «mail.jar», que nous avons téléchargé et copié dans «dossierInstallation\Tomcat 5.5\common\lib\».
Pour vérifier la bonne installation et éventuellement accéder aux services web existants sur « Axis », nous avons redémarrer Tomcat et taper dans un navigateur : http://localhost:8080/axis

Ensuite, pour vérifier, il suffit de cliquer sur « Validate » puis «List»:



La validation nous permet de tester l'existence des librairies requises par Axis:



Création du service web

La première étape consiste à définir la classe du service web qui retournera les informations liées au profil sélectionné sur le formulaire.

La définition de cette classe est la suivante :

Le web service (serviceweb.jws)

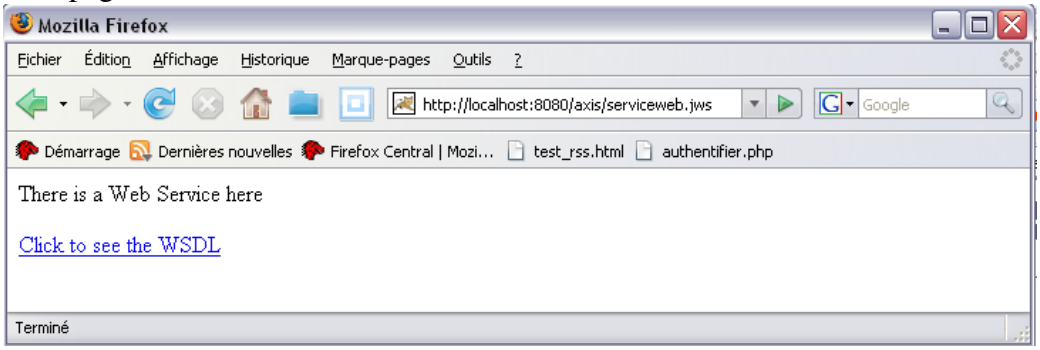
```
Ici, je présente un service web pour permettre de compter :
public class serviceweb {
        public int getsomme(int a, int b) {
        return a+b;
        }
}
```

Déploiement dans Axis

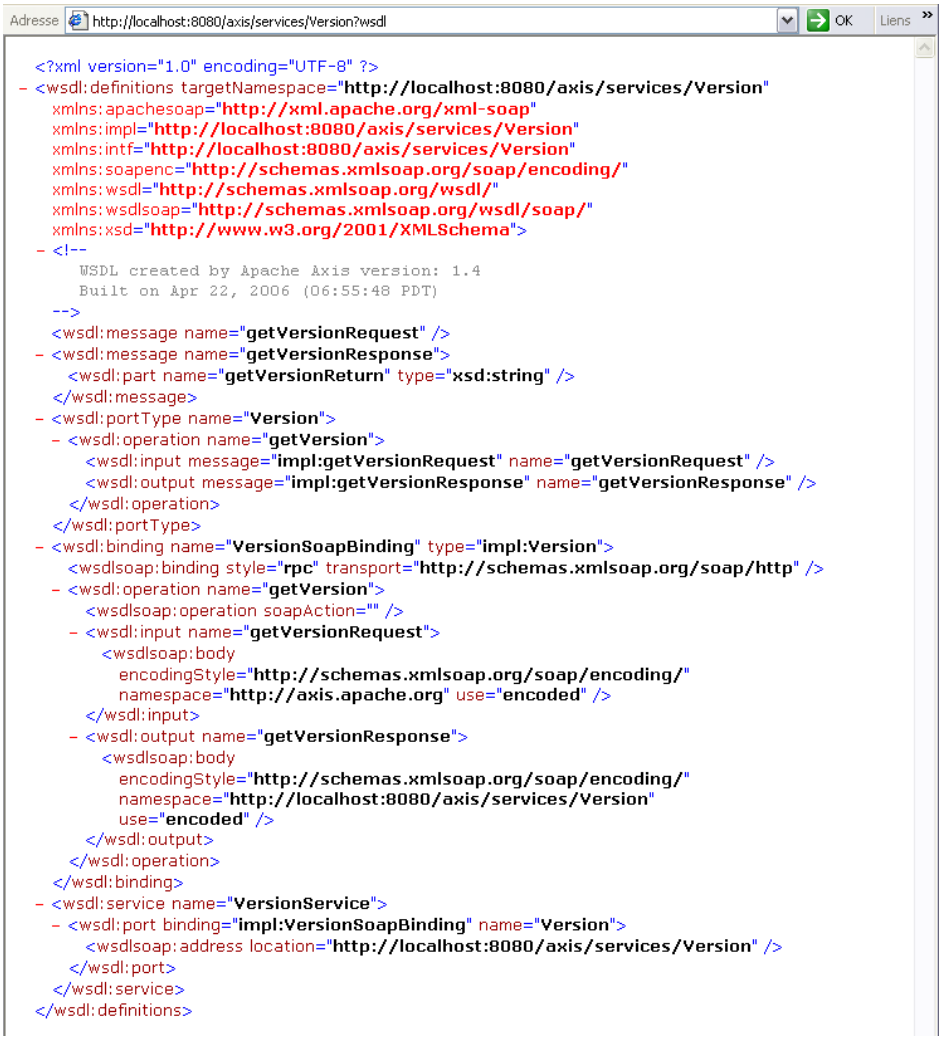
Le déploiement dans Axis est la deuxième étape qui consiste à déployer le service au sein d'un fournisseur de services web.
L'environnement d'exécution et de déploiement des services web que nous utilisons est, comme vous l'avez déjà constaté, l'outil «Axis ».

Deux façons sont possibles pour déployer un service web dans Axis.
1ère méthode de déploiement :
Le premier mode de déploiement sous « Axis » que nous avons mis en œuvre et du abandonner est le plus simple qui soit :
C'est le déploiement instantané par fichier « JWS ». Pour que le déploiement soit réalisable, il est nécessaire de compiler le source java.
Pour réaliser le déploiement, il suffit de copier le fichier «service.jws» dans le domaine applicatif de « Axis ». Le domaine d'application de « Axis » est : «dossierTomcat/webapps/axis», vous devez donc copier le fichier «service.jws» dans ce dossier.

Vous êtes désormais en mesure d'accéder à votre service à l'URL suivante : http://localhost:8080/axis/serviceweb.jws.
Pour constater que notre service a bien été déployé sur « Axis », il était nécessaire d'obtenir en retour la page « html » suivante :



Si on clique sur ce dernier lien, vous verrez la définition « WSDL » (générée automatiquement par « Axis ») de votre service web.




```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://localhost:8080/axis/services/Version"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/services/Version"
    xmlns:intf="http://localhost:8080/axis/services/Version"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <!--
      WSDL created by Apache Axis version: 1.4
      Built on Apr 22, 2006 (06:55:48 PDT)
    -->
    <wsdl:message name="getVersionRequest" />
  - <wsdl:message name="getVersionResponse">
      <wsdl:part name="getVersionReturn" type="xsd:string" />
    </wsdl:message>
  - <wsdl:portType name="Version">
    - <wsdl:operation name="getVersion">
        <wsdl:input message="impl:getVersionRequest" name="getVersionRequest" />
        <wsdl:output message="impl:getVersionResponse" name="getVersionResponse" />
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="VersionSoapBinding" type="impl:Version">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    - <wsdl:operation name="getVersion">
        <wsdlsoap:operation soapAction="" />
      - <wsdl:input name="getVersionRequest">
          <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://axis.apache.org" use="encoded" />
        </wsdl:input>
      - <wsdl:output name="getVersionResponse">
          <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://localhost:8080/axis/services/Version"
            use="encoded" />
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="VersionService">
    - <wsdl:port binding="impl:VersionSoapBinding" name="Version">
        <wsdlsoap:address location="http://localhost:8080/axis/services/Version" />
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Test du service web

La dernière étape a consisté de mettre en œuvre le service.
Pour exécuter une méthode de votre service et obtenir la réponse « SOAP » correspondante, vous tapez l'expression suivante dans votre navigateur :
Exemple d'URL :
http://localhost:8080/axis/serviceweb.jws?method=getparameters&X=value1&Y=value2

2ème méthode de déploiement :

Le premier mode de déploiement automatique présente les contraintes suivantes :
-Nécessité de disposer des sources des classes dont on veut définir le ou les services.
En effet, le mode de déploiement automatique d'Axis travaille sur la source des classes Java,
Impossibilité de décrire des particularités de déploiement (classes publiques et privées).
Ce sont pour ces raisons, que nous avons du réaliser un déploiement explicite.

Cela implique la définition d'un fichier particulier, appelé descripteur de déploiement du service web. Ce fichier porte l'extension « wsdd » pour «Web Service Deployment Descriptor».

La première étape consiste à compiler la classe que nous avons définie précédemment, il faut donc renommer le fichier " serviceweb.jws" en " serviceweb.java" et le compiler par commande "javac serviceweb.java", après nous avons copier la classe résultante dans le dossier des "jwsClasses".

Définition du descripteur de déploiement

Le descripteur de déploiement doit être placé dans le même dossier que le « .wsdl » définissant le service.
Appelons ce descripteur « deploy.wsdd ». Son contenu sera minimalement celui-ci :

Déploiement du web service

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<service name="serviceweb" style="java:RPC">
<parameter name="className" value="serviceweb"/>
<parameter name="allowedMethods" value="*"/>
</service>
</deployment>
```

Le descripteur de déploiement doit être maintenant pris en compte par le serveur « Axis » (i.e. le fournisseur de services) pour réaliser le déploiement du service. Pour ce faire, il convient d'utiliser l'utilitaire « AdminClient » du serveur « Axis ».
La ligne de commande est alors : «java org.apache.axis.client.AdminClient deploy.wsdd»

Si les librairies Axis ne sont pas spécifiées dans le classpath, vous devez le faire comme suit dans un fichier «.bat» de préférence:

```
set OLD_CLASSPATH=%CLASSPATH%
set CLASSPATH=%CLASSPATH%;%CATALINA_HOME%\common\lib\activation.jar
set CLASSPATH=%CLASSPATH%;%CATALINA_HOME%\common\lib\mail.jar
set CLASSPATH=%CLASSPATH%;%AXIS_HOME%
set CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\axis.jar
set CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\jaxrpc.jar
set CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\wsdl4j.jar
set CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\commons-discovery.jar
set CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\commons-logging.jar
set CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\saaj.jar
set CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\log4j-1.2.4.jar
set CLASSPATH=%CLASSPATH%;%CATALINA_HOME%\common\lib\xerces.jar
set CLASSPATH=%CLASSPATH%;%CATALINA_HOME%\common\lib\servlet-api.jar
set CLASSPATH=%CLASSPATH%;%CATALINA_HOME%\common\lib\naming-factory.jar
set CLASSPATH="%CLASSPATH%"
java - CLASSPATH="%CLASSPATH%" org.apache.axis.client.AdminClient deploy.wsdd
set CLASSPATH=%OLD_CLASSPATH%
```

Lors de l'étape précédente, demandé au serveur « Axis » d'être en mesure de traiter toutes les requêtes « SOAP » correspondant à notre service « serviceweb». Cela signifie donc qu'à la réception d'une requête «HTTP-SOAP », le serveur pourra appliquer la méthode spécifiée dans la requête à une instance de la classe correspondant à notre service (en lui passant, le cas échéant des valeurs).
Vous êtes désormais en mesure d'accéder à votre service à l'URL suivante : http://localhost:8080/axis/services/serviceweb.
Le nom « serviceweb» correspond au nom du service que nous avons indiqué dans le descripteur de déploiement.
Nous avons constaté que notre service a bien été déployé sur « Axis » en ayant en retour la page « html » suivante :

Consommer le web service en Java

Jusqu'à ce stade, nous avons toujours utilisé le navigateur Web pour invoquer les services et visualiser (sous le format SOAP) les résultats retournés. Nous allons étudier maintenant, un autre mode de communication avec les services plus adapté à leur mise en œuvre, et surtout à l'intégration avec d'autres applications.
Ce deuxième modèle consiste à la mise en œuvre des Services Web depuis le langage Java en générant automatiquement la définition du Proxy côté client.
Cela sous-entend la création de toutes les interfaces et classes nécessaires à la mise en œuvre du service depuis Java. Ce modèle est caractérisé par:
· Utilisation du « stub » côté client et du « skeleton » côté serveur
· Utilisation des interfaces représentant les types des objets Java à manipuler

Création du fichier de description du web service « wsdl »

Pour réaliser cette tache, nous utiliserons le service vue précédemment, et utiliser le menu du navigateur afin d'enregistrer le wsdl en le renommer serviceweb.wsdl dans le dossier «Tomcat 5.5\webapps\axis\WEB-INF».

Utilisation du générateur « WSDL2Java »

L'outil « Axis » qui permet la génération des définitions Java côté client et côté serveur, s'appelle «org.apache.axis.wsdl.WSDL2Java ».
L'invocation depuis une fenêtre DOS, est la commande:
«java org.apache.axis.wsdl.WSDL2Java serviceweb.wsdl».
L'outil « WSDL2Java » génère l'ensemble des définitions dans un sous-dossier correspondant au nom «targetNamespace="http://127.0.0.1:8080/axis/services/serviceweb"» du descripteur « wsdl ». En effet, les espaces de noms sont mappés en packages Java.

Définition du client Java

Vous devez ensuite définir le client Java qui met en œuvre les classes générées lors de l'étape précédente. Pour cela, votre client ressemblera à ceci :

```
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import _1._0._0._127.axis.sommer_jws.*;
public class ServicewebClient {
public static void main(String[] args) {
// Création du service depuis le endpoint
// ServicewebService correspond au nom du service dans le fichier #wsdl#
// C'est la balise : wsdl:service name="servicewebService"
ServicewebService service = new ServicewebServiceLocator();
try {
// Utilisation du service pour obtenir un stub qui implemente le SDI
// (Service Definition Type ; i.e. PortType).
// Pour le typage, c'est la balise : wsdl:portType name="sommer"
// Pour le getsommer(), le sommer correspond à la balise :
// wsdl:port binding="impl:sommerSoapBinding" name="sommer"
Serviceweb port = service.getparameter();
int s;
try {
// Mise en oeuvre du service par application directe des méthodes
s = port.getparameter(“X”, “Y”);
System.out.println("X”+”Y”+ s);
} catch (RemoteException e1) {
e1.printStackTrace();
}
} catch (ServiceException e) {
e.printStackTrace();
}}
}
```

Sous Eclipse vous obtiendrez le résultat suivant (dans notre cas, nous avons utilisé «http://127.0.0.1» au lieu de «http://localhost», c'est pourquoi nous obtenons le chemin généré «_1._0._0.127#») :

V.3 Mise en place d'une base de données XIndice d'Apache

Cette base de données XML native est une des plus connues. Open Source, écrite en langage Java, cette base gère le stockage et la restitution de documents XML.

Pour installer XIndice, il nous a fallu télécharger xml-xindice_1.0.zip se trouvant sur le site http://xml.apache.org/xindice/ à la rubrique « Download », et le décompresser.

Nous avons eu des difficultés à mettre en place le module XIndice, nous avons du passer par un autre intermédiaire qui l'installation du kit JWSDP se trouvant à cette adresse :
http://java.sun.com/webservices/webservicespack.html
Nous avons installé ce kit de développement pour pouvoir utiliser XIndice.
(Utilisation d'une j2sdk-1_3_1_19-windows-i586.exe qui correspond à une jdk 1.3).

Pour lancer l'exécution de la base de données XIndice, vous pouvez utiliser l'icône du menu du kit JWSDP Start XIndice.
Vous pouvez également exécuter Xlndice en lançant la commande xindice-start disponible dans le sous-répertoire *bin* de la distribution du kit JWSDP.

Vous devez également lancer Tomcat pour pouvoir utiliser la base Xlndice.

Dans la terminologie Xlndice, le terme « collection » identifie une base de données hébergée par le serveur de bases de données Xlndice. Chaque collection peut comporter des sous-collections et ainsi constituer une hiérarchie de collections. La collection de haut niveau db est une collection propre à XIndice.
Une collection comprend un ensemble de documents XML où chaque document contient plusieurs éléments XML. Pour récupérer un élément ou un ensemble d'éléments, on applique une requête exprimée avec le langage XPath.
Nous avons installé une interface graphique d'administration de la base XIndice.
XlndiceBrowser.
Pour exécuter cette application, lancez la commande XIndiceBrowser.bat depuis Windows.
Cet utilitaire va se connecter sur la base Xlndice précédemment lancée et délivrée avec le kit JWSDP.
La première colonne affiche la liste des collections existantes (vous ne verrez pas la collection db qui est la collection de plus haut niveau). Le nom *system* identifie un répertoire qui contient d'autres collections créées par XIndice pour la gestion des deux autres collections existantes authinfo et uddi. Ces deux collections correspondent respectivement aux utilisateurs existant pour l'utilisation d'UDDI et aux informations stockées par le référentiel UDDI.
En sélectionnant une collection, la deuxième colonne affiche la liste de documents XML contenus dans cette collection. Enfin, en sélectionnant l'un des documents, on peut accéder aux informations contenues dans ce document (sous divers formats de visualisation).

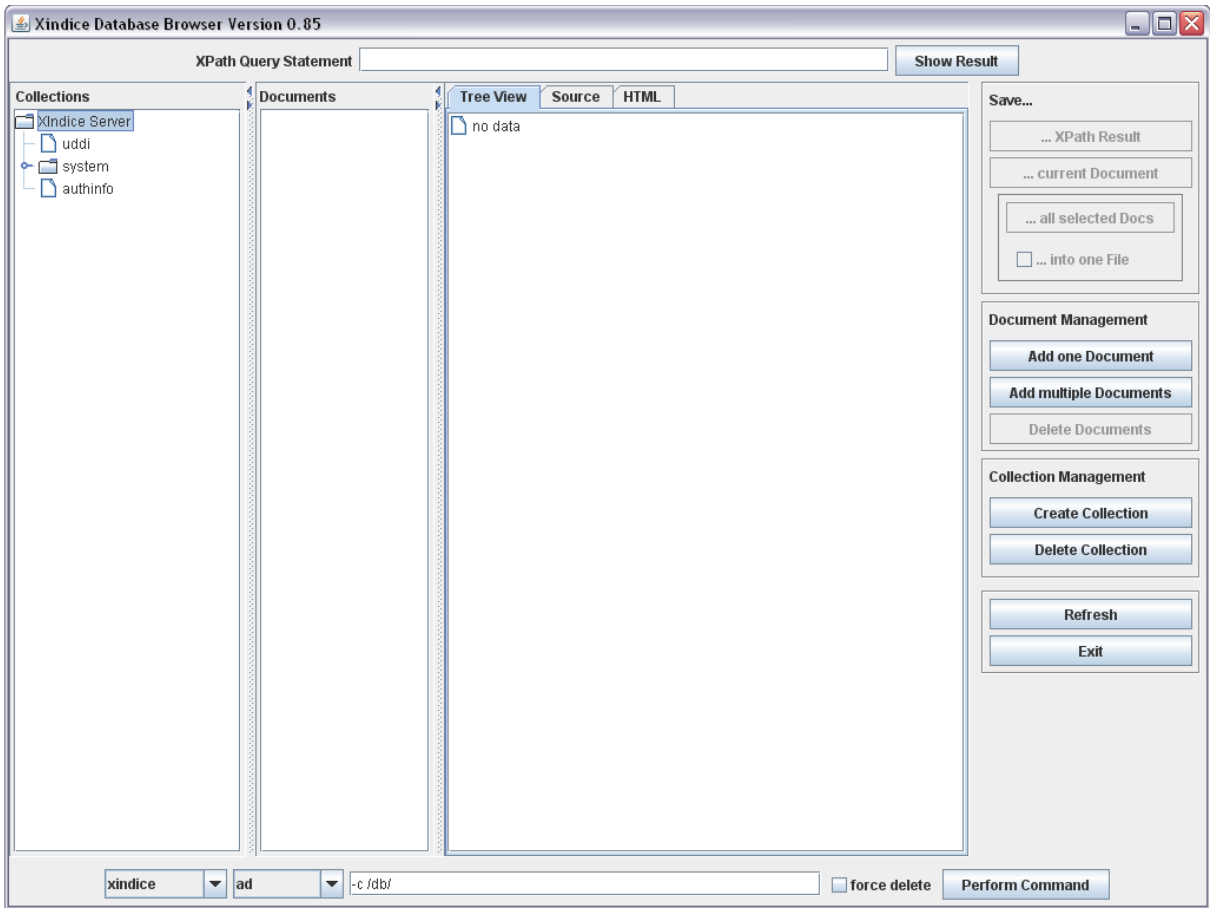


Figure 6 : **Interface d'administration de la base XIndice**

Pour simuler l'existence des différents CV, une collection va être créée puis on va y ajouter plusieurs données. Pour cela, nous avons sélectionné « Create Collection » dans la liste des boutons disponibles à droite de l'interface. Nous avons saisie les noms de la nouvelle Collection « CV ».

Figure 7 : **Création d'une nouvelle collection**

Il est essentiel lors de la création de la collection de vérifier que le nœud principal XIndice Server est sélectionné dans la colonne des collections.

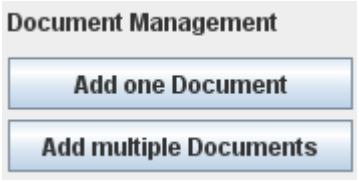
Ajoutons à présent un document XML à la nouvelle collection. Ce document contiendra la liste des CVs.

Ainsi, un document XML respectant ce format aura l'aspect suivant :

```
<CVS>
        <CV id="CV0000001">
            <PersonName>
                <GivenName>Francois</GivenName>
                <FamilyName>Parlant</FamilyName>
                <Telephone>0169042567</Telephone>
                <Mail>fparlant@laposte.net</Mail>
            </PersonName>
            <Service>DGSI</Service>
            <Qualifications>
                <Languages>
                    <Language>Anglais</Language>
                </Languages>
                <Competencies>
                  <info>
                        <infolang>C++</infolang>
                        <infolang>Java</infolang>
                        <database>DB2</database>
                        <database>Oracle</database>
                        <analyse>UML</analyse>
                  </info>
                </Competencies>
            </Qualifications>
            <Statut>Libre</Statut>
        </CV>
```

;

Ce fichier CVs.xml contient la description de plusieurs CVs. Ce document XML va être ajouté dans la collection. Sélectionnez « Add one Document » dans la liste des boutons de droit puis, à l'aide de la boîte de dialogue, sélectionnez le fichier *CVs.xml*. Pour terminer, confirmez ce choix.



Pour rechercher et tester la récupération d'un élément dans le document XML, nous avons utilisé « XPath Query Statement » pour écrire nos requêtes et « Show Result » pour l'exécuter.

Figure 8 : **Exécution d'une requête exprimée avec XPath**

Par exemple, pour récupérer la liste de tous les CVs, on a exécuté /CVS/CV en sélectionnant la collection CV.

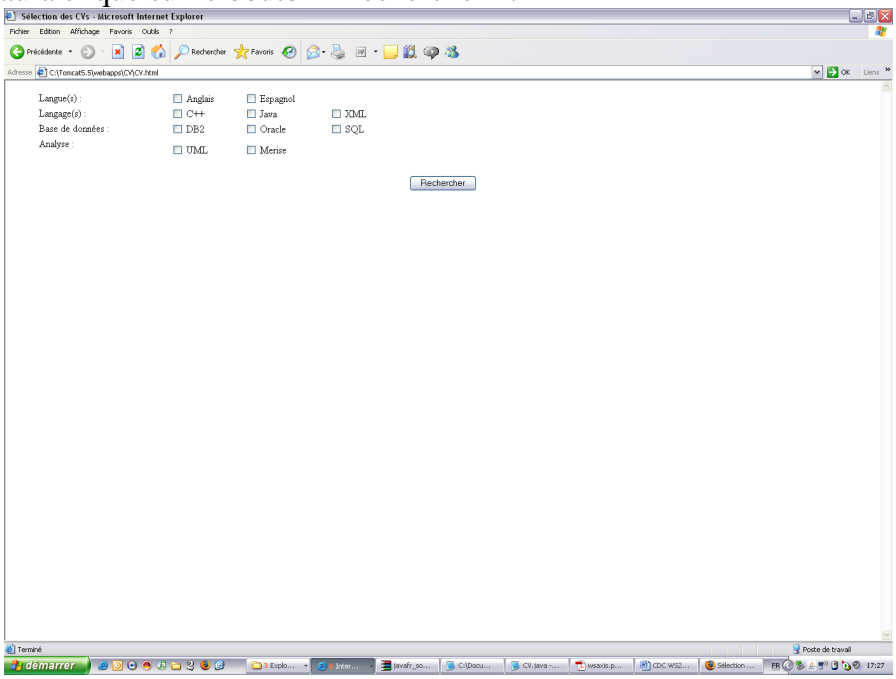
Le résultat de la requête est affiché au niveau de la colonne des éléments. On constate que le résultat est exprimé sous forme d'une liste de documents XML ou chaque document est un élément résultat de la requête. L'exemple ci-dessous montre un des résultats :

```
<?xml version="1.0"?>
<CV id="CV0000001" xmlns:src="http://xml.apache.org/xindice/Query" src:col="/db/CV"
src:key="CV.xml">
<PersonName>
        <GivenName>Francois</GivenName>
        <FamilyName>Parlant</FamilyName>
        <Telephone>0169042567</Telephone>
        <Mail>fparlant@laposte.net</Mail>
</PersonName>
<Service>DGSI</Service>
<Qualifications>
        <Languages>
        <Language>Anglais</Language>
        </Languages>
        <Competencies>
         <info>
                <infolang>C++</infolang>
                <infolang>Java</infolang>
                <database>DB2</database>
                <database>Oracle</database>
                <analyse>UML</analyse>
        </info>
        </Competencies>
</Qualifications>
<Statut>Libre</Statut>
</CV>
…
```

Dans la suite de l'implémentation, nous avons utilisé la base XIndice depuis une application Java, ou nous avons mis en place une connexion à la base pour pouvoir au niveau de l'application requêté nos fichiers se trouvant dans la base XIndice.

V.4 Interfaces Clientes

Pour permettre de soumettre les paramètres au service web et comme stipuler dans le cahier des charges, la mise en place d'une interface de type formulaire est utile.
Nous avons opté pour les boites à cocher pour la sélection des différents paramètres.
Les informations liées à la saisie seront envoyées au niveau du service via HTTP dès lors que l'utilisateur aura cliqué sur le bouton «Rechercher ».



L'interface liée à celle-ci, correspond à l'interface de récupération des informations sur la personne qui répond aux exigences.
A la date de rendu du rapport, nous n'avons pas encore eu le temps de développer cette interface, cette partie est encore en étude et en développement.

Utilisation des différents liens internet suivant :

· http://www.developpez.net/forums/
· http://dico.developpez.com
· http://ws.apache.org/axis/
· http://xml.apache.org/xindice/

Difficultés techniques rencontrées

Nous avons mis beaucoup de temps à pouvoir obtenir un serveur d'application intégrant les deux composants AXIS et la base XIndice.
Après avoir administré les composants, la seconde difficulté a été de reformuler le puzzle afin de lier les différents composants vu séparément.
Tout au long de notre projet, nous avons réalisé des exemples de services web avec différents moyens de déploiement ce qui nous a beaucoup ralenti dans nos travaux.
De plus, nous avions axé notre partie tout d'abord sur AXIS que nous avons abandonnée pour ensuite la retrouver.

VI. Conclusion

V.1 Difficultés rencontrées

Tout au long du projet, nous avons eu une grande difficulté à obtenir un serveur d'application qui répondait aux attentes d'un service Web. Nous avions durant l'implémentation commencé à travailler sur une partie appelé « Axis » auquel nous avons du abandonner à la suite d'une réunion avec M. BOCCON-GIBOD et M. GODEFROY.

Après avoir pris en compte ce que M. GODEFROY nous a indiqué, nous avons axé nos recherches sur la servlet que nous avons développée.

Puis lors d'une rencontre avec M. RIZK, il nous a été informé que les entreprises utilisaient systématique le module Axis car celui-ci permettait de se débarrasser des contraintes de création du fichier WSDL.

Donc nous nous sommes remis à travailler sur la partie Axis, cette épisode nous a couter un temps énorme car le développement n'est pas le même et les procédés non plus.

V.2 Conclusion

Malgré cela, la réalisation d'un projet à deux nous a permis de compléter mutuellement nos connaissances afin de fournir un travail efficace.

L'élaboration de tâches communes a été un élément moteur pour mener à bien ce projet. Ainsi nous avons mis nos idées et nos différentes expériences en commun pour mieux organiser le projet, et résoudre les problèmes rencontrés au cours de la programmation.

Sur le plan de la programmation, nous avons acquis des connaissances sur le développement client / serveur mais surtout sur l'architecture qui caractérise un service web.

Sur le plan personnel, nous avons maintenant acquis la mise en place d'un service web en utilisant un outil (Axis) qui a fait ses preuves dans le monde de l'entreprise.

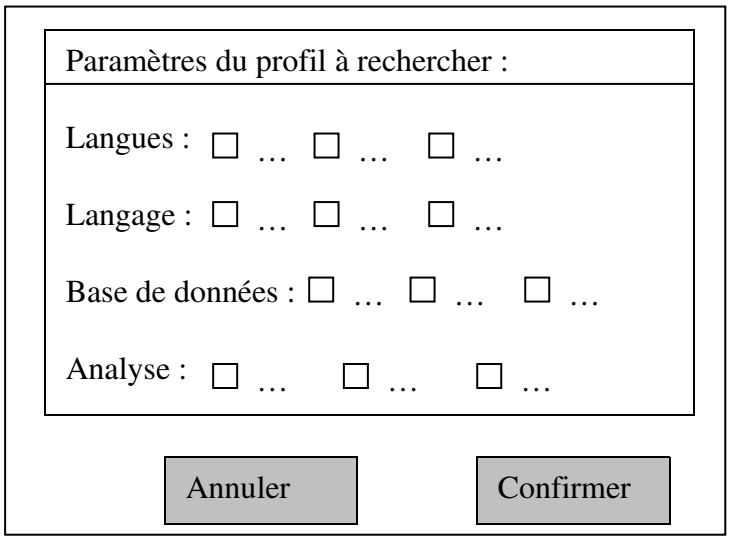
Pour conclure sur ce projet, nous pouvons dire que nous sommes fiers d'avoir compris comment fonctionne un service web et comment le mettre en place.

Toutefois avec plus de temps il aurait été possible d'apporter des améliorations au projet en augmentant les recherches sur les compétences des individus.

V.3 Résultats obtenus

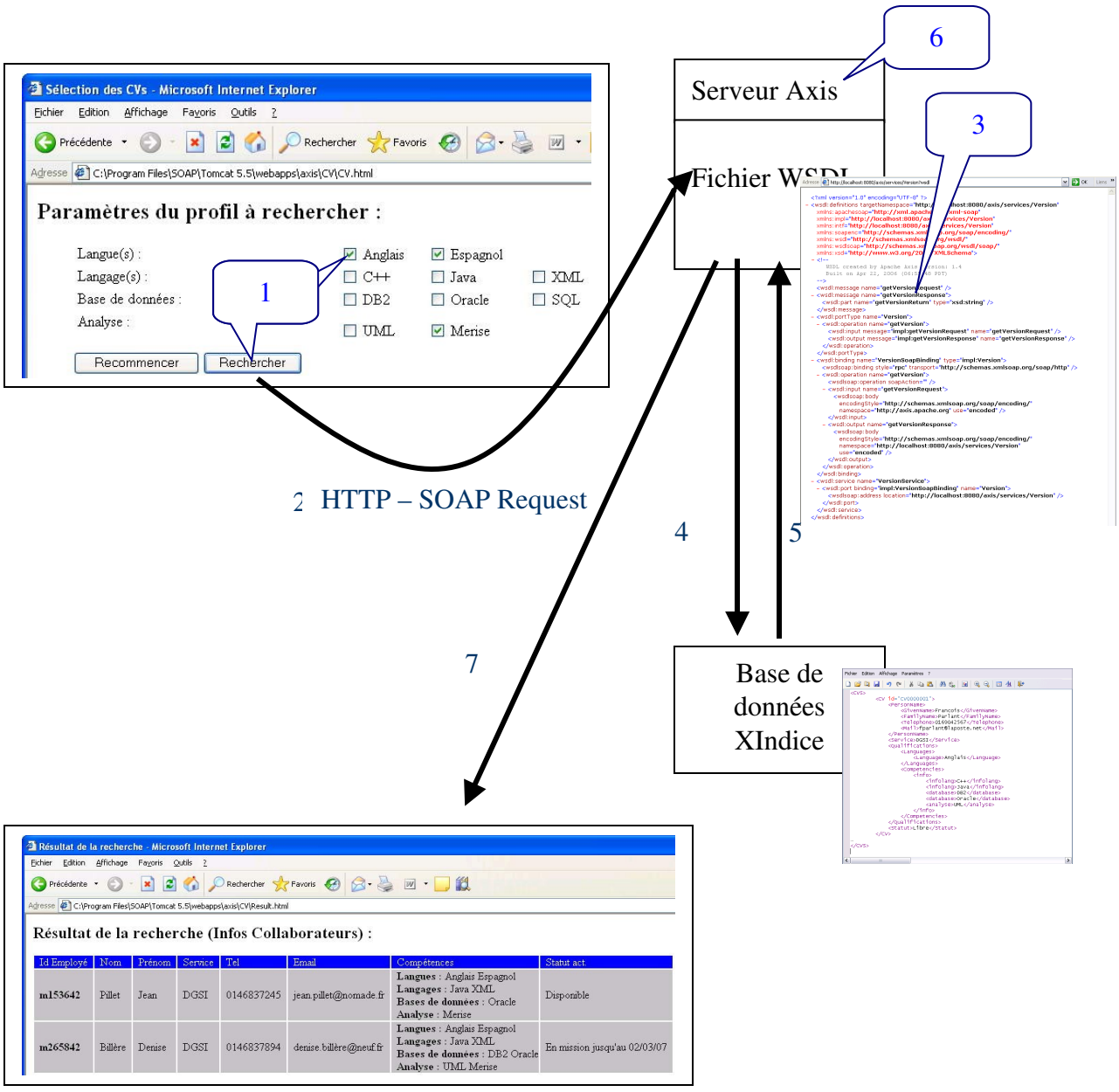
- **Interface d'entrée du système** :

Lors de la réalisation du cahier des charges, et avec l'accord du client, nous avions déterminé que l'interface de recherche des profils serait de la forme :



Voici donc, à l'issu de notre projet, l'interface de recherche que nous mettons à la disposition de notre client :

- **Fonctionnement du système** :



Les étapes du fonctionnement du système :

Etape 1 : Saisie et soumission du formulaire
Etape 2 : Passage des paramètres via HTTP-SOAP au niveau du serveur AXIS
Etape 3 : Recherche et appel les méthodes du service web via le fichier WSDL
Etape 4 : La méthode appelée va réaliser une connexion puis une requête au niveau de la base de données XML Native «XIndice». (Requête sous XPath)
Etape 5 : Réponse de la base de données XIndice
Etape 6 : Traitement pour l'obtention d'un résultat
Etape 7 : Résultat de la recherche

- **Interface de sortie du système** :

Lors de la réalisation du cahier des charges, et avec l'accord du client, nous avions déterminé que l'interface de résultats des recherches serait de la forme :

Résultat de la recherche (Infos Collaborateurs) :

Réponse de la requête							
Id Employé	Nom	Prénom	Service	Tel	Email	Compétences	Statut act.

Lors de l'exécution de la recherche précédente, le client visualise les résultats de sa recherche, à travers l'interface suivante :