heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Présentation personnelle

Web Services
Concepts, composants et mise en pratique

Reynald Borer
classe IL2007

16 janvier 2007

Table des matières

1 Introduction

Services Web, ou *Web Services* en anglais. Sous ce terme en vogue, lancé vers l'an 2000, se cache en fait une réponse à un besoin toujours plus fort ressenti sur *Internet*, à savoir l'échange d'information. Afin de comprendre les enjeux de cet ensemble de technologies, il est important de comprendre en premier lieu la problématique à laquelle ce terme est censé répondre.

Le réseau *Internet* connaît un essort toujours plus important depuis sa création car ses possibilités semblent infinies. Chaque jour, une nouvelle utilisation innovante de ce réseau mondial d'ordinateurs est découverte.

Son succès provient du fait qu'il facilite grandement l'échange d'information à l'échelle mondiale. En effet, depuis un simple ordinateur connecté au réseau, on accède à une multitude de contenus, allant de livres aux possibilités d'e-banking et d'achats en ligne. Cependant, tout cela n'aurait pas été possible sans la définition de standards de communication, de protocoles réseaux communs et de dialectes d'échange d'information.

Parallèlement à ces échanges d'information, de plus en plus d'entreprises fournissent des services, gratuits ou payants, par l'intermédiaire d'*Internet*. Il est donc important pour ces entreprises que leurs services soient le plus utilisé possible, et donc qu'ils soient le plus indépendant possible de la plateforme cliente.

C'est dans cette optique d'échange de l'information pure que le terme **services web** est né. Censé répondre à cette problématique en se débarrassant de toute mise en forme (ce que le *HTML* ne permet pas), les **services web** regroupent un ensemble de technologies et de normes créées à cette fin. Grâce à cette technologie, il devient désormais possible d'accéder à l'information brute, puis de la traiter et de l'utiliser comme bon nous semble.

Dans la suite de ce document nous allons en premier lieu analyser les concepts fondamentaux inhérents aux **services web**. Puis nous regarderons plus en détails les composants de base généralement utilisés pour obtenir des **services web** ainsi que les extensions et autres normes touchant ce domaine.

Viendra ensuite une partie dédiée aux langages de programmation `Java` et `Python` dans laquelle nous découvrirons quelques exemples simples de mise en œuvre des **services web** dans ces deux langages. Dans la suite du document nous verrons brièvement deux autres technologies similaires aux **services web**, avec leurs avantages et inconvénients respectifs. Puis nous terminerons par une conclusion sur le sujet en tentant de donner un avis critique et objectif sur cette technologie.

2 Concepts des services web

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.* (citation de [6])

Traduit simplement, cette citation indique qu'un **service web** est un logiciel conçu pour permettre des interactions entre machines à travers le réseau. Ce que la citation ne précise pas mais ce qui est quand même un point très important est le fait que cette technologie assure l'indépendance du langage de programmation. Il est donc possible qu'un **service web** soit un jour développé dans un certain langage, puis le jour suivant dans un autre sans que les utilisateurs ne puissent faire la différence. L'**interopérabilité** et l'**indépendence de la plateforme** sont deux des concepts fondamentaux des **services web**.

Un troisième concept concerne les échanges d'information. En effet, les services basés sur le contenu délivrent des pages web destinées à une « consommation humaine » , tandis que les **services web** délivrent du contenu à des ordinateurs. Il est bien évidemment possible d'extraire l'information depuis une page web, cependant chaque fois que la mise en page est modifiée il faut alors modifier l'algorithme d'extraction de l'information. C'est ce point là que les **services web** désirent résoudre.

À cela s'ajoute d'autres concepts peut-être moins important mais qui méritent d'être souligné. Citons entre autre la **réutilisabilité**, permettant d'exporter des fonctions d'une application en tant que **services web** pour les utiliser, à distance, dans une autre application. De même, la **flexibilité** offerte par un couplage faible entre les applications permet de faire évoluer un application cliente facilement sans que les **services web** associés n'aient besoin d'être modifiés.

3 Spécifications des composants

Une caractéristique qui a permis un grand succès de la technologie des **services web** est qu'elle est construite sur des technologies standards de l'industrie. En se basant sur des technologies déjà connues et maîtrisées, le temps d'adaptation d'une application s'en trouve grandement réduit.

Cependant, toutes ces technologies ont d'abord été mises en place par plusieurs grosses sociétés informatiques de renom avant que la standardisation ne soit reprise par le W3C (*World Wide Web Consortium*[1]) ainsi que par l'organisation OASIS (*Organization for the Advancement of Structured Information Standards*[2]). Il a donc fallu attendre un certain temps avant que des normes ne soient édictées, ce qui a donné lieu à plusieurs implémentations différentes pas forcément compatibles entre elles. Ce problème est à l'heure actuelle résolu, même si certaines implémentations logicielles ne prennent pas forcément en compte l'ensemble des normes.

3.1 Brique de base : XML

L'ensemble de ces composants et les **services web** en général s'appuient sur le langage **XML** (*eXtensible Markup Language*). Il s'agit d'un langage de balisage générique, sous forme textuelle, défini par le W3C. Il est reconnaissable par l'utilisation de balises imbriquées, définies par les caractères < et >. Son objectif initial est de faciliter l'échange automatisé de contenu entre systèmes d'information hétérogènes, notamment, sur *Internet*, car il permet de séparer entièrement l'information de son éventuelle présentation.

Les points clés de ce langage sont les suivants :

1. Un document *XML* est **entièrement transformable** dans un autre document *XML* avec une feuille de style *XSLT* (*Extended Stylesheet Language Transformations*) ;
2. La structure d'un document *XML* est **définissable et validable par un schéma**, soit par l'utilisation d'une *DTD* (*Document Type Definition*) ou par un *XML Schema*, ce dernier pouvant être directement intégré au document à valider de part sa syntaxe *XML* ; notons cependant qu'il existe d'autres standards, moins connus, permettant de définir la structure d'un document.

Pour en savoir plus au sujet de *XML* je vous invite à consulter le site web [2].

3.1.1 Schéma XML

Les **services web** se basent en grande partie sur **XML Schema** afin de pouvoir représenter tous les types de paramètres et de valeurs de retour des méthodes distantes.

[1] http://www.w3.org/
[2] http://www.oasis-open.org/

XML Schema est un langage de description de format de document *XML* permettant de définir la structure d'un document *XML*. Un **schéma XML** est lui-même un fichier *XML*.

La connaissance de la structure d'un document *XML* permet notamment de vérifier la validité de ce document. Grâce à un tel document il est donc possible de définir l'imbrication des balises *XML*, lesquelles sont obligatoires ou optionnelles, leur ordre, leur nombre d'occurrences, et il est aussi possible de définir des types complexes (en opposé aux types simples que sont les entiers, les chaînes de caractères, etc.). Un type complexe est composé d'un ou plusieurs types simples, que l'on peut contraindre.

Pour en savoir plus sur les **Schémas XML**, consultez les sites web [3] et [4].

3.2 Communication : SOAP

SOAP (à l'origine l'acronyme de *Simple Object Access Protocol*, devenu par la suite *Services Oriented Architecture Protocol*) est un protocole d'échange de messages au format *XML* se basant sur d'autres protocoles de communication (tels *HTTP*, le plus utilisé, mais aussi *SMTP*, *FTP*, etc.). La version actuelle de la spécification de ce protocole est la 1.2, recommendation édictée le 24 juin 2003, disponible sur [7].

Les messages **SOAP** sont le plus fréquemment utilisés pour permettre la communication entre un objet fournisseur de services et un objet client afin que l'objet client puisse appeler des méthodes distantes sur l'objet fournisseur. Ce principe est appelé invocation de méthodes distantes (*Remote Procedure Call*). Cependant ce n'est pas sa seule utilité, car ce protocole est de plus en plus utilisé dans des architectures orientées services (*Services Oriented Architectures*).

3.2.1 Format d'un message

Le format d'un message **SOAP** est présenté dans le schéma ci-dessous (figure 1).

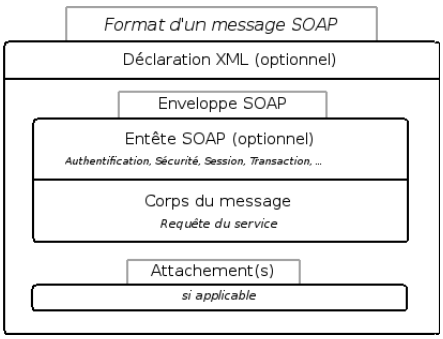

Fig. 1 – Format d'un message SOAP

Le format d'un message est défini de manière modulaire et contient plusieurs niveaux, grâce au format *XML*. La structure se présente comme suit :

1. **Déclaration XML** : cette déclaration est optionnelle et implicite, et permet d'indiquer qu'il s'agit d'un document *XML*. Seule cette déclaration est permise, aucun autre code *XML* ne doit être introduit. Sa forme est généralement la suivante :
   `<?xml version="1.0" encoding="UTF-8" standalone="no"?>`
2. **Message SOAP** : vient ensuite le message **SOAP**, composé d'une enveloppe et de zéro ou plusieurs attachements (par exemple si une image est jointe à la requête). L'enveloppe contient les informations nécessaires à l'acheminement et au traitement du message.

L'enveloppe elle-même se découpe en deux parties différentes, ayant des buts totalement différents :

- **l'entête** : cet élément, optionnel, contient certaines informations spécifiques aux logiciels utilisés comme, par exemple, l'authentification, une gestion de session ou de transaction ou tout autre détail important. Il est aussi possible d'ajouter à cet endroit une liste de nœuds de transit devant effectuer un traitement sur le message, le message pouvant ne pas aller directement de l'émetteur au récepteur. Si l'élément d'entête est présent il doit nécessairement être le premier élément enfant de l'enveloppe ;
- **le corps du message** : contient l'actuel message adressé au destinataire final. Cet élément contient un unique enfant, représentant soit la méthode distante à appeler (avec ses paramètres), soit un élément de type *fault* indiquant un message d'erreur.

Un exemple de requête **SOAP** simple peut être trouvé ci-dessous (figure 2).

```
- <soap:Envelope
    xmlns:s0="http://www.quisque.com/fr/chasses/crypto/"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  - <soap:Body>
      - <s0:Cipher>
          - <s0:text>VIVELAHEIGVVD</s0:text>
          - <s0:key>PPE</s0:key>
        </s0:Cipher>
    </soap:Body>
</soap:Envelope>
```

Fig. 2 – Exemple d'une requête au format SOAP

3.2.2 Possibilités d'utilisation

Basé sur les standards du web, **SOAP** vise à être un protocole aussi simple que ceux déjà édictés par le W3C sans pour autant perdre en fonctionnalités. Cependant, face à d'autres protocoles d'invocation de méthodes distantes tels que Corba et RMI, **SOAP** reste

fortement limité. Mais cette limitation sur les possibilités est compensée par sa simplicité, surtout au niveau du transport des messages. En effet, si **SOAP** utilise le protocole *HTTP* alors il suffit d'avoir un serveur web à disposition pour pouvoir l'utiliser. Et ce point est relativement important dans le monde de l'industrie, la plupart des sociétés de développement ayant déjà une grande expérience dans le déploiement d'applications web, sur lesquelles il suffit d'ajouter quelques mécanismes pour pouvoir faire des **services web**.

3.3 Description : WSDL

WSDL (*Web Services Description Language*) est un langage basé sur *XML* permettant de décrire les **services web** ainsi que leurs méthodes disponibles. Il existe à l'heure actuelle deux versions de cette norme. La version 1.1, qui date du 15 mars 2001, et la version 2.0, toujours en attente d'approbation du W3C. Les explications suivantes se basent sur la version 1.1.

Le format d'un document **WSDL** est représenté dans le schéma suivant (figure 3).

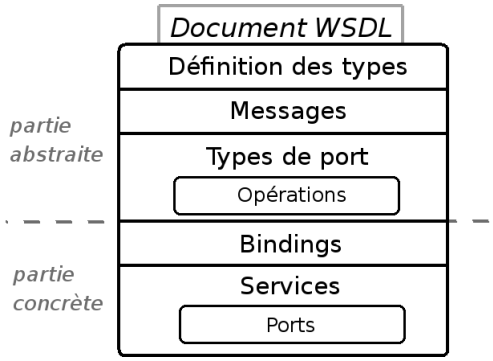

Fig. 3 – Format général d'un document WSDL

Un document **WSDL** se décompose en deux grandes parties, une partie abstraite et une partie concrète. Cette séparation, utilisée conjointement avec *XML*, permet de favoriser la réutilisation de la description et de séparer les préoccupations de conception de celles du transport.

La partie abstraite décrit un **service web** en terme des `messages` qu'il envoie et reçoit. Ces messages sont définis par un ensemble de couples (nom, `type`) définissant les paramètres d'un message (par l'utilisation d'un **schéma XML**). Les définitions de `types de port` représentent les interfaces disponibles, chaque instance étant une collection logique d'`opérations`. Une `opération` définit les `messages` entrants et sortants de l'`opération` ainsi que leur ordre. Un `message` est une unité de communication avec un **service web**. Il représente les données échangées dans une unique transmission logique.

Remarquons que la définition d'une `opération` permet de spécifier quatre types différents, en fonction des `messages` et de leurs ordres :

- requête seule : le **service web** reçoit un message ;
- requête - réponse : le **service web** reçoit un message et envoie une réponse en corrélation :
- sollicitation - réponse : le **service web** envoie un message et attend une réponse ;
- notification : le **service web** envoie un message.

Au niveau concret, un `binding` indique des détails de format et de transport pour un ou plusieurs `messages` et `opérations`. Un `service` quant à lui regroupe un ensemble de `types de port` qui implémentent une interface commune en spécifiant l'adresse à utiliser pour joindre le **service web**.

3.4 Découverte : UDDI

UDDI (*Universal Description, Discovery, and Integration*), est un protocole définissant un ensemble de méthodes invoquables avec **SOAP** afin de publier une liste de **services web** et d'effectuer des recherches de **services web** selon certains critères. La norme **UDDI** est édictée par OASIS.

Lancé en 1999 de l'initiative d'un certain nombre d'entreprises, cet annuaire permet de stocker à la fois des informations techniques et formelles telle que l'adresse pour accéder aux **services web** mais également des informations beaucoup plus contextuelles, comme le nom de la personne qui s'occupe de leur gestion, la description sommaire de leurs fonctionnalités ou encore le nom et la branche d'activité de l'entreprise dont ils dépendent. C'est pourquoi cet annuaire est consultable de différentes manières :

- **les pages blanches** listent les entreprises ainsi que leurs informations associées (nom de l'entreprise, coordonnées, description de l'entreprise, etc.) ;
- **les pages jaunes** recensent les services web de chacune des entreprises sous le standard **WSDL** ;
- **les pages vertes** fournissent des informations techniques précises sur les services fournis (descriptions de services et d'informations de liaison, processus métiers associés).

3.5 Vue globale

Le schéma 4 représente la mise en place de tous ces composants et comment ils interagissent entre eux. L'interaction la plus utilisée se déroule selon les étapes suivantes :

1. Le serveur (*service provider*) publie une liste de ses **services web** dans un document au format **WSDL** sur l'annuaire **UDDI** (*service broker*) ;
2. le client (*service requester*) désire appeler une méthode distante ; il effectue donc une requête **SOAP** vers l'annuaire **UDDI** afin d'obtenir un ou plusieurs **services web** correspondants à la méthode demandée ;

3. l'annuaire **UDDI** transmet au client un document **WSDL** avec la méthode, ses paramètres ainsi que l'adresse du serveur sur lequel effectuer l'appel ;
4. le client envoie alors sa requête au serveur, sous forme d'un message **SOAP** ; le serveur répond ensuite à la requête par l'intermédiaire d'un second message **SOAP**.

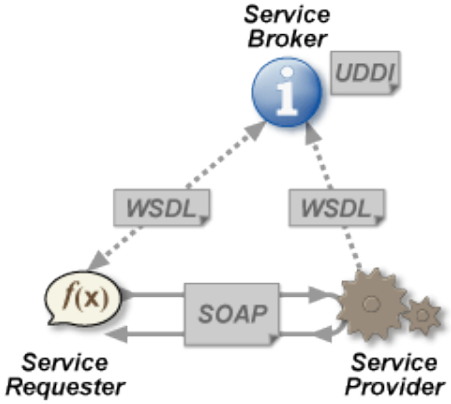

Fig. 4 – Interaction entre les différents composants d'un service web (*image tirée de Wikipédia*)

3.6 Autres spécifications

Ces composants représentent la base des **services web**, qui a volontairement été laissée modulaire afin de pouvoir facilement introduire de nouveaux composants et de nouvelles normes.

Il existe une multitude d'autres normes, certaines recommandées par le W3C. Les sujets couverts concernent entre autre la découverte des services, l'échange de messages, la gestion de leur intégrité ainsi que celle des transactions et la sécurité. Citons par exemple *WS-Addressing*, permettant de véhiculer des messages **SOAP** de manière bi-directionnelle mais aussi *WS-Security* qui couvre l'authentification, la gestion de l'intégrité et le cryptage des échanges. N'hésitez pas à effectuer une recherche sur *Internet* afin d'en apprendre un peu plus sur les autres normes et recommandations.

4 Exemples de code

4.1 Java

4.1.1 Librairie utilisée

Il existe plusieurs librairies pour le langage `Java`, certaines plus abouties que d'autres, certaines étant propriétaires et d'autres libres. Nous utiliserons ici la librairie **Axis2** en version 1.1.1 (disponible sur [16]). Cette librairie, issue des anciennes librairies *Apache Axis* et *Apache SOAP*, est certainement la librairie libre la plus aboutie. Elle supporte un ensemble conséquent de normes (**SOAP**, mais aussi **REST**[17] pour la communication, WS-Addressing, WS-Security et d'autres) grâce à une conception modulaire et flexible (il est possible d'ajouter dynamiquement des modules de traitement pour ajouter des fonctionnalités). Elle permet de plus des interactions asynchrones avec les services distants.

Les points clés de cette librairie sont les suivants :
- système de modules ajoutables afin de supporter d'autres normes ;
- flexibilité totale concernant la conversion des messages en code `Java` (accès direct au *XML*, invocation simple d'une méthode `Java`, redirection du message, etc.) ;
- flexibilité dans les schémas d'échange des messages (la librairie fournit les schémas *entrant seul* et *entrant-sortant*, mais il est possible d'en ajouter d'autres) ;
- supporte les interactions synchrones et asynchrones ;
- possibilité d'utiliser directement une classe `Java` comme **service web** ;
- multiples utilitaires afin de faciliter la création d'un **service web** rapidement.

Pour démontrer la facilité de mise en place d'un **service web** et son exploitation nous allons mettre en place un service de calculatrice en notation polonaise inversée invoquable à distance. Pour cela, le **service web** s'appuiera sur *Jakarta Tomcat* qui est un conteneur de servlet `Java` fournissant nativement un serveur web. Pour plus d'informations sur ce serveur vous pouvez consulter [18].

4.1.2 Partie serveur

Axis2 fournit plusieurs méthodes afin de créer un **service web**. La plus simple, que nous allons découvrir ici, ne nécessite aucune modification du code d'une classe afin de la transformer en **service web**. Cette technique est appelée **POJO** (*Plain Old Java Objects*). Le code de cette classe est présenté ci-dessous.

Remarque : il est important de noter que le constructeur de la classe n'est pas utilisé par **Axis2**. La documentation mentionne cependant d'autres méthodes permettant d'initialiser les données membres afin de simuler un constructeur.

```
01 : /** Classe Calculator - calculatrice en notation polonaise inversée
02 :  * Simple classe transformée, grâce à Axis2, en service web */
03 : package exemple;
04 : import java.util.Stack;
05 :
06 : public class Calculator {
07 :    protected Stack Pile = new Stack(); /** pile de nombres */
08 :
09 :    /** Ajout d'un nombre dans la pile */
10 :    public void push ( double n ) {
11 :       Pile.push ( new Double ( n ) );
12 :    }
13 :
14 :    /** Sortie d'un nombre de la pile */
15 :    public double pop () {
16 :       return ((Double) Pile.pop() ).doubleValue ();
17 :    }
18 :
19 :    /** Addition des deux nombres au sommet de la pile */
20 :    public double add ( ) {
21 :       double n = pop() + pop();
22 :       push ( n ); return n;
23 :    }
24 :
25 :    /** Soustraction des deux nombres au sommet de la pile */
26 :    public double sub ( ) {
27 :       double t = pop();
28 :       double n = pop() - t;
29 :       push ( n ); return n;
30 :    }
31 : }
```

Fig. 5 – Classe Java transformée en service web grâce à Axis2

La transformation d'une telle classe en **service web** nécessite les manipulations suivantes (consultez la documentation de **Axis2** pour plus de détails) :

1. Créer la structure de répertoire ci-contre.
2. Créer un document *XML*, nommé `services.xml` définissant la classe `Java` à utiliser pour le **service web** ainsi que son nom.
3. (optionnel) Générer le document **WSDL** avec l'utilitaire `java2wsdl`.
4. Créer une archive *jar* avec la structure de répertoire et la renommer avec une extension `.aar`.
5. Déployer l'archive avec l'interface d'administration de **Axis2**.

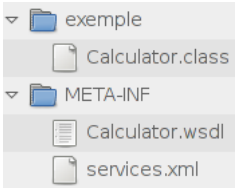

Fig. 6 – Structure de répertoire pour le service web

4.1.3 Partie client

La partie cliente est légèrement plus complexe car il faut utiliser certaines méthodes permettant de créer et de lire des messages **SOAP** en respectant le format voulu. Cependant, afin de faciliter les manipulations, la librairie **Axis2** vient avec un utilitaire, `wsdl2java`, permettant de générer à partir du document **WSDL** un ensemble de classes correspondantes ainsi qu'un script de compilation *Ant* (afin de faciliter la compilation de votre programme client). Un simple exemple est présenté ci-dessous et devrait être facilement compréhensible. Il a été obtenu en suivant la documentation de la librairie.

```
01 : /** Classe Client - Client simple pour notre service de calculatrice */
02 : package exemple;
03 : import exemple.CalculatorStub; /* généré avec wsdl2java */
04 :
05 : public class Client {
06 :    public static void main ( String args[] ) {
07 :       try {
08 :          /* adresse du service web */
09 :          CalculatorStub stub = new CalculatorStub (
10 :                "http ://localhost :8080/axis2/services/CalculatorService" );
11 :
12 :          /* ajout de deux nombres dans la calculatrice */
13 :          CalculatorStub.Push push = new CalculatorStub.Push ();
14 :          push.setN ( 1.0 );
15 :          stub.push ( push ); /* envoi */
16 :          push.setN ( 4.0 );
17 :          stub.push ( push ); /* envoi */
18 :
19 :          /* récupération du résultat de l'addition */
20 :          System.out.println("resultat du calcul : " + stub.add().get_return());
21 :
22 :       } catch ( Exception e ) {
23 :          System.err.println ( e );
24 :       }
25 :    }
26 : }
```

Fig. 7 – Application cliente pour notre service web

Notons qu'il existe plusieurs *formats* de représentation des messages **SOAP** au sein même de la librairie, avec pour chacun ses spécificités de création et d'accès. Tous ces formats sont gérés par l'utilitaire `wsdl2java`, consultez la documentation de la librairie afin de connaître les spécificités de chacun.

4.2 Python

4.2.1 Librairie utilisée

L'exemple suivant, en langage `Python`, utilise la librairie ZSI (*Zolera SOAP Infrastructure*), disponible sur [19]. Cette librairie, dont la version 2.0 utilisée ici est toujours en développement, fournit un ensemble de méthodes permettant d'utiliser les **services web**. Plus particulièrement, et comme avec la librairie *Axis* pour `Java`, elle fournit un utilitaire nommé `wsdl2py` permettant, à partir d'un document **WSDL**, de créer un ensemble de classes permettant de lier les types et opérations du **service web** avec le langage Python. Les fonctionnalités annoncées de cette librairie sont les suivantes :

1. Implémentation de la norme **SOAP** en version 1.1.
2. Permet de convertir les types utilisés dans les messages en types natifs `Python` (avec accesseurs pour les types complexes).
3. Supporte les messages **SOAP** avec attachements.
4. Ne fournit aucune implémentation du transport du message, cela étant laissé à d'autres modules spécialisés.

4.2.2 Client simple

L'exemple suivant représente un simple client pour un **service web** permettant d'effectuer du chiffrement avec le carré de Vigenère. Le document **WSDL** est disponible à l'adresse `http://www.quisque.com/fr/chasses/crypto/vigenere1.asmx?WSDL` et a été converti en classes python avec la commande suivante :

```
wsdl2py -complexType -url "http://www.quisque.com/fr/chasses/crypto/vigenere1.asmx?
WSDL"
```

La ligne `9` permet d'importer les modules `Python` créés avec l'utilitaire `wsdl2py`. Les lignes `14` et `15` créent une instance du proxy, permettant d'obtenir l'adresse du serveur sur lequel effectuer la requête, tandis que la ligne `21` crée une nouvelle instance de la méthode distante à invoquer. C'est grâce aux accesseurs utilisés aux lignes `22` et `23`, qui correspondent aux noms des paramètres définis dans le document **WSDL**, que nous donnons les paramètres à utiliser pour la méthode distante.

Finalement, la méthode distante est appelée et le résultat peut être consulté, ligne `29`, comme s'il s'agissait d'une variable `Python`.

Cet exemple, bien que très simpliste, met bien en évidence l'abstraction qui est faite quant aux divers composants des **services web**. En effet, il suffit d'avoir le document **WSDL** (et de le comprendre bien évidemment) pour pouvoir très facilement effectuer des appels de méthodes distantes, sans se soucier du format des messages ni du *XML*.

```
01 : #!/usr/bin/python
02 : # -*- coding : utf-8 -*-                                                    #
03 : #                                                                          #
04 : # vigenere.py                                                             #
05 : # Un simple exemple d'un appel à un web services en pyhon (le service web #
06 : # chiffre un texte avec le carrè de vigenère)                              #
07 :
08 : # importation des fichiers générés par WSDL2Py
09 : from Vigenere1_services import *
10 : import sys
11 :
12 : def main(args) :
13 :     # instance d'un proxy pour les requêtes
14 :     loc = Vigenere1Locator()
15 :     port = loc.getVigenere1Soap()
16 :
17 :     print '-- Chiffrement avec le carré de Vigenère par web services --'
18 :     print 'Chiffrement de', sys.argv[1], 'avec', sys.argv[2], 'comme cle'
19 :
20 :     # nouvelle requête de cryptage
21 :     cipher = CipherSoapIn()
22 :     cipher.set_element_text(args[1])
23 :     cipher.set_element_key(args[2])
24 :
25 :     # appel de la procédure distante
26 :     resp = port.Cipher(cipher)
27 :
28 :     # résultat
29 :     print 'Texte chiffré :', resp.CipherResult
30 :
31 : if __name__ == '__main__' :
32 :     if len(sys.argv) < 3 :
33 :         sys.exit('Usage : vigenere.py texte cle')
34 :     main(sys.argv)
```

Fig. 8 – Exemple de client pour un service web en Python

5 Autres technologies

Ce document s'est principalement focalisé sur l'utilisation des **services web** en tant que possibilité d'appels de procédures distantes. Dans ce cadre d'utilisation, il existe d'autres technologies, plus anciennes, et dont les choix techniques et possibilités sont différentes. Cette section va tenter de résumer deux technologies importantes en les comparant aux **services web**.

5.1 CORBA

CORBA, acronyme de *Common Object Request Broker Architecture*, est une norme définie par l'OMG (*Object Management Group*[3]) permettant à des composants logiciels développés dans différents langages et déployés sur différentes machines de communiquer. Ces composants sont généralement assemblés afin de construire des applications complètes.

La norme **CORBA** a été initiée en 1992 et en est désormais à la version 3.0. Elle adopte une approche essentiellement objet, chaque méthode étant décrite sous forme d'une interface en langage IDL (*Interface Description Language*, à comparer à **WSDL**). Il est ensuite possible d'effectuer une correspondance entre cette interface et un langage de programmation par l'utilisation d'un précompilateur.

CORBA utilise un ensemble de protocoles réseaux pour communiquer, appelés **ORB** (*Object Request Broker*). Les plus connus sont IIOP (*Internet InterORB Protocol*, protocole propre sur TCP/IP) et HTIOP (*HyperText InterORB Protocol*, qui utilise HTTP). Le schéma de la figure 9 résume le fonctionnement général du système, même s'il n'est pas complet. En effet, **CORBA** définit une multitude de composants permettant la répartition de charge, l'interception de messages afin de les modifier, et autres. Notons aussi que la norme fournit des services tels que l'authentification, la gestion des transactions et la sécurité nativement.

Cette norme a été édictée afin que les différents composants distribués d'une application collaborent avec *efficacité, fiabilité, transparence* et avec une capacité de *montée en charge* importante. Les principes de **CORBA** sont les suivants :

- séparation stricte de l'interface et de l'implémentation ;
- transparence de la localisation et de l'accès aux objets ;
- typage des objets par les interfaces ;
- héritage multiple d'interfaces.

5.1.1 Comparaison avec les services web

CORBA n'est pas une technologie récente, elle a donc eu le temps de faire ses preuves et d'être remaniée afin de s'adapter aux besoins des utilisateurs. Cette technologie a néan-

[3] http://www.omg.org/

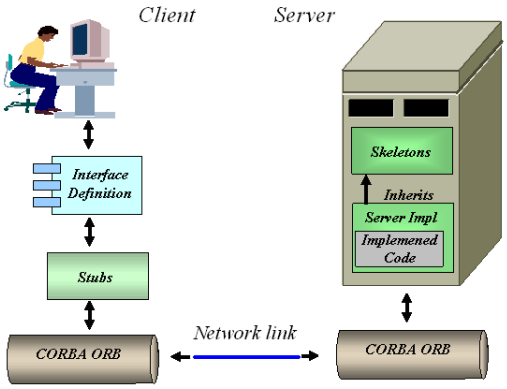

Fig. 9 – Schéma de fonctionnement général de CORBA (*image tirée de Wikipédia*)

moins une image très négative, car les buts n'ont jamais été clairement définis mais aussi car la norme semble être un amalgame de toutes les propositions faites, plutôt qu'une véritable étude sur ce qui est nécessaire et ce qui n'est pas cohérent. De plus, il subsiste des problèmes avec les implémentations de cette norme, aucune n'étant vraiment complète par rapport aux spécifications publiées. Cette technologie reste néanmoins grandement utilisée à l'heure actuelle dans le monde de l'industrie.

Les différences avec les **services web** sont les suivantes :

Fonctionnalités Au niveau fonctionnalités il semble clair que **CORBA** offre beaucoup plus de possibilités que les **services web**, ne serait-ce que parce que ces derniers sont plus récents. **CORBA** est clairement orienté sur l'invocation de méthodes distantes, qui, couplée avec les principes objets, permet un grand nombre d'opérations qu'il ne serait pas possible de faire facilement avec les **services web**. De plus, **CORBA** gère nativement les transactions et la sécurité.

Couplage client - serveur Les **services web** bénéficient d'un couplage faible, ce qui n'est pas le cas de **CORBA**. Le premier transmet des messages tandis que le second peut transmettre des objets. Il est donc possible d'appliquer une autre méthode sur l'objet reçu, tandis que ce n'est pas le cas du message.

Performance **CORBA** est nettement plus performant lors des transactions qu'il effectue car il dispose de son propre protocole de communication, optimisé, et qu'il n'utilise pas *XML*, qui est relativement lourd à traiter.

Sécurité Même si plusieurs normes existent concernant la sécurité des **services web**, peu d'entre elles disposent d'une implémentation logiciel répandue. Ce n'est pas le cas de **CORBA**, qui dispose d'un ensemble des fonctionnalités propre à la sécurité.

Mise en place Les **services web** sont relativement simple à mettre en place par rapport à **CORBA**, qui nécessite de définir au préalable correctement toutes les interfaces. Dans le cas des **services web**, il est nettement plus simple de les rajouter à une application existante. De plus, comme ils s'appuient généralement sur le protocole *HTTP* il n'y a pas besoin de reconfigurer un firewall d'entreprise pour permettre l'interaction avec des clients, pour autant que l'on dispose déjà d'un serveur web.

Au vu de ces diverses comparaisons **CORBA** semble être nettement plus abouti et évolué que les **services web**, tout en offrant une gamme de fonctionnalités beaucoup plus importantes. Cependant, **CORBA** n'est pas la solution à tout car sa mise en place est relativement lourde et demande un couplage fort entre clients et serveurs. Il convient donc de bien évaluer la solution à choisir entre les **services web** et **CORBA**. Si l'ensemble des composants est développé au sein du même groupe de développeurs, et que ceux-ci maîtrisent déjà la technologie **CORBA**, alors celle-ci peut être un bon choix. Mais si l'application doit fournir des services à un ensemble de clients inconnus, ou si l'on recherche la facilité de mise en place, alors il faudra s'orienter plutôt vers les **services web**.

5.2 RMI (spécifique à Java)

Java RMI (*Java Remote Method Invocation*) est une interface de programmation (API) pour le langage `Java` qui permet d'appeler des objets distants comme s'il s'agissait d'objets locaux. L'utilisation de cette API nécessite l'emploi d'un registre RMI sur la machine distante hébergeant ces objets que l'on désire appeler au niveau duquel ils ont été enregistrés.

Les connexions et les transferts de données dans **RMI** sont effectués par `Java` sur TCP/IP grâce à un protocole propriétaire (JRMP, *Java Remote Method Protocol*) sur le port 1099. À partir de `Java` 2 version 1.3, les communications entre client et serveur s'effectuent grâce au protocole RMI-IIOP (*Internet Inter-Orb Protocol*), un des protocole utilisé dans l'architecture **CORBA**. La transmission de données se fait à travers un système de couches afin de garantir une interopérabilité entre les programmes et les versions de `Java`, comme montré sur le schéma suivant (figure 10).

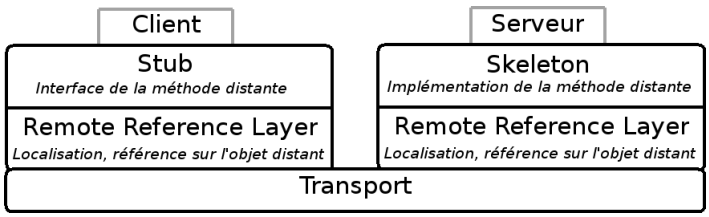

Fig. 10 – Transmission des données avec RMI

Cette technologie permet aussi aux divers acteurs de transférer directement leurs objets. Il est donc possible, pour une application cliente, de recevoir un objet *inconnu* du serveur puis de récupérer son interface afin d'interagir directement avec lui sans passer par le serveur. Un exemple de fonctionnement de **RMI** avec ce principe est présenté à la figure suivante :

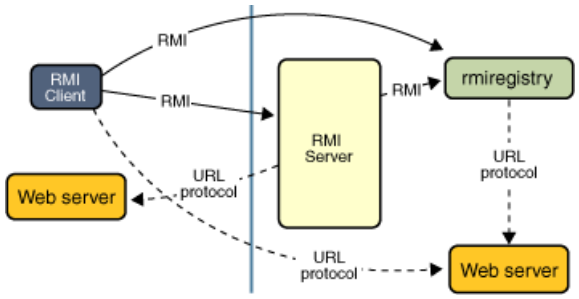

Fig. 11 – Schéma de fonctionnement général de RMI (*image tirée de* `http://java.sun.com/`)

Ce schéma illustre une application distribuée avec **RMI** utilisant le *registre RMI* afin d'obtenir une référence sur un objet distant. Le serveur associe un objet avec son nom dans le registre tandis que le client recherche un objet avec son nom et invoque ensuite ses méthodes. Le serveur et le client dialoguent aussi avec un serveur web afin de publier et obtenir les interfaces des objets.

5.2.1 Comparaison avec les services web

RMI étend encore plus le principe du langage `Java`, à savoir la possibilité pour le même code d'être exécuté n'importe où grâce à la machine virtuelle. Cette technologie apporte une dimension distribuée à travers le réseau, tout en gardant tous les avantages de `Java`. Il est donc possible d'utiliser toute la puissance de ce langage tout en effectuant de la programmation distribuée.

Fonctionnalités La technologie **RMI** offre certaines fonctionnalités qui sont très clairement orientées objets ce qui n'est pas le cas des **services web**. Elle permet par exemple de transférer directement les objets ainsi que leurs interfaces afin d'interagir avec eux en local. De plus, **RMI** gère en natif la sécurité et permet d'exploiter toutes les possibilités du langage `Java`. Cette technologie est donc fonctionellement plus avancée.

Couplage client - serveur Le couplage est relativement faible avec **RMI**, les clients pouvant recevoir des objets dont ils ne connaissent pas encore l'interface. Cependant, les **services web** offrent un couplage encore plus faible de par leur indépendance du langage.

Performance Comme le protocole réseau utilisé par **RMI** est spécifique à son fonctionnement les performances de cette technologie sont généralement meilleures. De plus, comme il est possible de rapatrier un objet distant afin d'exécuter certaines de ses méthodes en local cela donne un gain de performance non négligeable.

Sécurité Cette technologie gère en natif la sécurité en utilisant les mécanismes de sécurité mis à disposition par le langage `Java`.

Mise en place **RMI** est simple à mettre en place car il s'agit principalement de faire du `Java`. Une fois les interfaces des modules du serveur à partager définies il suffit d'effectuer l'implémentation côté serveur ainsi que côté client. Le code à ajouter afin de pouvoir utiliser cette technologie est relativement faible, et de plus il n'y a, à aucun moment, besoin de se soucier de la communication, celle-ci étant gérée de manière transparente.

Au vu de ces diverses comparaisons **RMI** semble être une technologie à envisager si l'ensemble des acteurs d'une application distribuée sont développés en `Java` et qu'il n'existe pas de firewall trop restrictif entre les diverses machines. Cependant, les **services web** offrent une plus grande souplesse quant aux langages utilisés par les divers acteurs, et sera une technologie vers laquelle il faudra se tourner dans le cas ou un autre langage que `Java` est utilisé. Notons de plus qu'il est relativement simple de faire des **services web** avec `Java` grâce à la librairie *Axis*, présentée à la section 4.1.

6 Conclusion

Au fil de ce document nous avons découvert les **services web**, le climat dans lequel cette technologie a été lancée ainsi que les divers problèmes qu'elle devait résoudre et les concepts fondamentaux de leur approche. Nous avons aussi eu un aperçu des briques de base composant un **service web**, des extensions possibles et des solutions logicielles mettant actuellement en pratique ces composants. Quelques petits exemples de code nous ont montré à quel point une telle technologie était simple à mettre en place, comparé aux technologies concurrentes telles que **CORBA** et **RMI**.

Les **services web** fournissent une solution élégante à la demande toujours plus croissante de partage d'informations et de services en mettant en place des normes permettant de ne se soucier ni du langage de programmation ni de la plateforme sur laquelle le **service web** tourne. Les quelques exemples pratiques ont montré qu'il existe désormais des librairies complètes et performantes pour la gestion de tels services.

Il est cependant difficile d'évaluer si les **services web** sont véritablement utilisés dans le monde de l'industrie. En effet, à part quelques petits exemples simples (vérification d'email, cryptage basique) je n'ai pas trouvé de véritable application accessible gratuitement avec **SOAP**. Même Google™, qui fournit un **service web** pour son moteur de recherche, a finalement décidé de se tourner vers la technologie *AJAX*. Néanmoins le nombre de petits services accessibles est relativement important (consultez http://www.xmethods.net/ pour vous en convaincre).

Les **services web** ont cependant le mérite d'avoir fait émerger un nouveau modèle d'interaction applicative, l'*architecture orientée services*. Cette architecture met à disposition des services (exécutés par des fournisseurs) pour lesquels le couplage avec le client est très faible. Elle est donc très efficace pour les problématiques que rencontrent les entreprises en termes de reutilisabilité, d'interopérabilité et de réduction de couplage entre les différents systèmes qui implémentent leurs systèmes d'information et les clients. Et ces problèmes s'inscrivent directement dans la lignée des **services web**. Certains analystes prévoient l'explosion de ce type d'architectures d'ici 2008, ou, d'après eux, plus de 75% des projets auront centré leurs logiciels sur une telle architecture.

Par l'intermédiaire de ce document j'espère vous avoir permis de découvrir ou de redécouvrir les **services web**, leurs principes et fonctions ainsi que leurs points forts et faibles. C'est une technologie qu'il vous sera peut être demandé un jour de mettre en place, et qui sait, vous vous référerez peut-être à ce document.

Table des figures

Références

[1] Journal Du Net - Dossier Web Services, Dossier coordonné par Antoine Crochet-Damais, Novembre 2003, http://solutions.journaldunet.com/dossiers/webservices/sommaire.shtml

[2] eXtensible Markup Language sur Wikipédia, http://fr.wikipedia.org/wiki/XML

[3] XML Schema - Wikipédia, http://fr.wikipedia.org/wiki/XML_Schema

[4] Initiation aux Schema XML, http://gilles.chagnon.free.fr/cours/xml/schema.html

[5] Web Services Activity, http://www.w3.org/2002/ws/

[6] Web Services Architecture, W3C Working Group, 11 February 2004, http://www.w3.org/TR/ws-arch/

[7] SOAP Specifications, W3C Working Group, 24 June 2003, http://www.w3.org/TR/soap/

[8] SOAP - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/SOAP

[9] WSDL - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Web_Services_Description_Language

[10] Web Services Description Language (WSDL) 1.1, W3C Working Group, 15 March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[11] Web Services Description Language (WSDL) Version 2.0, W3C Working Group, 27 March 2006, http://www.w3.org/TR/wsdl20-primer/

[12] Introdution to WSDL at W3School, http://www.w3schools.com/wsdl/wsdl_intro.asp

[13] Universal Description Discovery and Integration - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/UDDI

[14] Home of the Universal Description Discovery and Integration standard, http://www.uddi.org/

[15] Découverte et Sélection de Services Web pour une application Mélusine, Ricardo DE LA ROSA-ROSERO, 15 septembre 2004

[16] Apache Axis2/Java - Next Generation Web Services, http://ws.apache.org/axis2/

[17] Representational state transfer - Wikipédia, http://fr.wikipedia.org/wiki/REST

[18] Tomcat (serveur) - Wikipédia, http://fr.wikipedia.org/wiki/Tomcat_(serveur)

[19] Python Web Services, http://pywebsvcs.sourceforge.net/

[20] Common Object Request Broker Architecture - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/CORBA

[21] Java remote method invocation - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Java_remote_method_invocation

[22] X-Methods - publicly available web services list, http://www.xmethods.com/

[23] Architecture orientée services - Wikipédia, http://fr.wikipedia.org/wiki/Service_Oriented_Architecture

[24] Service-Oriented Architecture and Web Services : Concepts, Technologies, and Tools - Sun Developper Network, Ed Ort, http://java.sun.com/developer/technicalArticles/WebServices/soa2/