# [**Introduction**](#f1) 2

## Architecture du service web2

# [**Définition**](#f2) SOAP 4

## Historique de SOAP4

# Fonctionnement SOAP5

# Architecture7

## Comment construire un message SOAP ? 7

### L’enveloppe SOAP 9

### L’entête SOAP10

### Le corps SOAP12

## Règles de syntaxe14

# Comment traiter SOAP-FAULT ? 14

# Quelques différences entre SOAP et REST17

## Points communs17

## Pourquoi choisir REST ? 17

## Pourquoi ne pas choisir REST ? 18

## Pourquoi choisir SOAP ? 18

## Pourquoi ne pas choisir SOAP ? 19

# Conclusion20

Introduction

Un Web Service est un composant logiciel identifié par une URI, dont les interfaces publiques sont définies et appelées en XML. Sa définition peut être découverte par d'autres systèmes logiciels. Les services Web peuvent interagir entre eux d'une manière prescrite par leurs définitions, en utilisant des messages XML portés par les protocoles Internet.

(W3C)

Une technologie permettant à des applications de dialoguer à distance via Internet

Indépendamment des plates-formes et des langages sur lesquelles elles reposent.

(Définition : <http://www.dicodunet.com/>)

Un service web est un programme informatique permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine, et en temps réel.

(Définition : Wikipédia)

Web Services peut accélérer l'intégration de différents types de systèmes gouvernements, des entreprises et de l'entreprise, réduire les coûts et améliorer la prestation, mais ils conduisent également à de nombreux problèmes de sécurité. L’intérêt pour les services Web a rapidement augmenté de leur début d'utilisation.

Un service Web est un ensemble de méthodes Web qui sont hébergés sur un serveur d'application. Méthodes Web sont des méthodes qui peuvent être invoquées à distance sur un réseau ou Internet. Pour invoquer une méthode Web, il faut créer un message et de l'envoyer au service Web via HTTP ou un autre mécanisme de transport. Le plus souvent, un service Web envoie un message de réponse en retour pour chaque appel de méthode; le message de réponse contient des informations utiles ou des informations de défaut en cas d'erreur.

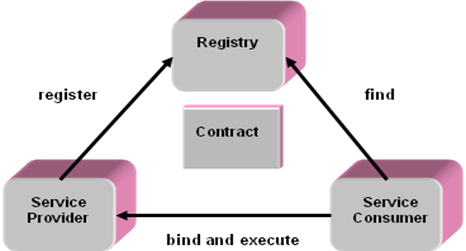
# Architecture du service web

L'architecture de services Web SOAP est illustrée dans la figure 1.0. Qui définit 3 entités:

1. Fournisseurs de services
2. Registre de service
3. Demandeur de services

Le fournisseur de services est l'entité adressable du réseau qui accepte et exécute les requêtes du consommateur. Le demandeur de service est une application, un service ou un autre type de module logiciel qui a besoin d'un service.

Un registre de service est un annuaire basé en réseau qui contient des services disponibles. Le consommateur de service trouve la description du service dans le registre qui est publié par le fournisseur de service. Grâce à cette description, le consommateur commence à interagir avec le service. La communication entre ces entités est basée sur XML et protocole SOAP.



|  |  |  |
| --- | --- | --- |
|  | Point de vue commercial | Point de vue technique |
| **Fournisseur de services** | Propriétaire de la de service | Plate-forme qui héberge le service |
| **Demandeur de service** | Entreprise qui nécessite certaines fonctionnalités | l'application qui appelle ou interagit avec le service |
| **Registre du service** | Registre consultable des descriptions de service où les fournisseurs de services publient leurs descriptions de service | |

Définition SOAP

**SOAP** (Object Access Protocol) est un protocole de **RPC** (Remote Procedure Call) permettant d'invoquer des méthodes d'objets distants. Il est comparable à DCOM ou CORBA mais contrairement à eux, il s'appuie sur des standards très connus. Il utilise XML pour définir les fonctions et les définitions disponibles. Il prend en charge divers protocoles de transport, tels que HTTP et SMTP, ainsi que différents formats comme MIME. Ces derniers sont très répandus sur de multiples plates-formes, ce qui donne à SOAP une grande portabilité et interopérabilité.

SOAP est une spécification non propriétaire. Il n'est pas lié à un protocole particulier. Il n'est pas non plus lié à un système d'exploitation ni à un langage de programmation.

SOAP étant un protocole d'échange d'informations entre diverses machines sur un réseau, elle nécessite un format pour transporter les données. Pour cela elle utilise des messages SOAP qui sont en fait des documents XML.

# Historique de SOAP

**SOAP 0.9** (Septembre 1999)

* Editeur (Microsoft, DeveloppMentor, UserLand)

**SOAP 1.0** (Novembre 1999)

* IETF

**SOAP 1.1** (Avril 2000)

* IBM & Soumission W3C

**W3C SOAP 1.2** (Sept 2000 à Mai 2002)

* Refonte SOAP 1.1
* Groupe de travail : 40 entreprises (IBM, Microsoft, Sun, Intel, Xerox, Canon ...)

Les spécifications de SOAP 1.2 sont composées de plusieurs parties :

* [Partie 0 : Primer](http://www.w3.org/TR/soap12-part0/)
* [Partie 1 : Messaging Framework](http://www.w3.org/TR/soap12-part1/)
* [Partie 2 : Adjuncts](http://www.w3.org/TR/soap12-part2/)
* [Specification Assertions and Test Collection](http://www.w3.org/TR/soap12-testcollection/)

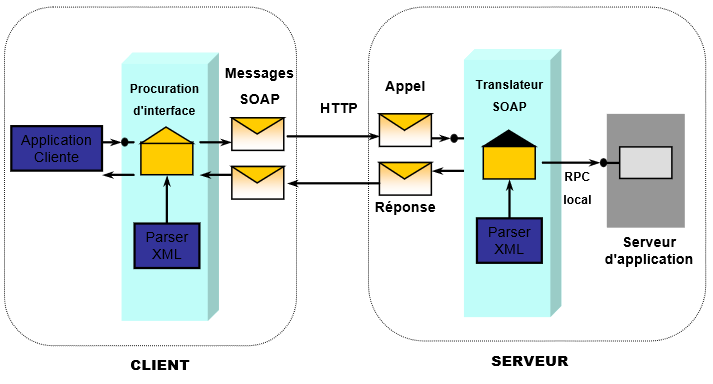
La version 1.2 des spécifications de SOAP est plus précise pour réduire les ambigüités qui pouvaient conduire à des problèmes d'interopérabilité entre différentes implémentations.

SOAP 1.2 propose un support pour des protocoles de transport différents de HTTP. La sérialisation de messages n'est pas obligatoirement en XML mais peut utiliser des formats binaires (XML Infoset par exemple).

Fonctionnement SOAP

Les services web de type Soap permettent l'appel d'une méthode d'un objet distant en utilisant un protocole web pour le transport (http en général) et XML pour formater les échanges. Les services web fonctionnent sur le principe client / serveur :

* un client appelle les services web
* le serveur traite la demande et renvoie le résultat au client
* le client utilise le résultat



L'appel à un service web de type SOAP suit plusieurs étapes :

1. Le client instancie une classe de type proxy encapsulant le service Web XML.
2. Le client invoque une méthode du proxy.
3. Le moteur SOAP sur le client crée le message à partir des paramètres utilisés pour invoquer la méthode
4. Le moteur SOAP envoie le message SOAP au serveur généralement en utilisant le protocol HTTP
5. Le moteur SOAP du serveur réceptionne et analyse le message SOAP
6. Le moteur fait appel à la méthode de l'objet correspondant à la requête SOAP
7. Le moteur SOAP sur le server crée le message réponse à partir de la valeur de retour
8. Le moteur SOAP envoie le message SOAP contenant la réponse au client généralement en utilisant le protocole http
9. Le moteur SOAP du client réceptionne et analyse le message SOAP
10. Le moteur SOAP du client instancie un objet à partir du message SOAP contenant la réponse

Un des intérêts des services web est de masquer aux développeurs la complexité de l'utilisation des standards sous-jacents. Ceci est réalisé grâce aux développements d'API et de moteurs pour la production et la consommation de services web.

Ces API sont dépendantes des plate-formes utilisées (Java, .Net, PHP, Perl, ...) mais elles mettent toutes en oeuvre avec plus ou moins de complétude les standards de l'industrie relatifs aux services web notamment SOAP et WSDL.

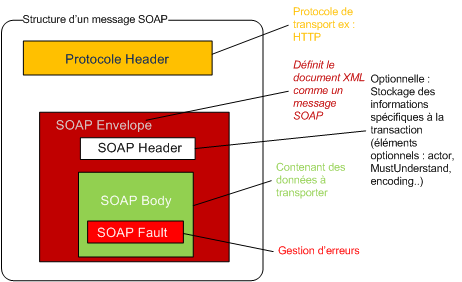
Ainsi les développeurs peuvent se concentrer sur l'écriture des traitements proposés par les services et par leur consommation sans se soucier de la tuyauterie sous-jacente. Un minimum de compréhension est cependant nécessaire pour bien appréhender les mécanismes mis en oeuvre.

Architecture

La grammaire de SOAP est assez simple à comprendre. Elle procure un moyen d'accès aux objets par appel de méthodes à distance. Les deux plus fortes fonctionnalités de SOAP sont sa simplicité et le fait que tout le monde a accepté de l'utiliser.

# Comment construire un message SOAP ?

Un message SOAP est composé de deux parties obligatoires **: l'enveloppe SOAP** et **le corps SOAP** ; et une partie optionnelle : **l'en-tête SOAP.**

****

**SOAP envelope** (enveloppe) :

Est l'élément de base du message SOAP. L'enveloppe contient la spécification des espaces de désignation (**[namespace](http://fr.wikipedia.org/wiki/Namespace%23En_informatique)**) et du codage de données.

**SOAP header** (entête) :

Est une partie facultative qui permet d'ajouter des fonctionnalités à un message SOAP de manière décentralisée sans agrément entre les parties qui communiquent. C'est ici qu'il est indiqué si le message est mandataire ou optionnel. L'entête est utile surtout, quand le message doit être traité par plusieurs intermédiaires.

S’il est présent, ce bloc doit toujours se trouver avant le bloc **Body** à l'intérieur du bloc **Envelope**.

**SOAP body** (corps) :

Est un container pour les informations mandataires à l'intention du récepteur du message, il contient les méthodes et les paramètres qui seront exécutés par le destinataire final.

**SOAP fault** (erreur) :

Est un élément facultatif défini dans le corps SOAP et qui est utilisé pour reporter les erreurs.

Par la suite, nous étudierons plus en détail le contenu de **l’enveloppe SOAP**, **le corps SOAP**, **l’entête SOAP**.

## L’enveloppe SOAP

L'enveloppe SOAP joue le rôle d’un conteneur pour les autres éléments du message SOAP, elle est définie au début par la balise **<soap:Envelope>** et se termine par la balise **</soap:Envelope>.**

Les messages SOAP ne peuvent pas être envoyés en lots, autrement dit l'enveloppe contient un seul message constitué **d'un entête facultatif** (SOAP header) et d'un **corps obligatoire** (SOAP body).



Toutes les balises XML associées à SOAP ont le préfixe ***soap*** .L'entête est **<soap:Header>** et le corps **<soap:Body>.**

SOAP repose entièrement sur les espaces de noms XML. Dans cet exemple, les espaces de noms sont introduits à l'aide d'un mot-clé « xmlns » *XML namespace* qui signifie espace de noms XML. L'espace de noms est utilisé pour identifier toutes les balises afin d'éviter les conflits. La spécification impose que tous les attributs contenus dans l'enveloppe SOAP soient explicitement associés à un *namespace*, de manière à supprimer toute ambiguïté. Par convention, la spécification SOAP définit deux *namespaces* fréquemment utilisés :

1. **soap-env ou soap** associé à l'URI « [..]schemas.xmlsoap.org/soap/envelope » pour définir le *namespace* de l'enveloppe dans la version 1.1, et à « [..]wwww.w3.org/2001/06/soap-envelope » dans la version 1.2 reprise par le W3C.
2. **soap-enc:encodingStyle**  associé à l'URI « [..]schemas.xmlsoap.org/soap/encoding » pour la définition des formats de types de données dans la version 1.1, et à « [..]www.w3.org/2001/06/soap-encoding » dans la version 1.2.

Deux autres espaces de noms fortement utilisés dans SOAP sont « **xsd** » et « **xsi** ».

* **xsd namespace** précise que les balises proviennent de la définition de schéma XML.
* **xsi namespace** indique que les balises viennent d'une instance d'un schéma XML.

## L’entête SOAP

L'en-tête SOAP est un élément facultatif dans un message SOAP. Toutefois, si un en-tête est présent, il doit être le premier élément qui apparaît dans l'enveloppe SOAP.

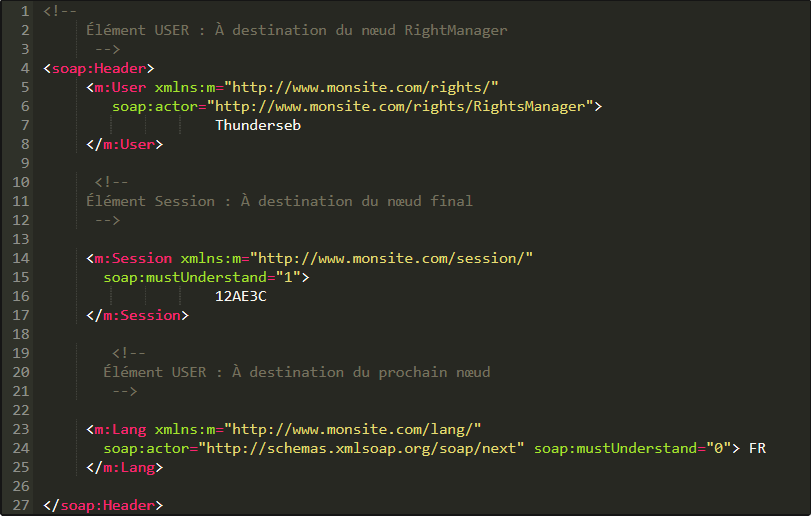
Son rôle est de communiquer des informations authentifiant l'émetteur ou bien encore le contexte d'une transaction dont le message SOAP doit passer par plusieurs intermédiaires SOAP pour arriver au destinataire final. Un intermédiaire SOAP est toute entité capable de recevoir et transmettre des messages SOAP.

L'en-tête d'un message SOAP commence avec la balise **<soap:Header>** et se termine avec la balise **</soap:Header>,** on peut aussi faire **<sopa-env:Header></sopa-env:Header>.**

Trois attributs associés à l'en-tête SOAP peuvent être utilisés :

1. **soap:mustUnderstand** : cet attribut prend la valeur 1 ou 0. La valeur 1 signale que le récepteur doit reconnaître l'information présente dans l'en-tête et que son traitement est obligatoire. La valeur 0 indique que l'en-tête peut être ignoré par le récepteur.
2. **soap:role** : sert à indiquer le destinataire SOAP auquel un bloc d'en-tête SOAP particulier est destiné.
3. **soap:relay** : est utilisé pour indiquer si un bloc d'en-tête SOAP ciblé sur un récepteur SOAP doit être réacheminé (relayé) s'il n'est pas traité.

**<soap:role>** et **<soap:relay>** sont utilisés conjointement par l'ensemble des nœuds SOAP intermédiaires qu'un message SOAP doit traverser pour arriver au destinataire final.



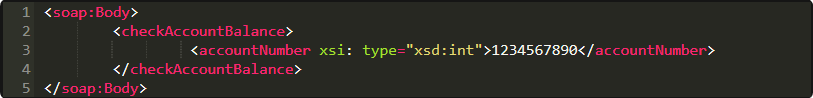
## Le corps SOAP

Le corps SOAP est un élément obligatoire dans le message SOAP. Il contient l'information destinée au receveur.

Le corps (*body*) doit fournir le nom de la méthode invoquée par une requête ainsi que les paramètres associés à celle-ci.

Le contenu du corps SOAP est utilisé pour spécifier un appel de méthode à un ordinateur distant avec les valeurs de paramètre. Par exemple, la demande du solde d'un compte bancaire.

L'extrait suivant représente un corps SOAP qui fait appel de procédure distante (RPC) à une méthode appelée **checkAccountBalance().**



Le corps du message SOAP commence avec la balise **<soap:Body>** et se termine avec la balise **</soap:Body>.**

L'élément **<checkAccountBalance>** fournit le nom de la méthode à appeler : **checkAccountBalance**. L'élément **accountNumber** est un paramètre qui est passé dans la méthode **checkAccountBalance**.

Les attributs **xsi** et **xsd** définissent les espaces de noms qui vont être utilisés dans le corps du message. La définition de xsi permet d'utiliser **xsi:type** dans le corps du message, **le xsd:int** signifie que cette valeur est de type **entier**. 1234567890 est la valeur donnée au paramètre.

L'ensemble de ces caractères représente un appel de méthode qui a la forme suivante en langage C :

exampleC.png

Une différence importante entre SOAP et XML-RPC est que les méthodes SOAP prennent des paramètres nommés. L'ordre des paramètres, lui, n'a pas d'importance, contrairement à XML-RPC.

# Règles de syntaxe

* 1. un message SOAP doit être codé en XML
  2. un message SOAP doit respecter l’architecture ci-dessus.
  3. un message SOAP doit utiliser l'espace de désignation de l'enveloppe SOAP
  4. un message SOAP doit utiliser l'espace de désignation d'encodage SOAP
  5. un message SOAP ne doit pas contenir une référence à une DTD
  6. un message SOAP ne doit pas contenir des instructions de type XML Processing

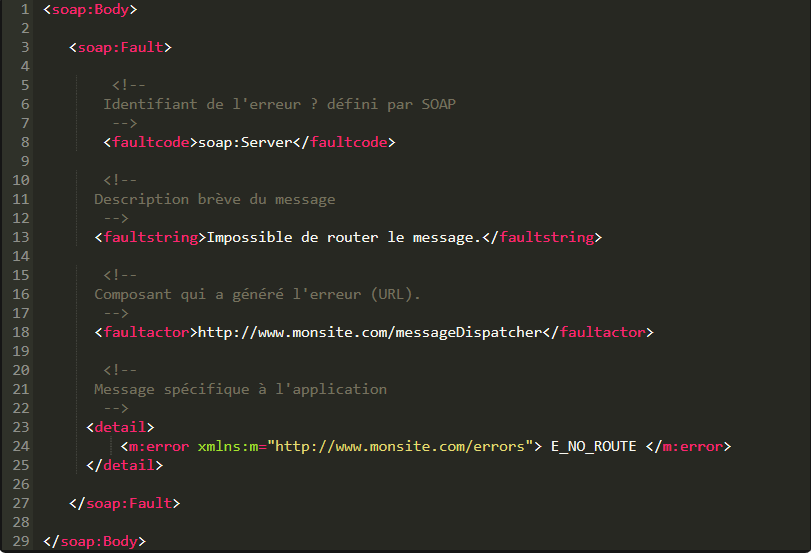
Comment traiter SOAP-FAULT ?

Afin de récupérer le plus grand nombre d'erreurs, l'approche SOAP se base essentiellement sur le bon usage de la balise **<soap:fault>** qui est contenue dans le corps SOAP. Cette balise est utilisée pour communiquer un problème qui a eu lieu dans la tentative de réalisation de la demande adressée au service Web. L'élément d'erreur est facultatif et figure uniquement dans les messages de réponse, il ne peut y apparaître qu'une seule fois. La balise **<soap:fault>** peut contenir quatre autres balises facultatives :

* **faultcode** : Il contient un code indiquant la nature du problème.
* **faultstring** : est la version lisible par l'homme de la balise *faultcode*. C'est la traduction en langage naturel du code d'erreur.
* **faultactor** : indique le service qui a généré l'erreur. Cela est important lorsqu'une chaîne de services a été utilisée pour traiter la demande.
* **detail** : cet élément doit contenir autant d'informations que possible sur l'état du serveur à l'instant de l'apparition de l'erreur. Il contient souvent des valeurs de variables au moment de l'échec.

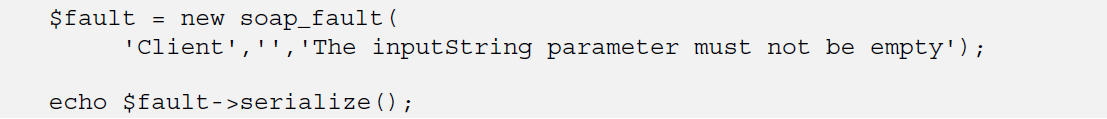
Quatre types de codes d'erreur sont définis par la spécification :

* **soap: Server** : indique qu'une erreur s'est produite sur le serveur, mais pas avec le message lui-même.
* **soap: Client** : signifie que le message reçu contient une erreur.
* **soap: VersionMismatch** : cette erreur se produit lorsque les versions des protocoles SOAP utilisés par le client et le serveur sont différentes.
* **soap: MustUnderstand** : cette erreur est générée lorsqu'un élément dans l'en-tête ne peut pas traiter alors qu'il est marqué comme obligatoire.

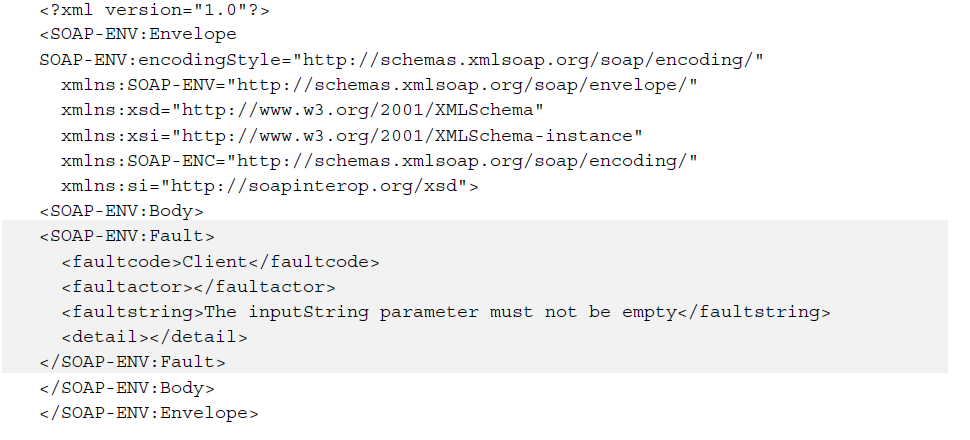


La classe du **soap\_fault** possède une seule méthode autre que le constructeur qui s’appelle **serialize ()** .cette méthode prend l’information d’erreur et la sérialiser.

Un exemple d’utilisation du **soap\_fault** est montré ci-dessous :



Le message SOAP ci-dessous est le résultat de la méthode **Serialize** sur l’objet **soap\_fault** instancie ci-dessus :



Quelques différences entre SOAP et REST

**REST** et **SOAP** sont des éléments souvent comparés l'un à l'autre dans la conception des applications client-serveur, mais c'est une erreur. En effet, ces éléments ne sont pas d'un même type : **SOAP** est un protocole tandis que **REST** est un style d'architecture.  
  
La différence majeure entre ces 2 éléments est le degré de liaison entre le client et le serveur. Un client développé avec le protocole **SOAP** ressemble à un logiciel d'ordinateur, car il est étroitement lié au serveur. Si une modification est effectuée d'un côté ou de l'autre, l'ensemble peut ne plus fonctionner. Il faut effectuer des mises à jour du client s'il y a des changements sur le serveur et vice-versa.  
  
Un client de type **REST** sait utiliser un protocole et des méthodes standardisées. Son application doit rentrer dans ce modèle. On ne crée pas de méthodes supplémentaires, on utilise les méthodes standardisées.

Il y a en conséquence beaucoup moins de couplage entre le client et le serveur : un client peut utiliser un service de type **REST** sans aucune connaissance de l'API. A l'inverse, un client **SOAP** doit tout savoir des éléments qu'il va utiliser pendant son interaction avec le serveur, sinon cela ne fonctionnera pas.

http://amine.benkirane.over-blog.com/2012/05/architecture-logicielle-soap-vs.-rest-web-services

# Points communs

Les deux fonctionnent parfaitement avec HTTP, le protocole de transfert de données entre clients et serveurs utilisé sur le Web. HTTP est un protocole simple, bien connu, donc cela peut être considéré comme un avantage : facile à comprendre, implémenté sur la plupart des plateformes, il n’est pas d’habitude bloqué par les pare-feu.

# Pourquoi choisir REST ?

* **REST est léger et simple** : les messages sont courts, faciles à décoder par le navigateur et par le serveur d’application.
* **REST est auto-descriptif :** vous pouvez naviguer à travers ses ressources comme vous le feriez avec une page Web. Il y a une URL intuitive unique pour chaque ressource. On peut facilement en déduire la structure des ressources sans avoir besoin de beaucoup de documentation.
* **REST est « stateless »** (il n’est pas adapté aux transactions longues et complexes) : il est parfait pour les opérations simples (créer, lire, mettre à jour, effacer).
* **REST peut être géré en cache (antémémoire)** : les ressources simples, identifiables sont aussi faciles à mettre en cache qu’un article sur un site Internet. L’adaptation aux variations de charge et la performance que cela apporte ne doivent pas être négligées.

# Pourquoi ne pas choisir REST ?

**REST** est généralement critiqué pour son manque de complexité exigée pour les transactions d’affaires : sécurité, support transactionnel.

**REST** est considéré en quelque sorte comme un artifice du « web 2.0″ plutôt qu’une vraie solution d’entreprise.

# Pourquoi choisir SOAP ?

* SOAP a été conçu comme un mécanisme d’appel de procédure distante. On utilisera SOAP en toute confiance pour appeler une méthode sur un autre serveur et recevoir la réponse dans un mode contractuel.
* **SOAP supporte le transactionnel :** il supporte les transactions ACID, de sorte qu’il peut gérer un commit à deux phases sur des ressources transactionnelles distribuées (WS – Atomic Transactions ).
* **SOAP a quelques caractéristiques de sécurité supplémentaires :** il supporte l’identification à travers des intermédiaires, plus que SSL ne peut offrir.
* **SOAP est formel** : les méthodes à distances et leurs résultats sont bien spécifiées via des contrats.
* **SOAP est fiable :** il dispose des mécanismes pour garantir la fiabilité de la transmission et de la réception de données.
* **SOAP offre de nombreux outils :** étant donné qu’il a d’abord été publié par Microsoft en 1998, beaucoup de fournisseurs ont créé des outils au fil des années. Aussi la plupart des plates-formes de développement sont pourvues d’un moyen facile d’exploiter une fonction distante via SOAP.

# Pourquoi ne pas choisir SOAP ?

SOAP a été à l’origine imaginé pour faire un protocole RPC qui passe les pare-feu (remplaçant donc le DCOM de Microsoft avec XML sur HTTP).

SOAP est considéré trop complexe et verbeux pour les besoins ordinaires.

SOAP est d’habitude utilisé dans des framework propriétaires. Bien que cela fournisse un point d’entrée apparemment plus facile (quelquefois basé sur un assistant) pour créer des services Web, sa complexité rend ardu le débogage des messages, donc tout l’avantage du XML lisible par l’utilisateur est perdu assez vite.

SOAP nécessite une compréhension claire des APIs spécifiques au site. Son utilisation est comparable à l’utilisation d’un nouveau framework. Il demande du temps et de la documentation.

L’interopérabilité de SOAP a diminuée au fil des années, avec des normes différentes et les implémentations des fournisseurs de logiciels. Désormais il est certain que REST est de loin plus interopérable.

Conclusion

Technologiquement, les WebServices sont matures pour être utilisés dans un environnement de développement distribué. Leur promotion faite par les acteurs principaux du marché informatique en fait une des technologies sur lesquelles il faudra compter dans le futur.  
On peut notament trouver des implémentations WebService dans les framework .Net [Microsoft], WebSphere [IBM], ou WebLogic [BEA], et il existe aussi certains solutions libre (Apache SOAP, Apache Axis ou uddi4j [IBM/HP]).

Cependant, d'un point de vu innovant, les WebServices n'apportent réellement que peu de choses par rapport aux technologies distribuées déjà existantes telles CORBA ou JNI, les concepts novateurs introduits par les WebServices étant pour le moment exclusivement réservés au domaine du théorique. Aussi, on est en droit de se demander si l'apparition de cette technologie est réellement motivée par une envie d'évolution ou bien uniquement par une stratégie marketing de la part des grand groupes informatiques partenaires de cette technologie et coutumiers de genre ...