Initiation au couple gagnant WSDL/SOAP

Lionel Tricon - lionel.tricon@free.fr

La meilleure définition à donner au format WSDL est qu'il permet de décrire, de façon abstraite et indépendante du langage et des architectures, l'ensemble des interfaces d'entrées et de sorties d'un Service Web. En résumé, c'est un langage de description de services qui trouve naturellement sa place au coté du protocole auto-proclamé des Services Web, SOAP.

Article publié dans GNU/Linux Magazine France, numéro 76 (Octobre 2005).

Préambule

Pouvoir comprendre et interpréter correctement un fichier descriptif au format **WSDL** [1] est indispensable au bagage de toute personne souhaitant mettre en oeuvre le protocole **SOAP** [2] ou s'initier au monde merveilleux des **`Services Web`** (pour ceux qui sortiraient d'un long comma et donc n'auraient pas consulté la presse spécialisée durant les quatre dernières années, on peut considérer qu'un service web est un modèle abstrait d'échange de données basé sur des protocoles universels comme **`HTTP`** ou **`XML`**, donc fortement orienté Internet).

Une introduction assez complète de la notion de Service Web a été faite dans le *numéro 72* de ce magazine suivi d'une introduction pratique au préprocesseur **gSOAP** [3] (framework SOAP en **`C`** et **`C++`**). Même si ce n'est pas indispensable, sa lecture vous aidera sans doute à mieux apprécier certains aspects de cet article qui peut, d'une certaine façon, être considéré comme sa suite logique.

Initialement, le format WSDL a été pensé dans le but de pouvoir supporter n'importe quel type de Service Web : On pense alors assez logiquement aux protocoles **XML-RPC** [4] (l'alternative la plus sérieuse à SOAP) ou **REST** [5] (acronyme de *REpresentational State Transfer*, qui bien que largement dépassé reste le plus simple des modèles de Services Web jamais conçu) mais, dans les faits, celui-ci a été architecturé presque intimement autour du seul protocole SOAP. Exit donc les autres protocoles ; dans cet article nous allons nous attacher exclusivement à la mise en oeuvre du protocole SOAP avec WSDL car c'est à la fois le présent (les Services Web ont connus leur véritable essor depuis l'avènement de SOAP) et l'avenir (aucun concurrent sérieux n'est apparu depuis sur les radars).

L'objectif de cet article est donc de vous initier à la structure du langage WSDL mais aussi à vous permettre d'évaluer son impact sur le format des messages SOAP avec l'aide de nombreux exemples afin de rendre le plat le plus digeste possible (de par sa nature XML, ce format peut rapidement tourner à l'usine à gaz).

Pour mieux comprendre la suite, il nous faut nous initier à la notion d'**`espace de nommage`** et de **`schéma XML`**.

Comme nous l'avons précisé, le format WSDL (actuellement en version 1.1) fait une utilisation massive du format XML (qui est donc de facto le socle technique des Services Web car utilisé aussi directement dans les messages SOAP) ; une bonne connaissance des espaces de nommages XML (namespaces) et des schéma XML (XML Schema) est donc un pré-requis indispensable à la lecture d'un document WSDL :

- Un **schéma XML** est un document XML (lui même décrit en XML) qui permet de valider le format et le contenu d'un fichier XML (on parle alors de conformité à un schéma XML). Ce format suit un modèle hiérarchique similaire à une grammaire et qui spécifie en particulier des contraintes de typage sur les données,
- Un **espace de nommage** permet de regrouper sous un identifiant unique **URI** (*Uniform Resource Identifier*) un sous-ensemble d'éléments et d'attributs dans le but d'isoler dans le document XML le domaine de définition d'un objet : On peut donc mélanger dans un même document XML plusieurs familles d'objets ; la distinction entre les balises (qui peuvent avoir le même nom) se faisant sur l'espace de nommage utilisé.

Le document suivant est un exemple de schéma XML :

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="LIVRE">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="TITRE"  type="xsd:string" minOccurs="1"/>
      <xsd:element name="AUTEUR" type="xsd:string" minOccurs="1"/>
      <xsd:element name="PRIX"   type="xsd:int"    minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="ISBN" type="T_ISBN"/>
  </xsd:complexType>
  <xsd:simpleType name="T_ISBN">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\[0-9]-[0-9]{4}[-][0-9]{4}[-][0-9]"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

</xsd:schema>
```

On définit une balise LIVRE qui possède un attribut ISBN (contrainte sur le format) et plusieurs sous-balises TITRE, AUTEUR et PRIX (optionnelle pour cette dernière). De plus, cet exemple nous permet d'introduire la notion d'espace de nommage (xsd est l'espace de nommage usuellement utilisé pour délimiter un schéma XML).

Plus précisément, le document XML suivant est conforme au schéma XML précédent :

```
<LIVRE ISBN="2-2640-3079-8">
  <TITRE>Passage des miracles</TITRE>
  <AUTEUR>Naguib Mahfouz</AUTEUR>
</LIVRE>
```

Un exemple plus parlant d'espace de nommage est apporté par le document XML suivant :

```
<LIVRE ISBN="2-2640-3079-8">
  <TITRE>Passage des miracles</TITRE>
  <AUTEUR>Naguib Mahfouz</AUTEUR>
</LIVRE>
```

L'espace de nommage principal utilisé est XHTML et l'on introduit notre propre espace de nommage **my** (définie par un schéma XML) pour délimiter les deux familles d'objets (c'est une introduction rapide à ces deux notions mais qui demeure indispensable pour la suite et qui vous donnera peut être envie d'aller plus loin sur le sujet).

Puisque nous sommes dans les préliminaires et avant d'attaquer à pleines dents ce langage, il nous reste encore deux points fondamentaux à aborder qui servent de fondations au langage WSDL et au protocole SOAP :

- La façon dont les données sont formatées dans les requêtes SOAP,
- Le choix du modèle d'interaction pour l'échange de données.

RPC/Document et Literal/Encoded

Il faut savoir que la spécification SOAP définit deux types de format possibles pour le transport des données entre applications : Les messages de type **RPC** et ceux de type **Document**. On retrouve ce choix tout naturellement dans le format WSDL où il faut explicitement choisir l'un des deux (le choix va directement impacter la façon dont les données vont être structurées au sein des messages SOAP).

Historiquement, l'un des objectifs fondateurs du format SOAP était de faciliter l'échange de messages dans un modèle d'invocation d'objets distants (de type RPC) puisque c'est le modèle d'interaction le plus fréquemment utilisé sur le réseau à l'heure actuelle. Une section de la spécification SOAP lui est donc intégralement dédiée et l'on parle alors de **messages de type RPC** (cf. spécification SOAP 1.1, section 7) : Les données encodées au format XML sont stockées dans le corps du message SOAP avant d'être envoyées à la machine destinataire ; cette dernière traduisant alors les données XML dans le format de l'application cible.

Il peut aussi arriver que les données que l'on souhaite transporter soient déjà au format XML ou que l'on souhaite s'abstraire du formalisme du modèle RPC : On parle alors de **messages de type Document** (aucune règle d'encodage ou de formatage spécifique ne s'applique alors en dehors de la conformité des

données à un schéma XML).

Notez que l'on peut tout à fait reproduire le modèle "RPC" dans un message de type "Document" : Tout est une question d'interprétation des données (c'est toutefois plus complexe à traiter pour la couche logicielle que dans le modèle RPC).

Afin de mieux appréhender ultérieurement la façon dont ce choix va impacter le format des messages SOAP, il nous faut jetez un oeil sur la structure type d'un message SOAP (hors en-tête HTTP) qui est composé des éléments suivants :

- Une enveloppe (**Envelope**) qui définit le document XML comme un message SOAP,
- Un en-tête optionnel (**Header**) pour stocker les informations métiers spécifiques à la transaction (authentification, jeton d'autorisation, état de la transaction, ...),
- Un corps (**Body**) contenant les données à transporter,
- Une gestion d'erreur (**Fault**) qui identifie la condition d'erreur (si applicable),
- Et enfin des attachements optionnels (format **DIME**).

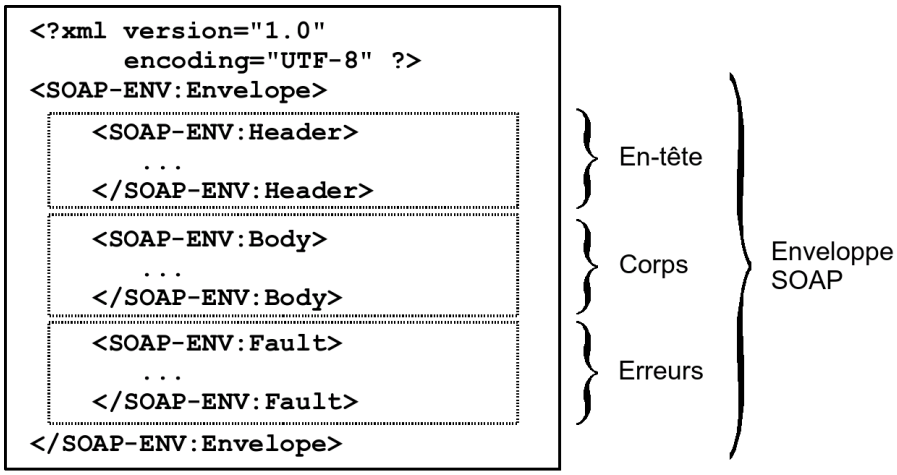

Figure 1 : Format d'un message SOAP

En pratique, les types « Document » et « RPC » se réfèrent exclusivement à la façon dont l'élément **`<Body>...</Body>`** est mis en forme dans le message SOAP (cela n'impacte pas le reste du message) : Soit un message directement au format XML dans le cas du modèle "Document", soit les arguments d'entrées-sorties des fonctions encodées au format XML dans le cas du modèle "RPC".

Mais pour montrer que rien n'est définitivement simple dans le monde des Services Web, il existe encore deux styles différents de mise en forme pour les paramètres au sein d'un message SOAP : **Encoded** et **Literal**. Là encore, il faut choisir l'un ou l'autre modèle d'encodage dans le format WSDL et donc comprendre l'impact que cela peut avoir sur le format du message SOAP.

Encoded fait référence à la mise en forme des paramètres à l'aide des règles de codage définies par l'attribut **`encodingStyle`** (en général, les règles définies par la spécification [1] SOAP 1.1, section 7)) alors que **Literal** implique que les données doivent être conformes à un schéma XML (présent dans le fichier WSDL) sur l'attribut **`type`** ou **`element`** des données.

Le mélange des différents types de messages et styles d'encodage nous donnent au final quatre combinaisons possibles :

- RPC/Encoded,
- RPC/Literal,
- Document/Encoded,
- Document/Literal.

Pas évident de s'y retrouver mais plusieurs éléments vont nous permettre d'y mettre un peu d'ordre :

- L'association entre le style "Encoded" et le modèle "Document" n'est pas implémenté,
- La norme **WS-I Basic Profile 1.0** (ensemble de spécifications permettant de garantir l'interopérabilité des Services Web) a choisi le style "Literal" comme standard au détriment du style "Encoded" (plus généralement, WS-I exclut l'encodage SOAP) : A ce stade, il nous reste donc les combinaisons "RPC/Literal" et "Document/Literal",

- Microsoft supporte dans .NET les Associations « RPC/Encoded » et « Document/Literal » (cette dernière étant proposée par défaut dans sa variante **`Document/Literal wrapped`**) et supporte « RPC/Literal » à partir de sa version 2.0 (à la différence de JAVA/AXIS et gSOAP qui supportent nativement les trois).

Il nous reste donc deux possibilités : Le support des associations « Document/Literal » et « RPC/Literal » si l'on veut être compris par les implémentations de référence du marché (.NET, AXIS ou encore gSOAP) mais pas forcément des implémentations plus modeste (perl, python, php, ...) ou encore « RPC/Encoded » pour quasiment l'ensemble des acteurs du marché même si dans ce dernier cas on n'est plus conforme à la norme WS-I (cruel dilemme !).

Cominaisons	**Framework SOAP**	**Conformité WS-I**
RPC/Encoded	.NET, Axis, gSOAP	Echec
RPC/Literal	.NET 2.0, Axis, gSOAP	Succès
Document/Encoded	-	-
Document/Literal	.NET, Axis, gSOAP	Succès

Figure 2 : Conformité des différents Framework SOAP

En tout état de cause, cela ne va généralement rien changer pour le programmeur qui doit juste s'assurer du niveau d'interopérabilité des différents outils SOAP mis en oeuvre en regard du modèle de messages implémenté (RPC ou Document et Literal ou Encoded).

Tout le reste est affaire de protocole et de traitement logiciel des données par le framework SOAP utilisé : Fonctionnellement, cela ne change rien pour l'utilisateur.

Choix du modèle d'interaction

La spécification WSDL en version 1.1 définie quatre modèles standards d'interactions entre le noeud appelant (client) et le noeud destinataire (fournisseur du service) :

- **One-way** : Le client envoie une requête SOAP au fournisseur du service mais n'attend pas de réponse en retour,
- **Request-response** : Le client envoie une requête SOAP au fournisseur du service et attend une réponse en retour,
- **Solicit-response** : Cette fois, c'est le fournisseur du service qui envoie une requête SOAP au client et qui attend une réponse en retour,
- **Notification** : Le fournisseur du service envoie une requête SOAP au client.

Attention, ne vous méprenez pas, ce n'est pas parce que la spécification WSDL décrit ces quatre modèles que ceux-ci sont réellement implémentés dans la spécification SOAP : Dans les faits, seuls les deux premiers modèles sont implémentés. Pour enfoncer le clou, la spécification WS-I n'autorise de toute façon l'utilisation que des deux premiers modèles et exclu explicitement l'implémentation des deux autres.

Si vous souhaitez tout de même implémenter les deux derniers modèles ou mettre en place un mode de réponse asynchrone, cela ne sera pas chose aisée car cela inverse de facto le modèle client-serveur synchrone de SOAP basé en général sur le protocole HTTP.

Plusieurs solutions sont toutefois possibles avec un peu d'astuce :

- Pour implémenter les modes **solicit-response** ou **notification** :
- Vous pouvez mettre en place un serveur SOAP sur votre client et croiser les requêtes (pas forcément aisé si ce n'est parfois impossible),
- Vous pouvez mettre en place un algorithme de polling avec un client qui interroge à intervalle régulier le serveur pour y récupérer des messages et éventuellement y répondre,
- Vous pouvez implémenter un mode **request-response "asynchrone"** en retournant au client un numéro de ticket pour la réponse (par exemple dans l'en-tête) et une évaluation du temps de traitement : Lorsque le temps de traitement est révolu, le client se reconnecte et présente son ticket ; si la réponse n'est pas encore prête, on retourne au point de départ avec un nouvelle temporisation.

En résumé, bien que spécifiés mais non-implémentés, tous ces modèles peuvent donc être conçus par abstraction des modes **`one-way`** et **`request-response`** dans le cas du protocole SOAP.

Il est tout à fait probable que le protocole SOAP implémente à terme ce type de fonctionnalité dans une

énième version ce qui le rapprochera alors un peu plus d'un véritable bus logiciel, ce qui est loin d'être le cas à l'heure actuelle (et souvent source de confusion dans l'esprit de certains).

/// Début encadré ///

La norme **WS-I Basic Profile** apparaît souvent lorsque l'on aborde le domaine si touffu des Services Web. Voici, en résumé, quelques contraintes imposées par cette spécification issue d'un organisme de normalisation prônant l'interopérabilité entre les différentes implémentations des Services Web :

- Impose le méta-langage XML en version 1.0,
- Impose l'utilisation du binding WSDL SOAP (version 1.1) avec HTTP comme couche transport,
- Impose l'utilisation de la technologie POST HTTP pour le protocole de communication,
- Impose l'utilisation du code de réponse HTTP 500 pour les messages d'erreur SOAP ,
- Impose l'utilisation du langage descriptif WSDL en version 1.1 pour décrire les interfaces d'entrées et de sorties d'un service web,
- Impose l'utilisation des combinaisons **RPC/Literal** ou **Document/Literal** pour la déclaration d'un Service Web sous WSDL,
- La règle précédente exclu implicitement le style **Encoded** mais la norme est encore plus claire :
    - Exclusion explicite de l'utilisation de l'encodage SOAP,
    - Exclusion explicite de la combinaison **RPC/Encoded** sous WSDL,
- Exclusion des modes d'interaction **Solicit-response** (le serveur envoi un message et le client répond) et **Notification** (le serveur envoi un message).

/// Fin Encadré ///

Le format WSDL

Globalement, le format WSDL définie quatre familles principales :

- Les données ayant trait aux **interfaces** disponibles publiquement,
- Les **types de données** utilisées au sein des messages échangés,
- Les données sur les **informations de liaisons** du protocole de transfert utilisé,
- Les **données d'adresse** permettant de localiser le service spécifié.
- Plus spécifiquement, un document WSDL se composent de plusieurs composants (sous forme de balises XML) appartenant à l'une de ces familles et s'imbriquant mutuellement dans l'arborescence XML (dans chaque composant, l'attribut **name** permet de référencer directement le composant) :
- Un **portType** (définition abstraite des méthodes) qui dérive du composant **binding** (définition concrète de l'appel des méthodes) et qui est composé d'une à plusieurs **opérations**,
- Une opération qui est définie en terme de **message**,
- Les types de données d'un **message** qui sont définies par l'intermédiaire de schémas XML ou en suivant les règles d'encodage SOAP (qui reposent, pour information, en grande partie sur les schémas XML).

Le tableau de la **figure 3** reprend en détail chaque composant majeur d'un document WSDL :

Composant XML	Description
wsdl:definitions	Noeud racine du document qui donne le nom du service et déclare les différents espaces de nommage
wsdl:documentation	Contient de la documentation ou des commentaires (en général, du texte) destinés à être exploités par la personne souhaitant mettre en oeuvre le Service Web. Peut être utilisé par n'importe quel composant WSDL
wsdl:types	Déclaration ou import de tous les types de données référencés par les attributs **element** ou **type** d'un message. A noter que c'est le plus souvent l'encodage XML Schema qui

	s'applique (préfixé par l'espace de nommage **xsd**) en particulier dans le cas du modèle **Document/Literal** (le message est alors intégralement défini par un schéma XML)
wsdl:message	Un message est composé de un à plusieurs composants **wsdl:part** (réunis, ils définissent les paramètres du message ou les valeurs retournées par le message) et décrit un message unique (requête ou réponse)
wsdl:part	Fait référence à un paramètre ou à un ensemble de paramètres d'un message. Détermine le type de données par utilisation de l'attribut **type** ou **element** : On peut alors directement spécifier son type d'encodage dans le cas d'un paramètre unique (type) ou référencer le type de données présentes dans le composant **wsdl:types** (élément)
wsdl:portType	Définition abstraite des prototypes des méthodes à appeler. Combine plusieurs composants **wsdl:message** afin de spécifier une opération (composant **wsdl:operation**)
wsdl:operation	Définition abstraite du prototype d'une méthode. Chaque operation fait en général référence à un message en entrée (**wsdl:input**) et à un ou plusieurs messages en sorties (**wsdl:output** et **wsdl:fault**) selon le modèle d'interaction du message (**one-way** ou **request-response**)
wsdl:binding	Définition concrète de l'appel à une ou plusieurs méthodes. Spécifie le protocole de communication, le type de l'échange (**RPC** ou **Document**) et le style de chaque opération (**Literal** ou **Encoded**)
wsdl:service	Point d'entrée du document WSDL. Regroupe un ensemble de services définis par le ou les composants **wsdl:port**
wsdl:port	Définition d'un point d'entrée pour un service et référence le composant **wsdl:binding** permettant son traitement

Figure 3 : Composition d'un fichier WSDL

Cela n'est pas très clair dans votre esprit ? C'est un peu normal mais la **figure 4** devrait mieux vous aider à visualiser les dépendances car présentant une décomposition d'un fichier au format WSDL en mettant en évidence les relations entre composants.

La lecture du fichier se fait de bas en haut et on peut en déduire immédiatement les éléments suivants :

- L'espace de nommage propre au fichier est **urn:myXML**,
- Le fichier WSDL est de type **rpc** avec un style **Encoded** : **RPC/Encoded**,
- Le prototype de la fonction mise à disposition sur le serveur, **fonction1**, prend en entrée deux chaînes et en sortie une chaîne et un entier,
- Le service est à disposition sur l'adresse **http://myhome.org/myXML**

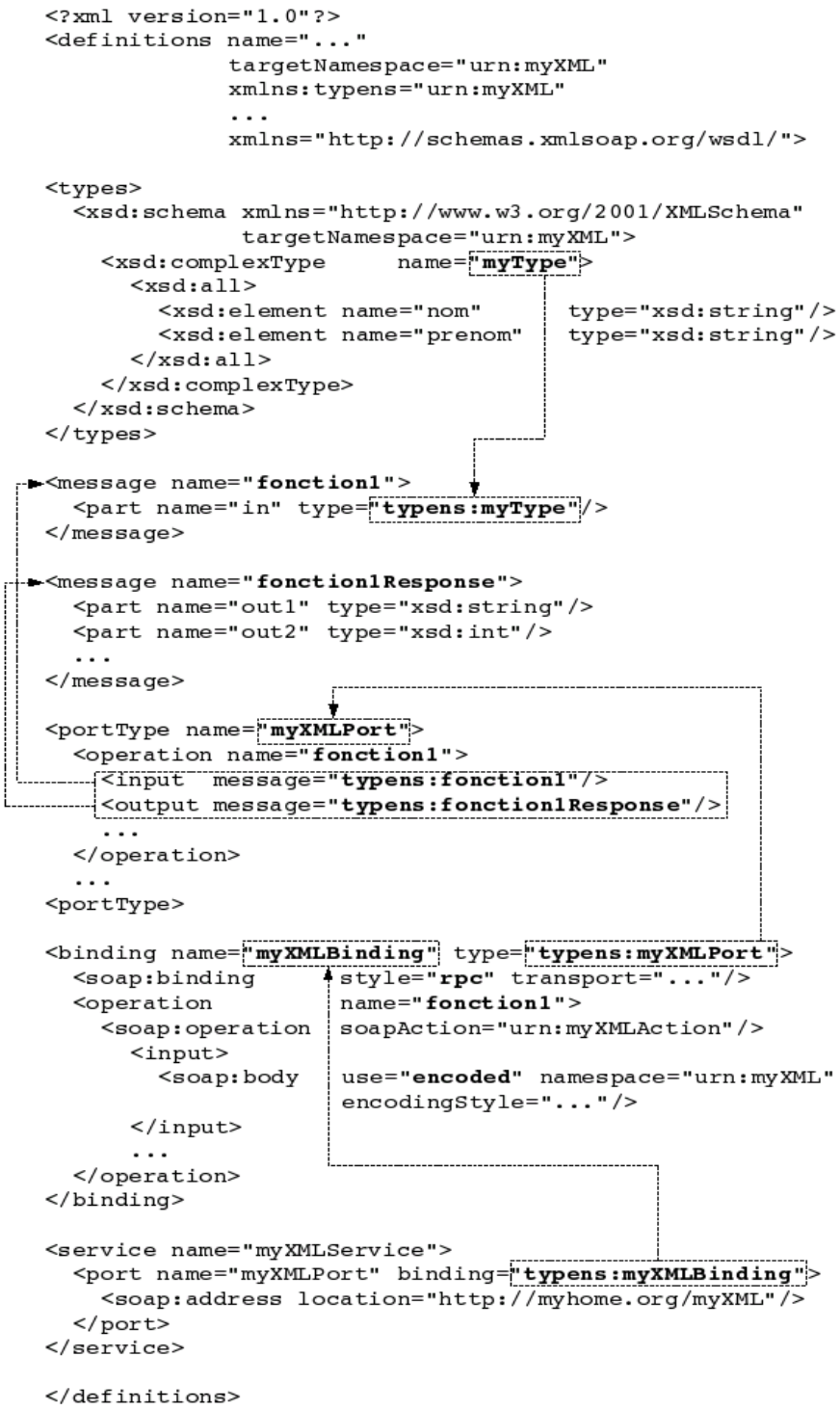

Figure 4 : Décomposition d'un fichier WSDL

Dans cet exemple, la couche transfert utilisée est HTTP (imposée de toute façon par défaut par la norme WS-I): Il faut alors le préciser dans l'attribut **`transport`** du composant **`binding`** :

```
<soap:binding style="rpc|document" transport="http://schemas.xmlsoap.org/soap/http"/>
```

De la même façon, l'attribut **encodingStyle** doit être précisé uniquement dans le style **Encoded** (cela ne s'applique pas au style **Literal**) pour chaque message référencé dans le composant **binding**. Généralement, on utilise l'encodage SOAP que nous avons abordé plus tôt dans l'article : **http://schemas.xmlsoap.org/soap/encoding/**.

Les curieux auront remarqués le présence d'un attribut **`soapAction`** dans la définition concrète d'une opération : Il s'agit d'une chaîne de caractères arbitraire de type URI qui, lorsqu'elle est présente, doit être transmise dans l'en-tête HTTP lors de l'appel au Service Web dans le champ **`SOAPAction`** (cas de la version 1.1 de SOAP) ou directement dans un attribut (cas de la version 1.2 de SOAP). Normalement, cet attribut est rendu obligatoire par la norme WSDL dans le cas de l'utilisation sur le binding HTTP mais elle

est le plus souvent optionnelle dans les faits suivant les implémentations (à l'exception notable des implémentations SOAP de Python ou celle de .NET).

Une bonne approche consiste à saisir par défaut cet attribut avec comme URI l'espace de nommage du document suivi du nom de la fonction appelée avec le caractère **#** comme séparateur. Dans le cas de notre exemple on aurait pu mettre **urn:myXML#fonction1** pour la fonction **fonction1** même si la valeur peut être commune à tout le document.

Son utilité réelle paraît peu évidente au début mais cet attribut peut-être perçu comme une action qui identifie le but du message afin, par exemple, de n'autoriser que certains types de messages à passer au travers d'un pare-feu.

Revenons aussi rapidement à la déclaration du modèle d'interaction.

Au sein d'un document WSDL, la grammaire permettant de déclarer un modèle **one-way** suit la syntaxe suivante (un seul élément **input**) :

```
<wsdl:definitions .... >
  <wsdl:portType .... >
   <wsdl:operation name="nmtoken">
     <wsdl:input name="nmtoken"? message="qname"/>
  </wsdl:operation>
   </wsdl:portType >
</wsdl:definitions>
```

Pour sa part, la déclaration d'un modèle **request-response** implique la présence supplémentaire des éléments **output** et **fault** (c'est le modèle mis en oeuvre par notre exemple) :

```
<wsdl:definitions .... >
  <wsdl:portType .... >
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
      <wsdl:input name="nmtoken"? message="qname"/>
      <wsdl:output name="nmtoken"? message="qname"/>
      <wsdl:fault name="nmtoken" message="qname"/>*
    </wsdl:operation>
  </wsdl:portType >
</wsdl:definitions>
```

Avant d'aborder dans le détail l'impact du format WSDL sur le protocole SOAP dans les combinaisons RPC/Encoded, RPC/Literal et Document/Literal, parcourons ensemble les différents espaces de nommage communément utilisés dans un document WSDL 1.1 et SOAP 1.1 (les préfixes sont variables et peuvent changer suivant la déclaration) :

Préfixe	Namespace	Description
wsdl	http://schemas.xmlsoap.org/wsdl/	Espace de nommage du format WSDL
soap	http://schemas.xmlsoap.org/wsdl/soap	Espace de nommage pour le binding SOAP (en version 1.1) du format WSDL
http	http://schemas.xmlsoap.org/wsdl/http/	Espace de nommage pour le binding HTTP du format WSDL
soapenc	http://schemas.xmlsoap.org/soap/encoding/	Espace de nommage pour l'encodage SOAP (style Encoded)
soapenv	http://schemas.xmlsoap.org/soap/envelope/	Espace de nommage de l'enveloppe d'un message SOAP
xsd/xsi	http://www.w3.org/2001/XMLSchema	Espace de nommage des schémas XML

On utilise aussi souvent le préfixe **tns** (*this namespace*) pour se référer à l'espace de nommage du document courant (ce n'est qu'une convention ; dans l'exemple considéré nous avons utilisé **typens**).

L'enveloppe SOAP (**soapenv**) définie l'enveloppe, l'en-tête, le corps, la gestion des erreurs ainsi que les attachements propres à un messages SOAP.

L'encodage SOAP (**soapenc**), pour sa part, définie l'élément **Array** et l'attribut **arrayType** utilisés pour encoder les vecteurs et les tableaux dans le style **Encoded**. Cette technique d'encodage est recommandée pour tout type de liste d'objets.

L'espace de nommage **XML Schema Instance** (**xsi**) définie des types d'attributs qui identifie les types de

données d'un élément. Par symétrie, l'espace de nommage **XML Schema namespace** (**xsd**) définie plusieurs types de données utilisés comme valeur pour l'attribut **xsi:type**. Par exemple, on peut citer les types **int**, **string** ou **double**.

Format des messages SOAP

Afin de mieux apprécier les différences entre les combinaisons RPC/Encoded, RPC/Literal et Document/Literal, nous allons mettre en oeuvre un Service Web en C/C++ et constater de visu ce que cela implique dans les messages SOAP. Nous pourrons alors corréler ces observations avec les fichiers WSDL correspondants.

Nous allons utiliser l'utilitaire **soapcpp2** issu du framework **gSOAP** présenté dans le numéro 72 du magazine pour nous aider à concevoir nos documents WSDL et à évaluer les différences de format (l'utilitaire génère aussi les trames XML de la requête et de la réponse).

Afin de visualiser les trames échangées, nous avons utilisé pour nos tests l'utilitaire **socat** :

```
# socat -v TCP4-LISTEN:8081,reuseaddr,fork TCP4:localhost:8080
```

Configurée de cette façon et accrochée dans une console, cette commande écoute sur le port 8081 et transfère le flux TCP reçu sur le port 8080 sur lequel écoute réellement notre serveur SOAP tout en affichant au passage sur la sortie standard les données ASCII transférées. En configurant nos clients pour écrire sur le port 8081, cela nous permet de comprendre dans le détail ce qui se passe.

Un usage intensif de cette approche peu intrusive est à conseiller lors du débogage d'un échange SOAP qui ne rend pas les résultats escomptés. En particulier, lors de l'utilisation de framework SOAP alternatifs (cf. hors .NET, Java/Axis ou gSOAP par exemple).

Le prototype de la fonction que nous allons mettre à disposition sur le réseau par l'intermédiaire d'un Service Web est le suivant (les directives gSOAP vont nous permettre de jouer sur le type **rpc/document** et le style **encoded/literal** du document) :

```
/// Fichier "myService.hh"
//gsoap ns service name: myService
//gsoap ns service location: http://myhome.org/myService
//gsoap ns service namespace: urn:myhome-myservice
//gsoap ns service style: rpc
//gsoap ns service encoding: encoded
//gsoap ns service method-action: getData "urn:myhome-myservice#getData"
int ns__getData(int a, int b, int &Result);
```

Il s'agit ni plus ni moins que d'appeler une fonction **getData** avec deux entiers en entrée et un entier en sortie : Rien de bien compliqué jusqu'à présent.

Pour générer le fichier WSDL, il faut saisir la commande suivante (l'option **-1** indique que l'on souhaite implémenter la spécification 1.1 de SOAP puisque c'est celle qui est supportée par la norme WS-I) :

```
# soapcpp2 -1 myService.hh
```

En modulant le type et le style du document, regardons maintenant ce que cela peut donner par type de combinaisons possibles.

RPC/Encoded

Si l'on exclu les parties communes aux trois combinaisons (à l'exception du composant **binding** qui spécifie le type **RPC/Encoded**) ainsi que le message de réponse (**getDataResponse**) afin d'alléger les exemples, un extrait du document WSDL nous propose le résultat suivant :

```
<message name="getDataRequest">
 <part name="a" type="xsd:int"/>
 <part name="b" type="xsd:int"/>
</message>

<portType name="myServicePortType">
 <operation name="getData">
  <input message="tns:getDataRequest"/>
  <output message="tns:getDataResponse"/>
 </operation>
</portType>
```

Dans ce modèle, les types de données sont directement déclarés dans les messages conformément à la spécification de l'encodage SOAP. Par contre, on n'utilise pas la section **types** de la spécification WSDL.

Un petit coup d'oeil sur le format du message échangé nous renseigne sur la façon dont le message sera encodé (pour éviter la surcharge, on évite de ré-afficher toute la déclaration des espaces de nommage) :

```
<?xml version="1.0"?>
<soapenv:Envelope ... xmlns:ns="urn:myhome-myservice">
 <soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ns:getData>
   <a xsi:type="xsd:int">...</a>
   <b xsi:type="xsd:int">...</b>
  </ns:getData>
 </soapenv:Body>
</soapenv:Envelope>
```

On retrouve directement le nom de la fonction ainsi que celles des variables, par contre, force est de constater que la re-déclaration des types de données alourdie un peu la requête inutilement car tout est déjà présent dans le fichier WSDL.

Vous apprendrez au passage que cela n'est pas le cas par défaut sous gSOAP qui ne re-déclare pas ses types de données dans les messages. En effet, et contrairement à ce qui est communément admis lorsque l'on parle du modèle RPC/Encoded, la spécification SOAP n'impose pas la déclaration des types de données (**`xsi:type`**) dans les messages SOAP du moment que les types des données peuvent être déduits du schéma XML présent dans le document WSDL.

Toutefois, afin de ménager la susceptibilité de certaines implémentations, gSOAP offre la possibilité de forcer la déclaration des types de données par l'utilisation de l'option « **`-t`** » (il est intéressant de noter que la spécification SOAP 1.1 inclus des exemples de messages SOAP qui sont partiellement ou pas du tout typés).

Il faut cependant savoir que cette fonctionnalité ne s'applique pas aux types de données standards (comme **`int`**, **`float`**, etc...) et qu'il faut alors penser à déclarer des types complexes si l'on veut profiter de la re-déclaration des types de données dans le corps des messages SOAP :

```
typedef int xsd__int;
int ns__getData(xsd__int a, xsd__int b, xsd__int &Result);
```

RPC/Literal

Un extrait du document WSDL nous donne le résultat suivant :

```
<types>
 <schema targetNamespace="urn:myhome-myservice"
  ...
  xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="a" type="xsd:int"/>
  <element name="b" type="xsd:int"/>
 </schema>
</types>

<message name="getDataRequest">
 <part name="a" element="ns:a"/>
 <part name="b" element="ns:b"/>
</message>

<portType name="myServicePortType">
 <operation name="getData">
  <input message="tns:getDataRequest"/>
  <output message="tns:getDataResponse"/>
 </operation>
</portType>
```

On remarque immédiatement la différence : Les types de données ne sont plus déclarés directement dans chaque message mais bien dans la section **types**.

Pour sa part, le message SOAP est similaire au précédent mais avec l'économie de la déclaration des types de données. Cependant, on retrouve toujours le nom de la fonction ainsi que l'intitulé de chaque variable :

```
<?xml version="1.0"?>
<soapenv:Envelope ... xmlns:ns="urn:myhome-myservice">
 <soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ns:getData>
   <a>...</a>
   <b>...</b>
  </ns:getData>
 </soapenv:Body>
</soapenv:Envelope>
```

Document/Literal

On va parler ici de la variante **Document/Literal wrapped** qui est devenue de facto le standard car proposée puis imposée par Microsoft au sein de son implémentation .NET. Cette façon d'écrire le schéma XML offre un avantage majeur sur le modèle original dans le cas d'une invocation de méthodes distantes (donc similaire à RPC) : Le nom de la méthode apparaît littéralement dans le message SOAP (sinon, il faut se référer au format WSDL pour retrouver ses petits).

Par contre, bien que tous les acteurs majeurs implémentent cette variante (par soucis de compatibilité), aucun organisme de normalisation n'a encore entériné cette approche ce qui est un peu troublant (pour information, l'utilitaire gSOAP **wsdl2h** permet d'importer les deux formes).

Cependant, rien ne nous interdit d'encapsuler un document XML arbitraire dans ce type de message SOAP (du moment que le schéma XML est présent dans le format WSDL) : Le message SOAP devient alors une simple enveloppe qui transporte des données (on ne parle plus alors d'invocation de méthodes distantes).

Pour en revenir à notre exemple, un extrait du document WSDL nous donne le résultat suivant :

```
<types>
 <schema targetNamespace="urn:myhome-myservice"
  ...
  xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="getData">
   <complexType>
    <sequence>
     <element name="a" type="xsd:int" minOccurs="1" maxOccurs="1"/>
     <element name="b" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
 </schema>
</types>

<message name="getDataRequest">
 <part name="parameters" element="ns:getData"/>
</message>

<portType name="myServicePortType">
 <operation name="getData">
  <input message="tns:getDataRequest"/>
  <output message="tns:getDataResponse"/>
 </operation>
</portType>
```

Cette fois, tout le message (encapsulé par la balise **soapenv:Body**) est déclaré par un schéma XML présent dans la section **types**. C'est intéressant car cela signifie qu'il n'y aura pas d'informations d'encodage dans les messages échangés et que vous pourrez alors directement valider le message SOAP par l'intermédiaire de ce schéma XML.

```
<?xml version="1.0"?>
<soapenv:Envelope ... xmlns:ns="urn:myhome-myservice">
 <soapenv:Body>
  <ns:getData>
   <ns:a>0</ns:a>
   <ns:b>0</ns:b>
  </ns:getData>
 </soepenv:Body>
</soepenv:Envelope>
```

Conclusion

Dans cet article, on aura appris qu'un document WSDL est tout simplement un ensemble de définitions permettant de spécifier les interfaces d'entrées/sorties d'un Service Web en mettant en concurrence deux formats (RPC/Document) et deux représentations possibles pour les données (Literal/Encoded) ; la mise en oeuvre du Service Web se conformant ensuite à un modèle d'interaction parmi quatre possibles (même si dans les faits cela n'est pas encore implémenté pour deux d'entre eux).

WSDL (en version 1.1) et SOAP (en version 1.1 ou 1.2) sont à l'heure actuelle indéfectiblement liés et représentent une norme industrielle solide et prometteuse même si les différentes combinaisons possibles complexifient sans doute inutilement leurs compréhensions et ne facilitent pas l'intéropérabilité immédiate entre implémentations.

Dans la jungle des standards, le norme WS-I apparaît comme une bouffée d'oxygène car limitant la spécification à un choix certes arbitraire (pourquoi supprimer l'encodage SOAP ?) mais sans doute nécessaire, vu l'exhaustivité du choix, pour imposer des limites et améliorer ainsi l'interopérabilité.

Sachez aussi que la version 1.2 de SOAP propose certaines modifications dont la plus dimensionnante (section 5) est le caractère désormais optionnel de l'encodage SOAP (pour coller à la norme WS-I ?) et de la représentation SOAP RPC (choix plus surprenant quand on connaît le nombre d'implémentations). Par contre, la norme WS-I ne prend pas encore en compte cette version donc l'impact sur les Services Web dans le court terme est réduit.

Suite à un échange avec **Robert Van Engelen** (l'auteur de gSOAP) qui a eu la gentillesse et la patience de répondre à quelque-une de mes questions, il m'a semblé intéressant de partager avec vous son point de vue (avec sa permission) : « *L'encodage SOAP utilisé au sein du modèle RPC/Encoding offre de multiples avantages par rapport au modèle Document/Literal car il propose une structuration des données qui est*

assez proche de ce que l'on attendrait du point de vue d'un langage de programmation. Les objets tableaux peuvent être sérialisées sans aucun problème (en utilisant id-ref). Sérialiser des objets en référence croisées ne présente aucun problème particulier (toujours avec id-ref) alors que cela est plus difficile avec le modèle Document/Literal (quoique pas impossible) depuis que, strictement parlant, le schéma doit inclure des définitions xsd:id et xsd:ref pour permettre le chaînage des éléments. Ce n'est pas aussi au bénéfice de l'intéropérabilité que d'avoir plusieurs façons de sérialiser des données en XML. Par exemple, est-ce que les types doivent être sérialisés comme attributs ou contenu dans les éléments ? Que faire des unions et <xs:choice> ? Comment gérer <xsd:any> ? [...] Cela serait une erreur que de déprécier l'encodage SOAP RC alors qu'il fonctionne si bien pour intégrer XML dans les langages de programmation. D'un autre coté, il semble raisonnable d'utiliser le modèle Document/Literal pour les méta-protocoles comme WS-Policy, WS-Notification, etc... ».

A vous de vous faire votre opinion mais, si le sujet vous intéresse, vous pouvez toujours consulter le lien [6] qui propose une intéressante discussion sur le sujet.

En conclusion, et au vu de tout ce que nous avons abordé, on peut facilement en tirer quelques enseignements :

- Le couple WSDL/SOAP a clairement pris le pas sur les autres implémentations de Services Web comme REST ou SOAP-RPC car plus complet (mais plus complexe) et bénéficiant d'une image plus moderne et du support de parrains prestigieux (IBM et Microsoft pour ne citer qu'eux),
- Le choix d'un framework SOAP de bonne qualité, conforme à la norme WS-I et disposant d'outils pour importer et exporter les documents WSDL, se révèle essentiel et engage souvent la réussite ou l'échec d'une migration vers les Services Web (souvent en reprenant une base de code, et donc un langage, existant) ; dans la même veine, toute personne qui met en oeuvre un Service Web se doit de connaître la structuration d'un document WSDL ainsi que le format des messages SOAP : Il arrive parfois de devoir encore mettre la main dans le cambouis (ce qu'adore faire un Linuxien !),
- On peut se demander quel crédit apporter à long terme à l'encodage SOAP vu que cela a été exclu de la norme WS-I et est rendu optionnel dans la version 1.2 de SOAP : Si la tendance ne s'inverse pas, on se dirige doucement mais sûrement vers une disparition de ce type d'encodage (quelques voix se sont élevées en ce sens) ou du moins son abandon progressif en environnement industriel (même si dans les faits c'est l'un des modèles actuellement les plus implémentés -en particulier dans le logiciel libre- et qu'il offre de nombreux avantages dont l'un des moindres n'est jamais que sa facilité d'implémentation),
- De même, si la représentation SOAP RPC (pourtant très répandue) est rendue optionnelle dans le version 1.2 du protocole SOAP, que pouvons-nous en déduire de son avenir ?
- Il faut s'attendre à ce que les normes WSDL et WS-I évoluent vers plus de simplicité (ne serait-ce que pour coller à la norme 1.2 de SOAP ou spécifier la variante **`wrapped`** au modèle **`Document/Literal`**) : Il faut donc suivre de prêt l'actualité,

Si l'on veut être pragmatique, il fait reconnaître que le modèle **`Document/Literal`** semble le moins sujet à obsolescence au vue des choix imposés dans les normes WS-I 1.0 et SOAP 1.2, et offre un certain nombre d'avantages non négligeables : Outre fédérer le traitement des messages autour de simples schéma XML, cette combinaison permet de traiter le modèle RPC et offre la possibilité d'échanger directement des documents XML et donc de gagner du temps (pas besoin de traitement particulier pour encoder puis décoder des données déjà en XML : Les applications qui modélisent déjà en XML n'ont absolument pas besoin du modèle de données SOAP). Son seul désavantage est qu'il sera peut-être plus difficile de faire dialoguer plusieurs implémentations SOAP entre elles : « Ce n'est pas aussi au bénéfice de l'intéropérabilité que d'avoir plusieurs façon de sérialiser des données en XML ».

Tout dépend au final de la palette technique proposée par vos outils de développement SOAP. Globalement, quelque soit le style sélectionné, cela n'aura que peu d'impact sur votre développement ; par contre cela pourra réduire ou améliorer vos capacités d'interopérabilités entre les différentes implémentations SOAP (en cas de besoin d'interopérabilité maximale, optez pour le modèle RPC/Encoded qui est le plus répandu).

Après la lecture de cet article, le langage de description WSDL ne devrait plus avoir de secrets pour vous (disons qu'il devrait être moins obscur) et sa compréhension devrait vous permettre de mieux appréhender et déployer le protocole SOAP (en particulier, en cas de problème d'intéropérabilité).

Vous pouvez vous amusez à essayer de comprendre la description (au format WSDL) du Service Web du moteur de recherche Google [7] qui est conçu à partir des briques **`RPC`** et **`Encoded`** : C'est une excellente occasion de mettre en pratique vos nouvelles connaissances.

Références

[1] Spécification WSDL 1.1 : http://www.w3.org/TR/wsdl

[2] Spécification SOAP : http://www.w3.org/TR/soap/

[3] Framework gSOAP : http://www.cs.fsu.edu/~engelen/soap.html

[4] Spécification XML-RPC : http://www.xmlrpc.com/

[5] Spécification REST : http://www.xfront.com/REST-Web-Services.html

[6] Débat sur l'encodage SOAP : http://www.w3.org/2005/06/21-xsd-user-minutes.html

[7] Accès à Google en Web Services : http://api.google.com/GoogleSearch.wsdl