Table des matières

Services Web et XML

Services Web et XML

Nicolas Gilson

Université de Namur, Namur, Belgique,
ngilson@info.fundp.ac.be

1 Introduction

L'Internet, devenu un acteur incontournable de l'économie, s'infiltre au sein des entreprises du monde entier sans la moindre discrimination. Celles-ci profitent des nouvelles technologies pour favoriser la communication avec leurs clients, leurs fournisseurs et leurs partenaires commerciaux.

Si, dans un premier temps, les entreprises ont été quelque peu bousculées par cette intrusion massive des nouvelles technologies, la tendance s'est inversée et, à présent, c'est la logique économique du "plus vite, meilleur et moins cher" qui dicte les transformations nécessaires de l'Internet.

Dans ce contexte, l'arrivée des Services Web correspond parfaitement aux attentes formulées par les entreprises. Le Web est, en fait, considéré comme un cadre de travail totalement indépendant des langages et des plates-formes où des opérations, appelées Services Web, peuvent être effectuées. Cette indépendance des Services Web vis-à-vis des langages et des plates-formes couplée à des protocoles de communication standardisés, tels que HTTP[1] ou encore SMTP[2], assure aux applications distribuées une interopérabilité maximale. De plus, ces applications supportant des Services Web s'avèrent être beaucoup moins lourdes à développer que des applications distribuées au-dessus d'un ORB[3].

Ces Services Web se présentent donc comme la solution permettant aux entreprises de fournir à moindre coût, au travers du Web, des services de qualité aux autres entreprises (B2B[4]) ou directement aux particuliers. Ils apportent donc un souffle nouveau aux entreprises désireuses de proposer leurs services au monde entier.

L'élément clef qui a permis le développement des Services Web est l'apparition de nouveaux standards technologiques basés sur XML[5] tels que SOAP[6], WSDL[7] et UDDI[8]. Nous étudierons ces standards plus en détail dans la suite de ce document.

[1] Hyper Text Transfer Protocol.
[2] Simple Mail Transfer Protocol.
[3] Object Request Broker.
[4] Business to Business.
[5] eXtensible Markup Language.
[6] Simple Object Access Protocol.
[7] Web Services Description Language.
[8] Universal Description, Discovery and Integration.

2 Etat de l'art

L'avènement du langage HTML en parallèle avec le protocole HTTP a engendré une révolution dans la manière dont les gens partagent l'information. Au-delà de cette révolution, une évolution des mentalités est aussi constatée puisque le monde des réseaux, regorgeant de systèmes propriétaires et de logiciels dépendant de leur plate-forme d'exécution, s'ouvre enfin aux standards.

Néanmoins, les limites du langage HTML, conçu principalement pour la représentation des données, se font très vite ressentir. En effet, il est impossible de définir avec rigueur la sémantique des données présentes au sein d'un fichier HTML. Heureusement, l'apparition du langage XML permet de remédier à cette carence grâce à l'utilisation de DTD[9] ou de "XML Schema". XML devient donc un élément de premier choix pour l'échange de données formatées au travers du Web.

En 1998, la firme UserLand Software saisit cette opportunité et utilise les deux standards XML et HTTP pour définir un protocole de communication permettant à différentes versions de leur logiciel, tournant distinctement sur des OS Windows et Mac, de communiquer. Ce protocole, première ébauche de XML-RPC[10], fait naître l'espoir d'une interopérabilité entre plates-formes de toutes origines reliées via l'Internet.

Ensuite, le géant de l'informatique Microsoft décide de continuer les travaux entamés en définissant un protocole dérivé de XML-RPC appelé SOAP. Il faudra attendre la version 1.1 de SOAP avant que SUN rejoigne Microsoft et participe activement au sein du "W3C[11] working group" à la définition de ce protocole. La version 1.2 est encore à l'heure actuelle à l'état de "draft". SOAP se présente comme la solution au problème d'interopérabilité entre la plate-forme .NET de Microsoft et la plate-forme J2EE de SUN...

Cependant, la création de Services Web nécessite d'autres types de standards technologiques que SOAP car celui-ci est uniquement limité à la communication entre composants distribués. Il est donc logique de voir apparaître d'autres standards tels que UDDI permettant la découverte des ces services dispersés sur le Web ou WSDL permettant la description de ces services.

En novembre 1999, une initiative de plus grande envergure ayant pour mission de fournir une infrastructure ouverte, basée sur XML et facilitant l'échange d'informations électroniques parmi des partenaires commerciaux, voit le jour. Il s'agit de ebXML[12]. Contrairement à ce que bon nombre de personnes imaginent, SOAP n'est pas un concurrent de ebXML, tout au contraire, il complète idéalement les travaux menés par les organisations UN/CEFACT[13] et OASIS[14]. En effet, ebXML est une suite de spécifications dans laquelle s'intègre parfaitement SOAP.

[9] Document Type Definition.
[10] eXtensible Markup Language - Remote Procedure Calling.
[11] World Wide Web Consortium.
[12] Electronic Business Extensible Markup Language.
[13] United Nations Centre for Trade Facilitation and Electronic Business.
[14] Organization for the Advancement of Structured Information Standards.

D'autres initiatives sont également à souligner. Nous notons notamment l'apparition de l'organisation sans but lucratif RosettaNet fondée en juin 1998 qui ne réunit pas moins de 400 compagnies dominantes dans la technologie de l'information et dont le but est de créer, implémenter et promouvoir de nouveaux standards ouverts pour le "e-business". Le "XML/EDI Group" fait aussi son apparition en proposant de combiner XML et EDI[15] pour développer une infrastructure destinée à l'échange de données formatées.

Finalement, dans un autre ordre d'idées, nous constatons l'intérêt croissant porté par les développeurs pour la technologie "peer to peer". Les services proposés par Napster, Gnutella ou encore KaAzA ont en effet un impact retentissant sur les internautes. C'est dans ce contexte, que le projet JXTA[16] a vu le jour. Il s'agit en fait d'une suite de protocoles, définis par une syntaxe dérivée de XML, établissant un réseau virtuel au-dessus des réseaux existants et standardisant la manière dont les "peers" se découvrent et communiquent.

3 Services Web : la technique

Afin de rendre un Service Web exploitable, nous devons avoir la capacité de définir ce service, de l'enregistrer dans une base de données, de le découvrir grâce à la base de données et finalement de l'invoquer et de l'exécuter. Heureusement, un bon nombre de standards, basés sur XML, rendant possible l'exploitation des Services Web, ont été développés : SOAP, WSDL, UDDI, ebXML... Nous aurons l'occasion de les passer en revue dans la suite de ce document.

3.1 XML

SOAP s'appuyant sur des concepts avancés du langage XML, il est pertinent d'approfondir quelque peu ces notions avant de plonger dans l'étude de ce protocole. Toutefois, nous ne nous attarderons pas sur les concepts de base de XML et nous encourageons le lecteur à se familiariser avec ces concepts[17] avant de poursuivre la lecture de ce document.

Au même titre que le langage HTML, XML est un sous-ensemble simplifié du langage SGML[18] et est destiné à décrire différents types de données à l'aide de "tags". Ces deux langages sont très similaires car ils partagent la même syntaxe pour définir les "tags" et les attributs. Toutefois, ils se différencient notamment par l'objectif qui leur est assigné. En effet, le langage HTML est essentiellement dédicacé au formatage de données et, par conséquent, ne permet pas une structuration efficace de celles-ci tandis que le langage XML est lui orienté vers la structuration des données. De plus, le langage XML peut être étendu selon la volonté du programmeur alors que le langage HTML est limité par le nombre

[15] Electronic Data Interchange.
[16] Jini XML Transaction Architecture.
[17] Vous trouverez, par exemple, des ressources sur le site suivant : http ://www.xml.org/xml/resources_focus_beginnerguide.shtml .
[18] Standard Generalized Markup Language.

de “tags” disponibles. XML permet au développeur de définir son propre vocabulaire pour décrire des structures de données particulières. Finalement, une dernière caractéristique différencie XML des autres langages utilisant des “tags” : XML est “case sensitive”.

La puissance du standard XML réside dans la simplicité avec laquelle il permet de décrire des données. Cette simplicité facilite l'interopérabilité entre systèmes totalement hétérogènes.

Un document XML est constitué de quatre parties distinctes : la déclaration, les règles, l'instance du document et la feuille de style. La déclaration[19] permet de définir la version de XML utilisée ainsi que le type d'encodage. Les règles définissent le type du document en spécifiant notamment les contraintes structurelles et les valeurs par défaut. Elles sont utilisées pour valider l'instance d'un document XML. Ces règles sont généralement écrites sous la forme d'un DTD. Cependant, depuis peu, le W3C propose une alternative aux DTDs : le “XML Schema”. Le “XML Schema”, au contraire des DTDs, est intégralement défini selon la syntaxe XML. Par ailleurs, il étend considérablement les capacités de ces DTDs : il supporte les “namespaces” que nous introduisons dans la suite du document, il permet la définition de types de données et il est extensible. Ces arguments en faveur des “XML Schemas” nous poussent à croire que les DTDs seront bien vite jetés aux oubliettes. La troisième partie du document XML, l'instance du document, contient notamment deux types de composants : des éléments et des attributs. Enfin, la feuille de style définit la présentation (XSL[20], CSS[21], DSSSL[22]) associée au document XML. Il existe un mécanisme de transformation (XSLT[23]) permettant d'obtenir la présentation (par exemple un document HTML) d'un document XML.

Tout développeur étant libre de définir son propre vocabulaire XML pour décrire des structures de données et XML étant appelé à devenir un acteur majeur de la communication entre systèmes distribués, le risque de voir apparaître des noms d'éléments identiques représentant des entités conceptuellement distinctes augmente dangereusement. Afin d'éviter ce problème, le W3C a étendu la syntaxe XML afin qu'elle supporte le concept de “namespaces”. Un “namespace” est un ensemble de noms dans lequel tous les noms sont uniques. Ces “namespaces” sont utilisés dans la syntaxe XML afin de fournir un contexte aux éléments. Ils permettent ainsi d'attacher une sémantique particulière à ces éléments en fonction de leur provenance. Les “namespaces” doivent également être représentés de manière unique afin de garantir que chaque élément puisse être identifié sans la moindre ambiguïté. A cet fin, chaque “namespace” XML doit se conformer à la syntaxe des URIs[24] définie dans le document RFC 2396. Ces URIs, généralement utilisées pour définir des ressources physiques (page

[19] Un exemple de déclaration : < `?XML version="1.0" encoding="UTF-8"?` > .
[20] eXtensible Stylesheet Language.
[21] Cascading Style Sheet.
[22] Document Style Semantics and Specification Language.
[23] eXtensible Stylesheet Language Transformations.
[24] Uniform Resource Identifier.

Web, fichier à "downloader"...), sont, dans ce cas, utilisées pour définir des ressources abstraites représentant le contexte dans lequel les éléments d'un document XML spécifique ont été définis par le développeur. Il ne faut donc pas confondre l'URI définissant un "namespace" avec une ressource distribuée accessible via le Web. A l'image du DNS[25] ou encore des "packages" JAVA, l'unicité des éléments est garantie par la concaténation du "namespace" de cet élément avec le nom de ce même élément. XML est donc nanti d'un mécanisme permettant de préfixer le nom des éléments avec une URI. Par soucis de clarté, les concepteurs de XML ont choisi d'associer l'URI à un identifiant afin de pouvoir utiliser cet identifiant, souvent plus intuitif et moins long, pour préfixer le nom des éléments. La syntaxe XML pour définir un "namespace" est la suivante : `xmlns :<prefix>=<namespace identifier>`. Ce préfixe peut être concaténé à un élément ou un attribut en les séparant par le caractère " : ". Le nom d'un élément préfixé avec un "namespace" est appelé un nom qualifié. Un exemple concret est sans doute utile pour appréhender ce nouveau concept. Imaginons que nous devions construire une application de gestion d'étudiants supportant plusieurs types de documents XML qui leur sont relatifs. Un des documents est constitué d'éléments "student" modélisant des étudiants en informatique et un autre est constitué d'éléments "student" modélisant des étudiants de l'école primaire. L'élément modélisant un étudiant en informatique est notamment composé d'un élément "language" indiquant le langage de programmation de prédilection de cet étudiant tandis que l'élément modélisant un étudiant de l'école primaire est composé d'un élément "language" indiquant cette fois la langue maternelle de l'élève. Ces deux éléments n'ont bien évidemment pas la même sémantique ! Comment permettre à notre application de les distinguer ? Vous l'aurez compris, il suffit d'associer ces éléments avec un "namespace" représentant soit l'étudiant en informatique soit l'étudiant de l'école primaire.

```
<p:student xmlns:p="http://info.fundp.ac.be/student/">
   <p:name>
      Nicolas
   </p:name>
   <p:language>
      Java
   </p:language>
</p:student>
```

3.2 SOAP

La version 1.2 de SOAP étant encore à l'état de "draft", nous nous concentrons essentiellement sur la version 1.1.

Principe général SOAP définit un mécanisme simple et léger pour échanger de l'information structurée et typée entre entités dans un environnement distribué décentralisé en utilisant le langage XML.

[25] Domain Name System.

Par analogie à d'autres systèmes où des objets distants s'échangent des messages, nous déduisons aisément l'essence du protocole SOAP. Ainsi, nous analysons que SOAP joue un rôle similaire au protocole IIOP[26] pour CORBA[27] ou encore au protocole JRMP[28] pour RMI[29]. SOAP est donc un protocole qui permet à des objets issus de différentes plates-formes et écrits dans différents langages d'intercommuniquer. Cependant, une différence majeure réside entre ces deux derniers protocoles et SOAP. Alors que IIOP et JRMP sont des protocoles "binaires", SOAP se base sur le langage XML pour encoder les données. Les messages échangés via ce protocole jouissent donc des avantages que lui procure le langage XML pour structurer les données. Ceci permet à des applications très faiblement couplées, n'ayant aucune connaissance a priori l'une de l'autre, de communiquer sans nécessairement faire usage d'un protocole compliqué.

Au-delà du message, il est nécessaire de disposer d'un protocole capable de véhiculer l'information sans que la mise en oeuvre de celui-ci ne soit trop exigeante. Le standard HTTP, défini à l'origine pour transporter du texte, se présente comme le candidat idéal. En effet, le système de requête et réponse de SOAP correspond parfaitement au système de requête et réponse de HTTP. De plus, HTTP est doté d'une base solide puisqu'il utilise comme protocoles sous-jacents IP[30] et TCP[31]. Cependant, il est tout à fait envisageable d'utiliser un autre protocole que HTTP, comme par exemple SMTP ou FTP[32], mais nous constatons que, pour l'instant, c'est HTTP qui a la cote. Dans la suite de ce document, nous traitons uniquement le cas où SOAP est lié à HTTP.

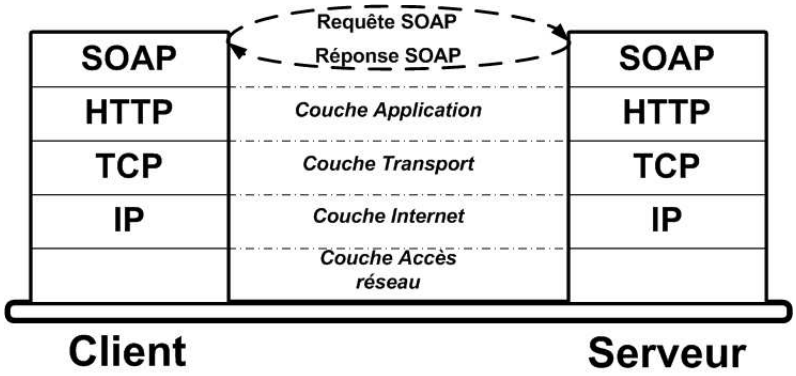

Fig. 1. Modèle en couche

[26] Internet Inter-ORB Protocol.
[27] Common Object Request Broker Architecture.
[28] Java Remote Method Protocol.
[29] Remote Method Invocation.
[30] Internet Protocol.
[31] Transmission Control Protocol.
[32] File Transfer Protocol.

Le développement d'un Service Web ne présente donc pas d'insurmontables difficultés techniques puisque la technologie utilisée est parfaitement maîtrisée. Nous retrouvons donc la logique client-serveur bien connue des développeurs de sites Web. Le client SOAP a pour rôle d'encoder les requêtes destinées à un service distant sous le format XML et de les envoyer via HTTP. Le serveur SOAP, quant à lui, est à l'écoute des messages SOAP qu'il interprète et convertit dans un langage compréhensible par l'objet auquel est destiné la requête. Ensuite, le serveur peut renvoyer la réponse encodée au format XML via HTTP. Ce processus permet donc à des applications écrites dans n'importe quel langage et exécutées sur n'importe quelle plate-forme de communiquer ; les efforts de traduction étant laissés au client et au serveur SOAP. Bien entendu, ces deux rôles ne sont pas exclusifs et il est tout à fait envisageable de rencontrer une application distribuée jouant à la fois le rôle de serveur et de client SOAP.

Fondamentalement, SOAP a été développé pour envoyer des requêtes "one-way". Toutefois, il profite du modèle d'échange de HTTP pour fournir des réponses aux requêtes. Un message est donc destiné à une transmission unidirectionnelle depuis un émetteur vers un récepteur, ceux-ci formant les noeuds du système. Dans un contexte d'environnement distribué un message peut transiter à travers un ensemble de noeuds qui forme un chemin. Un noeud peut être l'émetteur initial, le récepteur final, un émetteur ou un récepteur intermédiaire. Les noeuds peuvent se trouver sur une même machine ou sur des machines différentes. Ils représentent une ou plusieurs applications destinées à traiter les messages.

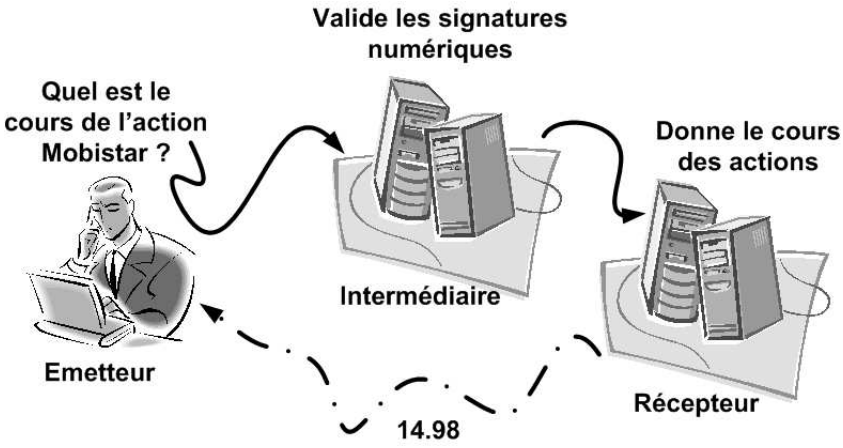

Fig. 2. Cotation en ligne

Pour illustrer le fonctionnement du protocole SOAP, nous avons choisi d'étudier le cas d'un Service Web permettant à des abonnés de consulter en ligne le cours

des actions. Ainsi, l'abonné (émetteur initial) envoie un message SOAP, comportant une requête, à ce service (récepteur final) afin de connaître le cours de l'action Mobistar. Ce message comporte également la signature numérique de l'abonné qui sera analysée par un organisme (récepteur intermédiaire) capable de valider de telles signatures.

Message SOAP SOAP définit à la fois les règles d'encodage des données qui traverseront les réseaux et les conventions de représentation des appels de procédure à distance. Il définit également l'enveloppe du message et fournit un mécanisme pour lier le message avec le protocole sous-jacent. Les règles utilisées pour la sérialisation au sein d'un message SOAP sont généralement dictées sur base d'un "XML Schema", supportant les types simples (int, float...) et composés (array, structure...), défini dans la section 5 de la spécification SOAP. Il est important de souligner que l'encodage des données selon ce "XML Schema" n'est absolument pas obligatoire et que les clients et serveurs sont libres d'utiliser des conventions différentes. Il existe également un mécanisme permettant de donner explicitement le type de chacune des données utilisées.

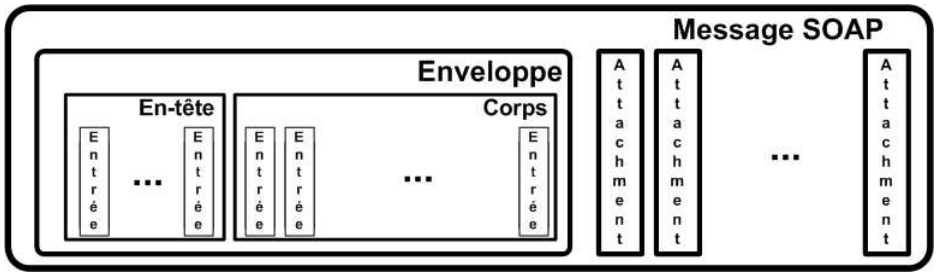

Fig. 3. Message SOAP avec "attachments"

Le message SOAP 1.1, entièrement défini au format XML, est contenu dans une enveloppe divisée en deux parties : l'en-tête optionnelle et le corps du message. Ces deux parties sont à leur tour décomposées en entrées. Il est envisageable d'accompagner le message SOAP de données au format "binaire" telles que des images JPEG. La version de SOAP avec "attachments" prévoit la possibilité d'associer ces données au message. Dans ce cas, les "attachments" sont référencés au coeur du message via une URI standardisée. Nous ne traitons pas cette version de SOAP avec "attachments" afin d'éviter de compliquer inutilement ce document.

L'en-tête de l'enveloppe fournit des informations concernant le contexte du message (authentification, transaction, paiement...). Ces méta-données peuvent être adressées à des intermédiaires ou directement au récepteur final. Le message SOAP pourra, comme dans notre exemple, transiter par un serveur intermédiaire

responsable de valider les signatures digitales avant d'être transmis au destinataire. SOAP a été conçu pour être extensible. Le développeur est libre de concevoir ses propres entrées et d'en définir la sémantique. Ainsi, "ebXML message service" exploite les fonctionnalités classiques de SOAP et définit ses propres entrées afin de produire un système de transfert de messages sécurisé.

Le corps de l'enveloppe contient, dans le cas d'une requête, les appels de procédure ainsi que les arguments correspondants. Dans le cas d'une réponse, il contient soit la réponse à la requête, soit un code d'erreur. Le corps de l'enveloppe est toujours destiné au récepteur final.

Considérons la requête simplifiée que l'abonné génère pour obtenir le cours de l'action Mobistar :

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header>
         <SOAP-SEC:Signature
            xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
            SOAP-ENV:actor="digitalsignature.org/Validity"
            SOAP-ENV:mustUnderstand="1">
               MC0CFFrVLtRlk=...
         </SOAP-SEC:Signature>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
         <m:getLastTradePrice xmlns:m="http://stocks.com/StockQuotes">
            <symbol>Mobistar</symbol>
         </m:getLastTradePrice>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ce message comprend un simple appel de méthode "getLastTradePrice" prenant comme unique paramètre un "string" symbolisant la société cotée en bourse. L'élément "Envelope" est la racine de ce document XML. Pour construire un appel de procédure distante au sein du corps du message SOAP, nous devons disposer de trois informations : l'URI de l'objet cible représentant la destination sur laquelle la méthode sera invoquée, le nom de la méthode et l'ensemble des paramètres de cette méthode. L'URI est utilisée comme "namespace" pour qualifier l'élément représentant la méthode. Dans notre exemple, l'objet distant est représenté par l'URI "http ://stocks.com/StockQuotes". La procédure distante est ainsi modélisée sous la forme d'une structure XML tout à fait indépendante du message SOAP.

L'attribut "encodingStyle" est nécessaire pour indiquer les règles utilisées pour la sérialisation. Cet attribut peut apparaître dans tout élément du message, sa portée étant limitée au contenu de l'élément ainsi qu'aux fils de cet élément. L'URI associé à cet attribut identifie les règles de sérialisation. Dans cet exemple,

l'URI "http ://schemas.xmlsoap.org/soap/encoding/"[33] indique que les règles de la section 5 de la spécification SOAP 1.1 sont en vigueur dans tout le message.

Les "namespaces" abondent dans les messages SOAP et pourraient effrayer le lecteur non coutumier de ce genre de notation. Cependant, une fois appréhendés, les "namespaces" facilitent la compréhension du message SOAP. Pour rappel, ce concept permet de grouper des éléments afin de définir leur contexte et d'éviter les collisions entre éléments de même nom possédant une sémantique différente. Il permet notamment de dissocier les éléments et attributs spécifiques au message SOAP[34] tels que "Envelope", "Header", "encodingStyle" de ceux spécifiques à l'application[35] hébergeant l'objet cible sur lequel la méthode sera invoquée. Ce mécanisme garantit donc la simplicité grâce à la modularité. Un dernier avantage des "namespaces" en relation avec SOAP est qu'ils permettent d'effectuer un contrôle de version. En l'occurrence, l'URI "http ://schemas.xmlsoap.org/soap/envelope/" caractérise la version 1.1 de SOAP.

Analysons maintenant l'en-tête du message. L'élément signature constitue l'unique entrée de cet en-tête. L'attribut "actor", toujours associé à une URI, identifie le noeud responsable du traitement de cette entrée. Cette URI peut éventuellement servir à router le message SOAP vers un noeud approprié. Mais SOAP ne spécifie pas de mécanisme de routage. L'absence de cet attribut impliquerait que l'entrée serait destinée au récepteur final. Le noeud traite tous les blocs d'en-tête qui lui sont destinés et, s'il s'agit du noeud final, il traite aussi le corps. Dans le cas d'un noeud intermédiaire, le noeud doit supprimer les blocs d'en-tête le concernant (éventuellement en rajouter d'autres pour fournir une information). Ensuite, il transmet le message au noeud suivant. Il est à noter que dans la spécification SOAP rien n'empêche les noeuds de traiter les entrées de manière non séquentielle. Finalement, l'attribut "mustUnderstand" est également remarquable. Lorsque celui-ci est positionné à 1, cela signifie que le traitement de l'entrée est obligatoire. Dans ce cas, le noeud responsable du traitement ne peut l'ignorer et doit soit effectuer le traitement, soit générer une erreur SOAP. Ceci favorise une évolution robuste de SOAP car tout changement de la sémantique des entrées de l'en-tête ne peut être ignoré par un noeud lorsque l'attribut mustUnderstand vaut 1.

Considérons maintenant la réponse renvoyée par le Service Web :

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <SOAP-ENV:Body>
         <m:getLastTradePriceResponse
            xmlns:m="http://stocks.com/StockQuotes">
               <Price>14.98</Price>
```

[33] Bien que cela ne soit pas obligatoire, cette URI référence un "XML Schema".

[34] Ces éléments et attributs appartiennent au "namespace" "http ://schemas.xmlsoap.org/soap/envelope/" .

[35] Ces éléments et attributs appartiennent au "namespace" "http ://stocks.com/StockQuotes" .

```
        </m:getLastTradePriceResponse>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

La convention SOAP pour structurer la réponse dans le corps du message est d'ajouter "Response" à la fin du nom de la méthode invoquée. Le corps du message peut également servir de récipient pour transporter les erreurs éventuelles survenues lors des divers traitements du message. SOAP définit une seule entrée du corps du message[36] : l'élément "Fault". Cet élément, pouvant apparaître, au plus, une fois dans le corps du message, définit quatre éléments fils : l'élément "faultcode" permettant un traitement automatisé de l'erreur, l'élément "faultstring" destiné à l'homme, l'élément "faultactor" spécifiant sous la forme d'une URI la source de l'erreur et l'élément "detail" relatif aux erreurs de traitement du corps du message. Dans le cas de SOAP 1.1, l'élément "faultcode" peut prendre quatre valeurs : "VersionMisMatch", "MustUnderstand", "Client" et "Server". La compatibilité de ces codes avec des versions futures de SOAP est garantie grâce à l'usage des "namespaces" servant à qualifier ces valeurs. Les entrées de l'élément "detail" sont spécifiques à l'application et sont donc développées de manière indépendante.

Lien avec HTTP Afin de transmettre le message SOAP au travers du réseau, il est nécessaire de le lier avec un protocole de transport. Même si la spécification SOAP 1.1 permet l'usage d'autres protocoles que HTTP, c'est communément HTTP qui est utilisé. SOAP suit naturellement le modèle de requête-réponse de HTTP. Cependant, les intermédiaires éventuels spécifiés dans l'en-tête du message HTTP ne sont pas les mêmes que les intermédiaires SOAP ; ceux-ci étant spécifiés par l'attribut actor dans l'en-tête de l'enveloppe. Ces intermédiaires HTTP ne sont donc pas habilités à traiter le message SOAP.

Une requête SOAP est, en général, liée avec une requête HTTP POST en spécifiant comme type de média : "text/xml". Le client HTTP doit impérativement inclure le champ "SOAPAction" dans l'en-tête HTTP. L'URI associée à ce champ indique simplement l'intention du message SOAP. Ceci peut par exemple permettre à un "firewall" de filtrer les requêtes SOAP. Néanmoins, la communauté SOAP débat encore le véritable usage de ce champ.

La requête de notre abonné est donc "emballée" dans une requête HTTP de la forme suivante :

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
```

Enfin, la sémantique du code utilisé par HTTP pour indiquer le statut de la requête au client est conservée dans le cadre d'une réponse SOAP.

[36] Cet élément appartient donc au même "namespace" que celui des éléments "Envelope", "Body"...

3.3 WSDL

En parallèle avec le protocole SOAP d'autres standards émergent et s'insèrent parfaitement dans le cadre des Services Web. Ces standards favorisent l'automatisation des phases préliminaires à l'invocation des Services Web. Ainsi, par exemple, le langage WSDL est exploité par le fournisseur de services pour donner une description technique de ses services. Cette description peut être interprétée, sans la moindre intervention humaine, par une entité distribuée désireuse de trouver et ensuite d'invoquer un service particulier.

WSDL se base sur le format XML pour décrire des Services Web sous la forme d'un ensemble de points finaux appelés ports. Un port est décomposé en deux parties distinctes. La première partie contient la définition abstraite des opérations ainsi que des messages contenus dans ces opérations. Une opération est ainsi considérée comme une action dont on peut extraire deux types de messages[37] : une requête et une réponse. Tandis que la seconde partie lie les messages à un protocole de communication concret. Cette distinction entre partie abstraite et concrète est requise afin de favoriser la réutilisabilité de cette définition en combinaison avec d'autres protocoles.

Un port est donc décomposé en un ensemble d'opérations. Chaque port est lié à un protocole de communication comme par exemple SOAP 1.1 ou SMTP. Enfin, le port représente une adresse à laquelle on peut joindre l'entité supportant les opérations définies.

Dans notre exemple, la définition abstraite de l'opération "getLastTradePrice" est décrite de la façon suivante :

```
<message name="getLastTradePriceInput">
   <part name="body" element="TradePriceRequest"/>
</message>
<message name="getLastTradePriceOutput">
   <part name="body" element="TradePrice"/>
</message>
<portType name="StockQuotePortType">
   <operation name="getLastTradePrice">
      <input message="getLastTradePriceInput"/>
      <output message="getLastTradePriceOutput"/>
   </operation>
</portType>
```

L'opération est donc décomposée en deux messages : la requête et la réponse. La définition des données échangées dans ces messages est décrite au préalable dans un "XML Schema". Ainsi, les types de données "TradePriceRequest" et "TradePrice" contiennent respectivement un élément "symbol" de type "string" et un élément "price" de type "float". Ce "XML Schema" définit également le "namespace"[38] représentant le contexte de l'opération.

[37] Un troisième type de message peut être extrait d'une opération. Il s'agit d'un message d'erreur.

[38] En l'occurrence, "http ://stocks.com/StockQuotes"

Ensuite, l'opération est liée avec le protocole de communication HTTP :

```
<binding name="StockQuoteSoapBinding" type="StockQuotePortType">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="getLastTradePrice"/>
</binding>
```

Finalement, le nouveau port ainsi créé est attaché à notre service Web en spécifiant le point de communication[39] nécessaire pour accéder à cette opération :

```
<service name="StockQuoteService">
   <port name="StockQuotePort" binding="StockQuoteSoapBinding">
      <soap:address location="http://www.stockquoteserver.com/StockQuote">
   </port>
</service>
```

La location correspond à l'adresse à laquelle nous avons installé le serveur SOAP. Si le développeur désire créer un site miroir, il peut réutiliser cette description. Seuls le nom et la location du service devront être mis à jour.

3.4 UDDI

Un autre standard émergeant et favorisant l'automatisation des phases préliminaires à l'invocation des Services Web, est UDDI. Ce standard est supporté par un grand nombre d'industries en provenance du monde de l'informatique. UDDI, signifiant "Universal Description, Discovery, and Integration", permet d'enregistrer de l'information concernant les types de services disponibles et les personnes pouvant utiliser ces services. Il fournit donc un mécanisme de publication et de recherche de services. UDDI s'appuie sur des standards tels que HTTP, SOAP et WSDL.

Il existe des registres publics[40], suivant le standard UDDI, où, à la fois, des fournisseurs de services peuvent publier leurs services et des demandeurs de services peuvent trouver ces services.

UDDI Version 2.0 définit un ensemble de spécifications : une spécification de l'interface d'un registre définissant environ une trentaine de messages SOAP pouvant être utilisés pour effectuer des requêtes et publier des services, une définition de la structure XML des données contenues dans ces messages, une description du processus de réplication des données contenues dans un registre...

Un registre UDDI est décomposé en pages blanches, jaunes et vertes. Les pages blanches incluent des informations tels que le nom et l'adresse d'une entreprise. Les pages jaunes catégorisent les entreprises en fonction du type d'affaires qu'elles effectuent. Finalement, les pages vertes décrivent le genre de services offerts ainsi que les techniques de communication à employer pour joindre ces services.

[39] Une URL pour HTTP, une adresse e-mail pour SMTP...

[40] Actuellement, il existe deux "UDDI Business Registry Version 1". Il s'agit des registres de Microsoft et de IBM. D'autres "UDDI Business Registry Version 2" sont d'ores et déjà disponibles en version Beta.

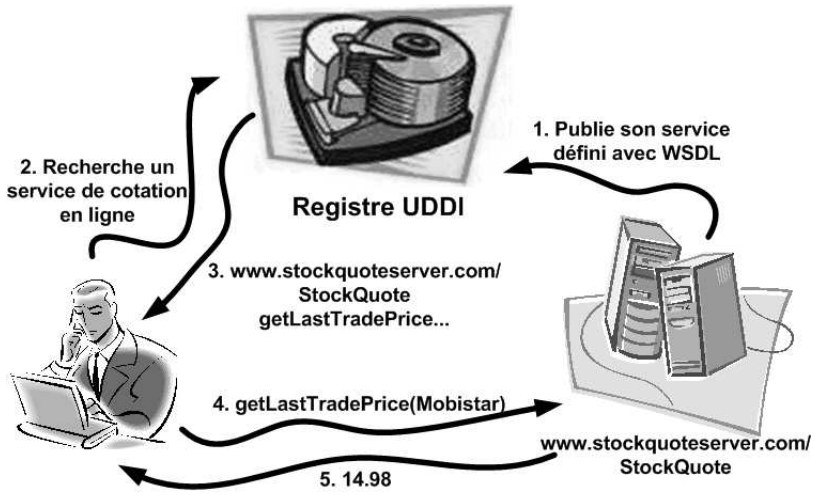

Fig. 4. SOAP, WSDL & UDDI

Nous concluons cette section concernant la technologie des Services Web avec un cas d'application intégrant les technologies étudiées. Dans cet exemple, un Service Web de cotation en ligne publie ces services dans un registre UDDI. A cette fin, il utilise le protocole SOAP pour envoyer la description de ces services définie selon le langage WSDL. Ensuite, un client désireux de connaître le cours de l'action Mobistar s'adresse via SOAP au registre UDDI pour découvrir les fournisseurs de ce type de service. Il reçoit ainsi l'information, spécifiée dans le langage WSDL, nécessaire pour invoquer le service désiré. Finalement, il s'adresse directement à ce Service Web via SOAP pour obtenir le cours de l'action Mobistar...

4 Critiques

Le succès des Services Web et donc de SOAP est dû à la volonté de leurs concepteurs de produire quelque chose à la fois d'utile et surtout de simple. Ceci répond à la demande grandissante des programmeurs pour des systèmes moins complexes à mettre en oeuvre. Ainsi, le protocole SOAP est très concis et facilement compréhensible pour les programmeurs et les organisations ne pouvant s'offrir le luxe d'attaquer d'autres technologies.

Ces concepteurs oublient ainsi volontairement de traiter des aspects inhérents à une infrastructure distribuée comme la sécurité, les transactions...Ils se contentent juste de laisser des portes entrouvertes pour que l'on puisse éventuellement étendre ce protocole en fonction des besoins ressentis par les programmeurs d'applications distribuées. Il incombe donc à d'autres associations de définir les standards de mise en oeuvre des aspects non traités par SOAP. De

plus, SOAP n'intervient pas non plus dans la définition des processus d'échange d'informations nécessaires au commerce électronique. C'est d'ailleurs une des raisons pour lequelles les Services Web proposés pour l'instant sont pour le moins simplistes...

Heureusement, ebXML – dont la mission est de fournir une infrastructure ouverte basée sur XML et facilitant l'échange d'informations électroniques d'une manière sûre et logique entre différentes parties – comble ces lacunes. ebXML a mis sur pied une suite de spécifications grâce auxquelles les compagnies disposent maintenant de méthodes standards pour échanger des messages électroniques, conduire des relations commerciales, définir et enregistrer des processus commerciaux. ebXML utilise en fait SOAP comme simple moyen de transport de messages électroniques et profite de l'en-tête de l'enveloppe SOAP pour intégrer de nouvelles fonctionnalités.

Au même titre que CORBA[41], DCOM[42], J2EE, les Services Web s'intègrent dans la lignée des "middlewares". En tant que protocole conçu pour le transport d'informations structurées et typées, SOAP peut être comparé au protocole IIOP de CORBA.

IIOP utilise le format binaire CDR[43] pour transférer les messages alors que SOAP utilise le langage XML. Le format texte des messages SOAP implique que chaque chiffre est encodé comme un caractère et par conséquent cela nécessite, en général, beaucoup plus de place pour stocker les données chiffrées au sein d'un message SOAP. Toutefois, ceci importe peu étant donné le débit actuel des réseaux sauf dans le cas d'infrastructures sans fil où la capacité du réseau est très limitée. Le processus d'emballage et de désemballage des données au format XML demande également plus de ressources.

Les Services Web sont placés au devant de la scène en tant que nouvelle technologie qui va révolutionner la manière dont les applications distribuées coopèrent. Toutefois, il est justifié de se demander si l'architecture orientée services sous-jacente aux Services Web est réellement une révolution. En effet, cette architecture basée sur le mécanisme "publier, trouver et lier" existe déjà au sein CORBA depuis des années ! Les objets CORBA publient leurs services dans des "Naming" et "Trader Services" grâce auxquels d'autres objets peuvent trouver ces services et ensuite délivrer des requêtes aux objets correspondants. La réelle révolution induite par les Services Web réside dans l'emploi des standards HTTP et XML et surtout dans la simplification du processus de communication.

Cependant, cette simplification engendre des limitations. Nous avons déjà constaté la faiblesse des Services Web, à laquelle essaie de remédier ebXML, en ce qui concerne la définition d'aspects clefs comme la sécurité ou encore les transactions. Seul UDDI est à l'heure actuelle parfaitement spécifié et accepté en tant que service standard exploitable par les Services Web.

En outre, nous constatons que les Services Web, au contraire de CORBA, n'intègrent pas du tout le modèle objet. Durant ces dix dernières années, le para-

[41] Common Object Request Broker Architecture.
[42] Distributed Component Object Model.
[43] Common Data Representation.

digme OO[44] est pourtant devenu un élément incontournable. Il est pratiquement impossible pour les développeurs familiers avec ce nouveau paradigme de jongler avec les concepts d'interface, d'héritage ou encore de polymorphisme lorsqu'ils utilisent la technologie SOAP...En fait, SOAP ne permet pas d'identifier un objet distant en tant que tel, il se contente juste de traduire de simples requêtes. La gestion du cycle de vie d'un objet est donc hors du contrôle de SOAP. Il est également impossible de gérer des objets par référence. De plus, même avec l'usage d'une API[45] SOAP, la transparence des appels distants laisse à désirer. Le programmeur ne jouit pas du niveau d'abstraction procuré par les "stubs" de RMI ou de CORBA. SOAP souffre également de l'absence d'une API standard permettant de traduire des données SOAP en objet ou inversément ce qui compromet fortement la portabilité du code. En fait, il n'existe pas de "mapping" unique entre objet et données SOAP tel qu'on le connaît dans CORBA avec le fameux langage IDL[46]. Finalement, le "type checking" ne peut s'effectuer que durant l'exécution de l'application ce qui rend les programmes utilisant SOAP moins aisés à "debugger". Tant de lacunes qui nous amènent à nous demander pourquoi préférer les Services Web à CORBA ?

Une explication, autre que la simplicité de mise en oeuvre, pouvant justifier l'engouement pour les Services Web au lieu de CORBA réside dans une faiblesse du protocole IIOP. En effet, CORBA souffre de l'absence d'un port standard pour le protocole IIOP. Le passage de "firewalls" est donc un problème pour les concepteurs d'ORBs. SOAP, au contraire, développé au-dessus du protocole HTTP, profite du port standard 80 pour traverser sans la moindre résistance les "firewalls".

A la lumière de ces critiques, il est possible d'envisager un usage pertinent de ces technologies. Ainsi, il semble que CORBA soit parfaitement adapté dans le cadre d'une intégration de systèmes au sein d'un environnement contrôlé tel qu'un intranet tandis que les Services Web seraient plutôt plus adaptés dans le rôle d'intégrateur entre les différentes technologies : CORBA, J2EE, .NET...Cependant, la réalité est tout autre et si SOAP peut en effet jouer un rôle intégrateur, il se présente également en tant que concurrent sérieux, grâce à sa flexibilité et son extensibilité, sur le marché des "middlewares".

[44] Object Oriented.
[45] Application Program(ming) Interface.
[46] Interface Definition Language.

Références

1. La spécification SOAP 1.1, www.w3.org/TR/SOAP.
2. La spécification SOAP avec “attachments”, www.w3.org/TR/SOAP-attachments.
3. La spécification SOAP 1.2, www.w3.org/TR/soap12.
4. La spécification SOAP “security extensions”, www.w3.org/TR/SOAP-dsig.
5. La spécification SOAP non officielle en français, reseau.sdbdc.org/soap/soap.htm.
6. Box, D. : A Brief History of SOAP, avril 2001, www.xml.com/pub/a/2001/04/04/soap.html.
7. Clements, T. : Overview of SOAP, Sun, janvier 2002, developer.java.sun.com/developer/technicalArticles/xml/webservices/.
8. Sommers, F. : A birds-eye view of Web services, JavaWorld, janvier 2002, www.javaworld.com/javaworld/jw-01-2002/jw-0125-webservices.html.
9. Shin, S. : Building Blocks of Web Services : SOAP, WSDL, UDDI, and ebXML , Sun Technology Audiocasts, developer.java.sun.com/developer/onlineTraining/webcasts/madrid/sshin2.html.
10. Modi, T. : Clean up your wire protocol with SOAP, Part 1, JavaWorld, 2001, www.javaworld.com/javaworld/jw-03-2001/jw-0330-soap.html.
11. Rommel, J. : Will Web services jump-start the software slump ?, JavaWorld, www.javaworld.com/jw-08-2001/jw-0831-webservice.html.
12. McLaughlin, B. : ebXML : Not just another acronym, JavaWorld, www.javaworld.com/javaworld/javaone00/j1-00-ebxml.html.
13. Le site ebXML officiel, www.ebxml.org/.
14. La spécification WSDL 1.1, www.w3.org/TR/wsdl.
15. Les spécifications UDDI 2.0, www.uddi.org/specification.html.
16. Tost, A. : UDDI4J lets Java do the walking, JavaWorld, www.javaworld.com/javaworld/jw-08-2001/jw-0824-uddi.html.
17. Winer, D. : Foreword for the XML-RPC book, 2001, www.xmlrpc.com/stories/storyReader$1726.
18. Randel, B., Skonnard, A. : A Guide to XML and Its Technologies, MSDN Library, Septembre 1999, msdn.microsoft.com/library/default.asp ?url=/library/en-us/dnxml/html/xmlguide.asp.
19. Skonnard, A. : Understanding XML Namespaces, MSDN Library, Juillet 2001, msdn.microsoft.com/msdnmag/issues/01/07/xml/xml0107.asp.
20. Sundaresan, N. : XML / Java Technology / Design Patterns, JavaOne Sun’s 2000 Worldwide Java Developer Conference, servlet.java.sun.com/javaone/javaone2000/pdfs/TS-555.pdf.
21. Schmidt, D.C., Vinoski S. : CORBA and XML – Part 3 : SOAP and Web Services, C/C++ Users Journal C++ Experts Forum, 2001, www.cuj.com/experts/1910/vinoski.htm.
22. Marcato, D. : Distributed Computing With SOAP, VisualC++, avril 2000, www.devx.com/upload/free/features/vcdj/2000/04apr00/dm0400/dm0400-1.asp.
23. Waterhouse, S., Doolin, D.M., Kan, G., Faybishenko, Y. : Distributed Search in P2P Networks, sun Microsystems.
24. Lurie, J., Belanger, R.J. : The great debate : .Net vs. J2EE, JavaWorld, mars 2002, www.javaworld.com/javaworld/jw-03-2002/jw-0308-j2eenet.html.