Intergiciel et Construction d'Applications Réparties

Chapitre 4

Les Services Web

L'accès aux systèmes d'information s'appuie aujourd'hui de plus en plus sur des technologies Internet. Les efforts de standardisation dans ce contexte ont accentué l'engouement des personnes et des organisations (aussi bien académiques, qu'industrielles, commerciales, ou institutionnelles) pour l'utilisation de l'Internet et ont permis l'émergence des services Web comme support de développement des applications accessibles par Internet. Ainsi, les technologies associées aux services Web sont devenues incontournables pour le développement d'applications interagissant les unes avec les autres par le biais de l'Internet. Nous proposons dans ce chapitre de faire un point sur le sujet des services Web. L'objectif de la discussion est de traiter des aspects conceptuels de la modélisation des services aussi bien que des aspects liés à leur implantation.

La section 4.1 commence par donner quelques définitions et principes nécessaires à la compréhension de la suite. Puis la section 4.2 discute de la modélisation des interactions supportées par un service Web en abordant les deux points de vue, structurel et comportemental. La section 4.3 donne un éclairage sur les principaux standards pour la description de services Web en les distinguant selon leur capacité à traiter des aspects structurels, comportementaux et non-fonctionnels des services. Dans la section 4.4 nous décrivons quelques outils existants pour la mise en œuvre de services Web. Finalement la section 4.5 donne, en forme de conclusion, quelques perspectives faisant actuellement l'objet de recherches.

4.1 Services Web : historique, définition, et principes

Plusieurs définitions des services Web ont été mises en avant par différents auteurs. Ci-dessous, nous citons une définition généralement acceptée et fournie par le consortium W3C [W3C-WSA-Group 2004] :

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL[1]). Other systems interact with*

[1] *Web Service Description Language*

> *the Web service in a manner prescribed by its description using SOAP*[2] *messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

D'autres définitions similaires adoptent un point de vue plus général en ne prescrivant pas l'usage exclusif de WSDL pour la description des interfaces de service ou celui de SOAP pour le traitement des messages[3]. Ainsi par exemple, certains auteurs proposent l'utilisation de messages XML (sans les extensions apportées par SOAP) échangés directement sur le protocole HTTP. De façon similaire, la définition officielle du consortium W3C fait spécifiquement référence au protocole HTTP, mais en pratique, d'autres protocoles sont aussi utilisés (voir Section 4.4).

Une étude plus approfondie des points communs partagés par les différentes définitions et par les usages qui sont faits des services Web, permet de dégager au moins deux principes fondamentaux :

- Les services Web interagissent au travers d'échanges de messages encodés en XML. Dans certains cas, les messages sont plutôt transmis dans un encodage binaire, même s'ils sont généralement produits et consommés en XML ou en utilisant des interfaces de programmation basées sur les concepts d'*élément* et d'*attribut*co définis par le modèle dit « XML Infoset ».
- Les interactions dans lesquelles les services Web peuvent s'engager sont décrites au sein d'interfaces. La définition W3C restreint la portée des interfaces des services Web aux aspects fonctionnels et structurels, en utilisant le langage WSDL qui, essentiellement, ne permet de décrire que des noms d'opérations et des types de messages. Cependant, d'autres auteurs proposent d'aller plus loin et de considérer aussi les aspects comportementaux et non-fonctionnels comme nous le verrons dans la section 4.3.

Par rapport à d'autres plates-formes pour le développement d'applications distribuées telles que CORBA (voir le chapitre 1) et Java RMI (voir le chapitre 5), l'une des différences primordiales est que les services Web n'imposent pas de modèles de programmation spécifiques. En d'autres termes, les services Web ne sont pas concernés par la façon dont les messages sont produits ou consommés par des programmes. Ceci permet aux vendeurs d'outils de développement d'offrir différentes méthodes et interfaces de programmation au-dessus de n'importe quel langage de programmation, sans être contraints par des standards comme c'est le cas de CORBA qui définit des ponts spécifiques entre le langage de définition IDL[4] et différents langages de programmation. Ainsi, les fournisseurs d'outils de développement peuvent facilement différencier leurs produits avec ceux de leurs concurrents en offrant différents niveaux de sophistication. Par exemple, il est possible d'offrir des environnements pour des méthodes de programmation de services Web minimalistes au-dessus de plates-formes relativement légères comme c'est le cas de NuSOAP (pour le langage PHP) ou d'Axis (voir la section 4.4). En revanche, des plates-formes plus lourdes telles que BEA WebLogic, IBM Websphere ou .Net (voir le chapitre 6) offrent un environnement pour des méthodes de développement très sophistiquées.

[2] *Simple Object Access Protocol*

[3] Il faut noter que WSDL et SOAP sont des standards définis par le consortium W3C, ce qui explique que la définition ci-dessus essaye d'imposer ces standards.

[4] *Interface Definition Language*

Les principes à la base des services Web, bien que simples, expliquent leur adoption rapide : le nombre d'outils associés au services Web a connu un essor considérable dans une période de temps réduite, assurant ainsi que les technologies associées aux services Web seront utilisées pour de nombreuses années à venir.

La description explicite des interactions entre les services Web, est aussi un point fort des services Web. Cependant, des efforts de recherche et de développement plus approfondis sont nécessaires afin de réellement exploiter ce principe dans des outils ou des méthodes de développement. En particulier, comme nous le discuterons dans la section 4.3, l'usage des descriptions comportementales et non-fonctionnelles des services requièrent davantage d'attention de la part de la communauté de recherche.

4.2 Modélisation de services Web

4.2.1 Points de vue dans la modélisation de services

Les services Web interagissent les uns avec les autres par le biais d'envois de messages. Dans ce cadre, une interaction entre deux services, un client et un fournisseur, est décrite par un envoi de message par le client et la réception de ce message par le fournisseur. De cette interaction peuvent découler d'autres interactions, au cours desquelles les rôles de client et de fournisseurs peuvent s'inverser. Une séquence de telles interactions est appelée une *conversation*.

De ce fait, la modélisation de services Web met en jeu la description d'interactions et de leurs interdépendances, aussi bien du point de vue structurel (types de messages échangés) que du point de vue comportemental (flot de contrôle entre interactions). De plus, ces interactions peuvent être vues d'une façon globale, c'est-à-dire du point de vue d'un ensemble de services participant à une collaboration, ou de façon locale, c'est-à-dire du point de vue d'un service donné. Selon cette dimension, il est possible de distinguer trois types de modèles de services :

– *Chorégraphie.* Une chorégraphie décrit, d'une part un ensemble d'interactions qui peuvent ou doivent avoir lieu entre un ensemble de services (représentés de façon abstraite par des *rôles*), et d'autre part les dépendances entre ces interactions.
– *Interface.* Une interface décrit un ensemble d'interactions dans lesquelles un service donné peut ou doit s'engager et les dépendances entre ces interactions. Dans une interface, les interactions sont décrites du point de vue du service concerné sous forme d'*actions communicationnelles*, c'est-à-dire d'envois et de réceptions de messages.
– *Orchestration.* Une orchestration décrit un ensemble d'actions communicationnelles et d'actions internes dans lesquelles un service donné peut ou doit s'engager (afin de remplir ses fonctions) ainsi que les dépendances entre ces actions. Une orchestration peut être vue comme un raffinement d'une interface qui inclut des actions qui ne sont pas nécessairement pertinentes pour les clients du service mais qui servent à remplir les fonctions que le service fournit et doivent donc être prises en compte dans son implantation.

A titre d'exemple, la figure 4.1 présente une chorégraphie correspondant à un processus de gestion de commandes dans le domaine commercial. Afin de laisser la description de cette chorégraphie à un niveau abstrait, nous utilisons des diagrammes d'activité UML dans lesquels les actions correspondent à des interactions entre des rôles correspondant à des types de service (client, fournisseur et entrepôt en l'occurrence). Pour simplifier, pour chaque échange de message le diagramme montre soit l'envoi de ce message, soit la réception, mais pas les deux. Par exemple, dans l'activité « Commande », nous faisons référence à l'envoi du Bon de Commande (BdC) par le client au fournisseur. Bien entendu, le fournisseur doit exécuteur l'action correspondant à la réception de ce message, mais nous ne présentons pas cette action duale dans le diagramme d'activités.

Il est à remarquer aussi que les diagrammes d'activité permettent de modéliser des aspects comportementaux, et en particulier des flots de contrôle. Ils ne permettent pas cependant de modéliser des aspects structurels tels que les types de données décrivant le contenu des messages échangés. Ceux-ci peuvent être décrits au travers de diagrammes de classe mais nous omettons ces détails car ils ne contribuent pas à la compréhension du concept général de chorégraphie.

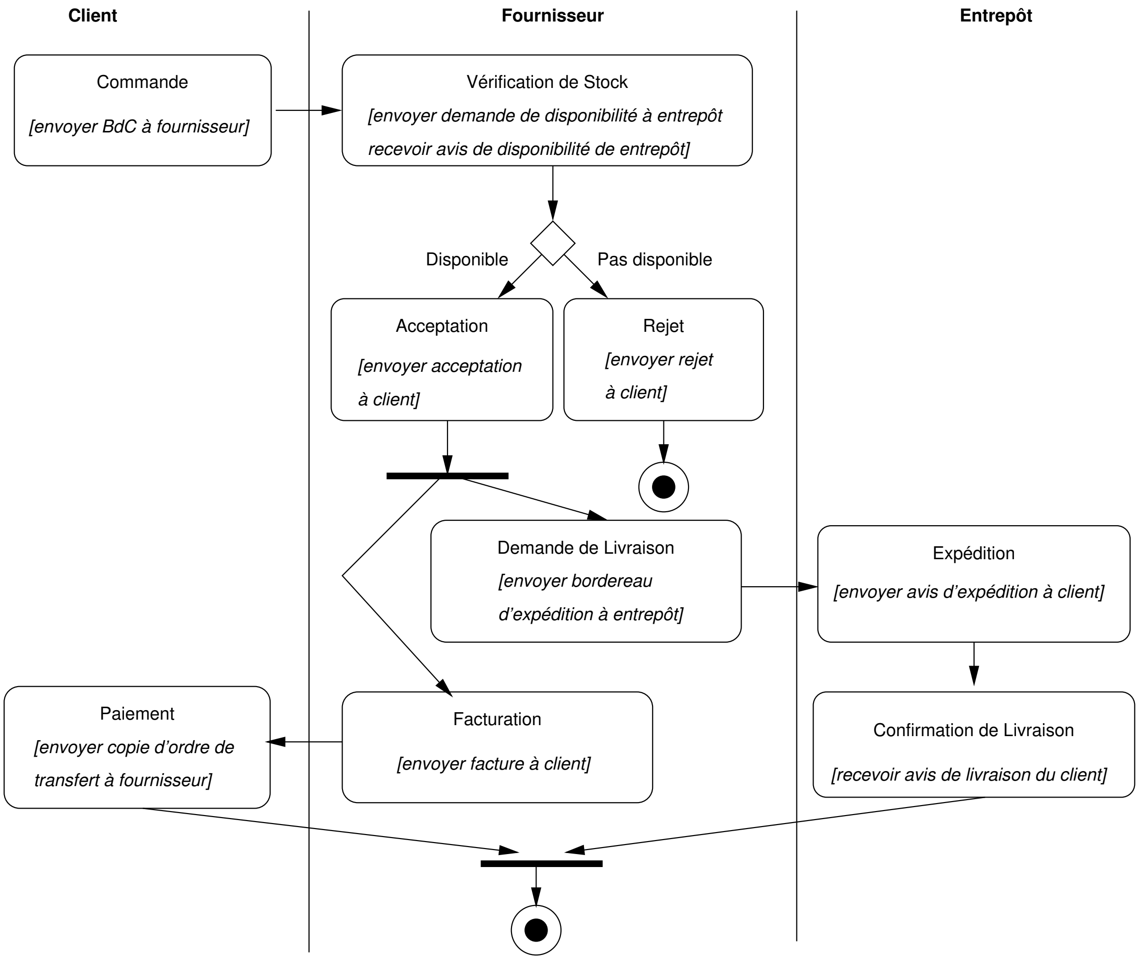

Figure 4.1 – Une chorégraphie

Il faut noter que la chorégraphie présentée dans la figure 4.1 n'adopte le point de vue d'aucun des services (ou plus précisément des rôles) concernés. En effet, cette chorégraphie inclut des interactions entre le client et le fournisseur, entre le fournisseur et son entrepôt, et entre l'entrepôt et le client. Lorsqu'on passe à des étapes ultérieures dans le cycle de développement de services, on s'intéresse à voir des interactions du point de vue d'un service spécifique, comme par exemple un service censé jouer le rôle de fournisseur dans l'exemple en question. Dans ce cas, on utilise la notion d'interface (voir chapitre 2). Une interface de service décrit un ensemble d'interactions dans lesquelles un service donné peut ou doit s'engager. Le concept d'interface est dérivé des langages de programmation et des intergiciels à base d'appels de procédures tels que CORBA. Mais alors que traditionnellement le concept d'interface est associé surtout à une description structurelle (c'est-à-dire une description des paramètres d'une opération), dans le cadre des services Web on s'intéresse à des descriptions plus riches, incorporant des aspects comportementaux. A titre d'exemple, l'interface correspondant au rôle de fournisseur dans la chorégraphie de la figure 4.1 est représentée sur la figure 4.2. Dans cette dernière figure, chaque élément de la séquence correspond à une réception (par exemple `Recevoir bon de commande`) ou un envoi de message (par exemple `Envoyer demande de disponibilité`). L'ensemble des types de messages échangés modélisent la dimension structurelle de l'interface alors que le flot de contrôle exprimé par la séquence modélise la dimension comportementale de l'interface.

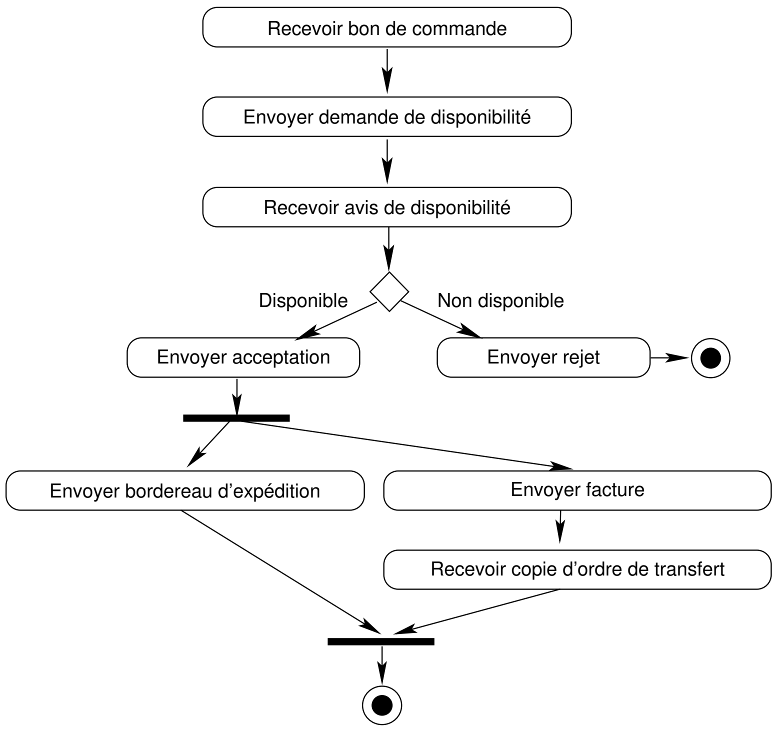

Figure 4.2 – Interface correspondant au rôle de fournisseur dans la chorégraphie de la figure 4.1.

Selon que l'on se réfère à l'interface qu'un service existant est capable de fournir, ou de l'interface qu'un service est censé fournir dans le cadre d'une chorégraphie, on parle d'*interface fournie* ou d'*interface requise* (voir chapitre 2). L'interface présentée dans la figure 4.2 correspond à l'interface requise du fournisseur, dérivée de la chorégraphie donnée dans la figure 4.1. Parfois, l'interface d'un service existant est identique ou compatible avec l'interface requise pour participer à une chorégraphie. Cependant, ceci n'est pas toujours le cas. A titre d'exemple, considérons le cas où l'interface représentée sur la figure 4.3 est l'interface fournie par un service que l'on voudrait utiliser pour remplir le rôle de fournisseur dans la chorégraphie de la figure 4.1. Dans ce contexte, l'interface fournie diffère de l'interface requise en deux points. Premièrement, alors que dans l'interface requise le service peut répondre par une acceptation ou un rejet, dans l'interface fournie il répond avec le même type de message quel que soit le cas. Deuxièmement, dans l'interface requise, l'ordre d'expédition et la facture sont envoyés dans n'importe quel ordre, alors que dans l'interface requise l'envoi de la facture et la réception de l'ordre de paiement précèdent l'envoi de l'ordre d'expédition. Dans ce contexte, il est nécessaire de réconcilier ces deux interfaces. Ce problème est connu sous le nom d'adaptation de services et fait l'objet de plusieurs efforts de recherche (voir section 4.5.1).

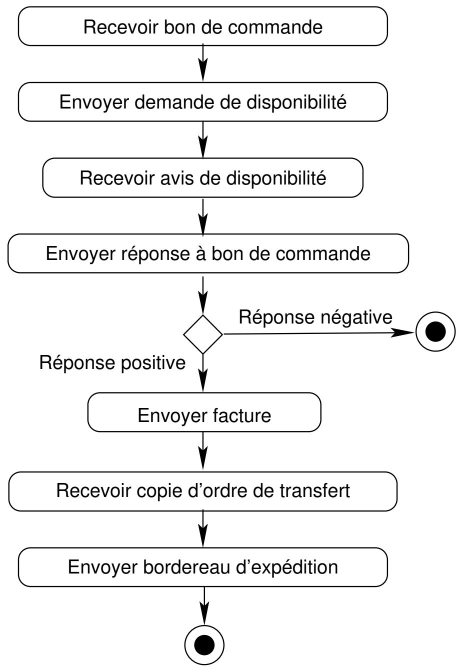

Figure 4.3 – Exemple d'interface fournie.

Enfin, la figure 4.4 présente une orchestration correspondant à l'interface de la figure 4.3. Cette orchestration rajoute une action (indiquée en lignes pointillées) qui n'est pas exposée dans l'interface. En réalité bien sûr, on peut attendre qu'une orchestration introduise davantage d'actions non-exposées, mais nous nous limitons ici à présenter une seule action non-exposée pour éviter de compliquer l'exemple. Plus généralement, une orchestration est un raffinement d'une ou plusieurs interfaces, interfaces qui à leur tour, peuvent être dérivées d'une chorégraphie. Certains auteurs considèrent ainsi qu'une orchestration est une implantation d'un service (voir par exemple [Peltz 2003]). Ceci peut faire penser qu'une orchestration est un programme exécutable, ce qui n'est pas tout à fait exact. Une orchestration peut être conçue comme un modèle décrivant le mode opératoire d'un service. Ce modèle peut manquer de quelques détails pour être exécutable, auquel cas il doit être raffiné davantage pour pour pouvoir être ensuite implanté dans un langage de programmation tel que Java ou BPEL (voir Section 4.4).

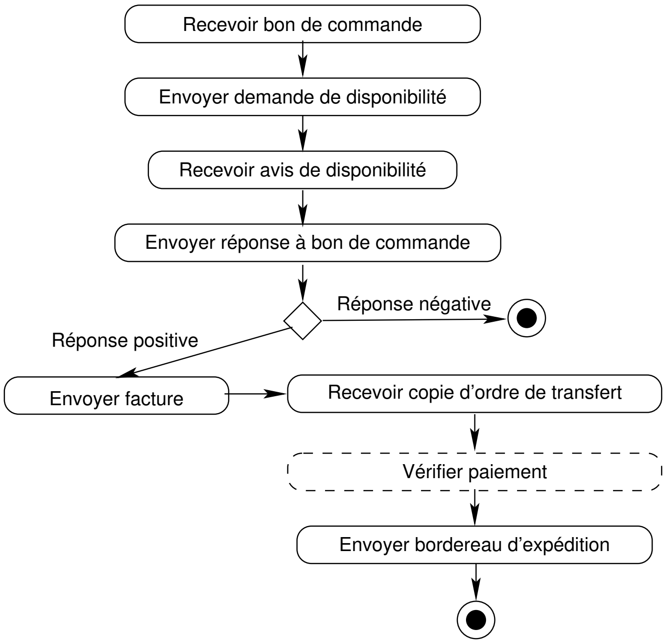

Figure 4.4 – Exemple d'orchestration raffinant l'interface de la figure 4.3.

4.2.2 Relations entre points de vue

Les points de vue discutés ci-dessus ne sont pas indépendants les uns des autres. Ces interdépendances peuvent être exploitées au cours du processus de développement pour effectuer des contrôles de cohérence entre les points de vue et pour générer du code. Par exemple, une interface requise peut constituer le point de départ de la génération d'une « ébauche d'orchestration »qui est ensuite complétée par la description détaillée des opérations internes pour enfin être raffinée jusqu'à obtenir le modèle complet d'orchestration. Ceci a pour avantage de faire coïncider l'interface fournie d'un service avec l'interface telle qu'elle est requise par les services qui l'utilisent. De plus, à condition de masquer ses opérations qui ne doivent pas être exposées, un modèle d'orchestration existant peut être utilisé pour générer son interface fournie. La cohérence de l'interface fournie résultante avec l'interface requise, peut ainsi être contrôlée. De cette manière, il est possible de détecter des situations où l'ordre des envois effectués par un service ne correspond pas à l'ordre attendu par les services destinataires avec lesquels il est censé collaborer. La résolution de ces incompatibilités s'appuie, soit sur la modification du modèle d'orchestration, soit sur la fourniture d'une enveloppe devant effectuer la médiation entre l'interface fournie et celle requise.

En résumé, les utilisations d'un modèle d'orchestration sont :

– La génération de l'interface requise pour chacun des services amenés à collaborer. L'intérêt de cette interface requise, au cours de la phase de développement des services a été souligné plus haut. Par exemple, étant donnée la chorégraphie décrite

figure 4.1, il est possible de dériver les interfaces requises des services fournisseur, client et entrepôt.
– Le contrôle, au moment de la conception, de la cohérence de l'interface existante d'un service avec le modèle de chorégraphie auquel il participe. Ainsi, la capacité du service à assumer le rôle qu'il est sensé jouer dans cette chorégraphie est vérifiée.
– La génération d'une ébauche d'un modèle d'orchestration pour chaque participant. Cette ébauche peut ensuite être détaillée dans la perspective, pour chaque participant, de remplir son rôle.

Pour une définition formelle des concepts de chorégraphie, d'interface et d'orchestration, le lecteur peut se référer à [Dijkman and Dumas 2004]. Il faut noter que ces concepts apparaissent parfois avec des noms différents dans la littérature. Par exemple, Casati et al. [Alonso et al. 2003] utilisent le terme « protocole de collaboration »pour se référer au concept de chorégraphie. De façon similaire, en ebXML [UN/CEFACT and OASIS 1999], une famille de standards pour le développement de services pour le commerce électronique, l'interface fournie est appelée « profil de protocole de collaboration »(*collaboration protocol profile* en anglais) alors que l'interface requise est appelée « accord de protocole de collaboration »(*collaboration protocol agreement*).

Le lecteur intéressé par les problèmes liés à la détection d'incompatibilités entre interfaces fournies et interfaces requises peut se référer à [Martens 2005]. La résolution de telles incompatibilités fait l'objet de recherches en cours (voir par exemple [Benatallah et al. 2005a, Fauvet and Ait-Bachir 2006]).

4.3 Description de services Web

Dans cette section sont discutées les questions liées à la description des interfaces des services. Cette description peut porter sur les aspects structurels (section 4.3.1) et/ou comportementaux (section 4.3.2) des services, aussi bien que sur leurs aspects non-fonctionnels (section 4.3.3).

Les langages présentés dans cette section sont fondés sur XML (*eXtensible Markup Language*) [W3C-XML]. XML est un standard de représentation de données semi-structurées. Dans un document XML, la structure des données est fournie par le biais de l'utilisation de balises (comme en SGML *Standard Generalized Markup Language* [Goldfard 1990], mais en s'affranchissant des aspects liés à la présentation des données). Cette structure n'est pas aussi rigide que celle d'une base de données relationnelle par exemple. Dans un document XML, les données peuvent être définies selon un schéma, mais cela n'est pas obligatoire et le schéma peut laisser quelques parties partiellement spécifiées.

Une problématique généralement associée à la description de services est celle de leur publication et de leur découverte. Un standard proposé pour s'attaquer à cette problématique est UDDI [OASIS UDDI 2005]. UDDI définit une interface de programmation pour publier des descriptions de services dans des répertoires dédiés, pour soumettre des requêtes à base de mots-clés sur ces répertoires et pour naviguer au travers des descriptions obtenues par le biais de ces requêtes. Étant donnée l'existence de moteurs de recherche sophistiqués aussi bien pour des Intranets qu'au niveau de l'Internet tout entier, et d'autres technologies pour la gestion de répertoires tels que LDAP [Koutsonikola and Vakali 2004], la valeur ajoutée apportée par UDDI est difficile à identifier. Peu d'architectures à base de services s'appuient aujourd'hui sur UDDI et le futur de ce standard est incertain. Par conséquent, nous ne détaillons pas UDDI dans ce chapitre, malgré les associations souvent faites entre UDDI et la problématique de la description de services.

4.3.1 Description fonctionnelle et structurelle : WSDL

WSDL (*Web Services Description Language*) [W3C-WSD-Group] est un langage de la famille XML permettant de décrire les types de données supportés et les fonctions offertes par un service Web. L'objectif est de fournir la description, en XML, des services indépendamment de la plate-forme et du langage utilisés et sous une forme que des personnes ou des programmes peuvent interpréter. Les descriptions WSDL sont en fait l'équivalent des interfaces IDL (*Interface Definition Language*) de CORBA par exemple.

Dans le langage WSDL, un service est vu comme une collection de messages pour les échanges et d'une collection de points d'entrée. Un point d'entrée consiste en la description abstraite d'une interface et de son implantation. La description abstraite contient : (i) la définition des messages qui sont consommés et générés par le service (les entrées et les sorties), et (ii) la signature des opérations offertes par le service. La mise en correspondance (*implementation binding*) entre l'interface et son implantation est fournie. Elle contient essentiellement l'indication du protocole utilisé pour échanger des messages avec le service (par exemple SOAP au-dessus de HTTP) et les associations entre la description de l'interface abstraite du service et les types de messages supportés par le protocole de communication sous-jacent (par exemple SOAP). La description WSDL de l'interface donnée figure 4.2 du service de réception d'une commande offert par le fournisseur est ébauchée ci-dessous.

L'élément `definitions` constitue la racine du document et fournit les espaces de noms.

```
<?xml version="1.0"?>
<definitions name="RecevoirBdC"
          targetNamespace="http://www.yothuyindi.fr:8080/exemple/fournisseur.wsdl"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema/"
          xmlns:wns="http://www.yothuyindi.fr:8080/exemple/fournisseur.wsdl"
          xmlns:xsdl="http://www.yothuyindi.fr:8080/exemple/fournisseur.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Les types de données, paramètres d'entrée et/ou de sortie sont éventuellement décrits ensuite en XMLSchema [W3C-XMLSchema]. La description d'une commande est décrite ci-dessous comme une date et une liste non vide de produits à commander, chacun dans une quantité non nulle donnée.

```
<types>
  <schema targetNamespace="http://www.yothuyindi.fr:8080/exemple/fournisseur.xsd">
    <xsd:complexType name="Commande"><xsd:sequence>
      <xsd:element name="dateCommande" type="xsd:date">
      <xsd:element name="LigneDeCommande" minOccurs="1" maxOccurs="unbounded">
        <xsd:complexType><xsd:sequence>
          <xsd:element name="RéférenceProduit" type="xsd:string"/>
          <xsd:element name="Quantité" type="xsd:positiveInteger"/>
        </xsd:sequence></xsd:complexType></xsd:element>
    </xsd:sequence></xsd:complexType>
```

```
...</schema>
</types>
```

Vient ensuite la description de la liste des définitions des messages échangés indépendamment de l'implantation du service et du protocole utilisé. Ici, le document décrit deux messages pour l'interaction : le premier message est la requête reçue par le service, l'autre est l'accusé de réception renvoyé. Les valeurs fournies pour les attributs `element` sont des noms de types XML Schema ou définis dans la partie `types` ci-dessus.

```
<message name="SoumissionBdC">
   <part name="body" element="xsdl:Commande"/>
</message>
<message name="RécépisséBdC">
   <part name="body" element="xsd:string"/>
</message>
```

Les opérations offertes par le service sont exposées par le biais de points d'entrée. Un point d'entrée (élément `portType`) fournit la signature de chaque opération et doit par la suite être associé à une implantation particulière (voir plus loin la description de la partie `binding`). WSDL permet l'existence de plusieurs points d'entrée dans un même document. Ainsi, la même opération peut être rendue disponible au travers d'implantations différentes.

```
<portType name=pt_RecevoirBdC>
   <operation name="op_Commande">
      <input message="wsn:SoumissionBdC">
      <ouput message"=wsn:RécépisséBdC">
   </operation>
   <operation name=...
   ...
</portType>
```

Dans la description ci-dessus les valeurs des attributs `message` font référence à un message défini plus haut (élément `message` du document WSDL).

La définition de la signature d'une opération s'appuie sur trois sous-éléments possibles : `input` pour le message en entrée, `output` pour celui en sortie et `fault` pour un message d'erreur émis par le service. Les combinaisons possibles de ces sous-éléments permettent de décrire divers modèles d'opérations :

- Réception de message (sous-élément `input`) : le service reçoit un message envoyé par un client ;
- Réception - émission (sous-élément `input` suivi de `output` et éventuellement de `fault`) : le service reçoit une requête sous forme d'un message puis émet une réponse sous forme d'un retour de message. C'est le modèle classique RPC (*Remote Procedure Call*, voir chapitre 1) ;
- Émission - retour (sous-élément `output` suivi de `input` et éventuellement de `fault`) : le service envoie une requête à un client (suite à une inscription préalable) sous forme d'un message ; le client répond sous forme d'un message de retour.
- Émission (sous-élément `output`) : le service envoie un message (d'alerte par exemple) à un client.

La dernière partie du document fixe la mise en correspondance entre chaque point d'entrée (un ensemble d'opérations) et son implantation, et permet de définir quels services sont associés à quelle mise en correspondance. Dans le fragment donné ci-dessous, l'implantation des points d'entrée s'appuie sur SOAP.

```
<binding name="bd_opCommande" type=pt_RecevoirBdC>
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="op_Commande">
      <soap:operation soapAction="http://www.yothuyindi.fr/exemple/op_Commande"/>
      <input>
         <soap:body
             use="literal"
             namespace="http://www.yothuyindi.fr/exemple/fournisseur.xsd"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </input>
      <output>
         <soap:body
             use="literal"
             namespace="http://www.yothuyindi.fr/exemple/fournisseur.xsd"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </output>
   </operation>
</binding>

<service name=ServiceFournisseur>
   <documentation> Service de réception des commandes </documentation>
   <port name="PortSoumissionCommande" binding="bd_opCommande">
      <soap:address location="//www.yothuyindi.fr/exemple/fournisseur"/>
   </port>
</service>
<!-- fermeture de la balise racine -->
</wsdl:definitions>
```

Le document final est obtenu par la concaténation des extraits fournis ci-dessus.

Comme le suggère l'exemple traité ici, la production de documents WSDL est un travail fastidieux dont le programmeur peut être déchargé grâce à l'utilisation d'outils pour la génération automatique de la description WSDL. Ces outils prennent en entrée la description du service sous forme d'une classe Java (ou une archive jar) ou d'un objet COM.

4.3.2 Description comportementale

La portée de WSDL est limitée la description des types des données incluses dans les messages qu'un service est capable de recevoir ou d'émettre. Dans de nombreuses applications, ces descriptions uniquement structurelles n'offrent pas assez d'information sur les contraintes régissant les interactions dans lesquelles un service peut ou est censé s'engager. Dans certains cas, ces contraintes sont assez simples, comme par exemple : « le fournisseur n'envoie le bordereau de livraison qu'après avoir reçu la confirmation du paiement ». D'autres fois, ces contraintes peuvent être relativement complexes. Par exemple, ci-dessous est donnée une description résumée d'une des interfaces définie par le consortium RosettaNet[5] pour le traitement de bons de commandes :

> Lorsqu'un fournisseur reçoit un « Bon de Commande (BdC) »de la part d'un client, il doit répondre avec un « Récépissé de BdC ». Par la suite, le fournisseur enverra zéro, une ou plusieurs « Réponses à BdC »au client jusqu'à avoir donné une réponse (une acceptation ou un rejet) pour chacune des lignes du BdC. Au cours de ce processus, le fournisseur peut recevoir une « Annulation de BdC »de la part du client, dans ce dernier cas le fournisseur répond avec un « Récépissé d'Annulation de BdC ». Dès lors, le fournisseur ne doit plus envoyer de « Réponses à BdC »au client.

En WSDL, il serait possible de définir chacun des messages échangés dans le protocole ci-dessus. Cependant, il ne serait pas possible d'exprimer que le fournisseur peut envoyer « plusieurs réponses à un BdC jusqu'à recevoir une annulation ». Dans des systèmes tels que CORBA, ces dépendances comportementales doivent être documentées en langue naturelle et c'est ensuite aux développeurs d'assurer que : (i) les services remplissent les contraintes exprimées dans leur documentation ; et (ii) les applications utilisent les services conformément avec leur documentation.

Plusieurs initiatives de recherche, de développement et de standardisation sont en cours afin de définir des langages de description de services prenant en compte des aspects comportementaux, et d'utiliser ces descriptions « comportementales »lors du développement et de l'exécution des services. De telles initiatives s'appuient sur des résultats de recherche dans le domaine des systèmes à base de composants [Yellin and Strom 1997] et des systèmes orientés-processus [Dumas et al. 2005]. Deux efforts de standardisation dans ce domaine sont sur le point d'aboutir : le langage de description de processus métiers exécutables pour services Web (*Web Services Business Process Execution Language*, BPEL) [Andrews et al. 2003] et le langage de description de chorégraphies de services Web (*Web Services Choreography Definition Language*, WS-CDL) [Kavantzas et al. 2005].

BPEL est un langage fondé sur XML et permettant de décrire aussi bien des interfaces comportementales (au sens défini dans la section 4.2) que des orchestrations complètement exécutables. En quelques mots, BPEL permet de décrire des actions communicationnelles (envois et réceptions de message dont les types sont décrits en WSDL et XML Schema), et de lier ces actions au travers d'opérateurs de flot de contrôle (par exemple la séquence, le choix, l'itération, et les clauses *throw/catch* pour le traitement des exceptions). Nous montrons ci-dessous, un extrait du code BPEL correspondant à l'interface RosettaNet énoncée plus haut :

```
<sequence>
  <receive operation="BdC"/>
  ...
  <while ...>
    <invoke operation="reponse-a-BdC"/>
  </while>
  ...
  <onMessage operation="annulation-de-BdC" ...>
    <throw faultName="annulationEnCours"/>
  </onMessage>
```

[5] Voir http ://www.rosettanet.com.

```
  ...
  <catch faultName="annulationEnCours">
    <invoke operation="RecepisseAnnulation-de-BdC" .../>
  </catch>
  ...
</sequence>
```

Dans cet extrait, on distingue deux types d'actions communicationnelles : `receive` pour la réception, et `invoke` pour l'envoi. Ces actions sont composées en utilisant les opérateurs de séquencement et d'itération (`sequence` and `while`) que l'on retrouve dans les langages de programmation impératifs. En outre, à tout moment (clause `onMessage`) un message de type `annulation-de-BDC` peut être reçu et alors une exception est levée et ensuite attrapée au travers des constructions `throw` et `catch` que l'on retrouve aussi dans des langages de programmation. On peut remarquer que le code BPEL ci-dessus se limite à décrire des actions communicationnelles et leurs dépendances. Si l'on voulait utiliser BPEL pour implanter le service correspondant, beaucoup plus de détails seraient nécessaires.

En tant que langage d'implantation de services, BPEL est incorporé dans plusieurs outils de construction d'applications à base de services (voir Section 4.4). Cependant, en tant que langage de description d'interfaces, l'outillage autour de BPEL est presqu'inexistant. Quelques travaux de recherche montrent comment des interfaces comportementales décrites en BPEL peuvent être utilisées pour la vérification statique [Fu et al. 2004] ainsi que pour le suivi et analyse d'exécutions de services, que ce soit en temps réel [Baresi et al. 2004] ou a posteriori [Aalst et al. 2005]. Cependant, ces techniques n'ont pas encore été adoptées dans des outils commerciaux.

Ce dernier commentaire s'applique également à WS-CDL, qui essaye d'aller plus loin que BPEL en s'attaquant non seulement à la description d'interfaces comportementales, mais aussi à la description de chorégraphies à différents niveaux de détails, allant de la modélisation conceptuelle jusqu'à l'implantation. L'idée de WS-CDL est que les chorégraphies décrites dans un langage semi-exécutable, peuvent être vérifiées statiquement, testées par simulation, et dans le cas des chorégraphies définies au niveau le plus détaillé elle peuvent être utilisées pour générer automatiquement les interfaces correspondant à chacun des rôles mis en jeu. A ce jour, les outils de développement de services basés sur WS-CDL sont très loin d'avoir atteint leur maturité[6]. Pour un aperçu et une analyse critique de WS-CDL, voir [Barros et al. 2005b].

4.3.3 Description d'aspects non-fonctionnels : « WS-Policy »

Les services Web étant généralement développés par des équipes indépendantes, ont besoin d'être décrits de façon précise selon des conventions standards, et de telle manière que leurs descriptions puissent être utilisées au maximum pour automatiser le processus de développement et de déploiement des futurs services devant interagir avec un service existant. WSDL et BPEL permettent de décrire les opérations fournies par un service Web, les types de données des messages devant être échangés pour invoquer ces opérations, et les dépendances comportementales entre ces opérations. Cependant, ils ne permettent pas de décrire des aspects non-fonctionnels des services tels que leur capacité à garantir

[6]Voir par exemple Pi4Tech : http ://www.pi4tech.com.

une certaine qualité de service par rapport à des préoccupations telles que la sécurité, la fiabilité, la journalisation des accès ou la gestion de transactions (voir le chapitre 1).

Ce manque est en partie compensé par le concept de *politiques d'usage*. Une politique d'usage est une énonciation explicite des possibilités et des restrictions d'usage d'un service Web. « WS-Policy »est un langage extensible permettant d'exprimer des politiques (ou règles) d'usage sous forme de conjonctions et de disjonctions (au sens logique) d'*assertions* [Kaler and Nadalin]. Dans ce contexte, une assertion est une donnée par laquelle un service exprime qu'il permet aux clients (ou qu'il requiert des clients) de procéder d'une certaine manière lorsqu'ils accèdent aux opérations du service. « WS-Policy »ne définit pas des types d'assertions d'usages particuliers. Cependant, ces types d'assertions sont définis par d'autres standards proposés tels que « WS-Security-Policy », « WS-Reliable-Messaging »et « WS-Addressing », qui définissent respectivement des types d'assertion liés à la sécurité, la fiabilité, et l'adressage. Ainsi, en utilisant les types d'assertion définis dans « WS-Security-Policy », il est possible d'exprimer qu'un service requiert que tous les messages soient signés ou qu'ils soient signés et encryptés avec une clé publique donnée. De même, en utilisant les types d'assertion définis dans « WS-Reliable-Messaging », on peut exprimer qu'un service donné opère selon un protocole de renvoi de messages permettant d'assurer que les messages arrivent au moins une fois, ou exactement une fois. Enfin, les types d'assertion définis dans « WS-Addressing »permettent par exemple d'exprimer qu'un service peut envoyer le résultat d'un appel d'opération à une entité différente de celle ayant initié l'appel.

« WS-Policy »constitue un pas en avant vers des modèles de développement d'applications réparties basées sur des descriptions détaillées couvrant des aspects aussi bien fonctionnels que non-fonctionnels. Cependant, pour réaliser la vision de services Web comme entités indépendantes, davantage d'efforts de recherche sont nécessaires afin de déboucher sur des modèles et des langages permettant de décrire l'environnement organisationnel dans lequel les services Web évoluent. Par exemple, pour faciliter l'adoption des services Web dans des applications de type *business-to-business*, il serait souhaitable d'avoir des descriptions de services couvrant des garanties de disponibilité et de temps de réponse, ainsi que des politiques de prix et de pénalités, de modalités de paiement, de réputation, etc. (voir [O'Sullivan et al. 2002]). Il ne faut pas s'attendre à ce que toutes ces propriétés fassent l'objet de standardisation ou soient prises en compte dans des outils de développement de services car la plupart d'entre elles touchent à des aspects socio-techniques et varient selon les domaines d'application. Cependant, de telles descriptions non-fonctionnelles pourraient constituer un outil d'évaluation et de sélection lors des phases initiales (c'est-à-dire au cours de l'analyse du domaine) dans la construction d'applications à base de services.

4.4 Implantation de services Web

Dans cette section, nous discutons des principaux outils permettant la mise en œuvre de services Web. Nous décrivons tout d'abord les principes de base du protocole SOAP (voir section 4.4.1), puis la spécification d'en-têtes permettant de transmettre des informations particulières (voir section 4.4.2). Nous donnons ensuite, dans la section 4.4.3, des éléments de comparaisons de SOAP avec REST, une solution concurrente pour l'implantation de services Web. Dans la section 4.4.4 nous introduisons rapidement deux implantations de

SOAP. Finalement la section 4.4.5 discute de BPEL un langage orienté processus.

4.4.1 SOAP (*Simple Object Access Protocol*)

Les interactions entre services Web s'effectuent par le biais d'envois de messages structurés au format XML. Le protocole SOAP (*Simple Object Access Protocol*) [W3C-XMLP-Group] fournit le cadre permettant ces échanges. SOAP est originellement issu de tentatives précédentes visant à standardiser l'appel de procédures à distance, et en particulier de XML-RPC [Winer 1999]. Mais à la différence des technologies RPC, SOAP n'est pas fondamentalement lié à la notion d'appel de procédure. En effet, SOAP vise à faciliter l'échange de messages XML, sans se limiter à des messages dont le contenu encode des paramètres d'appel de procédure et sans favoriser des échanges bidirectionnels de type requête-réponse comme c'est le cas des protocoles RPC. Dans le jargon des services Web, SOAP permet d'encoder des interactions orientées-RPC mais aussi des interactions orientées-document.

Une autre caractéristique de SOAP est de faire abstraction de la couche de transport sous-jacente. Bien que la pratique la plus répandue soit d'utiliser SOAP au-dessus de HTTP, il existe aussi des implantations de SOAP au-dessus d'autres protocoles tels que le protocole d'échange de messages électroniques SMTP, et les protocoles de transport orientés-message de Microsoft et IBM, à savoir MSMQ et MQSeries respectivement.

La manière d'implanter SOAP au-dessus d'un protocole de transport donné est appelée une *liaison SOAP* (« SOAP binding »en anglais). Une liaison SOAP définit, en particulier, l'encodage des messages (nécessaire en particulier lorsque le protocole sous-jacent utilise un format binaire), la méthode pour l'échange de messages, l'encodage des noms d'opérations (appelés « SOAP Actions »), et la façon dont différents messages (y compris les messages d'erreur) appartenant à la même interaction sont corrélés. Par exemple, la liaison SOAP au-dessus de HTTP définit que les messages sont encodés dans un « type de média »appelé « application/soap+xml »(c'est-à-dire en XML avec quelques extensions), que le nom de l'opération correspondant à une requête est donné dans un en-tête HTTP appelé « SOAPAction », et que les messages dans une interaction sont échangés au travers des méthodes POST et GET fournies par HTTP. D'autres règles (dont nous ne donnons pas les détails) définissent la manière dont les messages appartenant à un même échange (y compris les messages d'erreur) sont corrélés en exploitant les caractéristiques des méthodes POST et GET.

Outre la définition échanges de messages en faisant abstraction de la couche de transport, SOAP définit une structure standard de messages dans laquelle le contenu des messages est séparé des méta-données liées à l'échange des messages. Ainsi, un message SOAP est constitué de deux parties : un en-tête et un corps. L'en-tête indique l'objet du message (l'appel d'une opération ou le retour de résultats), la description de l'expéditeur, et l'information requise pour acheminer le message au destinataire. Le corps du message peut contenir : (i) un document quelconque ; (ii) l'appel d'une opération offerte par le service destinataire, avec les valeurs pour les paramètres d'entrée ; (iii) les valeurs produites en résultat d'un appel ; ou bien (iv) un message d'erreur. Ainsi, SOAP offre diverses possibilités d'interactions entre les services : soit des échanges de documents, soit des interactions de type RPC.

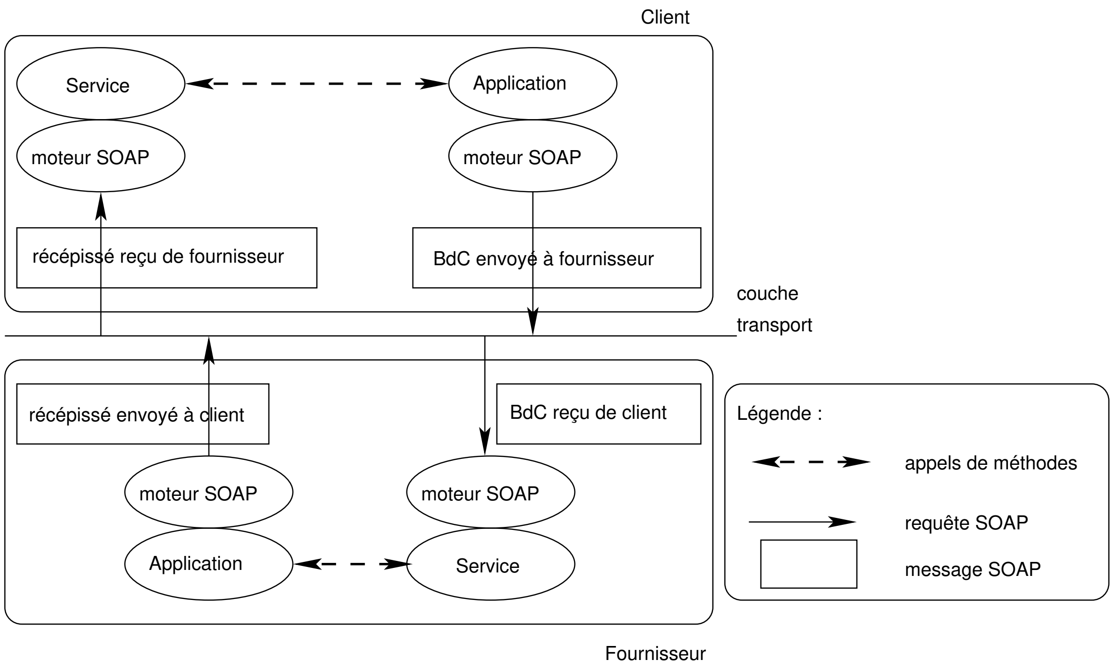

Figure 4.5 – Interactions entre les partenaires Client et Fournisseur

La figure 4.5 schématise un scénario d'interaction entre les partenaires Client et Fournisseur déjà introduits dans la section 4.2 (les interactions avec l'entrepôt ne sont pas montrées). Le client transmet une commande au fournisseur en lui envoyant un bon de commande. Ce bon de commande est transmis, par une application du client, sous forme d'une requête SOAP. Chez le fournisseur, le serveur d'application reçoit la requête (sous forme d'une requête HTTP) et la transmet au moteur SOAP. Ce dernier décode le message reçu et effectue les appels aux procédures correspondantes de l'application locale. Lorsque les vérifications de disponibilité des articles commandés et les interactions avec l'entrepôt sont terminées, l'application du fournisseur transmet, de la même manière, la facture au serveur d'application du client (les serveurs d'application ne sont pas figurés).

Il existe plusieurs mécanismes pour construire, analyser, et échanger des messages SOAP dans des langages variés tels que Java, C++, Perl, C♯, etc. Ces implantations permettent de générer les en-têtes de messages SOAP et de mettre en correspondance le contenu du corps du message avec les structures de données définies dans le langage hôte (voir Section 4.4.4).

4.4.2 Extensions de SOAP : spécifications WS-*

Comme indiqué ci-dessus, l'en-tête d'un message SOAP est destiné à contenir des méta-données sur l'échange des messages. Ces méta-données prennent la forme d'un nombre de (sous-)en-têtes, chacun encodé sous forme d'un élément XML. Chaque sous-en-tête permet de transmettre des informations liées à l'adressage, à la fiabilité, à la sécurité, ou à d'autres propriétés traditionnellement associées aux interactions distribuées (voir le chapitre 1).

Il est en principe possible d'inclure n'importe quel ensemble d'en-têtes dans un message

SOAP. Un fournisseur peut en théorie exprimer que son service est capable de recevoir des en-têtes X, Y, Z, avec une certaine sémantique pour chacun, et les programmeurs des applications qui accèdent à ce service sont alors libres de les utiliser ou pas. De façon symétrique, une application peut envoyer un message SOAP avec n'importe quel ensemble d'en-têtes, et c'est au service destinataire de déterminer ceux qu'il peut interpréter et ceux qu'il peut ignorer. L'attribut XML `mustUnderstand` peut être associé à chaque en-tête pour exprimer si cet en-tête doit obligatoirement être interprété par le destinataire ou si il peut être ignoré. Le destinataire est tenu de renvoyer un message d'erreur s'il détecte un en-tête qui devrait être interprété obligatoirement, et qu'il ne lui est pas possible d'interpréter.

Différentes spécifications (certaines en passe de devenir des standards), connues généralement sous le nom collectif WS-*, définissent des ensembles d'en-têtes SOAP perçus comme étant particulièrement importants pour un grand nombre d'applications. Trois des spécifications les plus avancées en terme de leur standardisation sont WS-Addressing (pour l'adressage), WS-Security (pour l'authentification et la non-répudiation) et WS-Reliable-Messaging (pour le renvoi répété de messages afin de garantir leur livraison fiable et ordonnée). Par exemple, WS-Addressing définit les (sous-)en-têtes suivants :

- `MessageID` : où l'on peut associer un identificateur au message.
- `RelatesTo` : où l'on peut faire référence à l'identificateur d'un message antérieur.
- `From, To` : qui fixent l'expéditeur et le destinataire du message.
- `Reply-to` : qui donne l'adresse de retour, c'est-à-dire l'URL à laquelle des réponses éventuelles au message doivent être envoyées. Cet en-tête est utile par exemple lorsque la réponse à une requête est envoyée au travers d'une deuxième connexion HTTP établie ultérieurement (selon le mode d'interaction dit *asynchrone*), au lieu d'être incluse dans le message de retour de la connexion HTTP courante (selon le mode d'interaction dit *synchrone*).

A titre d'exemple, le message HTTP suivant correspond à l'envoi d'un bon de commande par un client, d'après l'interaction décrite dans la section 4.2. Ce message inclut un identificateur de message et une adresse à laquelle l'éventuelle réponse doit être expédiée ultérieurement.

```
POST /orabpel/default/Supplier HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/#axisVersion#
SOAPAction: "BdC"
...
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <MessageID
            xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing"
            xmlns:orabpel="http://schemas.oracle.com/bpel" >
            bpel://www.yothuyindi.fr/default/Customer~1.1/301-BpInv0-BpSeq0.3-3
        </MessageID>
        <ReplyTo xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
            <Address>
                http://GA2550:9710/orabpel/default/Customer/1.1/Supplier/SupplierRequester
```

```
            </Address>
        </ReplyTo>
    </soapenv:Header>
    <soapenv:Body>
        <BonDeCommande> ... </BonDeCommande>
    </soapenv:Body>
</soapenv:Envelope>
```

La réponse du fournisseur peut alors être envoyée au travers d'une deuxième connexion HTTP. Cette réponse doit être envoyée à l'adresse indiquée dans l'en-tête « ReplyTo »du message ci-dessus, et doit se référer à l'identificateur du message du message ci-dessus au travers de l'en-tête « RelatesTo ».

Un exemple de message de réponse est donné ci-dessous :

```
POST /orabpel/default/Customer/1.1/Supplier/SupplierRequester HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/#axisVersion#
Host: GA2550:9710
SOAPAction: "ResponseBdC"
...

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <RelatesTo xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
            bpel://www.yothuyindi.fr/default/Customer~1.1/301-BpInv0-BpSeq0.3-3
        </RelatesTo>
    </soapenv:Header>
    <soapenv:Body>
        <ReponseBonDeCommande>
          ...
        </ReponseBonDeCommande>
    </soapenv:Body>
</soapenv:Envelope>
```

Pour une description relativement complète de WS-Addressing et autres spécifications WS-*, le lecteur peut se reporter à [Weerawarana et al. 2005].

4.4.3 Le débat SOAP vs. REST

Dans la section 4.4.1, nous avons indiqué que la façon la plus répandue d'utiliser SOAP consiste à s'appuyer sur le protocole HTTP en utilisant la liaison SOAP-HTTP. Dans cette liaison, il est possible d'associer plusieurs opérations à la même URL. Par exemple, un service de ventes peut être placé à l'URL `www.unservicedevente.fr/serviceweb`, et toutes les opérations de ce service (demande de devis, placement de bon de commande, suivi de bon de commande, etc.) peuvent être fournies sur cette même URL. Une application donnée peut alors appeler chacune de ces opérations en utilisant la méthode HTTP POST, et en incluant le nom de l'opération concernée par la requête dans l'en-tête HTTP « SOAPAction ».

Cette approche, ainsi que d'autres caractéristiques de SOAP, ont fait l'objet de nombreuses critiques. Les arguments principaux mis en avant contre SOAP reposent sur le fait que : (i) il rajoute peu de fonctionnalités au-dessus de ce qu'il est déjà possible de faire avec HTTP et XML (sans les extensions apportées par SOAP) ; et (ii) en associant plusieurs opérations à une même URL, il rend difficile, voire impossible, l'utilisation de l'infrastructure de « caching »associée au protocole HTTP, qui constitue sans doute l'un des points forts de HTTP.

Une approche alternative pour l'implantation de services Web appelée REST (REpresentational State Transfer) a été définie [Fielding 2000]. Dans cette approche, chaque opération d'un service est associée à une URL distincte et l'accès à chaque URL peut être réalisé en utilisant l'une des quatre méthodes fournies par HTTP : POST, GET, PUT et DELETE. Le contenu des messages est alors encodé en XML, et la distinction entre en-tête et contenu de message est laissée à la charge des applications qui requièrent cette distinction. Le résultat est un ensemble de conventions plus simples que celles de SOAP, et la possibilité d'utiliser des bibliothèques existantes pour l'échange de messages sur HTTP, éliminant ainsi le besoin de plates-formes implantant SOAP, qui dans certains cas peuvent être considérées comme étant plus « lourdes »et plus difficiles à utiliser.

Plusieurs services Web populaires disponibles à l'heure actuelle utilisent l'approche REST (appelée aussi « XML sur HTTP »). Ceci est le cas notamment de la maison de ventes au enchères en ligne eBay, qui rend une partie de ses fonctionnalités, accessibles sous forme de services Web « style REST »(voir `http ://developer.ebay.com/rest`). Il en est de même du site de vente par Internet Amazon (voir `http ://www.amazon.com/webservices`).

Le débat SOAP vs. REST est encore ouvert et les avis divergent considérablement dans la communauté. Les avantages et désavantages relatifs de SOAP et REST peuvent être résumés comme suit :

- REST ne requiert pas d'infrastructure spécifique pour le développement et l'exécution des services Web. Il repose sur l'infrastructure HTTP existante, qui a fait ses preuves dans le contexte des applications HTML classiques. De nombreux outils de programmation d'applications au-dessus de HTTP existent, et peuvent être combinés avec des outils de manipulation de documents XML pour construire des services Web « style REST ». Dans le cas de SOAP, il est nécessaire d'utiliser des outils spécifiques (voir la section 4.4.4) qui sont parfois difficiles à déployer et n'ont pas complètement fait leurs preuves.
- SOAP peut opérer au-dessus d'autres protocoles que HTTP, en particulier sur des protocoles permettant des communications asynchrones telles que SMTP, JMS, MSMQ et MQSeries.
- Lorsqu'il est utilisé en conjonction avec des spécifications WS-*, SOAP permet de prendre en compte des aspects liés à la sécurité, la fiabilité et l'adressage et routage de messages. Bien sûr, il serait possible de faire de même en utilisant l'approche REST, mais les standards et l'infrastructure correspondant ne sont pas en place.

En conclusion, l'approche REST est viable (voire préférable) dans le cas de services Web avec les caractéristiques suivantes : (i) ils n'ont besoin d'être exposés que sur HTTP ou HTTPS ; (ii) les communications asynchrones (et donc l'adressage explicite) ne sont pas requises ; (iii) ils ne requièrent pas de garanties de sécurité au delà de celles offertes par HTTPS ; et (iv) la fiabilité peut facilement être traitée au niveau des applications.

4.4.4 Exemples d'implantations de SOAP

Nous donnons ci-après un aperçu de deux implantations *Open Source* du protocole SOAP, qui se distinguent par leur principe : l'une, Axis, est à base de bibliothèques, l'autre, Beehive, est à base d'annotations (voir [W3C-XMLP-Group] pour obtenir une liste des implantations de SOAP).

Axis [Axis] est un projet du groupe Web Services d'Apache [Apache]. Le moteur Axis, qui joue le rôle de client et de serveur SOAP, se présente sous forme d'un ensemble de bibliothèques : une version Java est actuellement disponible, une autre en C++ est en cours de développement.

La version Java d'Axis est constituée d'un ensemble de bibliothèques qui implantent l'API Java JAX-RPC d'une part, et fournissent d'autre part, des outils pour faire de la journalisation, de la génération et de l'analyse de documents WSDL. Ces bibliothèques peuvent être empaquetées comme une application Web, puis rendues disponibles par le biais d'une *servlet*.

Axis propose un premier mode de déploiement grâce auquel un programme Java est immédiatement exposé comme un service Web. Il suffit de placer ce programme (non compilé) dans le répertoire dédié à l'application Web Axis. Cette méthode, si elle a le mérite d'être simple, a pour inconvénients d'obliger le fournisseur à montrer le code source et de ne pas être adaptée au déploiement d'applications complexes. Palliant ces inconvénients, le second mode de déploiement s'appuie sur la description des éléments d'un service (classes et méthodes) qu'il faut exposer. Le moteur Axis propose une opération de déploiement (et une autre inverse de suppression) qui prend en entrée le programme Java compilé et la description du déploiement (*Web Service Deployment Descriptor*). Cette seconde méthode, spécifique à Axis, permet de spécifier les classes et les méthodes à exposer. À l'inverse de la première méthode, elle permet surtout de pouvoir spécifier la durée de vie des objets générés par l'exécution du service et de permettre l'utilisation de composants Java (*Java Beans*). La durée de vie des objets générés par l'exécution du service peut être associée soit à la requête, soit à l'ensemble des exécutions du service, soit enfin à la session. Axis prend à sa charge la sérialisation/désérialisation des classes Java à condition qu'elles aient été implantées dans des composants Java.

Beehive [Beehive] est un autre projet Apache dont l'objectif global est de définir un modèle d'annotations pour Java dans la perspective de réduire la quantité de code Java à produire pour développer des applications J2EE (voir chapitre 5).

Une partie du projet Beehive est consacrée à l'implantation de JSR181 (*Java Specification Request - Web Services Metadata for the Java Platform*) [JSR 181] qui fixe un modèle d'annotations pour la construction de services Web développés en Java. L'idée est d'offrir un outil pour le développement de services Web qui seront ensuite rendus disponibles par le biais de n'importe quel moteur SOAP, comme par exemple Axis.

Beehive s'appuie sur trois types d'annotations : `@WebService` pour spécifier quelle classe est exposée comme service Web ; `@WebMethod` pour indiquer les opérations du service, et pour chacune la méthode qui l'implante. Pour chaque opération, l'annotation `@WebParam` permet d'en déclarer les paramètres.

4.4.5 Implantation à base d'un langage spécifique : BPEL

BPEL est aujourd'hui un standard de facto pour implanter des services Web selon un point de vue orienté processus. Des plates-formes matures et reconnues, telles que BEA WebLogic, IBM WebSphere, Microsoft BizTalk, SAP XI et l'outil de gestion de processus BPEL d'Oracle supportent BPEL à des degrés divers démontrant ainsi l'intérêt réel de ce langage. ActiveBPEL est une autre proposition assez complète dans le domaine de l'*Open Source*[7].

Comme brièvement indiqué dans la section 4.3, en BPEL, la logique des interactions entre un service et son environnement est décrite par le biais d'une composition d'actions élémentaires de communication (émission, réception ou émission/réception). Ces actions sont reliées les unes aux autres par un flot de contrôle spécifié par des constructions telles que la composition parallèle (avec un nombre fixé de branches), séquentielle, et conditionnelle, des règles de type événement-action et des clauses de capture/transmission d'erreur. La manipulation des données s'effectue par le biais de variables, comme dans un langage de programmation impérative.

Plus précisément, un processus exprimé en BPEL est construit à partir d'activités primitives qui peuvent être composées pour définir d'autres activités plus complexes. Les activités primitives sont : la réception (`receive`) qui bloque l'exécution jusqu'à l'arrivée d'un message correspondant à une opération ; l'appel (`invoke`) et la réponse (`reply`) qui chacune effectue un envoi de message correspondant à une opération ; l'affectation (`assign`) qui associe une expression (écrite en XPath ou XSLT) à une variable ; l'attente (`wait`) qui bloque l'exécution pour une période de temps fixée ; la transmission (`throw`) d'erreur et la fin (`exit`) pour terminer l'exécution.

La plupart des opérateurs de composition proposés par BPEL correspondent à ceux déjà présents dans les langages de programmation impérative : la séquence (`sequence`) pour imposer un ordre dans l'exécution, le choix (`switch`) pour exprimer un branchement conditionnel, et l'itération (`while`). Le concept de bloc issu des langages de programmation structurée est présent dans BPEL par le biais de la notion de portée (`scope`).

En plus de ces opérateurs de composition, BPEL offre plusieurs constructions pour la programmation concurrente : la composition parallèle (`flow`) pour exécuter plusieurs activités en parallèle ; la sélection (`pick`) pour choisir l'un parmi les messages entrants ou les *timers* ; des dépendances de précédences (`link`) peuvent être définies entre des activités qui autrement s'exécuteraient en parallèle. Finalement, des règles de type événement-action (`event handlers`) peuvent être associées à des blocs (`scope`). Ces règles sont déclenchées par la réception d'un message ou à l'expiration d'un délai, à n'importe quel moment au cours de l'exécution du bloc correspondant.

Dans le langage BPEL, tout est service : un processus exécutable est l'implantation d'un service qui s'appuie sur d'autres services. Lorsqu'une ressource, telle qu'une base de données, un fichier, ou une application patrimoniale, est utilisée par un service, il est nécessaire d'exposer le SGBD, le système de gestion de fichiers, ou l'application comme des services. Pour pouvoir utiliser BPEL dans des environnements où toutes les ressources ne sont pas forcément exposées comme des services, il est quelques fois judicieux de casser le paradigme « tout service »de BPEL. C'est dans cet esprit qu'a été définie

[7]Pour obtenir la liste des outils mentionnés ici, voir http ://en.wikipedia.org/wiki/BPEL

BPELJ [Blow et al. 2004], une extension de BPEL avec laquelle il est possible d'intégrer du code Java dans un programme BPEL. Cette approche est similaire à celle des JSPs (*Java Server Pages*) où du code Java est inclus dans des programmes HTML. La considération de BPEL dans les deux plates-formes .Net et Java laisse à penser que d'autres dialectes proches de BPEL vont émerger.

Ci-dessous, nous fournissons un squelette du processus exécutable BPEL correspondant au service « fournisseur »modélisé dans la figure 4.2. Ce squelette montre comment les opérateurs de flot de contrôle `sequence`, `switch` et `flow` sont combinés pour exprimer l'ordre dans lequel les messages sont échangés entre le fournisseur d'une part, et le client et l'entrepôt d'autre part. En particulier, les actions `envoyerBordereauExpédition` et `recevoirOrdreDeTransfert` peuvent s'exécuter en parallèle ou dans n'importe quel ordre puisqu'elles sont emboîtées dans une activité de type `flow`. Le squelette montre aussi comment BPEL est lié avec WSDL : les types et les opérations définis en WSDL dans la section 4.3.1 sont utilisés ici pour définir les types des variables et les types des messages envoyés et reçus par les actions `receive`, `reply` et `invoke`. De plus, les expéditeurs et destinataires des messages envoyés sont spécifiés par le biais des liens de communication (« partner links ») qui devront être associés aux services décrits dans la description WSDL.

```
<process name="ProcessusFournisseur">
  <partnerLinks>
    <partnerLink name="client" ... />
    <partnerLink name="entrepôt" ... />
  </partnerLinks>

  <variables>
    <variable name="BdC" type="xsld:Commande"/>
    <variable name="réponseBDC" messageType= "string" />
    <variable name="disponibilité" type="string"/>
  </variables>

  <sequence>
    <receive name="recevoirBdC"
             partnerLink="client" portType="pt_Commande"
             operation="op_Commande" variable="BdC"
             createInstance="yes"/>
    <invoke  name="demanderDisponibilité"
             partnerLink="entrepôt"
             ...
             inputVariable="BdC"
             outputVariable="disponibilité"/>
    <switch>
      <case ....> <!-- cas disponible -->
        <!-- Initialiser la variable réponseBdC avec réponse positive -->
        <reply name="RépondreBdC"
               partnerLink="client"
               portType="pt_Commande"
               operation="op_Commande"
               inputVariable="réponseBDC"/>
        <flow>
            <invoke name="envoyerBordereuExpédition" .../>
            <receive name="recevoirOrdreDeTransfert" .../>
```

```
        </flow>
      </case>
      <case ...> <!-- cas indisponible -->
        <!-- Initialiser la variable réponseBdC avec réponse négative -->
        ...
      </case>
    </switch>
  </sequence>
</process>
```

Bien que BPEL offre des constructions spécifiques pour le développement de services Web, sa complexité pose problème. Comme le montre [Wohed et al. 2003] l'opérateur `link` est dans une certaine mesure redondant avec les opérateurs `switch` et `flow` dans le sens que tout processus BPEL écrit en utilisant `switch` et `flow` peut être réécrit en utilisant un seul `flow` et un certain nombre de `links`. De plus, BPEL manque d'abstractions de haut niveau lorsqu'il s'agit de développer des services qui mettent en jeu des `multicasts` avec des conditions de synchronisation partielles, comme par exemple dans le cas d'un service « client »qui requiert des devis de plusieurs services « fournisseurs » [Barros et al. 2005a]. On peut s'attendre à ce que dans le futur, des efforts de recherche portent sur la résolution de ces problèmes et proposent soit des extensions de BPEL, soit des langages alternatifs.

4.5 Perspectives

4.5.1 Fouille d'interface, test de conformité et adaptation

Dans les architectures à base de services, nous l'avons déjà vu, la description structurelle et comportementale des services s'appuie sur la notion d'interface, il est donc d'attendu que les services respectent leur interface. Ces dernières peuvent ainsi être perçues comme un contrat entre les services. Cependant, les services sont indépendants les uns des autres et ne peuvent exercer aucun contrôle les uns sur les autres. Pour ces raisons, un service ne peut jamais être sûr que les autres se comportent comme convenu dans leur interface. Tout ce dont un service peut être sûr au sujet des autres services avec lesquels il interagit s'appuie sur l'ensemble des messages qu'il envoie et de ceux qu'il reçoit. L'existence de traces des événements qui se sont passés ainsi que la donnée des interfaces permettent de poser la question de la confrontation de « ce qu'il s'est passé »à « ce qui était prévu qu'il se passe ». Cette question peut être étudiée selon deux points de vue. Le premier consiste à prendre les interfaces des services comme étant la référence, puisqu'elles spécifient la manière dont les services partenaires doivent se comporter. La question est alors de savoir si les événements enregistrés dans la trace sont cohérents avec les interfaces. Par exemple, la trace peut contenir une séquence d'événements impossible selon les interfaces, ce qui traduit une violation du contrat. Le second point de vue est inverse : la trace des événements est supposée correcte car elle reflète ce qui se passe réellement. Dans ce cas, se pose le problème de savoir si les interfaces ne sont plus valides et en conséquence si elles doivent être modifiées. Il est alors extrêmement utile de pouvoir dériver les nouvelles interfaces des traces. Lorsque le problème est étudié selon la dimension structurelle des interfaces, cela revient à comparer un ensemble de messages en XML à un schéma exprimé dans un langage tel que XMLSchema [W3C-XMLSchema] (voir aussi [Bertino et al. 2004], un état de l'art sur ce sujet). Moins d'attention a été consacrée à la résolution des problèmes de test de conformité et de fouille d'interfaces en prenant le point de vue de la dimension comportementale des interfaces. Il est possible d'envisager la résolution de ces problèmes en appliquant des méthodes de fouille de processus [Aalst et al. 2003] ou de tests de conformité de processus [Rozinat and Aalst 2005]. Quelques investigations préliminaires sur le problème de la fouille d'interfaces comportementales sont rapportées dans [Dustdar et al. 2004, Gaaloul et al. 2004].

L'adaptation des interfaces est un autre problème saillant, complémentaire à celui de la conformité. Lorsqu'il s'avère, que ce soit a priori par une comparaison, ou a posteriori par un test de conformité, que l'interface fournie d'un service ne correspond pas à l'interface que ses partenaires requièrent, il y a deux solutions : (1) adopter l'interface requise comme interface fournie et modifier le service en conséquence ; (2) introduire un adaptateur qui réconcilie l'interface fournie avec celle requise par les partenaires. La première solution est en général mauvaise car le même service peut interagir avec plusieurs autres services partenaires qui considèrent son interface originale. Ce qui conduit à la situation où le même service peut participer à diverses collaborations qui nécessitent différentes interfaces fournies. La seconde solution pallie ce défaut par la fourniture d'autant d'adaptateurs que d'interfaces requises. Comme nous l'avons vu dans la section 4.3, une interface peut être décrite selon la dimension structurelle, comportementale ou encore non-fonctionnelle. Ainsi l'adaptation doit être étudiée selon chacune de ces dimensions. Le problème de l'adaptation structurelle se ramène essentiellement à celui de la réconciliation entre des types de messages. Il existe sur ce sujet de très nombreuses études et plusieurs systèmes sont commercialisés (par exemple Microsoft's BizTalk Mapper). A l'inverse, le problème de l'adaptation selon la dimension comportementale est en cours d'étude [Benatallah et al. 2005a, Altenhofen et al. 2005, Fauvet and Ait-Bachir 2006]. L'adaptation des règles d'usage (par exemple, la réconciliation de différentes politiques de sécurité) est par contre l'objet de peu de recherche. La technologie des services Web gagnant en maturité et tendant à être utilisée dans le cadre de l'intégration de projets à grande échelle, l'adaptation des règles d'usage devrait très vite devenir un sujet important et crucial.

4.5.2 Transactions

Certains services web, en particulier dans le domaine du commerce électronique, ont des propriétés transactionnelles inhérentes [Baïna et al. 2004]. Ceci est le cas notamment des services associés à la gestion de ressources (au sens large), comme par exemple la réservation de chambres d'hôtel, de places de spectacle, de services professionnels, etc. En principe, les propriétés transactionnelles de ces services peuvent être exploitées lors de leur composition pour répondre à des contraintes et des préférences établies par le concepteur et l'utilisateur final. Aujourd'hui cependant, les langages et outils disponibles permettant de programmer des transactions sur des services Web ne fournissent pas de concepts de haut niveau pour : (i) exprimer les propriétés transactionnelles désirées au niveau du service composé ; (ii) assurer ces propriétés de façon automatisée en exploitant les propriétés transactionnelles des services composants.

L'exécution de services composés avec propriétés transactionnelles s'appuie sur l'exécution de transactions distribuées, complexes, souvent de longue durée, qui éventuellement peuvent mettre en œuvre des mécanismes de compensation (une opération de compensation est une opération dont l'objectif est d'annuler les effets d'une autre opération qui n'a pas pu être terminée avec succès). De nombreux modèles de transactions ont été proposés dans le domaine des bases de données, des systèmes distribués et des environnements coopératifs (voir par exemple [Elmagarmid 1990, Gray and Reuter 1993, Alonso et al. 2003, Papazoglou 2003]).

Il est bien connu que les approches traditionnelles pour assurer les propriétés ACID d'une transaction (*Atomicity, Consistency, Isolation, Durability*) ne sont typiquement pas adéquates pour les transactions de longue durée telles que celles rencontrées dans le domaine des services web, puisqu'il n'est pas acceptable de verrouiller des ressources dans une transaction qui s'exécute sur une durée prolongée. De plus, le protocole de validation à deux phases, couramment utilisé dans les systèmes distribués, n'est pas applicable aux services composites car, dans ce protocole, il est fait l'hypothèse que tous les partenaires de la transaction supportent les opérations de préparation et de validation indispensables à sa mise en œuvre ; ce qui n'est pas toujours le cas dans le cadre des services web. Dans ce contexte, il peut être approprié de relaxer les contraintes d'atomicité *tout-ou-rien.* A cela s'ajoutent des problèmes d'intégration, puisque chaque service composant s'appuie sur un système de gestion de transactions choisi ou conçu pour le composant, considéré individuellement. Lorsqu'un service est intégré comme composant, il est fortement probable que son système de gestion de transactions ne réponde pas aux besoins de la composition vue comme un tout. Ce dernier aspect est la motivation principale pour l'émergence de protocoles tels que *WS-Coordination* [Cabrera et al. 2002a], *WS-AtomicTransaction*[8] [Cabrera et al. 2002b] et pour la conduite de travaux de recherche (voir par exemple [Arregui et al. 2000, Hagen and Alonso 2000, Vidyasankar and Vossen 2003, Limthanmaphon and Zhang 2004, Fauvet et al. 2005]).

4.5.3 Sélection dynamique

Il est courant que plusieurs services offrent les mêmes fonctionnalités (la même capacité de service). Le concept de « Communauté de services »a été proposé dans la perspective de composer un nombre potentiellement grand de services Web et ce, de manière dynamique [Benatallah et al. 2005b]. Une communauté décrit les capacités de services sans faire référence au fournisseur du service Web. Ainsi, une composition s'appuie sur des communautés et non plus sur des services directement. Au moment de l'exécution, la sélection d'un service parmi ceux disponibles est opérée par la communauté correspondante. Pour que les services deviennent disponibles au travers d'une communauté, ils doivent s'enregistrer auprès d'elle. Les services peuvent rejoindre et quitter une communauté à n'importe quel moment. Une communauté possède une interface au travers de laquelle sont accessibles les opérations qu'elle offre. Un service Web peut s'enregistrer auprès d'une ou de plusieurs communautés. Et une communauté peut s'enregistrer auprès d'une autre communauté.

Le choix d'un service entrant dans une composition peut ainsi être effectué dynamiquement, au moment de l'exécution du service composé, de sorte que l'ensemble des partenaires intervenant dans la composition n'est pas connu a priori. Cette possibilité n'est pas sans impact sur les problèmes soulevés dans les deux sections ci-dessus (voir sections 4.5.1 et 4.5.2) : par exemple, un service peut choisir d'interagir avec un autre par le biais de la

[8] http ://msdn.microsoft.com/library/default.asp ?url=/library/en-us/dnWebsrv/html/wsacoord.asp

sélection dynamique, au moment de l'exécution, en fonction de critères de coût, de qualité de services, de règles d'usages, ou encore en fonction des propriétés transactionnelles exposées (pour augmenter les chances de terminer la transaction avec succès par exemple).

Bibliographie

[Aalst et al. 2005] Aalst, W., Dumas, M., Ouyang, C., Rozinat, A., and Verbeek, H. (2005). Choreography Conformance Checking: An Approach Based on BPEL and Petri Nets. BPMCenter Report BPM-05-25, BPMcenter.org.

[Aalst et al. 2003] Aalst, W., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. (2003). Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267.

[Alonso et al. 2003] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2003). *Web Services. Concepts, Architectures and Applications*. Springer Verlag.

[Altenhofen et al. 2005] Altenhofen, M., Boerger, E., and Lemcke, J. (2005). An execution semantics for mediation patterns. In *In Proceedings of the BPM'2005 Workshops: Workshop on Choreography and Orchestration for Business Process Managament*, Nancy, France.

[Andrews et al. 2003] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S. (2003). Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, IBM Corporation and Microsoft Corporation.

[Apache] Apache. The Apache Foundation Software. http://www.apache.org.

[Arregui et al. 2000] Arregui, D., Pacull, F., and Rivière, M. (2000). Heterogeneous component coordination: The CLF approach. In *EDOC*, pages 194–2003. 4th International Enterprise Distributed Object Computing Conference (EDOC 2000), Makuhari, Japan, IEEE Computer Society.

[Axis] Axis. Projet Web Services - Apache - Axis. http://ws.apache.org/axis/.

[Baïna et al. 2004] Baïna, K., Benatallah, B., Casati, F., and Toumani, F. (2004). Model-Driven Web Services Development. In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAISE'04)*, Riga, Latvia. Springer.

[Baresi et al. 2004] Baresi, L., Ghezzi, C., and Guinea, S. (2004). Smart monitors for composed services. In *Proceedings of the 2nd Internation Conference on Service-Oriented Computing (IC-SOC)*, pages 193–202, New York, NY, USA.

[Barros et al. 2005a] Barros, A., Dumas, M., and Hofstede, A. (2005a). Service interaction patterns. In *Proceedings of the 3rd International Conference on Business Process Management*, Nancy, France. Springer. Extended version available at: http://www.serviceinteraction.com.

[Barros et al. 2005b] Barros, A., Dumas, M., and Oaks, P. (2005b). A critical overview of the web services choreography description language (ws-cdl). *BPTrends Newsletter*, 3(3).

[Beehive] Beehive. Projet Apache - Beehive. http://beehive.apache.org.

[Benatallah et al. 2005a] Benatallah, B., Casati, F., Grigori, D., H.R. Motahari-Nezhad, and Toumani, F. (2005a). Developing Adapters for Web Services Integration. In *Proceedings of the 17th International Conference on Advanced Information System Engineering, CAiSE 2005, Porto, Portugal*, pages 415–429.

[Benatallah et al. 2005b] Benatallah, B., Dumas, M., and Sheng, Q.-Z. (2005b). Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. *Distributed and Parallel Database*, 17(1).

[Bertino et al. 2004] Bertino, E., Guerrini, G., and Mesiti, M. (2004). A matching algorithm for measuring the structural similarity between an xml document and a dtd and its applications. *Information Systems*, 29(1):23–46.

[Blow et al. 2004] Blow, M., Goland, Y., Kloppmann, M., Leymann, F., Pfau, G., Roller, D., and Rowley, M. (2004). BPELJ: BPEL for Java. White paper.

[Cabrera et al. 2002a] Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., Langworthy, D., and Orchard, D. (2002a). Web Service Coordination, WS-Coordination. http://www.ibm.com/developerworks/library/ws-coor/. IBM, Microsoft, BEA.

[Cabrera et al. 2002b] Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., and Thatte, S. (2002b). Web service transaction, ws-transaction. http://www.ibm.com/developerworks/library/ws-transpec/. IBM, Microsoft, BEA.

[Dijkman and Dumas 2004] Dijkman, R. and Dumas, M. (2004). Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems*, 13(4):337–368.

[Dumas et al. 2005] Dumas, M., Aalst, W., and Hofstede, A. (2005). *Process-Aware Information Systems: Bridging People and Software through Process Technology*. John Wiley & Sons.

[Dustdar et al. 2004] Dustdar, S., Gombotz, R., and Baina, K. (2004). Web Services Interaction Mining. Technical Report TUV-1841-2004-16, Information Systems Institute, Vienna University of Technology, Wien, Austria.

[Elmagarmid 1990] Elmagarmid, A. K., editor (1990). *Database Transactions Models for Advanced Applications*. Morgan Kaufmann Publishers.

[Fauvet and Ait-Bachir 2006] Fauvet, M.-C. and Ait-Bachir, A. (2006). An automaton-based approach for web service mediation. In *Workshop on Web Services. 13th International Conference on Concurrent Engineering*, Antibes, France. Invited paper.

[Fauvet et al. 2005] Fauvet, M.-C., Duarte, H., Dumas, M., and Benatallah, B. (2005). Handling transactional properties in Web service composition. In *Proceeding of Web Information Systems Engineering (WISE)*, number 3806 in LNCS, New York City. Springer 2005, ISBN 3-540-30017-1.

[Fielding 2000] Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of Califormia, Irvine, USA.

[Fu et al. 2004] Fu, X., Bultan, T., and Su, J. (2004). Analysis of interacting BPEL web services. In *Proceedings of the 13th International Conference on World Wide Web (WWW)*, pages 621–630, New York, NY, USA.

[Gaaloul et al. 2004] Gaaloul, W., Bhiri, S., and Godart, C. (2004). Discovering Workflow Transactional Behavior from Event-Based Log. In *Proceedings of the OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004*, pages 3–18.

[Goldfard 1990] Goldfard, C.-F. (1990). *The SGML Handbook*. Oxford Clarendon Press.

[Gray and Reuter 1993] Gray, J. and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann.

[Hagen and Alonso 2000] Hagen, C. and Alonso, G. (2000). Exception handling in workflow management systems. *IEEE Transactions on Software Engineering*, 26(10):943–958.

[JSR 181] JSR 181. Web Services Metadata for the $Java^{TM}$ Platform. http://jcp.org/en/jsr/detail?id=181.

[Kaler and Nadalin] Kaler, C. and Nadalin, A. Web services security policy language (ws-securitypolicy) version 1.1. Standards proposal by IBM Corporation, Microsoft Corporation, RSA Security and VeriSign.

[Kavantzas et al. 2005] Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., and Barreto, C. (2005). Web Services Choreography Description Language, Version 1.0. W3 CCandidate Recommendation.

[Koutsonikola and Vakali 2004] Koutsonikola, V. and Vakali, A. (2004). LDAP: Framework, practices, and trends. *IEEE Internet Computing*, 8(5):66–72.

[Limthanmaphon and Zhang 2004] Limthanmaphon, B. and Zhang, Y. (2004). Web service composition transaction management. In Schewe, K.-D. and Williams, H., editors, *ADC*, volume 27 of *CRPIT*. Database Technologies 2004, Proceedings of the Fifteenth Australian Database Conference, ADC 2004, Dunedin, New Zealand, Australian Computer Society.

[Martens 2005] Martens, A. (2005). Analyzing Web Service Based Business Processes. In Cerioli, M., editor, *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, pages 19–33. Springer.

[OASIS UDDI 2005] OASIS UDDI (2005). Universal Description, Discovery and Integration version 3.0. http://www.uddi.org.

[O'Sullivan et al. 2002] O'Sullivan, J., Edmond, D., and ter Hofstede, A. (2002). What's in a Service? *Distributed and Parallel Databases*, 12(2–3):117–133.

[Papazoglou 2003] Papazoglou, M. (2003). Web services and business transactions. Technical Report 6, Infolab, Tilburg University, Netherlands.

[Peltz 2003] Peltz, C. (2003). Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52.

[Rozinat and Aalst 2005] Rozinat, A. and Aalst, W. (2005). Conformance Testing: Measuring the Alignment Between Event Logs and Process Models. BETA Working Paper Series, WP 144, Eindhoven University of Technology, Eindhoven.

[UN/CEFACT and OASIS 1999] UN/CEFACT and OASIS (1999). Electronic Business XML. http://www.ebxml.org.

[Vidyasankar and Vossen 2003] Vidyasankar, K. and Vossen, G. (2003). A multi-level model for web service composition. Technical report, Dept. of Information Systems, University of Muenster. Tech. Report No. 100.

[W3C-WSA-Group 2004] W3C-WSA-Group (2004). W3C Web Service Architecture Group. Web Services Architecture. http://www.w3.org/TR/ws-arch.

[W3C-WSD-Group] W3C-WSD-Group. Web Services Description Working Group. http://www.w3.org/2002/ws/desc/.

[W3C-XML] W3C-XML. XML coordination group. http://www.w3.org/XML/.

[W3C-XMLP-Group] W3C-XMLP-Group. XML Protocol working group. http://www.w3.org/2000/xp/Group/.

[W3C-XMLSchema] W3C-XMLSchema. XML coordination group. http://www.w3.org/XML/Schema.

[Weerawarana et al. 2005] Weerawarana, S., Curbera, F., Leymann, F., Story, T., and Ferguson, D. (2005). *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable-Messaging, and More*. Prentice Hall.

[Winer 1999] Winer, D. (1999). XML-RPC specification. http://www.xmlrpc.com/spec.

[Wohed et al. 2003] Wohed, P., Aalst, W., Dumas, M., and Hofstede, A. (2003). Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *22nd International Conference on Conceptual Modeling (ER 2003)*, pages 200–215. Springer.

[Yellin and Strom 1997] Yellin, D. and Strom, R. (1997). Protocol specifications and component adaptors. 19(2):292–333.