UNIVERSITÉ DU QUÉBEC À MONTRÉAL

INTEGRATION DU GRID ET DES SERVICES WEB

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE (GÉNIE LOGICIEL)

PAR

BASSEL NAFFAA

JUIN 2006

REMERCIEMENTS

Alors que j'arrive à la fin de cette aventure, j'aimerais remercier mon directeur de recherche Le Professeur Abdellatif Obaid, pour son soutien sans faille, son encadrement, ses encouragements, ses conseils et les discussions que nous avons eus.

Je tiens à remercier également tous les professeurs du département d'informatique de l'UQAM pour leurs enseignements et leur dévouement.

Mes remerciements tout spéciaux vont aussi à mes collègues et amis Walid, Hakim, Claude, Nabil et les autres pour leur soutien moral et matériel tout au long de ce travail.

Enfin, je remercie toutes les personnes qui me sont chères et qui tiennent une grande place dans mon coeur, en commençant par ma famille : mes parents tout d'abord, sans qui bien sûr je ne serais pas arrivé au bout de mon parcours et qui ont toujours été présent, mes frères et sœurs et ma femme pour leur patience et leurs encouragements dans les moments difficiles.

À mes parents et ma femme, je vous dédie ce mémoire en signe de reconnaissance.

Je le dédie également à mes frères, mes sœurs et mes neveux, qu'il soit pour vous une source d'inspiration et un exemple de patience et de persévérance.

Avec tout mon cœur,

TABLE DE MATIERES

LISTE DE FIGURES

RÉSUMÉ

Cette recherche vise à intégrer l'environnement du GRID avec le registre UDDI. La combinaison des technologies de services Web et l'environnement du Grid Computing est actuellement d'un majore révolution scientifique. Elle combine la solution de logiciel personnalisé des services Web et des solutions de partage des ressources du Grid Computing.

Ce mémoire explore en profondeur le nouvel environnement de technologie de GRID, qui constitue une innovation centrale de Globus ToolKit et les services Web. Les recherches effectuées s'appuient d'une part sur le Globus ToolKit4 du Grid et le service Web, spécifiquement, le registre UDDI.

Les Services Web forment la base pour la dernière technologie de calcule distribuée. Ils peuvent être décrit comme une plate forme indépendante, un langage indépendant et les components des logiciels sont couplés facilement et accessibles sur l'Internet.

Un Grid Computing se caractérise par des utilisateurs et des ressources dispersés et hétérogènes, un état des ressources matérielles (calcul, stockage...) et logicielles variable, ainsi que des groupes d'utilisateurs et une connectivité réseaux également variables au cours du temps.

Il apparaît clairement dans un tel contexte que l'utilisation de standards est un point particulièrement critique pour la réussite de la mise en place d'infrastructure de Grid, notamment pour faire face aux problèmes d'intégration avec les services Web.

Pour réaliser notre but, nous allons créer un système du répertoire basé sur le Grid Service et le registre UDDI afin de garantir l'interopérabilité de systèmes hétérogènes pour le partage et l'accès à des ressources de calcul et de stockage distribuées.

Chapitre I

INTRODUCTION

1.1 Évolution technologique

Le besoin de communication entre les applications informatiques n'a jamais cessé de croître avec la popularisation du réseau Internet et du web. L'architecture client/serveur a permis d'offrir divers services répartis entre des clients et des serveurs d'applications réparties (exemple : serveurs d'informations, messagerie, etc.)

.

À ses débuts, la technologie impliquait une programmation longue et difficile d'approche en utilisant des mécanismes de bas niveau tels que les sockets. Entre autres, elle imposait au concepteur de logiciels de mettre au point des protocoles de communication relativement complexes. Ces protocoles devaient contrôler la transmission des données et effectuer les vérifications d'usage et le codage/décodage des données échangées. Le programmeur devait donc réserver une bonne part de son temps à la conception du protocole, affectant le temps consacré au processus de conception du logiciel lui-même.

On a ensuite développé des technologies de plus haut niveau telles que RPC (*Remote Procedure Call*), qui permet de séparer une application en deux modules (le serveur et le client) de fonctionnalités différentes et de les distribuer sur des machines différentes sous forme de procédures qui peuvent être exécutées à distance à travers un réseau. Cette technologie permet aux applications d'exécuter des procédures appartenant à des applications distantes et ce, de façon transparente. Avec cette technologie, le programmeur n'a plus à développer des protocoles gérant les méthodes de transfert, de vérifications et de codage/décodage. Ces opérations sont entièrement

prises en charge par RPC.

L'évolution des langages de programmation a amené de nouveaux outils aidant à la conception d'applications. Par exemple, la venue du langage orienté objet a facilité l'abstraction des problèmes à résoudre en fonction des données du problème lui-même (par l'utilisation de classes et d'instances d'objets). Elle a aussi permit l'essor de nouvelles technologies de distribution des applications. Les concepts d'héritage, d'encapsulation et de polymorphisme sont mis à profit avec langage orienté objet. Java RMI (*Remote Methode Invocation*) est un exemple de technologie orientée objet permettant de distribuer une application. Grâce à ce paradigme (de manière semblable à RPC), un client peut invoquer des méthodes d'un objet distant.

Les applications distribuées peuvent prendre plusieurs formes. Le système le plus simple est constitué d'un client et d'un serveur échangeant des informations, et ce, sans autre interaction avec le reste du réseau. Le serveur attend et répond aux requêtes des clients, et les clients transmettent leurs requêtes. Les informations peuvent être échangées de façon bidirectionnelle entre les deux modules.

Des environnements de distribution d'applications ont également vu le jour, tels que CORBA (*Common Object Request Broker Architecture*) [WWW 9], qui permet à une application cliente d'effectuer une requête à un objet distant et de recevoir en retour le résultat de l'opération effectuée, le tout en utilisant un paradigme d'appel à distance tel que RMI. Cet environnement permet de séparer une application en plusieurs parties (composantes ou objets) et de les distribuer sur un réseau tout en fournissant les outils nécessaires à la gestion de la communication entre les modules et la gestion du cycle de vie des objets.

CORBA a permis de mettre au point des services génériques que les développeurs peuvent intégrer dans leurs applications. Il a également introduit un langage de description des interfaces (ou contrats) des services offerts. Ce langage, appelé IDL (pour *Interface Definition langage*) permet des descriptions abstraites de ces contrats

et est utilisé pour générer des codes écrits en divers langages (ex. Java, C++) de manière automatique.

CORBA a également introduit la notion de référentiel. On a défini, entre autres, un référentiel d'implantations d'objets (*Implementation Repository*) et un dépôt de définitions de contrats (*Interface Repository*). Ceux-ci permettent de déposer des objets en vue de leur découverte et leur utilisation.

Cette évolution des systèmes distribués a créé un nouveau défi. Par exemple, il fallait décider de la disposition des éléments qui constituent l'application : les interfaces avec les utilisateurs, les traitements et les données.

La venue du Web a permis de généraliser ces outils pour des applications hétérogènes en permettant de les modifier et de les incrémenter tout assurant l'accès aux ressources distribués sur l'Internet. Le web a été utilisé en tant que middleware permettant d'accéder à des données et à des applications. La disponibilité des serveurs web et des clients web (les navigateurs) a permis l'accessibilité de plus en plus grande à ces données et services.

La tendance qui a marqué cette évolution a poussé les concepteurs à utiliser le web pour stocker des services sous forme de composantes accessibles à travers le réseau, appelées *Services web*. Les services web comportent plusieurs composants :

- Un protocole d'échange de messages de requêtes et de réponses appelé *SOAP* (*Simple Object Access Protocol*).

- Un langage de description des contrats des services (une sorte de IDL) appelé *WSDL* (*Web Service Description Language*).

- Un référentiel de description des services offerts appelé *UDDI* (*Universal Description Discovery and Integration*).

Les services web ont émergé comme un mécanisme puissant assurant l'interaction entre des fournisseurs de services et des consommateurs sur différentes plates-formes technologiques sur le web. Ils fournissent des mécanismes flexibles, extensibles basés sur le langage XML (pour la représentation des données et des messages) [www 10] ainsi que les protocoles standard de l'Internet tels que HTTP et FTP. Ces mécanismes facilitent le développement des systèmes en utilisent une architecture dite *Architecture Orientée Services* (*Servive-Oriented Architecure ou SOA*) [Mailvaganam, 2005].

L'évolution des logiciels applicatifs des architectures client-serveur s'oriente de plus en plus vers des architectures avec services Web.

Plus récemment, des infrastructures plus performantes ont été envisagées pour traiter et stocker des masses de données et pour accéder à des ressources de différents types (logiciels, ressources de calcul, etc.) sous forme de grilles (ou *GRID*) [Foster, 2004]. Le Grid est un environnement de distribution des ressources et des traitements permettant aux ordinateurs de communiquer entre eux et aux applications de travailler ensemble, en permettant l'allocation des ressources, l'amélioration de la qualité de service et de la sécurité des transactions.

Par rapport aux systèmes répartis classiques, ces nouveaux systèmes possèdent deux caractéristiques fondamentales :

- La montée en échelle du nombre de ressources.
- La complexité des ressources d'où les ordinateurs peuvent exécuter des programmes complexes avec un niveau d'autonomie élevée.

Le paradigme de calcul sur le Grid (*Grid Computing*) a récemment évolué vers l'intégration ou l'interopérabilité avec les services Web. Ceci a donné naissance à une architecture appelée *Open Grid Service Architecte* (*OGSA*) [Li 2005] qui permet de bénéficier des avantages des services web et des architectures basées sur le Grid. D'autre part, l'intégration entre l'environnement du Grid et le registre UDDI pose des

problèmes d'interopérabilité. Le travail de ce mémoire s'intéresse plus particulièrement à cette problématique d'interopérabilité.

1.2. Problématique

Nous nous intéressons aux problèmes liés à la conception, au déploiement et à l'exploitation de l'environnement de Grid pour rendre l'accès aux informations de services web disponibles sur le registre du UDDI en particulier. Nous voulons que les utilisateurs du Grid puissent accéder à ce registre de manière transparente.

Notre objectif est de mettre en oeuvre des environnements qui intègrent aussi bien les besoins des utilisateurs (à travers l'expression de besoins de ressources ou d'applications) que les contraintes liées à la mise à disposition de services et ce, à travers la description de services et l'intégration ou le déploiement de serveurs.

Des différences essentielles rendent l'utilisation de registre UDDI dans un environnement de Grid inadéquate, à savoir le cycle de vie de service et l'état de service*. Tandis que la statique de registre de UDDI convient pour publier des informations sur des services statiques et persistants, elles nous paraissent inadéquates pour stocker des informations sur des services dynamiques et transitoires tels qu'on les trouve sur le Grid. Nous voulons également établir un lien entre l'environnement de GRID et le registre UDDI pour permettre aux utilisateurs du Grid de pouvoir accès au registre UDDI et de chercher des services disponibles ou ceux dont ils ont besoin.

1.3. Objectifs visés

L'objectif principal de cette étude est de concevoir et d'implanter un intermédiaire (c.-à-d. un middleware) entre l'environnement de Grid et le registre UDDI qui permette l'échange entre services.

---

* L'état dans le sens state (ex. statefull ou stateless).

Ce projet vise les sous-objectifs suivants :

- Établir une stratégie de découverte et d'annonce des services.
- Mettre en place une architecture qui facilite l'invocation des services dans un environnement de GRID.
- Développer un système de répertoire basé sur le UDDI pour les services sur le Grid qui permette d'utiliser les fonctions de stockage et de recherche selon le standard de services Web.
- Intégrer l'environnement du Grid et le registre UDDI.

Pour atteindre ces objectifs, le travail est divisé en trois parties:

- Partie utilisateur de Grid: Les utilisateurs des applications auraient une vue transparente qui expose l'information liée à un environnement des services.
- Partie environnement de Grid : Celle-ci inclut l'état des ressources disponibles, les outils, le contrôle et la surveillance des informations dans l'infrastructure de Grid.
- Partie implantation : Un système répertoire inclut le registre UDDI et les services du Grid pour établir un pont entre l'environnement du Grid et le registre UDDI qui permettra aux utilisateurs de Grid de chercher et stocker des informations dans ce registre et réciproquement.

1.4. Organisation du mémoire

Ce mémoire est divisé en plusieurs chapitres. Dans le premier chapitre, nous présentons les services Web en montrant leurs caractéristiques principales et les principaux standards (WSDL, SOAP et UDDI). Le second décrit l'environnement de Grid Computing et ses composants. Dans le troisième chapitre, nous discutons en

premier lieu de Globus ToolKit [Jacob 2004], l'outil que nous avons utilisé, et de ses fonctionnalités afin d'analyser les différentes composantes existantes. Dans le quatrième chapitre, nous discutons de la convergence entre l'environnement du Grid et les services Web. Ensuite, nous présentons les nouvelles normes pour supporter cette convergence comme *Open Grid Service Architecture* (*OGSA*), *Open Grid Service Infrastructure* (*OGSI*), *Web Service Resource Framework* (*WSRF*) et le *Grid Service* (*GS*). Le chapitre suivant traite des différents aspects d'interfaçage entre les Grid services et les services Web. Au cours du sixième chapitre, les détails d'implémentation du prototype d'un système de répertoire que nous avons implanté sont présentés, les étapes franchies lors de notre démarche d'intégration du Grid avec le registre UDDI et la mise en œuvre de notre proposition. Enfin, le septième chapitre conclut le travail du mémoire.

Chapitre II

LES SERVICES WEB

2.1. Introduction

Les services Web ont émergé comme une nouvelle génération de technologies sur le Web pour échanger l'information et accéder à des services. Ces services sont des applications pouvant se décrire sous forme de composants disponibles sur Internet, basées sur des normes ouvertes, et permettant d'établir des applications en utilisant différentes plateformes.

Les services Web sont des composants logiciels conçus pour soutenir l'interaction et l'interopérabilité entre les différentes machines via l'Internet en utilisant XML comme moyen de représentation de données ainsi que des protocoles de communication de l'Internet ([Mailvaganam, 2005], [Zhang, 2004]). Les utilisateurs et les fournisseurs de ces services communiquent entre eux selon le protocole d'échange SOAP (Simple Object Access Protocol) ([WWW 1], [WWW 2]).

La modularité et la flexibilité des services Web rendent ceux-ci bien adaptés pour l'intégration de leurs applications. Les entreprises peuvent installer et utiliser ces services avec un minimum de programmation. Une fois qu'un service est déployé, d'autres applications et services Web peuvent les découvrir et les appeler.

L'usage du protocole HTTP améliore l'accessibilité de ces services]. En effet, ce protocole existe sur tous les sites serveur (les serveurs HTTP sont largement déployés) et clients (les navigateurs web sont largement disponibles) et ce, sur tous les postes clients. Par ailleurs, l'usage XML offre plusieurs avantages. Il permet entre autres de décrire des messages complexes et adaptés aux applications client/serveur (Figure 1.2).

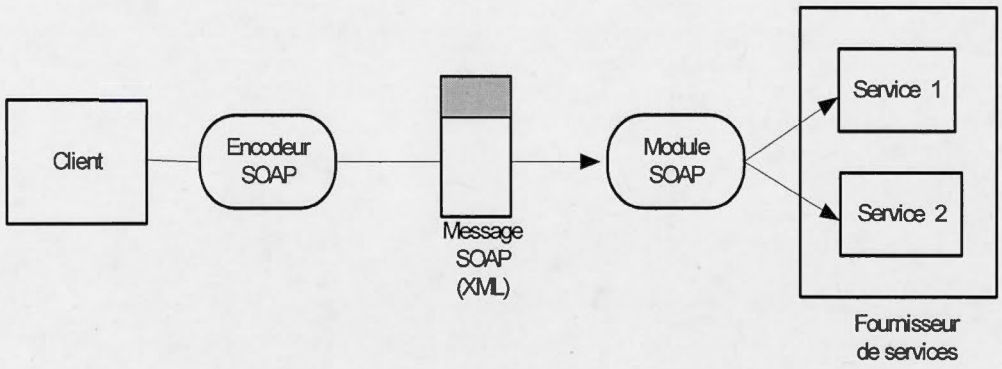


Figure 2.1. Service web et SOAP.

L'architecture des services web comportent des éléments servant à ~~pour~~décrire les services offerts (à la façon d'un IDL), à découvrir ces services et à véhiculer des messages pour les invoquer. Ces éléments sont :

- *SOAP* (*Simple Objext Access Protocol*) ~~:~~est un protocole servant à échanger des ~~de~~ données XML Il définit les éléments qui permettent d'effectuer des appels de type RPC en se basant sur HTTP comme protocole de transport.

- *WSDL* (*Web Services Description Language*) : est un vocabulaire pour définir ce que le service web fait, où il réside et comment l'invoquer. Il s'agit d'un langage IDL. Celui-ci permet aux fournisseurs d'un service de décrire les données et les commandes que ce service accepte. Ces descriptions sont stockées dans un registre que les clients potentiels peuvent consulter pour découvrir ces services et ces descriptions.

- *UDDI* (*Universal Description, Discovery and Integration*) : est un registre auquel on peut accéder pour trouver les descriptions des services ainsi que leurs caractéristiques d'accès. Lorsqu'un fournisseur veut rendre un service accessible, il le publie dans UDDI avec une commande de type *publish*. Les clients peuvent alors effectuer des recherches de services (avec une

commande de type *find*) pour les utiliser dans leurs applications (avec une commande de type *bind*). L'accès à UDDI se fait également au moyen du protocole SOAP.

2.2. Les interactions orientées services web

Tous les éléments de l'architecture des services web utilisent XML pour représenter les descriptions avec WSDL, les stocker dans UDDI et échanger les messages de requêtes et réponses avec SOAP. Ces éléments sont présentés dans la figure 2.2.

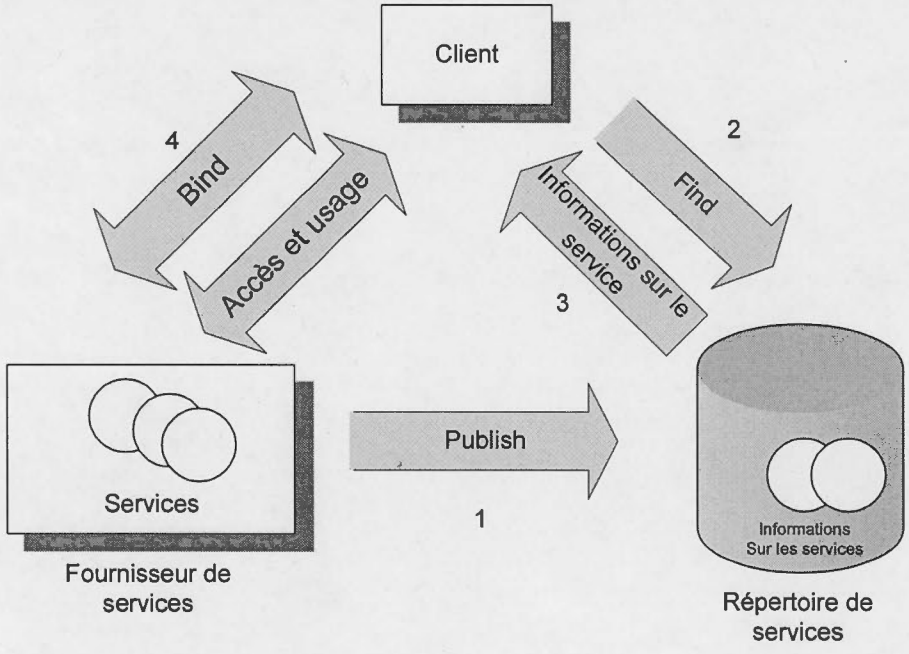


Figure 2.2 : Éléments de l'architecture basée sur les services web.

La figure 2.2 représente le paradigme appelé *Find, bind and publish.* Dans ce paradigme, les fournisseurs de services enregistrent ces services dans le registre (en l'occurrence UDDI). Celui–ci est consulté par les clients (aussi appelés *consommateurs*) pour retrouver des services qui satisfont certains critères. Si un tel service existe, le client obtient les informations de contact, l'adresse de son point d'accès, etc., et il peut alors l'invoquer.

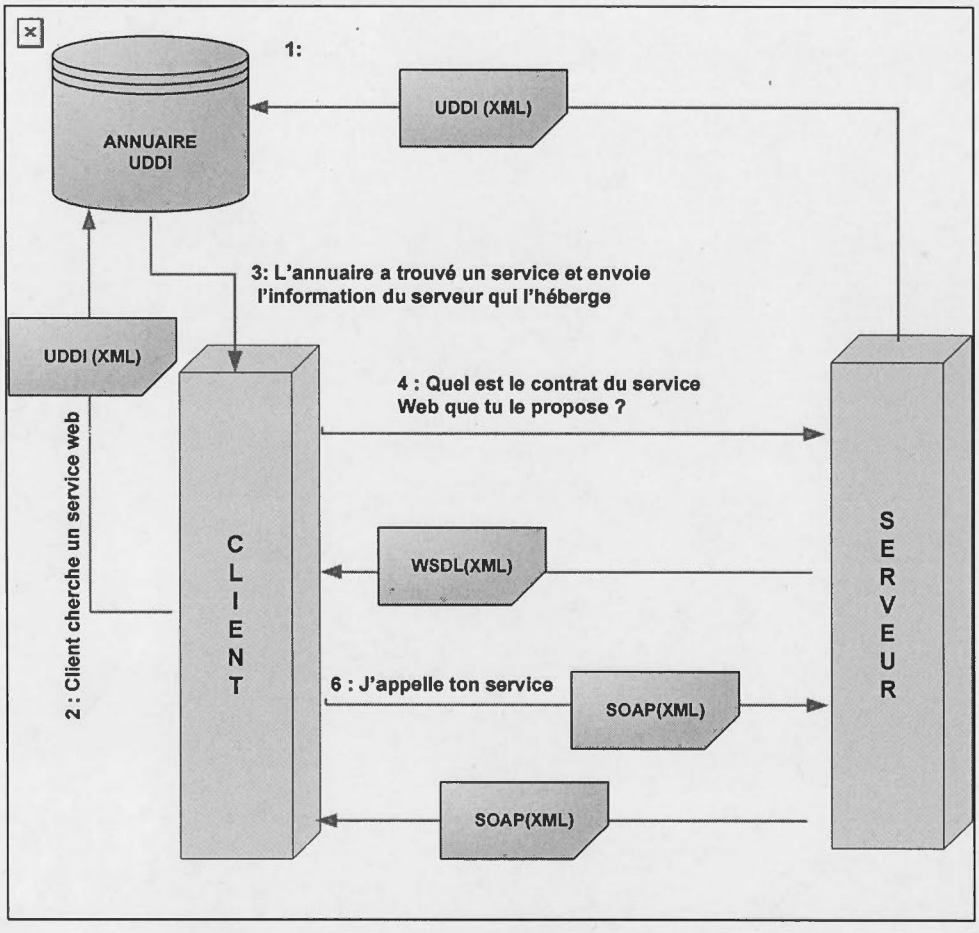


Figure 2.3. Le modèle de fonctionnement des services web.*

La figure 2.3 illustre le modèle de fonctionnement des services Web. Ce modèle présente les grandes phases de la mise en oeuvre d'un service Web. Le fournisseur doit d'abord réaliser ses services Web et produire leurs descriptions sous format WSDL.

1. Le fournisseur publie ses services dans l'annuaire UDDI en envoyant les messages encapsulés dans une enveloppe SOAP au registre UDDI via un protocole de transport (ex. HTTP). Les informations fournies par le

---

* Source : http://www.lamsade.dauphine.fr/~mellit

fournisseur contiennent les méthodes d'invocation, les paramètres, la localisation du service ainsi le format de la réponse.

2. Le client recherche les informations sur les services et dans le registre UDDI.
3. L'annuaire UDDI retourne la réponse (qui est un message WSDL) au client.
4. Dès que le client reçoit la réponse, il interroge le fournisseur pour pouvoir accéder au service. La demande se fait via un message de SOAP.
5. Le fournisseur répond en donnant le format que le client doit utiliser pour effectuer l'appel.
6. Le client envoie sa requête d'appel sous forme d'un message SOAP au fournisseur pour obtenir le service demandé.
7. Le serveur envoie la réponse à cet appel.

En vue de faciliter l'interopérabilité et l'accès aux services web, ceux-ci se basent sur des standards et des normes bien définies : langages de description, protocoles de transport, registre, etc. Ces protocoles et ces langages sont basés sur XML. Ces éléments sont décrits dans les sections ci-dessous.

2.3. Le langage XML

XML a été introduit en tant que langage de représentation de données dans le contexte de l'Internet. C'est un langage extensible, car chaque utilisateur peut définir ses propres éléments. Il a été créé pour structurer, stocker et échanger les informations. L'utilisateur décrit la façon dont ces éléments doivent être agencés dans un document XML grâce à des *DTD* (*Document Type Definition*) ou des schémas.

XML offre les caractéristiques suivantes :

- Tel qu'il est conçu, XML facilite le partage des données entre différentes plates-formes. Les données XML sont sous forme textuelle.
- Pour décrire les types de données contenues dans un document, on peut utiliser soit une DTD (Document Type Définition) ou un *schéma XML*. Les schémas sont décrits plus loin dans ce mémoire.

- XML est extensible, car les descriptions se basent sur l'utilisation de balises (comme dans le cas de HTM) et l'utilisateur peut introduire ses propres balises selon ce qu'il ait décrit dans le schéma ou le DTD.
- Avec XML, les données sont séparées de la présentation. Ces données peuvent être stockées dans un fichier séparé. On peut utiliser des techniques d'affichage (du type HTML) qui seront séparées de ces données. Et tout changement sur les données se reflétera sur la présentation en HTML.

XML joue également un rôle de plus en plus important dans l'échange d'une grande variété de données sur le Web. L'exemple ci-dessous montre un document XML qui décrit un menu:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <menu>
     <repas >
        <nom>Sandwich au poulet</nom>
        <prix>$6.00</prix>
        <description>Un morceau de viande de poulet avec salade</description>
        <calories>650</calories>
     </repas>
   <repas >
        <nom>Gateau au fromage</nom>
        <prix>$15.05</prix>
        <description>Gateau avec fromage et fraises</description>
     <calories>1000</calories>
   </ repas >
</menu>
```

Un document XML peut être utilisé pour décrire un service Web incluant l'information détaillant comment ce service peut être exécuté. Quand un client a besoin d'un service Web, le document XML est trouvé d'abord. Puis, les détails sur la manière dont le service peut être invoqué sont exploités.

XML emploie une DTD ou un schéma pour décrire les données. Il s'agit de métalangages qui permettent de définir tous les éléments qui formeront le document, de décrire les types de leurs données ainsi que les relations hiérarchiques entre eux. Ils permettent également de définir leur syntaxe. Le document XML associé à une

DTD ou un schéma doit satisfaire ces descriptions, sinon, il est considéré invalide. Le schéma est une alternative à un DTD et permet de décrie les types de données de manière plus expressive et possède un ensemble riche des types de données de base ainsi que des éléments de syntaxe qui permettent de construire des types plus complexes. De nous jours on utilise davantage les schémas que les DTD.

L'exemple ci-dessous présente un exemple simple de schéma XML d'une bibliothèque qui peut contenir un ou plusieurs livres. Un livre doit avoir un titre, un ou plusieurs auteurs, ainsi qu'un éditeur. La description décrit également la hiérarchie entre les éléments du document.

```
<?XML version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="librairie">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="livre" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="livre">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="titre"      type="xs:string" />
      <xs:complexType>
       <xs:sequence>
         <xs:element name="auteur"  type="xs:any" minOccurs="1"/ >
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="éditeur "          type="xs:string" />
   </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Les noms des éléments et des attributs dans un document XML doivent être uniques (c-à-d. on ne devrait pas donner le même nom à des éléments différents). Afin d'éviter des conflits de noms, XML ajoute des préfixes qui définissent des *espaces de noms*. Un espace de noms est déclaré dans un élément en utilisant un attribut spécial

appelé *xmlns* (*XML name space*) de la forme *<pref:element xmlns:pref="URL">*. L'espace de noms est désigné par le nom *pref*. L'attribut *xmlns* permet d'associer ce nom à un URL unique. Chaque fois que l'élément apparaît dans le document, il doit être précédé de ce préfixe. Dans l'exemple ci-dessus, on utilise l'espace de noms appelé *xs* associé à l'URL http://www.w3.org/2001/XMLSchema.

2.4. Le protocole SOAP

SOAP (*Simple Object Access Protocol*) [WWW 1] est un protocole qui permet d'échanger les informations sur les services web dans un environnement distribué via l'Internet. Il ne définit pas de protocole de communication, mais s'appuie sur des protocoles de communication tels que HTTP. SOAP permet d`échanger des messages d'appels de type RPC pour invoquer des méthodes des services. Ces appels sont accompagnés de paramètres et des valeurs de retour. Les messages et les réponses SOAP sont en XML et respectent des types d'encodage spécifiques. De plus, les éléments des messages SOAP sont définis grâce à des espaces de noms qui permettent de les distinguer des autres éléments XML.

SOAP se compose de trois parties:

- *Enveloppe SOAP* définit un cadre global pour exprimer ce qui est dans un message, qui devrait le traiter et s'il est facultatif ou obligatoire.
- *Encodage SOAP* définit un mécanisme d'encodage (une sorte de sérialisation) employé pour échanger les messages.
- *Représentation SOAP RPC* définit une convention pour représenter les appels et les réponses RPC.

Un message de SOAP contient les éléments illustrés dans la figure 2.4:

Figure 2.4. Structure des messages SAOP.

Le message SOAP

Un message SOAP, appelé *Enveloppe* (*SOAP Envelope*), est décrit par l'élément *SOAP-Envelope*. Il contient un en-tête optionnel suivi du corps du message. Ce corps contient les données du message encodées en XML selon un codage standardisé. La structure du message est constituée des éléments suivants :

- *L'en-tête*: il contient des éléments qui peuvent être appliqués au corps du message. Par exemple, on pourra les utiliser pour véhiculer des informations de sécurité le long d'un chemin parcouru par le message.

- *Le corps*: il contient les données du message SAOP. Ces données sont destinées au récepteur final. Le corps est généralement utilisé pour des appels RPC, leurs résultats ainsi que des messages d'erreurs.

- L'*enveloppe* contient deux attributs qui décrivent un espace de nom et un type de codage appelé *encodingStyle* :

- *L'espace de noms :* les messages SOAP doivent contenir un espace de noms dont l'URI est : *http://schemas.xmlsoap.org/soap/envelope/.* En fait, cet espace de noms correspond à la version de SOAP utilisée. Tout nœud qui reçoit un message avec un autre espace de noms est jugé invalide et génère un message d'erreur.

- *L'attribut encodingStyle :* cet attribut permet de spécifier la manière de représenter les données dans les messages. Ceci permet de réaliser une forme de sérialisation des messages.

Un exemple de message SOAP est donné ci-dessous. Ce message constitue un appel à la procédure Conv qui doit convertir une chaîne de caractères de minuscule à majuscule.

```
<soap:Envelope
      xmlns:soap =http://schemas.xmlsoap.org/soap/envelope/
      soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      < soap:Body>
        <mes:Dialog xmlns:mes="Conv">
              <Donnee xsi-type="xsd:string"> Hello! </Donnee>
        </mes:Dialog>
      </soap:Body>
</soap:Envelope>
```

L'en-tête SOAP

SOAP permet de faire passer les messages vers un destinataire final en passent par des récepteurs intermédiaires représentés par l'attribut *actor*. Ce message peut contenir des en-têtes. Certains de ces en-têtes peuvent être destinés à ces récepteurs intermédiaires. Un en-tête SAOP permet d'ajouter des informations spécifiques pour le traitement d'un message lorsque celui-ci traverse un chemin du client vers le destinataire final. Pour chaque récepteur intermédiaire, on doit spécifier son identité et indiquer s'il doit prendre en compte l'en-tête en question (attribut *mustUnderstand*). Les intermédiaires peuvent traiter les messages qui leurs sont destinés et les acheminer vers la prochaine intermédiaire. Le code qui suit montre un

exemple d'en-tête :

```
<soap:Header>
   <mes:Message
        xmlns:mes="URL"
        soap:actor="http://www.info.uqam.ca/application
        soap:mustUnderstand="1" >
              Ce message est pour toi et tu dois le prendre en compte !
     </mes:Message>
 </ soap:Header>
```

Le corps SOAP

Le corps véhicule des messages RPC entre un client et un destinataire final (le serveur). Ce message peut contenir des éléments propres à l'appel d'un service et à la syntaxe pour l'envoi de ses paramètres. Le corps du message est défini par une balise appelée *Body*. L'exemple qui suit illustre l'utilisation d'un message SOAP pour l'appel RPC de l'opération PrixDe d'un service ayant un seul paramètre, qui est le nom du produit dont on veut le prix (dans ce cas, Oranges) :

```
<?xml version="1.0"?>
<soap:Envelope
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
      <mes:PrixDe xmlns:mes="http://www.info.uqam.ca/exemples">
         <mes:Produit>Oranges</mes:Produit>
      </mes:PrixDe>
</soap:Body>
</soap:Envelope>
```

La réponse à une requête RPC avec SOAP est également contenue dans le corps d'un message SOAP. Par exemple, la réponse à la requête précédente pourrait être comme suit :

```
<?xml version="1.0"?>
<soap:Envelope
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
<soap:Body>
  <mes:ReponsePrixDe xmlns:mes="http://www.info.uqam.ca/exemples">
    <mes:Prix>4</m:Prix>
  </mes:ReponsePrixDe >
</soap:Body>
</soap:Envelope>
```

Les messages Fault

Ce message est émis par une destination lorsqu'une erreur se produit. Il contient les informations sur l'erreur en question ainsi que des informations d'état. Un message *Faut* contient les éléments pour identifier l'erreur et les informations sur qui l'a produite.

Transport des messages SOAP

Les messages SOAP sont généralement transportés par des messages HTTP. Ceci est du au fait que ce protocole est simple et extensible grâce à l'utilisation des en-têtes. D'autres protocoles tels que SMTP et FTP peuvent également être utilisés. Mais étant donné que HTTP est simple et disponible auprès des clients (grâce aux navigateurs), il est le protocole le plus utilisé.

Les messages SAOP sont transportés dans des requêtes HTTP avec la méthode POST uniquement (la méthode GET est limitée par la quantité de données qu'elle peut contenir). Une commande HTTP émet un bloc de données constitué du message SAOP lui-même. L'en-tête *Content-Type* dans la requête HTTP doit être *text/xml*.

Les réponses SAOP sont également retournées dans un message de réponse HTTP.

Un en-tête HTTP spécifique appelé *SOAPAction* est introduit pour désigner la composante spécifique qui doit traiter le message SOAP. Il permet de préciser une destination particulière du message SOAP transporté dans la requête HTTP. Il est utilisé, par exemple, par des firewalls pour décider de la destination finale du message. Si cet en-tête est vide, alors sa valeur et identique à l'URL de la ligne POST de HTTP. Nous avons ensuite l'enveloppe SOAP. Son corps indique le nom de

l'opération invoquée.

L'exemple qui suit montre un message SAOP transporté par une requête HTTP. Il montre également la réponse et le message HTTP qui lui est associé.

```
POST   http://www.info.uqam.ca/prix.asp HTTP/1.1
Content-Type: text/xml
Content-Length: 308
SOAPAction: "destination"
<?xml version="1.0"?>
<soap:Envelope
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
      <mes:PrixDe xmlns:mes="http://www.info.uqam.ca/exemples">
        <mes:Produit>Pommes</mes:Produit>
        </mes:PrixDe>
</soap:Body>
</soap:Envelope>
```

Suite à cette requête HTTP, on obtient la réponse HTTP contenant le message de réponse SAOP :

```
HTTP/1.1 200 OK
Content-Type: application/soap
Content-Length: 338
<?xml version="1.0"?>
<soap:Envelope
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
  <mes:ReponsePrixDe xmlns:mes="http://www.info.uqam.ca/exemples">
    <mes:Prix>4</m:Prix>
  </mes:ReponsePrixDe >
</soap:Body>
</soap:Envelope>
```

2.5. Le langage de description WSDL

WSDL (*Web Services Description Language*) [www 11] est un langage basé sur XML qui sert à décrire des services web. Il permet de décrire les fonctions de ces services et la manière d'y accéder à travers le web. Ceci contient des informations

opérationnelles concernant le service :

- L'interface du service;
- Les protocoles d'accès;
- Les points d'activation.

WSDL définit la description du service en termes de messages échangés dans une interaction avec ce service. Il y trois composantes principales dans cette description : le vocabulaire, le message et l'interaction. Le vocabulaire est le point de départ de cette interaction. WDSL utilise un système de types externes de XML pour décrire les types de données utilisés dans les échanges. Ces éléments sont disposés tel que le montre la figure 2.5.

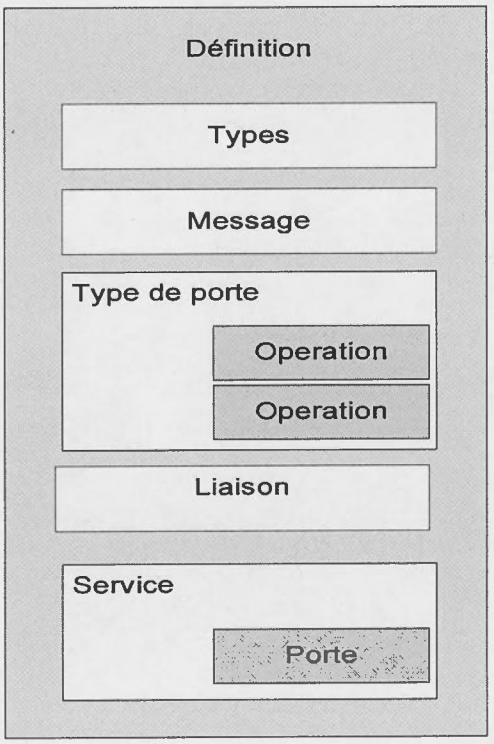


Figure 2.5. Structure de document WSDL.

Ces éléments sont décrits comme suit.

<definitions>

L'élément *<definitions>* est l'élément racine d'une description WSDL. Il spécifie le nom du service, accompagné de certains espaces de noms à inclure dans l'élément. Ces espaces de noms permettent d'accéder à des spécifications externes dont la spécification WSDL, SOAP et XML.

```
<definitions
    name="ServiceDePrix"
    targetNamespace="http://www.info.uqam.ca/ ServiceDePrix.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:ns=" http://www.info.uqam.ca/ServiceDePrix.wsdl"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Notons l'utilisation de l'attribut *targetNamespace* qui permet de spécifier les préfixes à utiliser dans la description WSDL. Notons également l'espace de noms par défaut, *http://schemas.xmlsoap.org/wsdl/*, qui sera utilisé pour les éléments qui ne font partie d'aucun espace de noms spécifique.

<types>

L'élément *<types>* contient la définition des types de données qui sont utilisées dans le service. Normalement, cet élément inclut l'élément *<schéma>* qui définit les différents types de données selon un schéma XML. Par exemple, si l'on veut définir un type ProduitQuantite pour représenter le nom et la quantité d'un produit dont veut obtenir le prix, on écrirait :

```
<definitions
    name="ServiceDePrix"
    targetNamespace="http://www.info.uqam.ca/ ServiceDePrix.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:ns=" http://www.info.uqam.ca/ServiceDePrix.wsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <type>
    <xs:schema
            xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
        <xs:element  name="ProduitQuantite">
                <xs:complexeType>
                        <xs:sequence>
                                <xs:element name="nom"  type="xs:string"/>
                                <xs:element name="quantite"  type="xs:decimal"/>
                        </xs:sequence>
                </xs:complexeType>
        </xs:element>
    </xs:schema>
    </type>
</definitions>
```

<message>

L'élément *<message>* décrit les données qui peuvent être échangées entre le fournisseur d'un service et ses clients. Chaque méthode (ou opération) du service a deux messages: *input* et *output*. Le *input* décrit les paramètres de l'opération alors que le *output* décrit les données retournées. Chaque message contient zéro ou plusieurs paramètres d'éléments *<part>*, qui désigne un paramètre de l'opération. Dans l'exemple qui suit nous décrivons les messages d'appel du service de prix. L'appelant donne le nom du produit comme paramètre d'entrée dans le message de requête et le service retourne un entier indiquant le prix dans le message de réponse. Notons que dans cet exemple, chaque message comprend à une seule partie indiquée par le sous-élément *<part>*.

```
<message name=" ServiceDePrixRequete">
  <part name="produit" type="xsd:string"/>
</message>
<message name=" ServiceDePrixReponse">
  <part name="Prix"  type="xsd:decimal"/>
</message>
```

<portType>

L'élément *<portType>* permet de spécifier l'ensemble des opérations d'un service :

les opérations qui peuvent être exécutées et les messages qui sont impliqués. Chaque opération inclut le message *Input* et *Output*. Par exemple pour avoir la température, en utilisant l'opération *getTemp*, de type requête/réponse on écrirait:

```
<portType name="TemperaturePortType"><operation name="getTemp">
  <input message="tns:getTempRequest" />
  <output message="tns:getTempResponse" />
 </operation>
</portType>
```

<binding>

Les éléments WSDL que nous avons décrits jusqu'à présent décrivent les fonctionnalités du service au niveau des applications. Cette description est plutôt abstraite. Pour compléter la description de l'interaction, on doit spécifier quel protocole de communication utiliser (ex SOAP sur HTTP) en établissant une *liaison* (*binding*) entre la description et le protocole. On doit également spécifier le protocole de transport utilisé et la façon de transporter les messages sur ce protocole. Ces éléments permettent de préciser les aspects concrets de la description. L'exemple ci-dessous montre l'élément *binding* de l'exemple Temperature.

```
<binding name="TemperatureBinding" type="tns:TemperaturePortType">
    <soap:binding style="rpc" transport="
    http://schemas.xmlsoap.org/soap/http" />
  <operation name="getTemp">
   <soap:operation soapAction="" />
    <input>
     <soap:body use="encoded" namespace="um:xmethods-Temperature"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
     <soap:body use="encoded" namespace="um:xmethods-Temperature"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
</operation>
</binding>
```

<service>

L'élément *service* est une collection de portes. Pour chaque port, on doit spécifier le protocole de transport et la liaison (binding) utilisés. L'exemple ci-dessous montre le port de l'exemple Température :

```
<service name="TemperatureService">
 <documentation>Returns current temperature in a given U.S.
     zipcode</documentation>
 <port name="TemperaturePort" binding="tns:TemperatureBinding">
  <soap:addresslocation="http://services.xmethods.net:80/soap/servlet/rpcrouter" />
 </port>
</service>
```

2.6. Le registre UDDI

UDDI (*Universal Description, Discovery and Integration*) [www 4] est une plateforme basée sur XML pour publier et localiser des informations sur des services Web, pour décrire des services, découvrir des entreprises et interagir avec des services. L'interaction entre un client et un service UDDI repose sur le protocole de SOAP.

UDDI permet de stocker des informations techniques et formelles telles que l'adresse pour accéder aux services et des informations contextuelles telles que le nom de la personne qui s'occupe de leur gestion, la description sommaire de leurs fonctionnalités ou encore le type d'activité de l'entreprise qui offre ces services.

UDDI définit également un ensemble d'interfaces de programmation (APIs) qui peuvent être employées par des applications et des services pour interagir avec des données de UDDI directement.

Structures de données de UDDI

UDDI utilise des structures de données spécifiques qui contiennent les informations nécessaires pour découvrir des services. La figure 2.6 montre ces structures et leurs relations.

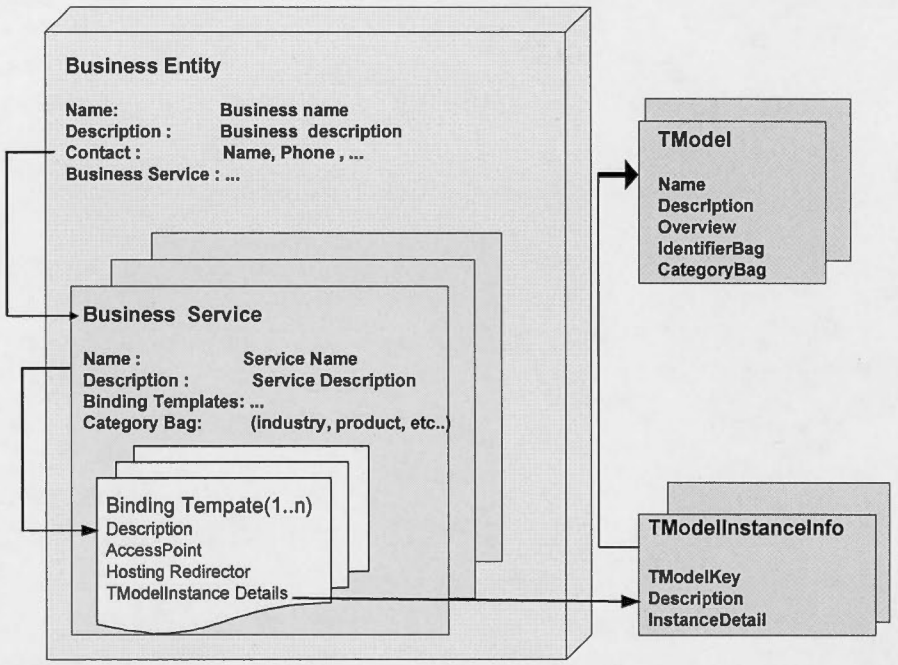


Figure 2.6. Structures de données de UDDI.

Le registre d'informations dans UDDI pour les entreprises et les services est groupé en trois groupes appelés :

- Les *pages blanches :* permettent de trouver des services et les entreprises qui les offrent par des informations telles que le nom, l'adresse, le numéro de téléphone et d'autres informations de contact.

- Les *pages jaunes :* permettent de trouver un service par sujet basé sur une taxonomie standard du domaine du service. Elles recensent les services web de chacune des entreprises selon le standard WSDL.

- Les *pages vertes* : permettent de trouver des services par caractéristiques techniques. Elles fournissent des informations techniques précises sur les services fournis et la façon d'y accéder..

Chacun de ces groupes utilise des structures de données spécifiques. Les pages blanches utilisent une structure appelée *businessEntity*. Les pages jaunes utilisent une structure appelée *businessService*. Les pages vertes utilisent les structures *bindingTemplate* et *tModels*.

La structure *businessEntity* est composée de plusieurs structures *businessService*. Celle dernière représente un service. Un tel service peut être de plusieurs types, pouvant aller d'un simple service manuel (ex. fax, email, ...) à un service web automatisé plus élaboré et basé sur XML. Une structure *businessEntity* contient une ou plusieurs structures *bindingTemplate*.

Les services sont représentés par la structure *businessService*. Les détails de l'accès à ces services sont définis par des instances de la structure *bindingTemplate*. Cette dernière spécifie les points d'accès au service (appelés *accessPoint*) et des instances de la structure *tModels* décrivant le service.

La structure *bindingTemplate* contient des pointeurs vers des descriptions techniques des services et de leurs points d'accès. Cette structure ne contient pas les détails de la spécification du service mais fait référence à la structure *tModel* qui contient les informations sur l'emplacement de ces spécifications.

Les données contenues dans un registre UDDI peuvent être conceptuellement divisées en quatre catégories, chacune représentant une entité dans UDDI. A chaque entité est assigné un identificateur unique appelé *UUID* (*Universal Unique IDentifier*).

Fonctionnement du registre UDDI

Un registre UDDI contient des descriptions structurelles accessibles aux entreprises et aux services qu'elles offrent. Il contient également des références aux caractéristiques d'industrie et ses services, aux définitions de taxonomie (utilisées pour la catégorisation des entreprises et des services) et aux systèmes d'identification par clés

uniques. UDDI fournit un modèle et un schéma de programmation qui définit les règles pour communiquer avec le registre.

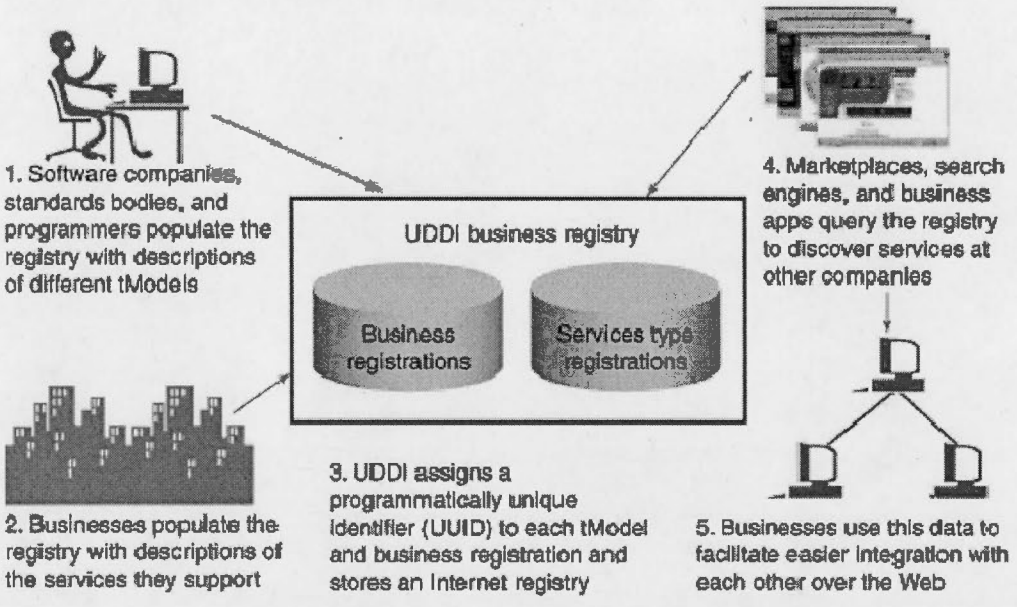


Figure 2.7. Utilisation du registre UDDI. *

La figure 2.7 illustre comment un registre UDDI est utilisé et comment les différents opérateurs découvrent et emploient ses informations. Plusieurs étapes peuvent être nécessaires pour rendre des données utiles dans UDDI:

1. L'enregistrement commence quand des compagnies et des organismes de normalisation définissent des caractéristiques concernant une industrie ou des affaires, qu'elles enregistrent dans UDDI. Celles-ci sont stockées sous forme de modèles techniques comme *TModels*.

2. Les entreprises enregistrent des informations sur elles-mêmes ainsi que les services qu'elles offrent.

---

* Source : http://www-128.ibm.com/developerworks/library/ws-featuddi/

3. Un registre UDDI maintient toutes ces entités en assignant à chacune une clé UUID.

4. D'autres clients, tels que des compagnies de commerce électronique, des moteurs de recherche ou des applications d'affaires électroniques, utilisent le registre UDDI pour découvrir des services qui les intéressent.

5. Alternativement, d'autres entreprises peuvent appeler ces services pour faciliter leur intégration sur le web.

Les APIs de UDDI

Une application qui utilise UDDI peut appeler les éléments d'une API pour effectuer des opérations de deux types:

- Découverte (*inquiry*), qui permet de chercher les informations contenues dans le registre.

- Publication (*publishing*), qui permet de stocker des informations auprès du registre.

Quelques-unes de ces opérations de recherche sont les suivantes:

- Les fonctions *find_business*, *find_service* et find_binding permettent, respectivement, de localiser les informations relatives à une entreprise, un service ou un binding.

- La fonction *find_tModel* permet de localiser les structures d'information *tModel* relatives à un service.

Ces fonctions font partie d'une API de programmation en Java très utilisée appelée JAXR (Java API for XML Registries) [WWW 12]. Celle-ci permet de formuler des requêtes UDDI et de les traduire en messages transportés par SAOP.

De manière générale, on doit d'abord accéder aux informations concernant l'entreprise avec l'opération *find_business*. On peut alors traiter le résultat ainsi retourné. Ensuite, on accède grâce à l'opération *find_service* à la liste des services offerts par cette entreprise en utilisant la clé qui l'identifie (*buisnessKey*). Les messages ainsi échangés sont transportés par le protocole SOAP.

Par exemple, l'appel de la fonction *find-service* (utilisée pour un service d'une entreprise dont on connaît la clé) s'effectuerait comme suit :

```
<?xml version='1.0' encoding='utf-8'?>
<s:Envelope
 xmlns:sd='http://schemas.xmlsoap.org/soap/envelope/'>
 <sd:Body>
  <find_service generic='1.0'  xmlns='urn:uddi-org:api' businessKey="CLE">
        <name> Services Web de UDDI </name>
  </find_service>
 </sd:Body>
</sd:Envelope>
```

Le résultat de cet appel est constitué d'une liste d'informations relatives aux services de l'entreprise contenues dans la structure *buinessInfo*. Ce résultat est retourné dans un message de réponse SOAP.

```
<?xml version='1.0' encoding='utf-8'?>
<sd:Envelope
 xmlns:sd='http://schemas.xmlsoap.org/soap/envelope/'>
 <sd:Body>
  <serviceList generic="1.0"
        operator="Microsoft Corporation"
        truncated="false" xmlns="urn:uddi-org:api">
         <serviceInfos>
                <serviceInfo businessKey="CLE1"  serviceKey="CLE2">
                <name> Service Web UDDI </name>
                </serviceInfo>
        </serviceInfos>
  </serviceList>
 </sd:Body>
</sd:Envelope>
```

Rôle de UDDI

UDDI permet aux organismes de normaliser la manière dont les entreprises organisent, découvrent, réutilisent et contrôlent des services Web. UDDI fournit un mécanisme normalisé pour découvrir et partager des interfaces de services Web entre les équipes de développement au sein de ces entreprises. Pour les professionnels et les gestionnaires des interfaces, UDDI fournit un moyen centralisé permettant le contrôle et le déploiement des services Web (Figure 2.8).

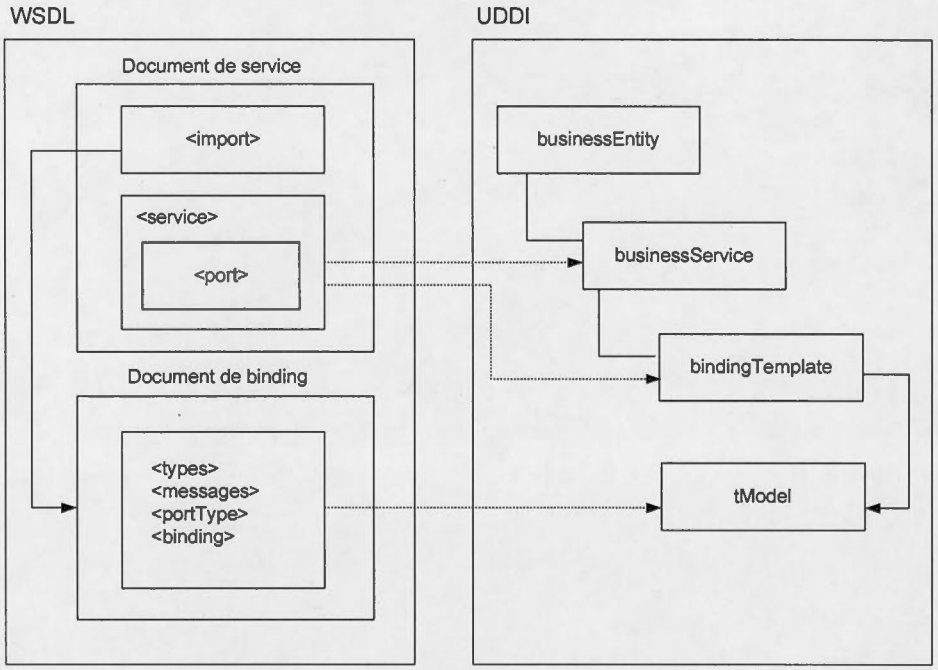


Figure 2.8. Lien entre WSDL et UDDI.

UDDI offre également des moyens puissants pour organiser et partager des services Web ainsi qu'un mécanisme pour la réutilisation des logiciels et des services. Il fournit aux administrateurs les moyens de faire les liens (*binding*) dynamiques entre une application et des services.

Chapitre III

LE GRID COMPUTING

3.1 Introduction

L'ubiquité de l'Internet ainsi que la disponibilité des technologies puissantes (ordinateurs, réseaux à haut débit, etc.) modifie rapidement le paysage des technologies de l'information et des communications. Ces opportunités technologiques ont donné la possibilité d'utiliser des ordinateurs distribués à grande échelle pour résoudre des problèmes de grande envergure.

En 1985, UNEP (*United Nations Environment Program*) a construit le GRID (*Global Resource Information Database*) comme un système global pour la surveillance climatique et l'étude des phénomènes reliés à l'environnement.

La technologie de *Grid computing* a ainsi été développée pour résoudre deux types de problèmes :

- Le grand nombre de ressources non utilisées existant sur Internet, incluant des unités de traitement (CPU), de l'espace disque, des logiciels, des données et des réseaux.

- La difficulté de l'intégration de différents systèmes déployés à l'échelle d'une grande compagnie. En effet, une technologie et une plateforme standard sont nécessaires pour soutenir une telle intégration.

Le Grid computing considère toutes les ressources disponibles dans le réseau comme un "super ordinateur" [Foster, 2004]. L'utilisateur peut, d'une manière transparente,

employer et contrôler ces ressources. Le Grid computing fournit également une série de normes pour intégrer ces systèmes hétérogènes.

Le Grid devient une technologie qui permet un accès simple et extensible à des ressources distribuées pour les partager et accomplir des tâches de collaboration. Le Grid computing fournit également une infrastructure principale pour la résolution des problèmes distribués dans des organisations virtuelles.

Le Grid Computing est également défini comme une infrastructure matérielle et logicielle qui fournit un accès sûr, uniforme et peu coûteux aux ressources informatiques avancées [Dikaiakos 2004]. C'est un outil de communication et une infrastructure informatique pour le partage transparent des ressources de calcul réparti et basé sur des protocoles et des interfaces d'usage universel et des normes ouvertes.

Le Grid computing utilise une collection de machines (parfois désignées sous le nom de noeuds, ressources, membres, etc.) pour permettre le partage du calcul, des applications, les données, les outils de stockage et les ressources de réseau à travers des organisations géographiquement dispersées [Jean-Marie 2002]. Ce partage de ressources se fait entre *les Organisations virtuelles (Virtual Organisation ou VO)*. Une VO est un groupe logique des membres qui peut être géographiquement distribués. L'idée de VO est employée depuis plusieurs années dans le domaine des technologies pour désigner des groupes de travail temporels afin de réduire des coûts. Normalement tous les membres ont des intérêts communs, par exemple un institut, un groupe de scientifique ou une entreprise commerciale. Dans ces VOs, les ressources sont partagées et l'utilisation est définie par des politiques du groupe (Figure 3.1).

3.2 Environnement de Grid Computing

L'environnement Grid est constitué de ressources hétérogènes, de systèmes de gestion locale (Système d'exploitation, système de communication, systèmes de file d'attente, etc.), d'applications spécifiques avec des exigences variées (en terme d'unité centrale, d'entrées-sorties, de mémoire, de ressources réseau, etc.). Le tout

forme une organisation virtuelle dans laquelle les échanges sont contrôlés. Ce contrôle touche autant les accès aux machines que les données renvoyées par celles-ci. Son rôle est d'automatiser les échanges entre les machines et de gérer les ressources [WWW 5].

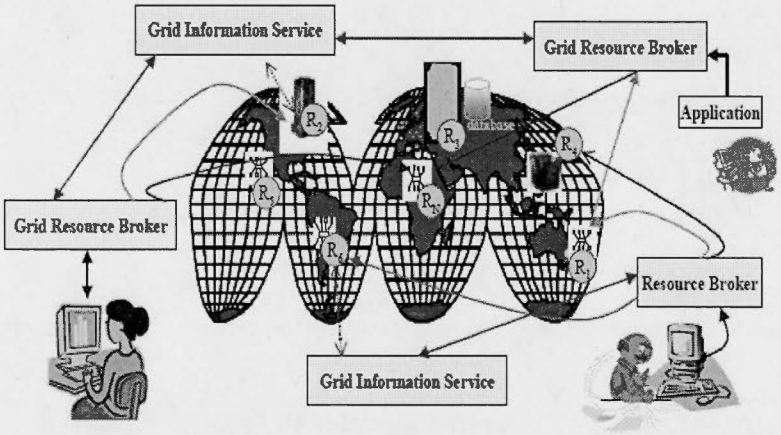


Figure 3.1: Modèle du Grid.*

Pour les opérations du Grid, on utilise un middleware spécifique pour découvrir des ressources et leurs propriétés, permettre l'accès des utilisateurs aux ressources à travers les serveurs d'information, négocier les ressources, envoyer des tâches aux ressources, orchestrer les applications et les données pour le traitement et enfin rassembler les résultats.

Le Grid est une infrastructure constituée d'un ensemble de composants proposant différents services (Figure 3.2).

---

* Source : http://developpeur.journaldunet.com/tutoriel/out/061205-fonctionnement-systeme-grille.shtml

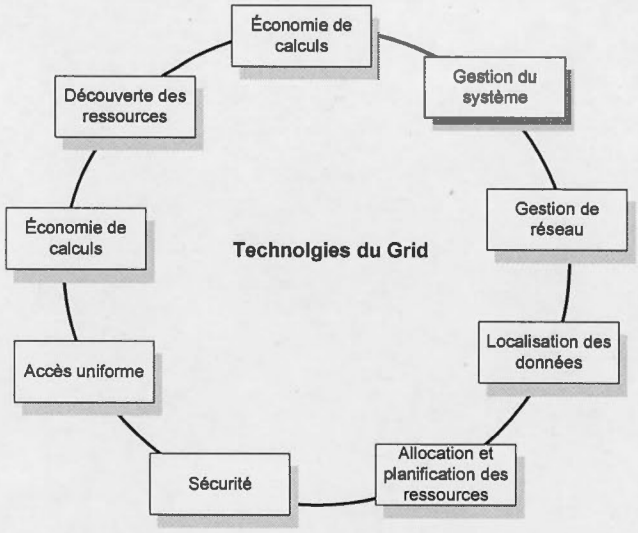


Figure 3.2. Technologies du Grid.

Ces services permettent de réaliser les tâches suivantes :

- *Sécurité* : un accès sécuritaire aux ressources et aux applications (identification, authentification, délégation, etc.) est fourni par un middleware du Grid tel que, par exemple, Globus [WWW 6].

- *Accès uniforme*: les technologies de Grid fournissent un moyen sécuritaire qui permet aux utilisateurs d'accéder grâce à un mécanisme de contrôle d'accès aux ressources de différentes organisations, d'où elles sont allouées (ex. Login).

- *Interface d'usager* permettant d'accéder aux applications et aux ressources disponibles sur la Grid dans un espace virtuel

- *Gestion du système*: présente une manière globale d'exprimer les demandes des ressources ou des applications que l'utilisateur veut exécuter et pour afficher les résultats. Une application peut communiquer avec un gestionnaire des tâches pour découvrir les

ressources disponibles et leurs états.

- *Découverte de ressources*: implique la découverte des ressources appropriées et leurs propriétés pertinentes aux exigences des utilisateurs.
- *Planification*: permet de localiser les ordinateurs, exécuter les applications et assigner les travaux. Cette tâche implique souvent de donner la priorité à des files d'attente, contrôler la charge, trouver les ressources réservées et afficher le progrès.
- *Gestion des données* : prend soin de déplacer les données au bon endroit à travers diverses machines.

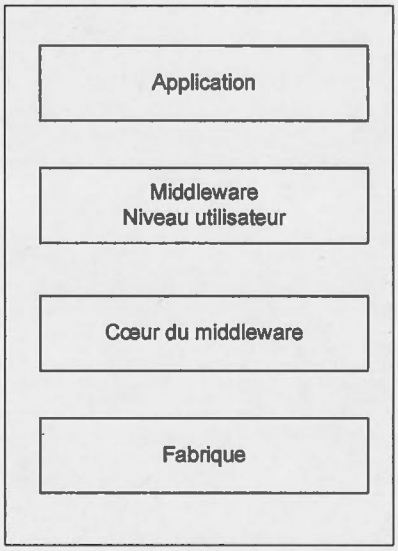


Figure 3.3: L'architecture multicouches du Grid.

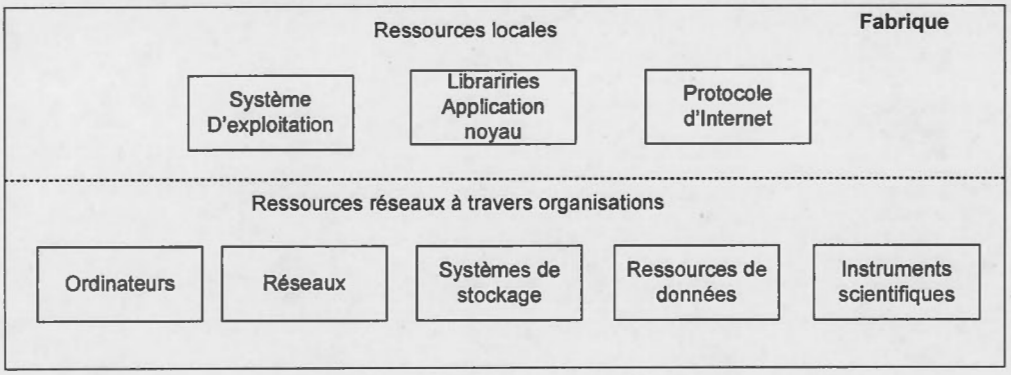
3.3 Architecture logicielle du Grid

L'architecture Grid définit le but et les fonctions de ses composants tout en indiquant comment ces composants interagissent entre eux [Kesselmam 1998]. L'essentiel de cette architecture est l'interopérabilité entre les fournisseurs des ressources et les

utilisateurs afin d'établir des relations pour partager les ressources. Cette interopérabilité offre des protocoles communs et définit des mécanismes, des interfaces et des schémas nécessaires à chaque couche du modèle architectural. Les composantes clés de Grid sont illustrées dans la figure 3.4.

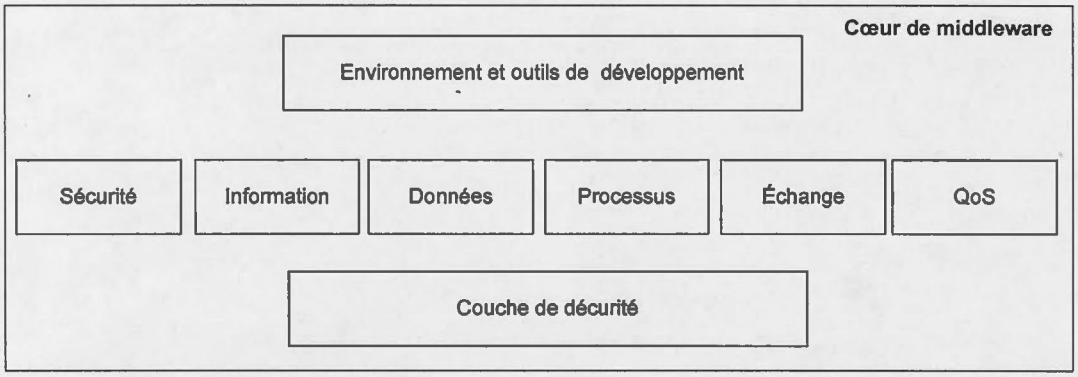
La couche Fabrique

Cette couche présente une interface de contrôle local et se compose de toutes les ressources globalement distribuées qui sont accessibles partout sur Internet. Ces ressources pourraient être physiques (telles que des ordinateurs, des dispositifs de stockage, des bases de données, des réseaux et des instruments scientifiques) ou logiques (telles que des systèmes d'exploitation et des systèmes de gestion de ressource - *Load Sharing Facility* ou *LSF*) qui vont être partagés via les protocoles du Grid.



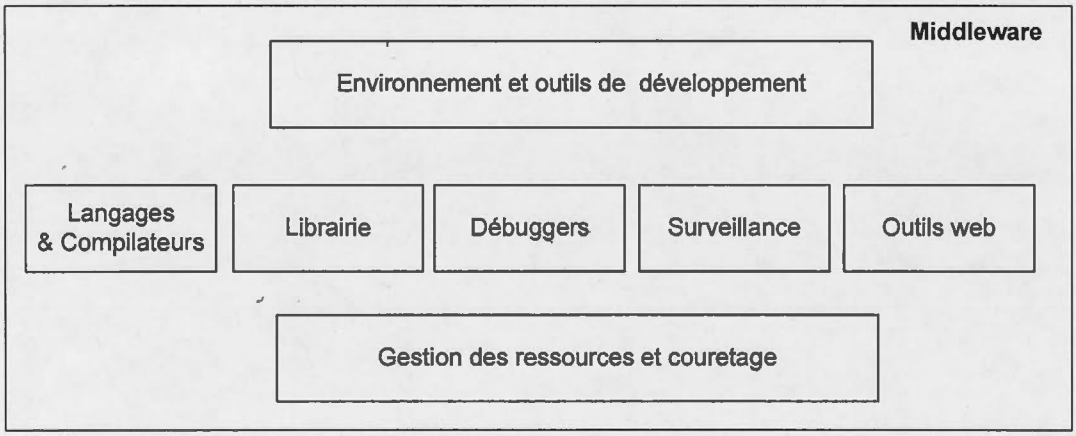
La couche Cœur du middleware

Cette couche offre une communication sécuritaire et des services essentiels tels que le contrôle de processus industriels à distance, la co-allocation des ressources, l'accès aux unités de stockage, l'enregistrement et la découverte des informations, la sécurité et des mécanismes de gestion de la qualité du service (QoS).



Le niveau utilisateur du middleware

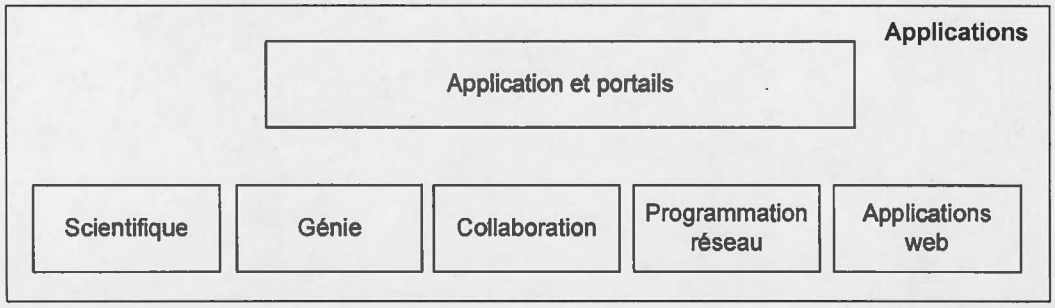
Ceci contient l'environnement de développement des applications, les outils de programmation et les ressources intermédiaires pour gérer les ressources et programmer les tâches d'application afin de permettre une exécution sur des ressources globales.



Les applications et les portails

Les applications du Grid sont développées en utilisant des langages et des utilitaires tels que *MPI* (*Message Passing Interface*). Un exemple d'application serait le contrôle à distance de l'accès aux ensembles de données et l'interaction avec les instruments scientifiques. Les portails Grid offrent l'application des services web, d'où les utilisateurs peuvent soumettre et rassembler des résultats pour leurs travaux à

distance via le web. Cette couche comporte les applications d'utilisateurs qui fonctionnent dans un environnement de VO.



3.4 Services du Grid

Les technologies de Grid visent à répondre aux problèmes suivants:

- Exploitation des ressources peu utilisées.
- Utilisation de capacités parallèles de CPUs.
- Développement d'applications.
- Collaboration entre ressources et organisations virtuelles.
- Accès à des ressources additionnelles.
- Équilibrage de charges.
- Fiabilité.
- Gestion.

Nous allons décrire chacune de ces fonctions.

Exploitation des ressources peu employées

Dans la plupart des organismes, il y a de grandes quantités de ressources

informatiques peu utilisées. Le Grid fournit un cadre pour exploiter ces ressources et offrir ainsi la possibilité d'augmenter sensiblement l'efficacité de leur utilisation. Il y a deux scénario préalables pour exploiter ces ressources :

- L'application doit être exécutable à distance.
- La machine à distance doit fournir le matériel, le logiciel et des ressources spéciales qui sont imposées par l'application.

Capacité parallèle de CPUs

La capacité pour le traitement parallèle offre des avantages attrayants dans le Grid. Cependant, des limitations existent souvent pour améliorer les performances des CPUs. La première limitation dépend des algorithmes utilisés pour diviser l'application en plusieurs segments parallèles pouvant s'exécuter sur des CPUs différents. La deuxième limitation apparaît si les parties d'un même programme ne sont pas indépendantes. D'autres sources qui peuvent limiter l'extensibilité dans une application parallèle de Grid sont la latence des communications sur le réseau, les protocoles de synchronisation ainsi que la largeur de bande d'entrée-sortie aux dispositifs de stockage.

Développement d'applications

L'un des avantages du Grid est le traitement parallèle des tâches. Cependant, les applications ne peuvent pas toutes être transformées en programmes parallèles dans le Grid. Il y a peu d'outils pratiques pour transformer des applications arbitraires et exploiter les possibilités du parallélisme offertes par le Grid. La transformation automatique d'un programme exige des talents spécifiques de programmation

Collaboration entre ressources et organisations virtuelles

Une autre contribution importante de Grid est de simplifier la collaboration entre utilisateurs tout en offrant des normes importantes qui permettent aux systèmes

hétérogènes de fonctionner ensemble et former un grand système de calcul virtuel. Les utilisateurs du Grid peuvent être organisés en organisations virtuelles qui peuvent partager leurs ressources comme un grand Grid.

Accès aux ressources additionnelles

En plus des CPUs et des sockage, le Grid permet d'accéder à un plus grand nombre de ressources et d'équipements spéciaux, logiciels et aux autres services.

Équilibrage des ressources

Le Grid peut offrir un effet d'équilibrage de ressources en programmant les travaux du Grid sur des machines moins utilisées.

Fiabilité

Les systèmes conventionnels de calcul utilisent des matériels chers pour augmenter la fiabilité. Ils contiennent la logique pour réaliser le rétablissement en cas de panne de matériel. Les systèmes de Grid sont relativement peu coûteux et sont géographiquement dispersés. Ainsi, s'il y a une erreur ou une panne dans un endroit, les autres parties du Grid ne seront pas affectées. Le Grid peut automatiquement re-soumettre les travaux aux autres machines dans le Grid. Dans des situations critiques et en temps réel, des copies multiples des travaux importants peuvent être lancées sur différentes machines dans le Grid.

Gestion

Le middleware du Grid offre la possibilité de gestion des priorités parmi différents projets. Dans le passé, chaque projet devait être responsable de son propre matériel. Le Grid devient une manière plus facile de commander et contrôler de telles situations. Les administrateurs peuvent changer les politiques qui affectent les différentes organisations qui partagent les ressources.

3.5 Les catégories de Grid

Il existe trois catégories de Grid :

a) Grid d'informations

Ce Grid ressemble au World Wide Web d'aujourd'hui qui fournit des informations sur un grand nombre de sujets. Ces informations seront recherchées pour être échangées par différents types d'infrastructures et d'équipements : réseaux, ordinateurs, portables, etc.

b) Grid de ressources

Le Grid des resources fournit des mécanismes pour coordonner l'usage de ressources telles que des ordinateurs, des archives de données, des applications et des dispositifs spéciaux. Le système Globus illustre ce type de Grid en donnant accès aux ressources sans exiger que l'utilisateur connaisse le nom, l'endroit et les autres attributs relatives aux ressources utilisées. Contrairement aux données fournies par le Grid d'informations, les équipements du Grid de ressources ne sont accessibles qu'aux utilisateurs autorisés.

c) Grid de services

Ce type de Grid fournit des services et des applications indépendamment de leur endroit d'exécution, de la plate-forme matérielle, etc. Ces services se basent sur les ressources concrètes disponibles dans le Grid de ressources. Alors que le Grid de ressources donne accès aux ressources concrètes, le Grid de services fournit des services abstraits et indépendants.

Chapitre IV

L'OUTIL GLOBUS TOOLKIT

4.1 Introduction

Globus ToolKit ([Jacob 2003], [Sotomayor 2005]) a été développé par *Globus alliance* [www 6] pour supporter le développement d'applications orientées services ainsi que les infrastructures de calcul distribué. Il fournit une infrastructure logicielle (middleware) qui permet à des applications de voir les ressources informatiques hétérogènes distribuées comme des machines virtuelles simples.

Globus Toolkit est un logiciel libre (*Open source*) utilisé pour construire des systèmes de Grid, des applications et pour éliminer les obstacles qui empêchent une collaboration simple et facile entre les compagnies, des ordinateurs et des applications. Il inclut des logiciels, des services et des bibliothèques pour distribuer les applications, la gestion de la sécurité, les ressources, les données ainsi que la découverte des ressources.

Ce toolkit est constitué d'un ensemble de composants qui mettent en application des services de base, tels que la sécurité, la localisation des ressources, la gestion des ressources et des données, la réservation des ressources ainsi que la communication. Il fournit également un ensemble de services à partir desquels des développeurs d'outils spécifiques ou d'applications peuvent construire leurs propres outils.

Globus présentent la forme d'une hiérarchie dans laquelle des services de haut niveau peuvent être développés en utilisant des services de noyau de bas niveau. Il met l'accent sur l'intégration hiérarchique des composants de Grid et de leurs services. Les ressources et les états des informations manipulées sont fournis via le *Monitoring*

Discovery System (MDS), qui comporte deux composants :

- *Grid Resource Information Service (GRIS)*, qui fournit des informations sur une ressource spécifique et des *pages blanches*.
- *Grid Index Information Service (GIIS)* qui fournit un ensemble d'informations récupérées par plusieurs serveurs GRIS. Il supporte des requêtes sur de l'information disséminée. Il fournit un mécanisme de *pages jaunes*.

Depuis son apparition, *Globus ToolKit* a beaucoup évolué, passant des versions GT1 à GT4. La table 1 donne les caractéristiques de la version GT4. Dans ce travail, nous avons adopté cette version car elle permet l'utilisation étendue des mécanismes de services Web pour définir ses interfaces et structurer ses composants. Elle fournit un ensemble de services d'infrastructure de Grid et implémente une interface pour la gestion de calculs, le stockage, et d'autres ressources.

Fonction.	Outils.
Format de données.	XML data document
Protocole pour interroger les services.	WSRF
GUIs disponible.	WebMDS.
Opérations de subscription/notification.	WS-Notification.
APIs pour subscription/notification.	WS Core APIs.
Index d'interroger d'information agrégée.	WS- MDS Index Server.
Interrogation de l'information non agrégée.	Service Web Individual

Mécanisme d'enregistrement d'index.	• WS MDS Index servers. • Maintien l'agrégation des groupes de services qui incluent l'enregistrement d'information. L'enregistrement est accompli en ajoutant une entrée à un agrégé d'un service de groupe via la commande mds-servicegroup-add.
Commande clients pour les interrogations.	Wsrf-get-property, Ws-query.

4.2 Globus ToolKit 4

Il est important de noter que GT4 est plus qu'un ensemble de services:

- Il fournit un mécanisme standard pour associer les ressources des propriétés écrites en XML avec des entités de réseau et permet d'accéder à ces propriétés via des requêtes de type *Pull* ou à la suscription via des opérations de type *Push*. Ce mécanisme est important pour l'efficacité du fonctionnement de WSRF et de la spécification *WS-Notification*, et peut être incorporé dans n'importe quel service développé par l'utilisateur. Les services peuvent également être configurés pour s'inscrire dans un container.

- Le GT4 fournit aussi deux services qui rassemblent les récentes informations sur l'état de ressources à partir de registres d'informations. Ces services collectent des données sur n'importe quelle ressources et offrent l'interface WSRF et *WS-Notification*.

La figure 4.1 montre les composantes principales de GT4.

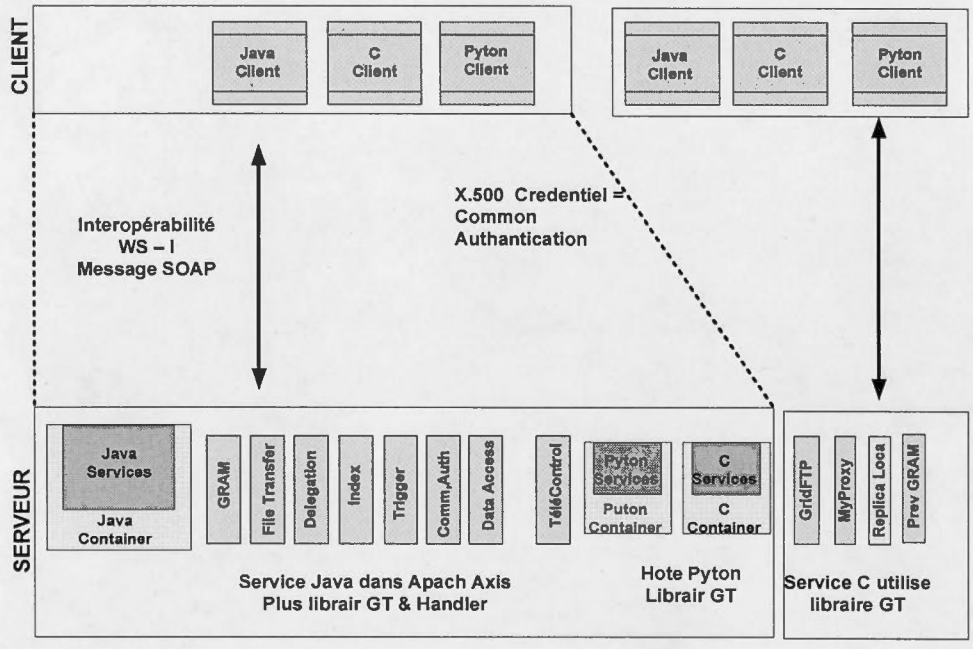


Figure 4.1. Architecture de GT4.

L'architecture de GT4 inclut trois ensembles de composants [19,20]:

- Un ensemble d'implémentations de services de gestion d'exécution (*GRAM*), l'accès et le mouvement de données (*GridFTP , RFT, OGSA-DAI*), la gestion des replicas (*RLS DRS*), le monitoring and discovery system MDS (*Index, Trigger, WebMDS*), la gestion des délégations ou *Credential management* (*MyProxy*, *Delegation* et *SimpleCA*) ainsi que la gestion d'instruments. La plupart des services sont implantés avec des technologies Web en Java.

- Trois containers peuvent être utilisés pour héberger les services développés par des utilisateurs écrits en Java, Python ou C. Ces containers fournissent, entre autres, l'implantation des services de sécurité.

- Des libraires qui acceptent des programmes client en Java, C, ou Python et peuvent invoquer des opérations sur GT4 et les services développés par l'utilisateur.

Chapitre V

LE GRID ET LES SERVICES WEB

5.1 Introduction

Il y a quelque temps, seul Globus pour le Grid existait. Mais depuis que les services Web ont vu le jour, l'équipe de Globus ToolKit a décidé de créer une nouvelle version qui fusionne les deux technologies créant le *Grid Service* (*GS*). Le *Forum Global Grid* (*GGF*) [www 13] a été fondé pour normaliser une infrastructure de services disponibles sur le Grid.

Les services Web apportent aux techniques de calculs distribués un fond commun pour la structure et l'échange de données ainsi qu'un paradigme par composants. Ceci devrait faciliter la diffusion des applications dans les entreprises. L'intégration de e-science et de e-business a besoin d'une passerelle qui puisse dynamiquement exécuter des services sur des plateformes hétérogènes et réparties. La flexibilité d'une approche ouverte, standard et disponible permet de rassembler des ressources hétérogènes dans un tel environnement.

Avec de nouvelles spécifications de services Web [Favali 2004], les solutions de type entreprise vont être prolongées pour que les clients puissent facilement déployer leurs applications d'affaires. Ces nouvelles spécifications sont importantes pour permettre aux clients d'utiliser une infrastructure basée sur les services web qui supporte le Grid et les solutions fournies par les entreprises.

5.2 Open Grid Service Architecture

OGSA (Open Grid Service Architecture) propose une solution orientée service pour le Grid [Li 2005]. OGSA présente la clé d'intégration de la technologie de Grid avec le

mécanisme des services web.

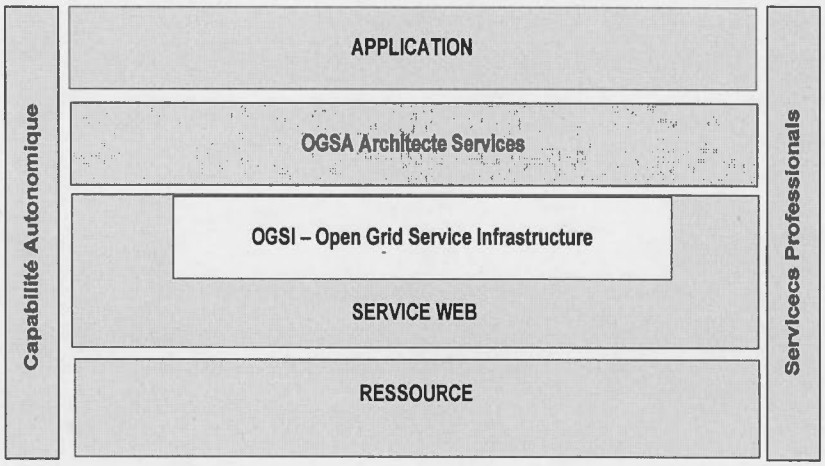
La couche de base de OGSA est fournie par le standard des services Web, mais l'infrastructure ne répond pas à toutes les exigences communes aux services de Grid comme : la gestion de cycle de vie, l'accès extérieur de l'état des services et l'interaction avec état (*statefull interaction*). La gestion de cycle de vie réfère à la possibilité qu'un client crée ou détruise de nouveaux services. L'interaction avec état satisfait le besoin de l'abstraction de session et des messages relayés entre les services.

Puisque OGSA est basé sur les services Web, il hérite de ses avantages et de ses inconvénients [www 13]. OGSA offre de nouvelles possibilités pour la découverte de services et leur gestion de vie. Il offre également des Services Web *avec état (stateful)* mais OGSA ne résout cependant pas totalement le problème d'interopérabilité. Par ailleurs il y a une opération appelée *FindServiceData* qui peut être exécutée pour un service de Grid. Ceci permet à un client de découvrir plus d'informations sur l'état d'un service, l'environnement d'exécution et les détails sémantiques additionnels, essentiellement, pour apprendre plus au sujet du service qui est important pour l'interopérabilité.

OGSA définit les mécanismes pour créer, contrôler et échanger l'information parmi les entités Grid Services.

5.3 Architecture de OGSA

Cette architecture [www 7] permet l'intégration et l'administration des composants hétérogènes, géographiquement distribués afin de former des systèmes de calcul virtuels pour livrer une QoS désirée (Figure 5.1).



La figure 5.1, Architecture de OGSA.*

a) La couche Ressource

Cette couche représente les ressources physiques et logiques. Le concept de ressource est central pour OGSA et le Grid en général. Pour la plupart, ces ressources sont rendues accessibles à la structure de OGSA en ajoutant des fonctions à ces logiciels pour abstraire les ressources matérielles ainsi que du système en tant que services.

Au-dessus des ressources physiques, on trouve les ressources logiques qui fournissent des fonctions additionnelles (telles que des systèmes de fichiers, des bases de données, un système de transmission de messages, des annuaires, etc.) et rassemblent les ressources dans la couche physique. Une passerelle, telle que le système de fichiers, la gestion de base de données, etc. peut fournir ces services abstraits sur le Grid physique. Ce logiciel exploite généralement la couche inférieure de la ressource physique et fournit également la fonction qui peut être abstraite comme service dans OGSA.

---

* Source : http://www-128.ibm.com/developerworks/webservices/library/gr-visual/

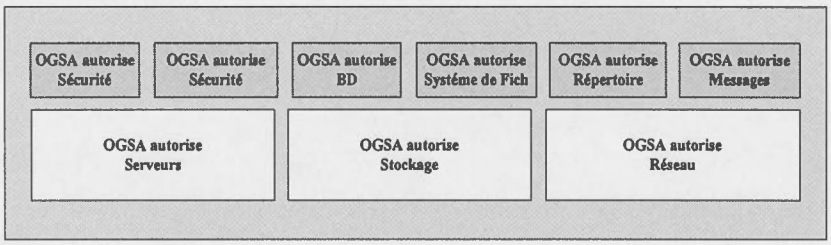


Figure 5.2. La couche Ressource.

b) La couche Web Services et OGSI

La couche Service Web avec l'extension de OGSI fournit une base d'infrastructure pour les autres couches de services de Grid. Les spécifications de *Open Grid Service Infrastructure (OGSI)* définissent le Service de Grid et sont construites sur la technologie standard de services Web. OGSI exploite les mécanismes des services Web comme XML et WSDL pour indiquer les interfaces, les comportements et l'interaction standardisée.

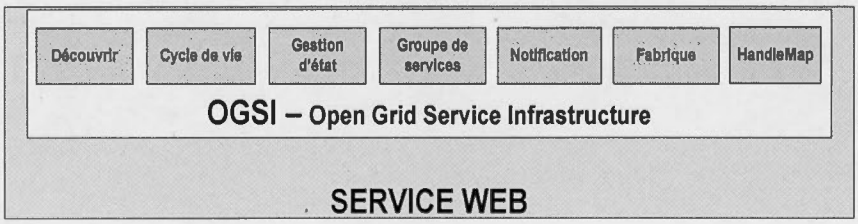


Figure 5.3. La couche service Web avec OGSI.

c) La couche OGSA

En tant qu'approche SOA, OGSA est une prolongation de l'infrastructure existante de services Web par des normes comme XML, WSDL, SOAP, etc. OGSA profite des avantages des middlewares des serveurs d'applications qui fournissent l'appui de programmation pour implémenter des services Web. OGSA repose sur ce support

des services Web. Le GGF travaille actuellement pour définir des architectures de services de Grid telle que l'exécution du programme, des services de données, et des services de noyau. Il est prévu que OGSA devienne une architecture SOA plus élaborée.

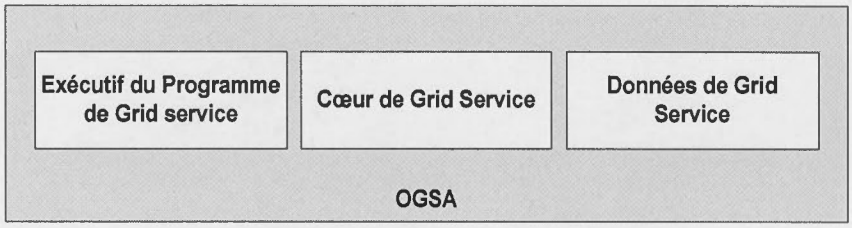


Figure 5.4. La couche OGSA et ses éléments.

d) La couche Grid application

Les applications de Grid à leur tour exploitent les services que OGSA définit pour contrôler et exploiter les ressources distribuées du Grid. Aussi, le développement des services dans le temps sera apparu grâce à la richesse de l'ensemble de l'architecture de Grid.

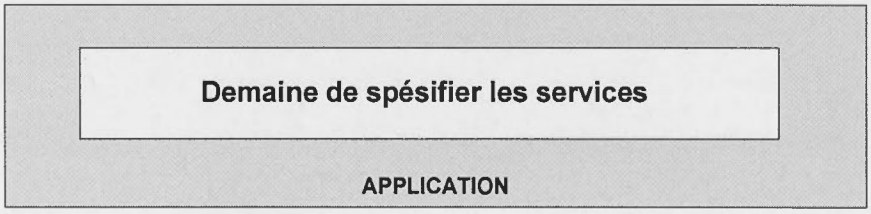


Figure 5.5. La couche Application.

5.4 Objectifs de OGSA dans l'environnement du Grid

OGSA joue un rôle essentiel dans le Grid pour interagir avec les services Web tels que :

- Contrôler les ressources à travers des plateformes hétérogènes distribuées.
- Permettre l'interaction entre ressources. La topologie du Grid est souvent complexe. L'interaction de ses ressources est habituellement dynamique en ce que le Grid fournit des services importants pour l'autorisation, le contrôle d'accès et la délégation.
- Fournir une base commune pour les solutions autonomes de gestion. Un Grid peut contenir beaucoup de ressources avec de nombreuses combinaisons de configurations, interactions, etc. Une forme d'autorégulation et de gestion autonome de ces ressources est nécessaire.
- Définir des interfaces ouvertes. OGSA est une norme ouverte contrôlée par l'organisme de normalisation GGF. Pour l'interopérabilité des ressources diverses, les services des Grids sont établis sur la base d'interfaces et de protocoles standards.

5.5 Web Service Resource Framework

Web Service Resource Framework (*WSRF*) [Taylor, 2005] présente le coeur de GT4, qui est l'implantation Java de WSRF.

L'objectif de WSRF est de définir un modèle pour accéder aux ressources persistantes en utilisant des services Web, de sorte que la définition, l'implantation d'un service, l'intégration et la gestion des multiples services soient facilitées [www 8].

WSRF définit un système pour créer des ressources avec état (*Statefull*) entre les

services Web en termes de modèle implicite de ressources et des fonctions qui supportent l'interaction avec les services Web (bien que les interactions avec les services compatibles avec WSRF soient plus riches et plus faciles à gérer). Le cœur de WSRF est le *WS-Resource*, qui est une composition de services Web et de ressources avec état décrits par XML et associés avec un *portType* de services Web et l'adresse par *WS-Addressing* et *EndpointReference*. WSRF est basé sur la spécification OGSI et est davantage compatible avec les concepts de services Web (c-à-d. XML, SOAP et WSDL).

WSRF définit une famille d'outils pour accéder aux ressources avec état en utilisant des services Web. Son avantage est que les services Web, typiquement, ne maintiennent pas de l'information d'état pendant leurs interactions. Leurs interfaces doivent tenir compte de la manipulation de l'état, c.-à-d., les valeurs de données qui persistent et évoluent en fonction des interactions avec les services Web. Par exemple, un système de réservation de vols en ligne doit se maintenir à jour au sujet de l'état des vols, des réservations faites par des clients spécifiques, et par le système lui-même: site, charge et exécutions en cours.

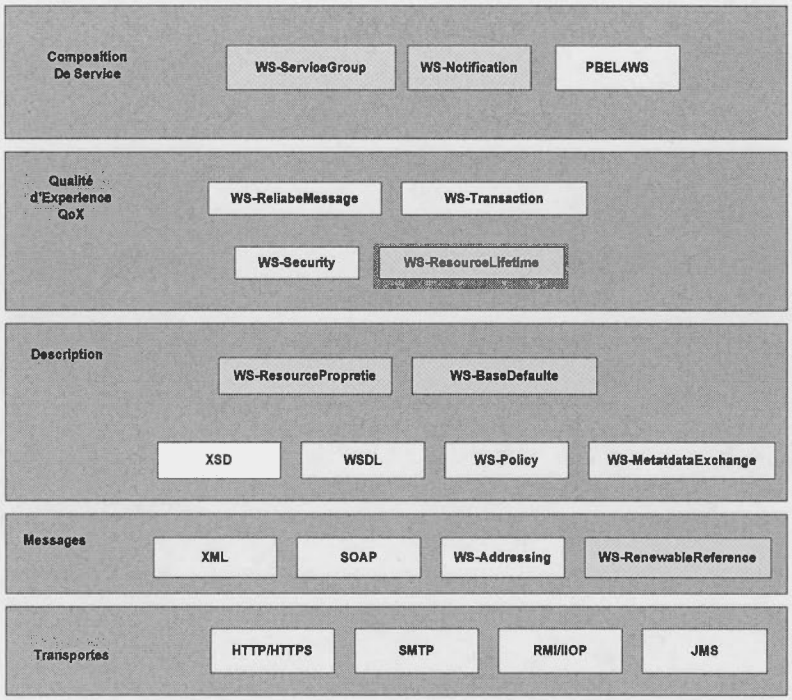


Figure 5.6. Élément de WSFR.

Le WSRF comprend les spécifications suivantes (Figure 5.6):

- *WS-Resource Properties :* Elle décrit la manière dont les propriétés des ressources sont définies et accédées. WSRP est décrit dans l'interface de WDSL de Service Web. Le but des *WS-ResourceProperties* est de normaliser des terminologies selon les concepts des opérations WSDL et XML requis pour exprimer les propriétés des ressources, et aussi de déterminer les possibilités d'interroger et de mettre à jour les propriétés d'un service Web ayant une association avec une ressource avec état. Ces ressources avec état sont définies par un type de document et une association exprimée par l'élément de WSDL *portType*.

- *WS-ServiceGroup :* Cette spécification définit des moyens de gérer des services dans un groupe (ajouter, supprimer et rechercher). *WS-ServiceGroup* décrit une interface pour relier et grouper des ressources ou des services ensemble.

- *WS-ServiceGroupRegistration :* Il s'agit d'un moyen permettant aux utilisateurs du service d'insérer explicitement de nouveaux membres dans un groupe.

- *WS-ResourceLifetime* : Cette spécification fournit des mécanismes de base pour gérer le cycle de vie des ressources. Elle normalise des moyens par lesquels une *WS-Ressource* peut être détruite et surveillée. Le but est de normaliser des terminologies comme les concepts, les échanges de messages, le WSDL et le XML requis pour surveiller la vie de ressources.

WSRF améliore plusieurs aspects des services web pour les rendre plus adéquats aux applications de Grid (Figure 5.7).

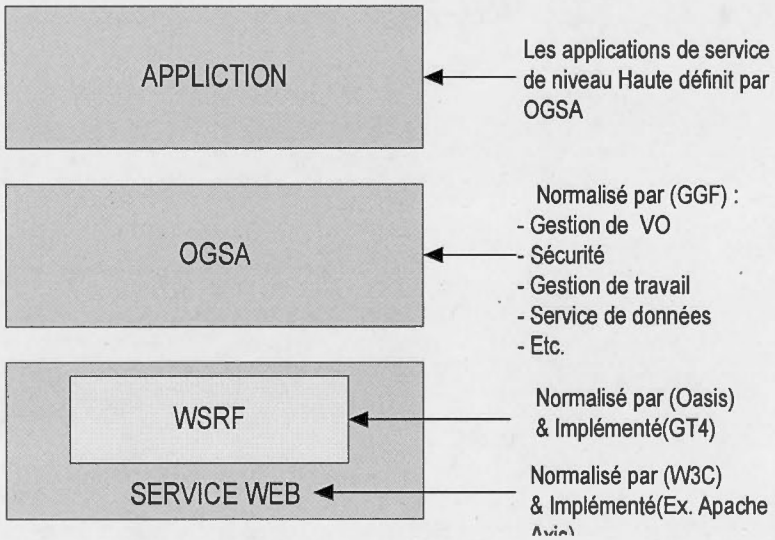


Figure 5.7. Intégration WSRF et les services web.*

* Source: http://dsd.lbl.gov/gtg/projects/pyGridWare.

Chapitre VI

L'INTÉGRATION GRID ET SERVICES WEB

Dans ce chapitre, nous proposons une solution pour résoudre le problème d'interaction entre l'environnement de GRID et les services web (en particulier le registre UDDI). Pour réaliser ce but, nous avons implanté un système de passerelle entre un environnement de déploiement de services web (comprenant des services, des descriptions WDSL et le référentiel UDDI) et l'environnement basé sur les services Grid (*Grid Service* ou *GS*).

6.1 Un prototype de mapping du Grid et des services web

Notre implantation repose sur une architecture à trois niveaux (3 tiers) tel qu'illustré dans la figure 6.1. Dans cette architecture, nous utilisons le *Toolkit GT4* avec MDS et WSRF ainsi qu'un générateur de requêtes SOAP et l'environnement Grid Services qui interagit avec UDDI.

6.2.1 Le GS

Le GS est un espace multidimensionnel de services avec un ensemble d'opérations de services uniformes [IBM 2004]. Cet espace constitue lui-même un service Web. Le GS se présente comme étant indépendant de la plateforme et des langages de programmation utilisés.

GS nous assure les fonctions suivantes :

- La création et la destruction dynamique des instances de services.
- Le support des services transitoires et des services avec état.

- La création de services à partir d'autres services, possiblement dans d'autres serveurs étant donné que GS est un objet de type *Factory*.

- La découverte et l'enregistrement des services qui permettent de publier des informations sur d'autres services.

- La capacité des services distribués de s'informer synchroniquement sur leurs changements d'états.

- La manipulation des références des services grâce au *Grid Service Handle* (*GSH*) et le *Grid Service Reference* (*GSR*). Le *Grid Service Handle* (*GSH*) est une adresse URI unique et globale alors que le *Grid Service Reference* (*GSR*) est un indicateur pour spécifier une instance de Grid Service.

- Une extension de WSDL appelée *Grid WSDL* (*GWSDL*).

6.2.2. Processus de développement

Ce modèle se compose de trois parties :

1. La première partie représente les utilisateurs d'applications du *Grid*. Ces utilisateurs ont une vue transparente du *Grid* à haut niveau qui expose l'information liée à un environnement de service. Ceci convient à la résolution des problèmes plutôt qu'au développement d'applications complexes.

2. La deuxième partie du modèle représente l'environnement de *Grid* et *Globus Toolkit4 (GT4)*. Cette partie inclut aussi l'état des ressources disponibles, les outils, le contrôle et la surveillance des informations reliées dans l'infrastructure de *Grid* afin de guider les processus de développement des applications.

3. La troisième partie représente l'implantation du Grid avec un système référentiel qui inclut le registre *UDDI* et le *Grid Service*. Cette implantation vise à établir un pont entre l'environnement du *Grid* et le registre de *UDDI*. Ceci permettra aux utilisateurs du *Grid* de chercher et stocker des services et des informations dans UDDI.

Le processus de traitement des requêtes est le suivant (Figure 6.2):

- La requête de l'utilisateur du Grid est envoyée à l'environnement de GT4.
- *Globus ToolKit4* traite la requête et l'envoie au générateur de messages SOAP.
- SOAP transfère la requête au système du répertoire, d'où la requête va passer au registre UDDI à travers GS. Il retourne ensuite la réponse au générateur du SOAP.
- *Globus ToolKit* enregistre la réponse de la requête. Il retourne alors la réponse à l'utilisateur.

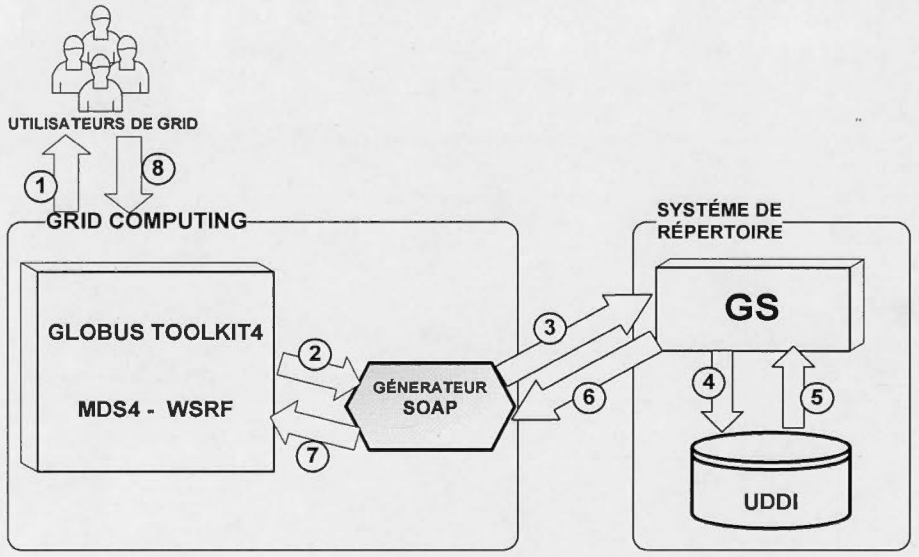


Figure 6.1. Schéma d'intégration entre le Grid et le UDDI.

Comme l'indique ce schéma, les utilisateurs font partie de l'environnement de Grid. Mais avant qu'ils ne soient capables d'accéder à cet environnement, il leur faut

s'identifier afin d'avoir l'autorisation d'effectuer des opérations. L'utilisateur se sert du GSI, qui utilise des certificats d'identification X.509 et des certificats de Proxy 509. Pour avoir un certificat de type CA (Certificate Authorty) dans l'environnement d'UNIX, on commence par télécharger le fichier *SimpleCA* (à partir de http://www.globus.org/security/simple-ca.html) et on construit le paquetage pour ce fichier dans son répertoire avec la commande :

```
$GLOBUS_LOCATION/sbin/gpt-build globus_simple_ca-latest-src_bundle.tar.gz gcc32dbg
```

Dés que le fichier est construit, on l'installe avec la commande :

```
$GLOBUS_LOCATION/sbin/gpt-postinstall
```

L'utilisateur doit ensuite configurer ce fichier avec la commande :

```
$GLOBUS_LOCATION/bin/grid-default-ca
```

Une fois le client identifié, il sera capable d'envoyer sa requête à travers le Grid pour chercher les ressources dont il a besoin.

Un client de Grid computing qui désire avoir un service commence par saisir sa requête en utilisant le navigateur WebMDS. En particulier, le service de WebMDS permet aux utilisateurs de surveiller l'information via une interface Web. WebMDS est implanté comme Servlet et utilise XSLT pour transformer les données aux utilisateurs dans un format lisible en HTML. WSMDS peut être accessible de n'importe quelle plate-forme.

La figure 6.2 montre comment un client peut avoir accès aux informations via un navigateur de Web qui utilise le protocole HTTP.

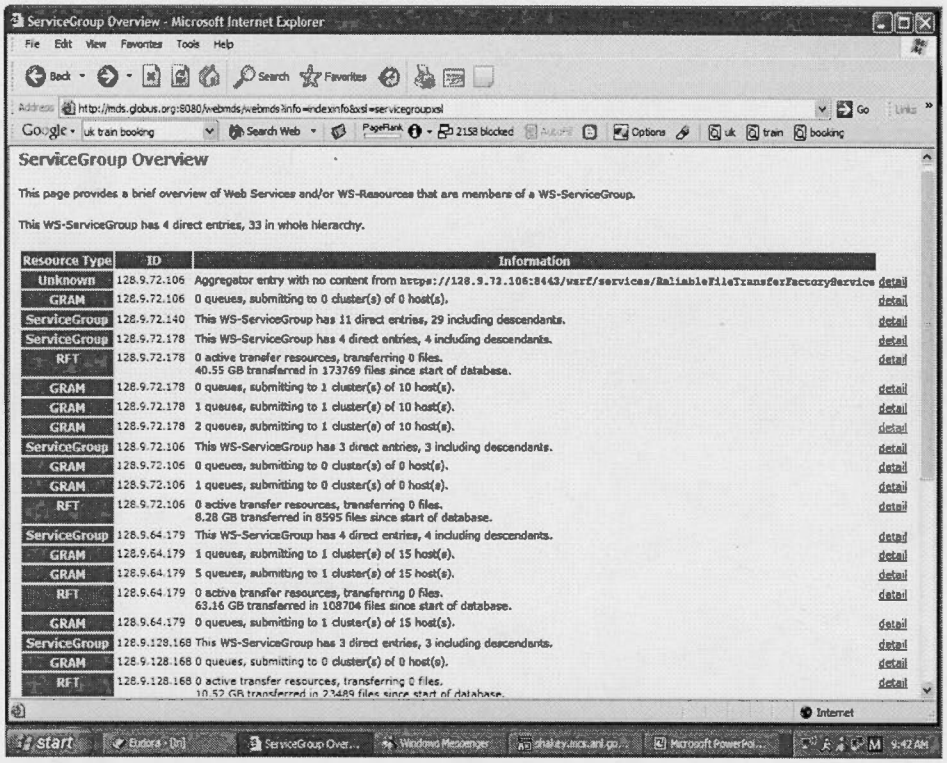


Figure 6.2. Un navigateur de WS-MDS.

On a utilisé l'environnement Axis [WWW 14] pour manipuler des services web avec tous les fichiers qui leur sont associés.

Une fois la requête du client saisie, celle-ci est transférée au Globus Toolkit. Les composants de GT4 s'occupent alors de traiter la requête et de l'envoyer à sa destination finale.

Les clients du Grid qui sont intéressés à chercher des ressources ou des services dans le registre UDDI utilisent GT4, qui fournit plusieurs interfaces de navigation, des commandes en ligne et une interface de service Web permettant d'interroger et d'accéder aux informations collectées dans le Grid. La figure 6.3 montre l'architecture de GT4 du côté client et du côté serveur.

- Écrire les classes Java avec les fonctionnalités requises.
- Importer le package de.fb12.grid.annotations.*
- Créer toutes les méthodes Java que le service pourra exporter.
- Créer les attributs de Java d'où ils seront compris dans les ressources de service avec *@GridAttribute*.
- Créer les getters et setters pour tous les services avec *@GridAttributes*.

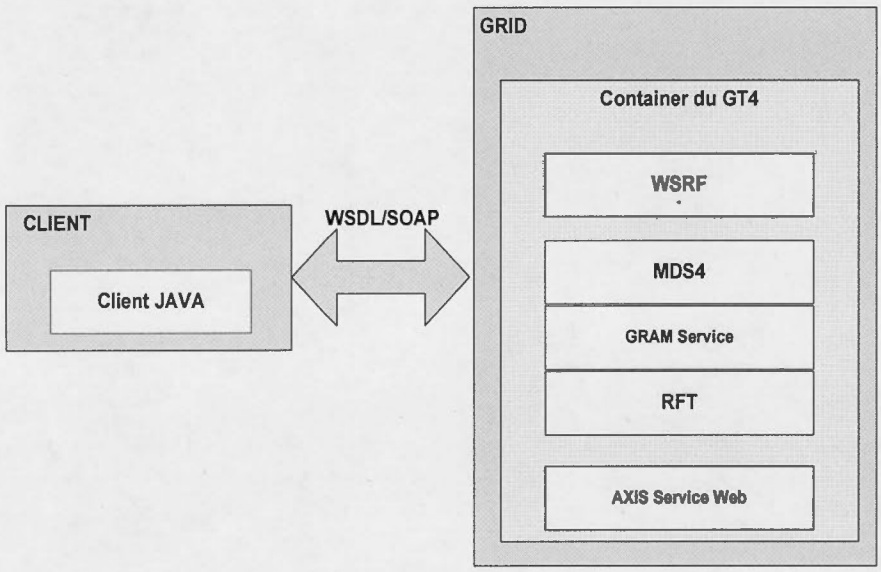


Figure 6.3. Architecture côtés client et serveur dans GT4.

Cette illustration montre un le container GT4, qui consiste en différents containers servant à gérer et déployer les services et les ressources selon leurs cycles de vie. Le GT4 utilise Axis, qui assure, entre autres, le support et la génération des applications de services Web ainsi que le traitement des messages SOAP avec JAX-RPC.

GT4 propose également des outils qui aident à générer automatiquement des fichiers WSDL à partir de classes Java.

6.3 Installation de GT4

Dans cette section nous allons discuter des détails de l'installation de GT4 et des composants nécessaires à note projet. Cette installation a été effectuée sur une machine UNIX selon les cinq étapes suivantes:

1) Installation des environnements TOMCAT et ANT. Tomcat est un serveur d'application. Il permet d'utiliser des servlets. ANT permet de construire des outils pour développer et exécuter de tâches. Ant est basé sur un fichier XML pour décrire des tâches et pour les programmes écrits en Java.

2) Écriture des services.

3) Génération de services. On a fait appel à la commande:

```
ant generateAndBuild
```

4) Déploiement des services. Pour déployer le service de GT4, on fait appel à la commande:

```
ant deployTomcat
```

5) Installation de l'environnement Eclipse [www 15] pour compiler le GT4. Une fois installé, nous pouvons commencer à créer le projet et les services.

Avec GT4, on utilise WSRF ainsi que le *Monitoring and Discovry System* qui a été adapté (MDS4). MDS4 se distingue par son utilisation étendue des interfaces et des comportements définissant les spécifications *WS-Ressource*, WS-Notification, et par son intégration des composants du Globus toolkit. En effet, la surveillance et la découverte des applications permettent de collecter des informations sur les ressources distribuées sur le Grid.

MDS4 fournit trois groupes de services avec différentes interfaces et comportements :

- *MDS-Index*, qui permet de faire la surveillance et la découverte des informations de ressources de Grid et des données de différentes ressources. Il fournit une interface

de requête/souscription pour ses données semblable au registre UDDI mais avec plus de flexibilité.

- *MDS-Tigger*, qui collecte et compare l'information de surveillance pour détecter des erreurs et informer les opérateurs des services (par exemple, par email).
- *MDS-Archive*, qui permet de stocker les données historiques de surveillance de données et d'interroger ces données.

La figure 6.4 montre comment les ressources sont agrégées et traitées à l'aide des composants de MDS4.

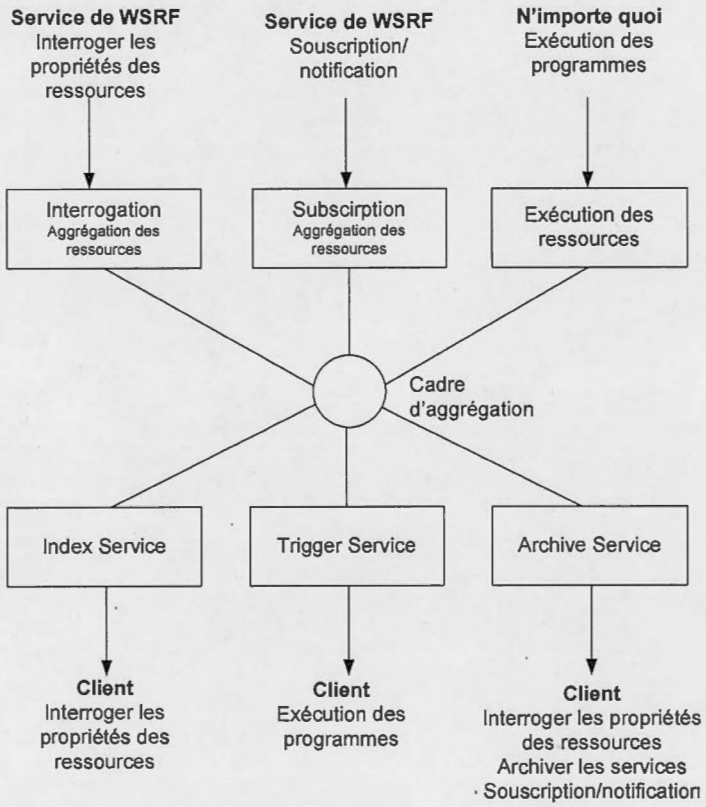


Figure 6.4. Agrégation des ressources via les éléments de MDS4

WSRF définit un cadre générique pour modeler et accéder aux ressources persistantes en utilisant des services Web de sorte que la définition, l'implantation de services, l'intégration et la gestion des multiples services soient plus faciles. Il définit aussi un

système pour créer des ressources avec état (*stateful*) entre les services Web et les standards qui rendent les ressources de Grid accessibles dans l'architecture de service Web.

L'utilisateur peut interroger le WSRF pour découvrir le type de matériel, le système d'exploitation, etc. Plusieurs types d'interrogations peuvent être configurées pour découvrir des propriétés d'une ressource.

6.4 Outils d'implantation

On a utilisé *WSDL2Java*, qui est un outil servant à construire des Proxys et des squelettes en Java. À partir d'un document de WSDL, il génère des petits programmes pouvant être utilisés aussi bien des côtés client et serveur, convertit la requête du client en une requête SOAP sur le réseau, puis reçoit la réponse. Il va ainsi permettre la création automatique de l'interface de Service Web.

La commande suivante permet d'utiliser WSDL2Java et génère des liens (*binding*) nécessaires pour le client:

```
% java org.apache.axis.wsdl.WSDL2Java WSDL-file-URL.
```

On a également utilisé un générateur de SOAP qui permet de manipuler les requêtes et les réponses de SAOP. Il sert à faire les opérations suivantes :

- Recevoir un message d'un protocole de transport.
- Vérifier la sémantique de SOAP.
- Traiter les en-têtes de SOAP.
- Décoder le message.
- Conduire le message au service.
- Coder la réponse.
- Traiter les en-têtes de SOAP de réponse.
- Retourner la réponse au-dessus du transport.

Pratiquement, il est plus commun d'utiliser un moteur générique de SAOP que de gérer un serveur à distance pour chaque service Web individuel. Un bon exemple d'un moteur de SOAP est Apache Axis, qui est en fait le moteur de SAOP employé par le Globus Toolkit.

6.5 Implantation du référentiel

Dans cette section, nous décrivons un système référentiel basé sur le Grid Service et le registre de UDDI qui représente un intermédiaire pour résoudre le problème d'intégration de l'environnement de Grid avec le registre UDDI d'où un utilisateur de Grid peut réaliser ses tâches dans UDDI.

L'avantage d'utiliser le GS est qu'il peut se présenter avec état (statfull) ou sans état (statless) et de manière transitoire (*transcient*) ou non transitoire ; ce qui n'est pas le cas pour les services Web classiques, qui sont persistants et sans état.

La figure 6.5 montre le schéma de notre modèle d'intégration de GS et UDDI :

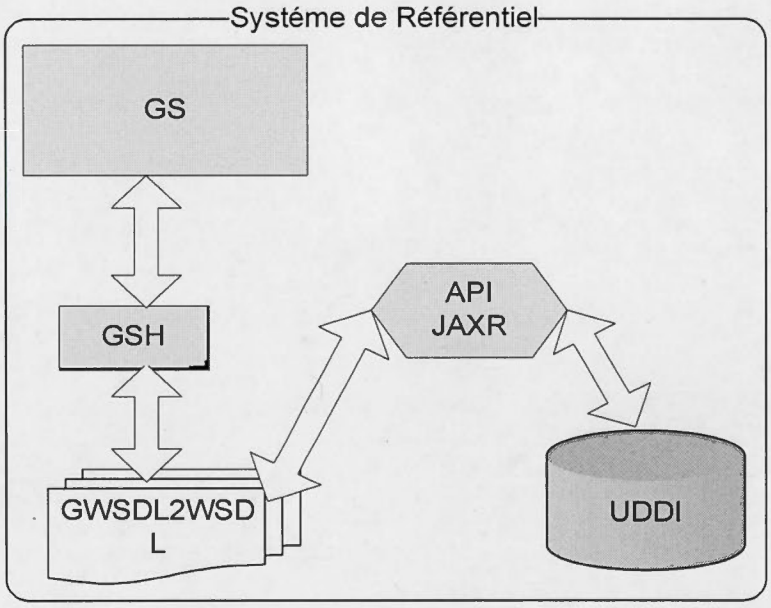


Figure 6.5. Intégration de GS et UDDI.

Selon ce modèle, le client envoie d'abord sa requête via l'environnement de Grid au système référentiel. La requête arrive ensuite au GS, qui prend en charge son traitement en créant une nouvelle instance de service avec un nouveau GSH, puis va chercher son GSR et enregistre le nouveau service dans son registre. Cette étape sera bien détaillée dans les figures 6.6 et 6.7. Pour générer les fichiers *Grid Service Handle* (GSH) et *Grid Service Reference* (GSR), le code ci-dessous montre comment définir le GSH et le GSR dans les fichiers de déploiement XML.

Dans GSH, on utilise la description XML suivante :

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"
<xsd:element name="handle" type="ogsi:HandleType"/>
        <xsd:simpleType name="HandleType">
       <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>
```

Dans GSR, on utilise plutôt :

```
targetNamespace = http://www.gridforum.org/namespaces/2003/03/OGSI"
```

```
<xsd:complexType name="WSDLReferenceType">
  <xsd:complexContent>
    <xsd:extension base="ogsi:ReferenceType">
     <xsd:sequence>
        <xsd:any namespace="http://schemas.xmlsoap.org/wsdl/"
             minOccurs="1" maxOccurs="1" processContents="lax"/>
     </xsd:sequence>
    </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

GSH peut aussi spécifier le *Handle Resolver*, qui s'appelle *WS-RenewableReferences* dans le WSRF, afin de mettre à jour la référence des ressources codées dans une adresse *WS-Addressing*, qui fournit des mécanismes pour localiser les services Web.

Le GS envoie la nouvelle instance de service avec son GSH au registre UDDI dans un fichier GWSDL (*Grid Web Services Description Language)*. En effet, un Grid Service est défini en *GWSDL*, qui contient quelques attributs prédéfinis tels que des types, des opérations et des liaisons (*bindings*). Une nouvelle étiquette *gwsdl* a été ajoutée au document de WSDL pour permettre la description des Grid Services. Une fois l'interface de Grid Services définie, cette étiquette doit être rendue disponible à l'usage des autres utilisateurs. Le code ci-dessous montre comment un service avec ses opérations définies en WSDL sera présenté en GWSDL :

```
<wsdl:definitions>
      <wsdl:types>...</wsdl:types>
         <wsdl:message>...</wsdl:message>
            ...
          <gwsdl:portType name="Service" extends="ns:bar ogsi:GridService">
                  <wsdl:operation name="Op1">...</wsdl:operation>
                  <wsdl:operation name="Op2">...</wsdl:operation>
                  <ogsi:serviceData ... />
          </gwsdl:portType>
  ...
</wsdl:definitions>
```

Une fois que le nouveau GSH écrit en GWSDL a quitté le GS vers le registre UDDI, un convertisseur du fichier GWSDL au fichier WSDL est utilisé. Le code ci-dessous montre un exemple de mécanisme de conversion de GWSDL au WSDL :

```
<ant antfile="${build.services}" target="GWSDL2WSDL">
    <property name="build.schema.dir" value="guide/Counter"/>
    <property name="wsdl.root" value="counter_port_type"/>
</ant>
<ant antfile="${build.services}" target="generateBinding">
    <property name="binding.root" value="counter"/>
    <property name="build.schema.dir" value="guide/Counter"/>
    <property name="porttype.wsdl" value="counter_port_type.wsdl"/>
</ant>
```

Pour que la requête puisse avoir accès au registre UDDI, il faut qu'elle passe par une interface. Dans notre travail on a utilisé l'API JAXR [WWW 12]. Pour que la requête définie en WSDL soit comprise par l'interface JAXR, nous avons adopté Apache Axis. Celui-ci est un intermédiaire pour convertir le fichier WSDL en Java et inversement grâce aux outils *WSDL2Java* et *Java2WSDL*.

WSDL2Java permet de générer des classes Java à partir d'un fichier WSDL local ou distant afin que le fichier GSH/WSDL soit compris par JAXR du registre UDDI. En pratique, il sert à générer des souches (*stubs*) du coté client à partir de la lecture du fichier WSDL du service. Le code ci-dessous illustre comment générer une classe Java d'un fichier WSDL :

```
<xsd:complexType name="phone">
 <xsd:all>
  <xsd:element name="areaCode" type="xsd:int"/>
  <xsd:element name="exchange" type="xsd:string"/>
  <xsd:element name="number" type="xsd:string"/>
 </xsd:all>
</xsd:complexType>
``** WSDL2Java génère le code Java suivant**``
public class Phone implements java.io.Serializable {
 public Phone() {...}
 public int getAreaCode() {...}
```

```
    public void setAreaCode(int areaCode) {...}
    public java.lang.String getExchange() {...}
    public void setExchange(java.lang.String exchange) {...}
    public java.lang.String getNumber() {...}
    public void setNumber(java.lang.String number) {...}
    public boolean equals(Object obj) {...}
    public int hashCode() {...}
}
```

Java2WSDL permet de générer un fichier WSDL à partir d'un code Java. Le code ci-dessous illustre comment générer une classe Java d'un fichier WSDL.

```
package samples.userguide.example6;
/** Interface describing a web service to set and get Widget prices.**/
  public interface WidgetPrice {
     public void setWidgetPrice(String widgetName, String price);
     public String getWidgetPrice(String widgetName);
}
```

L'API JAXR permet d'analyser le fichier WSDL et de le transférer au registre UDDI via le protocole SOAP pour permettre de stocker, chercher ou publier le service demandé dans UDDI par l'utilisateur du Grid.

6.6 L'implémentation de Grid Service dans le répertoire UDDI

L'implémentation de Grid Service dans le répertoire UDDI se fait en plusieurs étapes :

- Écrire une définition de WSDL *PortType* en utilisant le OGSA types (avec *Java2WSDL*).
- Écrire une définition du binding WSDL pour identifier des moyens de se connecter avec le service en utilisant par exemple, SOAP/HTTP, TCP/IP, etc.
- Écrire une définition de service en WSDL basée sur le *PortType*.

- Implanter une *Factory* qui fournit une extension du *FactorySkeleton* afin d'indiquer comment les nouvelles instances des services vont être créées.
- Configurer la *Factory* avec différentes options disponibles.
- Implanter les opérations du service comme extension de la classe de *ServiceSkeleton.*
- Implanter du code pour le client qui doit interagir avec le service.

À l'instar des archives pour les services web (fichier *web archive* ou *war*) utilisés pour leur déploiement, un service Grid est livré sous la forme d'une archive Grid service (fichier *grid archive* ou *.gar*). Cette archive contient les fichiers nécessaires pour l'exécution du Grid Service dans un environnement d'exécution de service compatible avec WSRF. Cet environnement contient les éléments suivants:

- Bibliothèques du service, bibliothèques des souches générées et bibliothèques existantes utilisées par le service (fichier .jar)
- Description du service (fichiers .wsdl et .gwsdl)
- Déclaration des services data en XML Schéma
- Descripteur de déploiement du service (fichier .wsdd).
- Fichiers de configuration (fichiers .properties', .xml, ...)

Le déploiement d'un Grid service se fait de façon à l'aide de la commande :

```
ant –Dgar.name=<fichier.gar> deploy
```

Pendant la phase de déploiement, des fichiers sont générés pour permettre la désinstallation du service à l'aide de la commande :

```
ant –Dgar.id=<identifiant> undeploy
```

6.7 Implantation du répertoire UDDI

UDDI a été conçu comme un système d'annuaire commercial mais comporte quelques limitations qui compliquent la découverte de ressources par le Grid. Cependant, UDDI permet d'effectuer la recherche sur les attributs de services, tels que le nom de service qui peut être choisi par le fournisseur de service, la clé de référence, des mots-clés spécifiques faisant partie d'une catégorie du domaine de l'entreprise qui offre le service (appelée *categoryBag*), etc.

Dans notre travail, nous avons installé le JUDDI de Apache [WWW 16] et les composants nécessaires pour son fonctionnement. Cette installation a été faite sur un système UNIX. Cette installation a nécessité l'utilisation de Tomact comme serveur d'applications. Pour le stockage des descriptions des services, on a eu recours à un système de bases de données MySQL, tel que requis par JUDDI.

Une fois ces étapes franchies, on crée une base de données et on lui accorde les privilèges appropriés grâce à la commande GRANT de SQL.

Pour invoquer un des services qui figurent dans le registre UDDI, GS traite la requête des usagers selon deux scénarios.

Scénario 1

Dans ce scénario (Figure 6.6), l'utilisateur envoie sa requête à l'environnement du Grid via HTTP (1). À son tour, le Grid traite la requête et la transfère au répertoire via le protocole de SOAP (2). Dès que la requête arrive au répertoire, le GS sera en mesure de la traiter et l'acheminer à sa destination. En particulier, certains éléments de GS tels que le Factory vont créer une nouvelle instance de services d'où l'utilisateur va interagir avec une autre instance qui maintient l'état des données

pendant la durée de l'interaction. Afin de distinguer cette autre instance, un nouveau Grid Service Handle *GSH* (qui se présente sous forme d'un URI) sera associé à chaque nouvelle instance comme un identificateur de services et spécifiera l'endroit où ce service se trouve ainsi que la manière dont il est employé. D'autres spécifications seront associées, tel que les protocoles d'attachement, l'adresse de réseau ainsi que le Grid Service Reference (GSR), qui contient les informations pour communiquer avec le GS à l'aide d'un HandleMapper. Ce dernier est un document WSDL dont le rôle est de chercher le GSR pour résoudre le GSH. Aussi, l'instance GS va chercher le GSR dans le HandleResolver (3 et 4). Le rôle du HandleResolver sera de retourner le GSR pertinent au GSH pour la nouvelle instance GS (5). Celle-ci sera envoyée avec son GSH au registre UDDI encapsulé par le fichier GWSDL. Ce fichier sera converti au fichier WSDL avec le GWSDL2WSDL et sera envoyé à l'interface de UDDI par l'API JAXR. De son côté, JAXR enverra la requête au répertoire UDDI pour trouver le service demandé par l'utilisateur via le protocole SOAP (6). UDDI retournera alors la réponse à l'instance GS en passant par JAXR, qui va à son tour retourner la réponse à l'aide de Axis qui va alors générer le fichier WSDL d'une classe Java (7). L'instance GS retournera le résultat à l'environnement du Grid avec SOAP (8), puis à l'utilisateur du Grid (9).

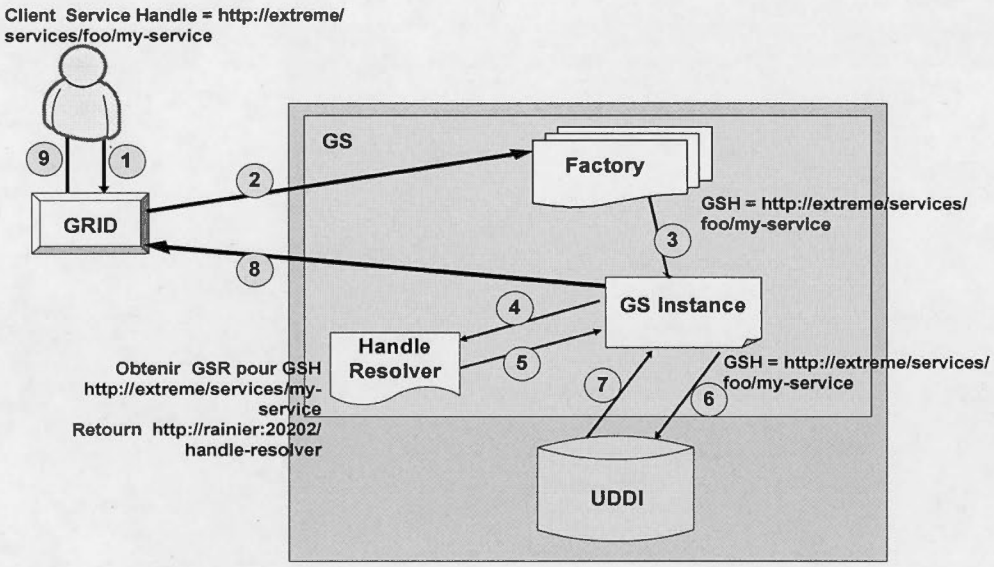


Figure 6.6. Traitement des requêtes des clients- Scénario 1.

Scénario 2

Ce scénario sera appliqué seulement si la nouvelle instance de services existe déjà dans le registre de GS (Figure 6.7). Les étapes 1 à 3 seront les mêmes que celles du scénario 1. À partir des étapes 4 et 5, la nouvelle instance de GS sera envoyée au registre de GS afin de sélectionner le GSH pour identifier cette instance à l'aide de son GSR. GSH sera envoyé au registre UDDI (6). UDDI retournera alors la réponse à l'instance GS, qui retournera le résultat à l'environnement du Grid (7). Finalement, l'instance GS retournera le résultat à l'environnement du Grid (8), puis à l'utilisateur du Grid (9).

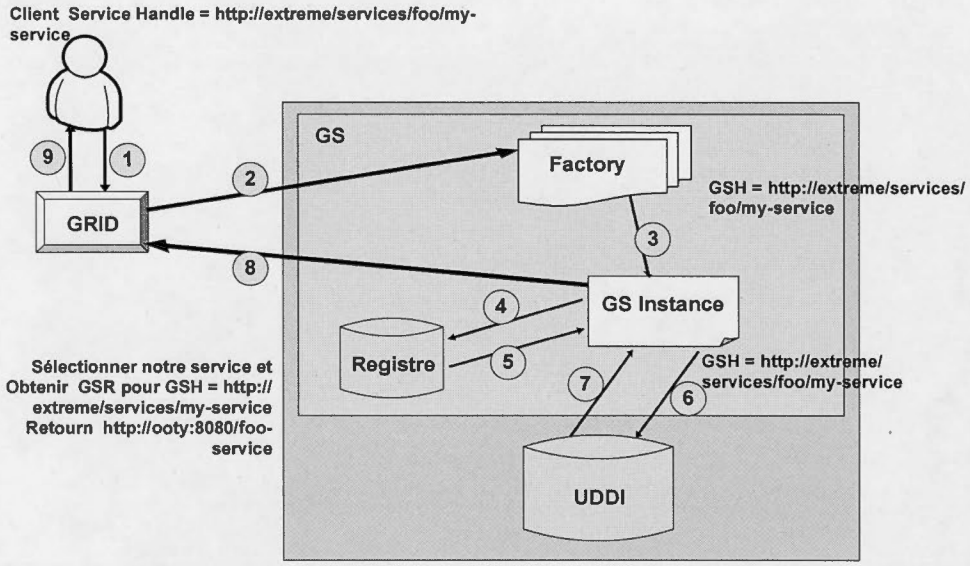


Figure 6.7. Traitement des requêtes des clients- Scénario 2.

Ces deux scénarios sont présentés à la figure 6.8.

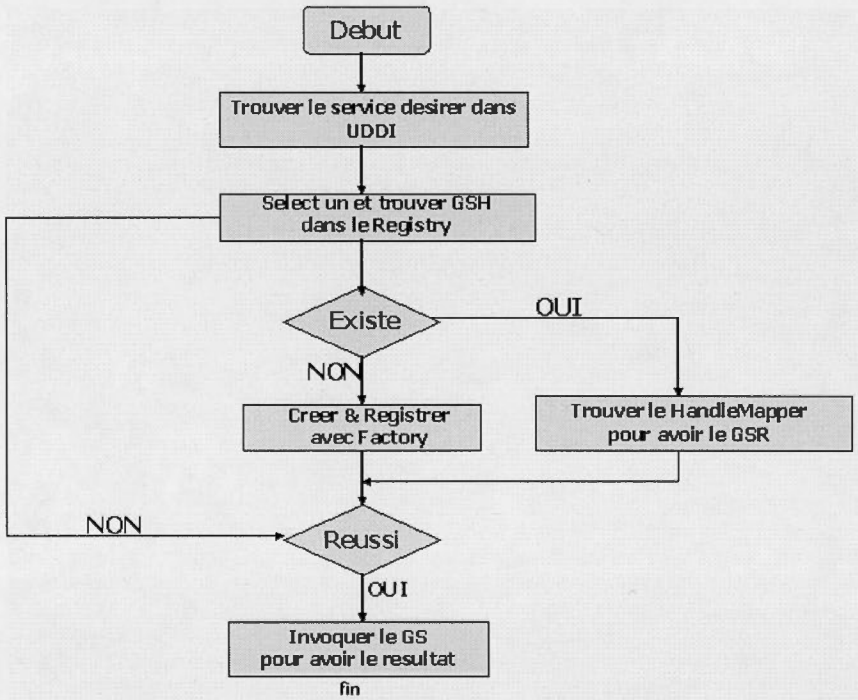


Figure 6.8. Modèle d'intégration GS et UDDI.

Pour publier et interroger UDDI à distance, nous avons utilisé JAXR. Cet API permet l'accès, l'envoi de requêtes et l'édition de registres XML tels que UDDI. Comme le montre la figure 6.9, JAXR est en même temps un API client et un serveur. Il fournit une exécution complète de l'interface de programmation d'application d'UDDI. Les classes de JAXR incluent une représentation complète des entités de données d'UDDI. Par exemple, la classe *TModel* représente une structure TModel UDDI, et la classe *BusnisseService* une structure un UDDI *BusnisseService*. JAXR inclut également une classe Proxy de registre UDDI (Figure 6.10). Cette classe est employée pour envoyer la publication et trouver les services dans le registre UDDI.

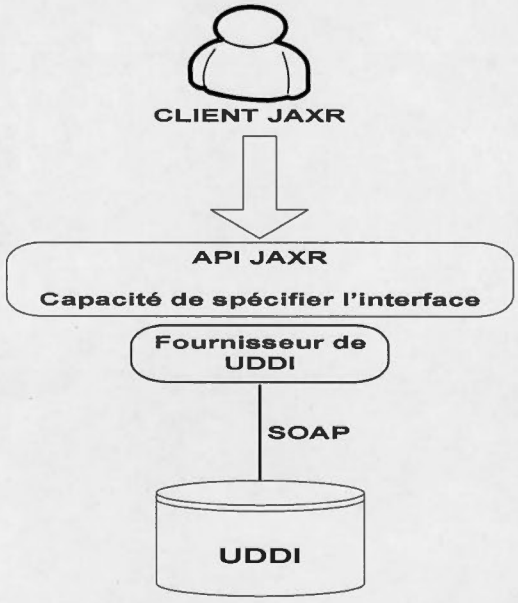


Figure 6.10. Accès à UDDI avec JAXR.

La figure 6.11 illustre par un exemple comment UDDI pourrait être employé dans un projet simple de Grid pour la découverte de ses propres *businessEntities* et de leurs services par des composants de niveau élevé, tels que des applications et des portails APIs (par exemple UDDI4J et JAXR), qui sont disponibles pour accéder à cette information dans des applications de Grid.

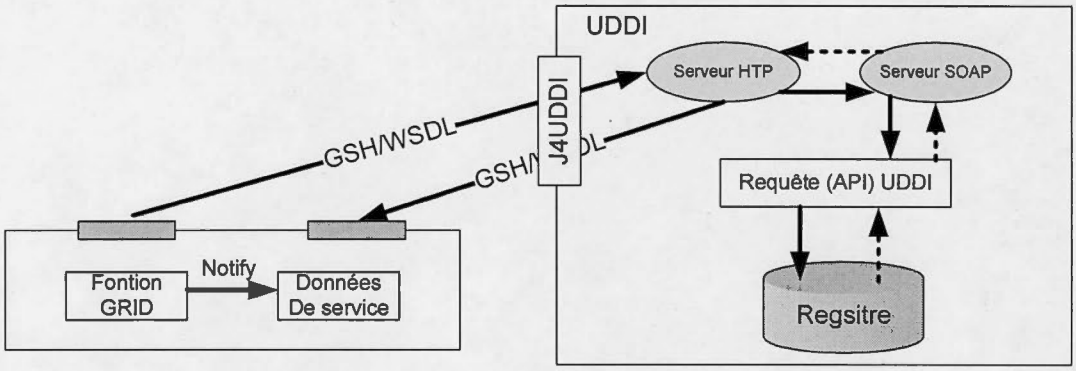


Figure 6.11. Échange de services entre le Grid et Service Web.

Chapitre VII

CONCLUSION

Nous avons proposé un middleware permettant l'intégration de l'environnement du Grid et des services web à partir de l'intégration de l'annuaire UDDI.

Nous avons tout d'abord étudié plusieurs standards et technologies permettant un traitement efficace et une utilisation optimale des ressources sur le réseau.

Le Grid constitue une plateforme et un environnement favorisant l'utilisation des ces ressources. Les services web, de leur côté, constituent une solution efficace pour l'intégration des applications réparties. Nous avons décrit des standards relatifs à ces deux technologies.

Le Grid Computing offre une vaste infrastructure qui permet à un ensemble de machines d'interagir et d'exploiter les ressources nécessaires au calcul réparti. Cet environnement exige de coordonner les applications, les données, le stockage ou les ressources de réseau à travers des organisations géographiquement dispersées. Pour cela, nous avons voulu exploiter le concept de services web et ses techniques standards, qui sont construits pour assurer une interopérabilité entre les systèmes et les applications.

Nous avons étudié la convergence entre la technologie du Grid et les services Web. Cette convergence nous a amené à fournir des solutions à l'intégration de ces deux environnements. Ceci nous poussé à utiliser et adapter les architectures OGSA basées sur le Grid et les services Web.

Nous avons également présenté une architecture pour cette intégration. Nous nous sommes intéressés à l'interconnexion entre le Grid et l'annuaire UDDI. Cette

architecture nous a aidé à développer un système relativement souple permettant l'échange des services entre l'environnement du Grid et l'annuaire de UDDI.

Dans le futur, nous voudrions introduire une solution plus complète qui permettrait cette intégration de manière transparente. Cette solution aurait pour objectif de rendre transparentes les composantes du Grid et des services web de manière à permettre aux usagers des services web d'accéder efficacement aux ressources du Grid.

Bibliographie

[Chauvet, 2002] Chauvet J-M., Services Web avec SOAP, WSDL, ebXML, Eyrolles (2002), ISBN 2-212-11047-2.

[Dikaiakos, 2004] Dikaiakos, M., Grid computing, Springer, c2004, ISBN 3-540-22888-8 LC 2004-113302

[Favali 2004] Favali R., Stuart S., New Specifications: Grid and Web Services Standards to Converge, http://xml.coverpages.org/WS-NotificationAnn.html

[Ferreira, 2003] Ferreira L., Viktors B., Introduction to Grid Computing with Globus, IBM October 01, 2003, ISBN: 0-7384-9988-9

[Foster, 2004] Foster I., The Grid, Blueprint for a New Computing Infrastructure, 2004. ISBN: 1-55860-933-4.

[IBM, 2004] Grid Services Programming and Application Enablement, IBM RedBook 2004, ISBN 0738498033

[Jacob 2003] Bart J, Luis F.: Enabling Applications for Grid Computing with Globus, IBM June 18, 2003, ISBN 0-7384-5333-1

[Jean-Marie, 2002] Jean-Marie A., Advances in computing science--ASIAN 2002, Springer, c2002, ISBN 3-540-00195-6 LC 2002-42483.

[Kesselman, 1998] Kesselman C., Foster I., The Grid Blueprint for a New Computing Infrastructure.

[Li 2005] Li M., Mark B.: The Grid: Core Technologies, John Wiley & Son's 2005, ISBN: 0-470-09417-6

[Mailvaganam, 2005] Mailvaganam H., Understanding SOA with Web Services, 2005. ISBN: 0321180860.

[Sotomayor 2005] Boja S., Lisa C., Globus Toolkit 4 (Programming Java Services), Morgan Kaufmann 2005, ISBN 0123694043.

[Taylor 2005] Taylor I.: Grid Computing: A Practical Guide To Technology And Applications, Springer 2005, ISBN: 1-85233-869-5.

[Tuecke, 2000] Tuecke S., The Anatomy of the Grid,

[Tueke, 2003] Tuke A. et all., Open grid services infrastructure (OGSI) version 1.0, june 2003.

[Zhang, 2004] Zhang L., Jeckle M., Web Services, Springer, C2004. ISBN 3-540-23202-8 LC 2004-112843

[www 1] http://www.trucsweb.com/Java/rmi.

[www 2] http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/

[www 3] http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[www 4] http://www.uddi.org/

[www 5] http://developpeur.journaldunet.com/fonctionnementsysteme- grille.shtml

[www 6] http://www.globus.org

[www 7] http://www-128.ibm.com/developerworks/webservices/library/gr-visual

[www 8] http://www-128.ibm.com/developerworks/library/specification/ws-resource/

[www 9] http://www.corba.org/

[www 10] http://www.w3.org/XML/

[www 11] http://www.w3.org/TR/wsdl

[www 12] http://java.sun.com/webservices/jaxr/.

[www 13] http://www.gridforum.org.

[www 13] http://www.sei.cmu.edu/isis/guide/technologies/ogsa.htm#GGF05

[www 14] http://ws.apache.org/axis/.

[www 15] http://www.eclipse.org/.

[www 16] http://ws.apache.org/juddi/.