Analyse et synthèse de XML

Philippe Jost

15 mars 2000

Table des matières

Chapitre 1

Buts

Nous allons étudier, dans ce rapport, les possibilités de XML dans les systèmes distribués. La question principale est de savoir si XML est, en utilisant HTTP comme protocole oui ou non un middleware et nous étudierons plusieurs solutions existantes. Nous nous occuperons aussi de la possibilité de l'utiliser en combinaison avec CORBA. L'analyse de ces diverses solutions fera l'objet d'une section. Pour la suite du document, XML comme middleware signifiera XML et HTTP comme middleware. Nous verrons aussi quelques autres applications intéressantes, de même que des outils et des librairies disponibles à ce jour. Mais avant cela, nous commencerons par introduire brièvement les notions de base de XML.

Les règles typologiques de ce document sont les suivantes:

1. Le code XML sera affiché comme suit: `<tag>...</tag>`

2. Les mots de langue anglaise sont en *italique*

Chapitre 2

Introduction

2.1 Historique de XML

En juillet 1996, le World Wide Web Consortium (http://www.w3c.org) crée le XML Working Group avec, entre autre des représentants de chez Sun, Netscape, Microsoft et Hewlett Packard, afin de définir les spécifications de XML. Dès novembre 1996, les premiers documents de travail concernant XML sont publiés, mais il faudra attendre février 1998 pour que les recommandations finales de XML 1.0 soient rendues publiques. Dès lors, toutes les grandes entreprises du domaine se sont intéressées au XML. IBM, Oracle, Sybase, Adobe, Macromedia... ont annoncé que leurs produits destiné au Web supporteraient XML.

2.2 Petite introduction à XML

Commençons par quelques généralités sur XML. XML est du SGML sans les parties qui sont difficiles d'implémentation, c'est la seule différence technique entre XML et SGML. Il existe encore une autre différence: XML a été créé pour une utilisation sur le web.

Nous avons pu assister au développement rapide d'internet ces dernières années. Le langage de formatge HTML y est pour beaucoup, mais il s'est vite avéré assez limité malgré le nombre de *tags* qui y ont été ajoutés. Le besoin d'une nouvelle solution est né. XML est la réponse à ce besoin.

XML est un langage de formatage qui est en passe de devenir le standard pour l'échange de données sur le web. De même qu'avec HTML, les données sont identifiées avec des *tags* (des identifiants placés entre `<...>`).

Mais à l'inverse de HTML, les *tags* de XML décrivent le sens des données plutôt que la manière de afficher. Voici un petit exemple d'une application pour envoyer des messages:

```
<message>
    <to>personne@domaine.com</to>
    <from>moi@monDomaine.com</from>
    <subject>XML c'est sympa</subject>
    <text>
      On peut vraiment faire beaucoup de choses avec XML.
    </text>
</message>
```

Les tags peuvent aussi contenir des attributs, des informations additionnelles qui sont inclues dans les *tags*. L'exemple suivant montre une structure du message qui utilise des attributs pour `to`, `from` et `subject`:

```
<message to=''personne@domaine.com'' from=''moi@monDomaine.com'' sub-
ject=''XML c'est sympa''>
    <text>
      On peut vraiment faire beaucoup de choses avec XML.
    </text>
</message>
```

Une grande différence entre HTML et XML est que XML a des règles syntaxiques plus contraignantes. Il y a quelques règles qui permettent de déterminer si le document est correctement formaté. Voici deux des règles les plus importantes:

1. Chaque *tag* doit avoir un *tag* qui le referme. Dans notre premier exemple, le *tag* `</to>` n'est pas optionnel. De plus le tag `<to>` ne doit jamais être fermé par un autre *tag* que `</to>`.

2. Une autre contrainte importante est que tous les tags doivent être emboîtés. Il est possible d'avoir
<message>..<to>..</to>..</message>, mais pas
<message>..<to>..</message>..</to>.

Une liste complète des contraintes se trouve dans la liste des FAQ de XML à l'adresse suivante: http://www.ucc.ie/xml/#FAQ-VALIDWF.

Un fichier XML commence toujours avec un prologue. Le prologue minimal contient une déclaration qui identifie le document:

```
<?xml version=''1.0''?>
```

La déclaration peut aussi contenir des information additionnelles:

```
<?xml version=''1.0'' encoding=''ISO-8859-1'' standalone=''yes''?>
```

version : Identifie la version de XML utilisée dans le document.

encoding : Identifie l'encodage utilisé pour le document. "ISO-8859-1" est "Latin-1".

standalone : Indique si le document référence une entité extérieure ou une spécification du type de donnée.

XML permet de définir ses propres *tags* à l'aide d'un DTD (*Document Type Definition*). Regardons maintenant le processus de développement d'un DTD. Une personne voulant écrire un fichier XML peut définir le rôle de chaque élément selon un modèle formel: un DTD. Un DTD permet à un ordinateur qui est un *parser* de contrôler la structure logique du document. La présence d'un DTD n'est pas obligatoire. Dans le cas où aucun DTD n'est utilisé, une définition par défaut est attribuée au composants du fichier qui ne sont pas déclarés.

Reprenons le premier exemple de ce chapitre et écrivons un DTD qui lui sera attribué.

```
<!DOCTYPE message [
 <!ELEMENT message  (to, from, subject?, text+) >
 <!ELEMENT to       (#PCDATA) >
 <!ELEMENT from     (#PCDATA) >
 <!ELEMENT subject  (#PCDATA) >
 <!ELEMENT text     (#PCDATA) >
]>
```

Ce modèle explique au *parser* qu'un message est constitué d'une séquence de *tags*. Le point d'interrogation indique que ce *tag* est optionnel.

Un standard intéressant développé par le W3C est le *namespace*. Voyons en premier comment on le déclare:

```
<tagname xmlns[:name]=URI>
```

L'URI ne doit pas forcément se référer à un DTD ou à toute autre définition possible; l'URI fournit simplement un identifiant unique pour le *namespace*. Reprenons l'exemple de notre message ci-dessus, le tag `text` est, dans notre cas, le message à envoyer; mais dans d'autres applications XML, il peut avoir des significations différentes. L'utilisation d'un *namespace* permet de le rendre unique et d'éviter ainsi toute confusion. Nous verrons plus loin l'utilisation des *namespaces* et les avantages qu'ils amèment pour l'extensibilité du langage.

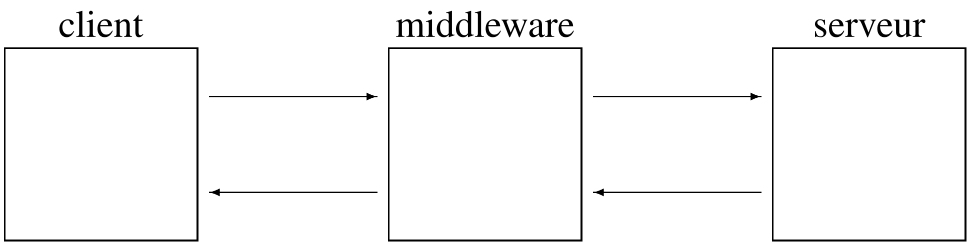
Chapitre 3

XML dans les systèmes distribués

3.1 XML comme middleware

Ces dernières années, nous avons assisté à une révolution dans le monde informatique, la montée en puissance des ordinateurs, la baisse des prix et le développement fulgurant des réseaux. Cela a entraîné l'éclosion de nouvelles applications distribuées comme le *World Wide Web* et la messagerie électronique. Toutes ces nouvelles ressources sont accédées par de nombreux utilisateurs répartis sur un ensemble de machines hétérogènes. Cela a entraîné la définition de normes, pour la communication entre ces machines, prenant en compte l'hétérogénéité, l'intégration et l'interopérabilité des applications distribuées.

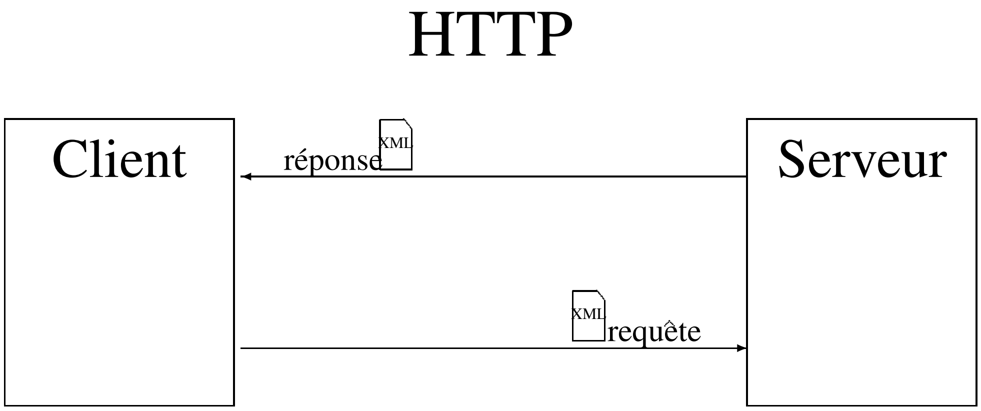
Un middleware est un software qui connecte un système à un autre. Nous allons étudier ici plusieurs nouvelles solutions basées sur XML et voir ce que la combinaison de CORBA et de XML peut apporter de nouveau.



3.1.1 XML comme nouvelle solution

Internet est maintenant utilisé comme plateforme pour une nouvelle génération d'applications distribuées. Il existe le besoin d'une architecture qui adopterait certaines caractéristiques de CORBA.

HTTP a été originellement conçu comme un protocole relativement simple pour la communication entre un *browser* et un serveur. Un serveur HTTP supporte un petit nombre de requêtes, par exemple GET, POST. Des données XML peuvent être insérées dans une requête à un serveur HTTP; la réponse du serveur peut de même inclure un document XML contenant les résultats à la requête. Voyons maintenant l'aspect qui nous intéresse de la possibilité offerte par l'utilisation de XML avec le protocole HTTP: les appels de procédure à distance (*remote procedure calls* ou RPC). La figure suivante illustre ce principe:



A un niveau primaire, les besoins pour envoyer un appel à une procédure sont un protocole de communication et un format de message. Sur le web, ces tâches étaient initialement accomplies avec HTTP et CGI. Il existe maintenant plusieurs propositions pour des applications qui utilisent HTTP pour transporter les requêtes et XML comme syntaxe de message. Nous allons voir ici trois de ces solutions: WebBroker, SOAP et XML-RPC. Pour plus de facilité de comparaison, nous regarderons à chaque fois le même exemple: on appèle une fonction doSomething qui prend en paramètre un *integer* et un *array* contenant deux *strings*. La réponse est un *boolean* qui vaut *true*.

WebBroker

Il n'y a pas de spécifications finales pour WebBroker, mais seulement un rapport technique soumis au W3C le 11 mai 1998. Le but est d'avoir un modèle qui bénéficierait des avantages de CORBA et de DCOM et qui serait moins compliqué et plus puissant que la combinaison de HTML et CGI.

Voyons un petit exemple d'une requête faite avec WebBroker:

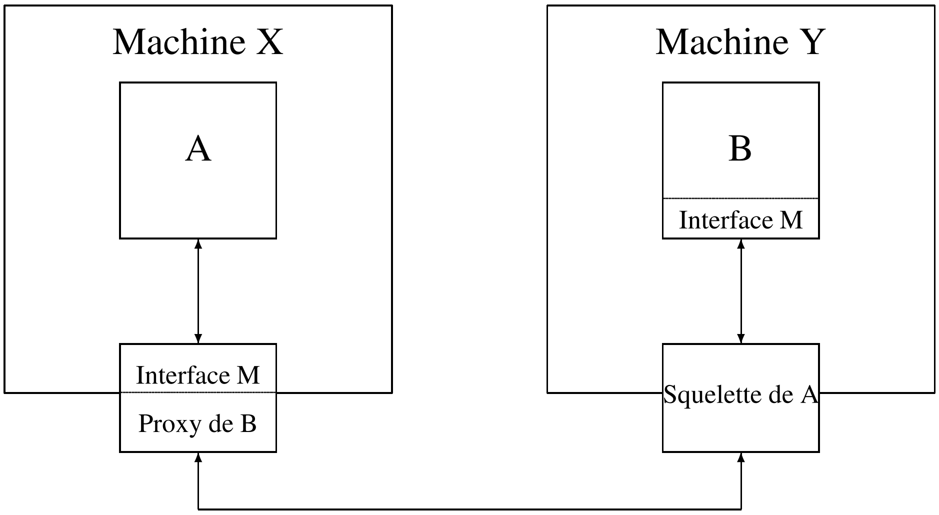
```
<objectMethodRequest version="0.9">
  <methodCall methodName="doSomething">
    <int>1234</int>
    <stringArray length="2">
      <string>little</string>
      <string>test</string>
    </stringArray>
  </methodCall>
</objectMethodRequest>
```

Voyons maintenant ce que pourrait être la réponse:

```
<objectMethodResponse version="0.9">
  <methodReturn>
    <boolean value="true" />
  </methodReturn>
</objectMethodResponse>
```

Il existe plusieurs styles de communication entre composants dans des systèmes distribués; le style utilisé par WebBroker est basé sur les interfaces. Le but d'une communication par interface est qu'un 'objet' puisse faire un appel à un autre 'objet', (potentiellement) sur une autre machine, sans que le programmeur ou les objets aient à traiter le fait que les objets se trouvent sur des machines différentes. L'appel à une méthode se fait sur un intermédiaire. Cet intermédiaire est appelé un *proxy* dans WebBroker et il implémente la même interface que la cible.

Le cas typique est qu'un objet A sur une machine X à besoin d'appeler une méthode sur l'objet B sur une machine Y:



- Le proxy de l'objet B tourne sur la machine X
- Le squelette de l'objet A tourne sur la machine Y
- Le *proxy* et le squelette sont les deux bouts d'une connection virtuelle
- Le *proxy* de B à la même interface que l'objet B
- Le *proxy* et le squelette envoient les requêtes et les réponses par le réseau sous forme de document XML

WebBroker a besoin d'un DTD pour définir les messages et d'un autre pour définir les interfaces sur les objets.

Simple Object Access Protocol (SOAP)

SOAP est, selon Microsoft, une façon d'utiliser l'infrastructure d'internet pour permettre aux applications de communiquer directement entre elles sans être bloquées par un firewall. HTTP est un protocole flexible, mais il n'a pas été conçu pour l'invocation de procédures à distance.

Dans une architecture TCP/IP, chaque protocole est assigné à un numéro de port, chaque requête utilisant ce protocole transporte ce numéro. La plupart des firewalls bloquent un protocole en rejetant toute demande qui accède au port de ce protocole. Dans la majeure partie des cas, les firewalls permettent le trafic uniquement sur le port 80 (HTTP).

C'est là que se situe un des problèmes avec les *distributed object protocols*; généralement, ils n'ont pas de numéro de port qui leur est attribué. Ces protocoles utilisent plutôt des ports dynamiquement attribués. Cela ne pose aucun problème si il n'y a pas de *firewall* entre le client et le serveur. Il existe plusieurs approches pour régler ce problème. Certain *firewalls* peuvent être configurés pour laisser passer le trafic de certains ports donnés. De plus il n'est pas trop difficile ni trop coûteux de reprogrammer un *firewall*.

Un des buts premier de SOAP est d'être directement utilisable avec les infrastructures d'internet déjà en place (HTTP, des *proxies*, des *firewalls*). En ce qui concerne la sécurité, SOAP utilise SSL (Secure Socket Layer).

Voyons maintenant un exemple d'une requête SOAP:

```
POST /Something /HTTP/1.1
```

```
Host: www.anyhost.com
Content-Type: text/xml
Content-Length: nnnn
SOAPMehtodName: Some-Namespace-URI\#doSomething

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:doSomething xmlns:m="Some-Namespace-URI">
      <integer>1234</integer>
      <ArrayOfstring> xsd:type=''u:string[2]''>
        <string>little</string>
        <string>test</string>
      </ArrayOfstring>
    </m:doSomething>
  </SOAP:Body>
</SOAP:Envelope>
```

La réponse est:

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:doSomethingResponse xmlns:m="Some-Namespace-URI">
      <return>1</return>
    </m:doSomethingResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

XML-RPC

XML-RPC est un protocole d'appel de procédure à distance fonctionnant sur Internet. Un message XML-RPC est une requête HTTP-POST. La requête est sous forme de fichier XML. La procédure invoquée est exécutée sur le serveur et la valeur en retour est aussi en XML. Les paramètres de la procédure peuvent être des scalaires, des nombres, des dates, des *strings*, ...

Voyons en premier un exemple d'une requête:

```
POST/RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host:ica118pc45.epfl.ch
Content-Type: text/xml
Content-length:181

<?xml version="1.0"?>
<methodCall>
  <methodName>doSomething</methodName>
  <params>
    <param>
      <value><i4>1234</i4></value>
      <array>
        <data>
          <value><string>little</string></value>
```

```
        <value><string>test</string></value>
      </data>
    </array>
  </param>
</params>
</methodCall>
```

Examinons maintenant cette requête, voyons en premier l'entête. Un utilisateur (User-Agent) doit être spécifié de même qu'un hôte (Host). Le format du contenu (Content-Type) est text/xml. La longueur du contenu (Content-Length) doit être spécifiée et correcte.

En ce qui concerne la partie en xml est constituée d'une seule structure `<methodCall>`. Cette dernière le tag `<methodName>` contenant le nom de la méthode appelée. Le nom spécifié peut être composé de caractères (majuscule ou minuscule), des chiffres (0-9), des points, des soulignés et des *slash*. C'est le serveur qui décide à quoi se réfère le nom ainsi donné. Il peut s'agir du nom d'un fichier contenant un script, le nom d'une cellule dans une base de donnée ou encore le chemin d'accès à un fichier.

Voyons maintenant la réponse:

```
HTTP/1.1 200 OK
CONNECTION: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 11 Feb 2000 08:25:46 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version=''1.0''?>
<methodResponse>
  <params>
    <param>
      <value><boolean>1</boolean></value>
    </param>
  </params>
</methodResponse>
```

Si l'appel à la procédure a des paramètres, `<methodCall>` doit contenir un `<params>` qui contient un certain nombre de `<param>` qui ont chacun une `<value>`.

Nous verrons dans le chapitre '*outils de développement* les librairies disponibles pour XML-RPC.

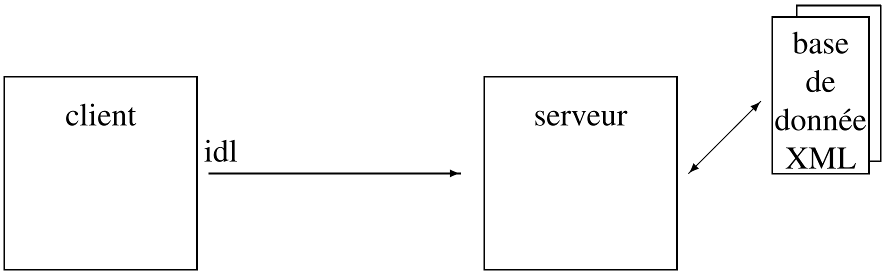
3.1.2 XML en collaboration avec CORBA

XML ne peut en aucune façon à ce jour remplacer CORBA, mais comme nous l'avons déjà remarqué, XML est un langage très flexible dans la description de données et de structures. Nous allons essayer de voir ce que cela pourrait apporter à CORBA.

XML se profile vite comme un outil viable pour l'échange de données entre des systèmes incompatibles comme CORBA et COM. Les grands constructeurs d'ORB ont déjà annoncé leur volonté d'intégrer un "niveau de communication XML" dans leurs produits. L'OMG publiera en février 2000 un standard sur l'utilisation de XML avec CORBA. Cela permettra à d'autres *middlewares* d'utiliser les mêmes spécifications, donc de cohabiter avec CORBA.

Voyons maintenant comment ont peut combiner XML et CORBA.

IDL est un document qui décrit les interfaces dans CORBA. Un fichier IDL ressemble fortement à du code C. Un fichier IDL peut être exprimé sous forme de document XML. De même, il est possible d'exprimer l'*interface repository* sous forme de fichier XML et le stocker dans une base de donnée. Il est libre au constructeur d'implémenter l'*interface repository* comme il le désir. La figure ci-dessous illustre ce principe:



3.1.3 Avantages et désavantages

Voyons en premier les avantages de xml.

XML est facile

XML est très facile à lire et à écrire; cela fut un des facteurs de l'acceptation rapide de XML. Tout un chacun peut créer un message XML avec un simple éditeur de texte. Il existe maintenant aussi des outils (*parsers*) qui permettent de créer facilement et de vérifier la syntaxe des fichiers XML corrects. Il est aussi possible de générer du XML avec n'importe quel langage de programmation. XML étant lisible par l'homme, cela rend le débugage plus simple.

XML est extensible

XML fourni un mécanisme relativement simple d'extension. L'exemple suivant illustre ce fait:

```
<order orderno="11234">
  <customer custno="4462" />
  <item itemno="5566" />
  <item itemno="2468" />
</order>
```

Dans ce fragment de code, nous voyons que le client 4462 commande les objets 5566 et 2468. Si celui qui envoie la commande et celui qui la reçoit comprennent la signification de ce bout de code, tout fonctionne bien. En revanche, que ce passe-t-il si l'auteur de ce message décide de rajouter une information supplémentaire? Supposons que le message qu'il envoie soit le suivant:

```
<order orderno="11234" transid="99321">
  <customer custno="4462" />
  <item itemno="5566" />
  <item itemno="2468" />
</order>
```

Il peut alors se poser plusieurs problèmes à la réception. En premier, le destinataire peut ou ne peut pas permettre l'addition d'attribut ou de nouveaux éléments. Si le destinataire interprète la présence de ce nouveau attribut comme une erreur, alors la requête échouera. Pour résoudre ce problème, certaines nouvelles technologies (comme SOAP) permettent de définir le vocabulaire XML de deux façons: soit ouvert ou fermé. Un vocabulaire fermé ne peut pas être étendu plus que ce qui est contenu dans le schéma de base (DTD). Un vocabulaire ouvert peut être étendu et c'est l'application qui reçoit la requête qui décide comment interpréter les attributs ou les éléments ajoutés. Dans la majeure partie des cas, ce qui n'est pas reconnu est ignoré.

Supposons maintenant que le message précédent est une partie d'un vocabulaire ouvert, il est alors possible d'ajouter l'attribut `transid`. Que ce passe-t-il si le destinataire voulait aussi ajouter un attribut `transid` pour l'interaction avec une base de données par exemple? Dans ce cas, l'attribut serait alors mal interprété. Pour résoudre ce problème, le W3C a ajouté des *namespaces* à XML. Leur utilisation permet d'enlever toute ambiguïté à l'attribut `transid`. Voyons ce que cela donne:

```
<order orderno="11234"
       xmlns:fin="http://money.org/XML/ns"
       fin:transid="99321">
  <customer custno="4462" />
  <item itemno="5566" />
  <item itemno="2468" />
</order>
```

Quand le destinataire traite ce fragment de code XML, il détecte que l'attribut `transid` est complété par le namespace suivant: `http://www.money.org/XML/ns` et que ce n'est pas le même que l'attribut `transid` utilisé pour la base de données. Les deux attributs `transid` peuvent donc apparaître dans la même requête sans confusion:

```
<order orderno="11234"
       xmlns:fin="http://money.org/XML/ns"
       xmlns:db="urn:xmlpc:XMLTransactions"
       fin:transid="99321"
       db:transid="12345">
  <customer custno="4462" />
  <item itemno="5566" />
  <item itemno="2468" />
</order>
```

XML permet de résoudre certains problèmes d'interopérabilité

Adopter XML ne résout pas tous les problèmes d'interopérabilité; pas toutes les organisations utiliseraient le même vocabulaire pour représenter la même information. Il existe plusieurs organismes qui tentent de standardiser le vocabulaire; mais cela ne garantira pas à 100 pour-cents son uniformité.

Heureusement, cette lacune dans l'utilisation d'un standard donné peut être résolue grâce au XML. En présence de deux formes de vocabulaire proches, il est très facile de créer une application qui transformerait le requêtes écrites à l'aide d'un vocabulaire "A" en une requête écrite à l'aide d'un vocabulaire "B". Nous avons vu au début du chapitre trois façons d'exprimer la même requête. Les trois syntaxes sont différentes mais il est possible de créer un tarducteur qui pourrait passer d'un langage à un autre.

Désavantages de XML

Voyons maintenant les désavantages de XML.

Limitation des les formes de données

Une des limitations souvent citée de XML est le fait qu'il s'agit d'un format ASCII et qu'il n'offre pas de solution native pour des types de données complexes ou binaires. Pour résoudre ce problème, plusieurs propositions ont été soumises au W3C pour lier de l'information binaire au XML.

Quand utiliser XML

CORBA est un middleware créé initialement pour des applications distribuées d'une unique organisation. Comme nous l'avons vu avant, XML a été conçu pour une utilisation inter-organisation. XML est nettement plus flexible, mais aussi moins efficace que CORBA.

Tableaux comparatifs

Voyons maintenant un tableau comparant les possibilités offertes par une solution basée sur XML et CORBA.

Capacité	Description	XML	CORBA
Indépendance du la plateforme	Est-ce que le client et le serveur peuvent être sur des architectures différentes?	Oui	Oui
Traiter des documents "*human readable*"	Les documents peuvent-ils être lus et édités?	Oui	Non
Traiter des données qui ne sont pas des *strings*	Y'a-t-il la possibilité d'utiliser des nombres entier, des nombres à virgules flottantes, des booléens, etc.?	Non, mais chaque DTD peut définir des types de données	Oui (*string, short, int, float, double, boolean, byte, char, structs*)
	Transparence, sécurité, transactions, etc.	Non	Oui
Orienté objet	Les données peuvent-elles être objet (structure de données + comportement)?	Non	Partiellement
Langages de programation	Peut-on implémenter ces solutions avec divers langages de programmation?	Oui, tous les langages qui manipulent des strings peuvent travailler avec XML.	Non, les langages disponibles sont: Java, C++, Smaltalk, C, COBOL

Dans des systèmes distribués, les données transmises d'un ordinateur à un autre ne sont généralement pas stockés et ne sont probablement jamais lues par des hommes. XML est parfaitement fonctionnel, mais dans les système où des applications échangent des informations entre elles sans que ces interactions aient besoin d'être *human-readable*, XML n'est pas la solution optimale. Utiliser XML dans ce cas est évidement moins efficace que les formats binaires natifs. Ce qui n'est pas tout à fait exact, il est possible de compresser les fichier XML avant de les envoyer et de les décompresser ensuite à la réception. Historiquement, les formats de web sont tous *human readable*. Cela amène un grand bienfait qui est la facilité de débuggage. De plus les apllication de ce type sont beaucoup plus robuste. Dans le cas de XML, c'est aussi un des facteur qui a permis le développement de nombreux outils.

Dans le cas d'un réseau comme internet, des RPC basé sur XML ont un avenir prometteur à cause de leur simplicité. Ils offrent la possibilité de remplacer des solutions actuelles comme les scripts CGI par exemple. XML est de loin plus simple que CORBA, mais est aussi moins complet.

3.2 XML pour décrire un système distribué

XML est un langage ayant une structure que nous pouvons définir, est-il alors possible de l'utiliser pour décrire un système distribué?

Nous allons étudier ici une application particulière de XML: la description d'un système distribué. XML étant un langage très flexible, il y a toutes les raisons de penser qu'en définissant un DTD approprié, il serait possible de décrire efficacement un système distribué. À la section suivante, nous verrons une application précise des efforts accordés ici à la description d'un système: le déploiement.

Un système distribué étant très complexe, nous utiliserons donc un modèle simplifié. Un ordinateur sera composé d'un nom (*name*), d'une adresse (*address*), d'un environnement (*environement*) et aura plusieurs services (*service*) décris par une architecture (*architecture*), par un nom (*name*) et par une version (*version*). L'exemple suivant nous montre quel genre de fichier nous obtenons alors:

```
<computer>
 <computerName>in1sun21</computerName>
 <computerAddress>128.178.??.??</computerAddress>
 <environement>solaris</environement>
 <service>
  <architecture>
   <architectureName>ftp</architectureName>
   <architectureVersion>wu-ftp</architectureVersion>
  </architecture>
 </service>
</computer>
```

Maintenant que nous avons les informations que nous voulons voir apparaître dans notre fichier XML, nous pouvons définir un DTD. Ce DTD nous pernettra de valider les informations à l'aide d'un *parser*. Il est ensuite possible de manipuler ce fichier à l'aide de n'importe quel langage de programmation. Voyons à quoi pourrait ressembler ce DTD:

```
<!DOCTYPE computer [
 <!ELEMENT computer       (computerName, computerAddress, ser-
vice+) >
 <!ELEMENT computerName       (#PCDATA) >
 <!ELEMENT computerAddress    (#PCDATA) >
 <!ELEMENT service (architecture+) >
 <!ELEMENT architecture (architectureName, architectureVersion) >
 <!ELEMENT architectureName (#PCDATA) >
 <!ELEMENT architectureVersion  (#PCDATA) >
]>
```

Il est évident que cet exemple ne reflète pas la réalité, mais cela nous donne une idée des possibilités de XML pour la déscription de systèmes complexes.

Il n'existe aucun standard ni aucune proposition à ce sujet, mais cet exemple me permet d'affirmer que c'est possible.

3.3 XML comme outil de déploiement

Nous sommes arrivés, dans la section précédente, que XML est un moyen de décrire un système distribué. Essayons maintenant de voir dans quel mesure ces données sont réutilisable pour une utilisation future. Dans quelle mesure est-il possible d'utiliser XML comme outil de déploiement?

Le fichier XML de la section précédente nous permet de décrire de manière simple et structurée un parc informatique; essayons maintenant de décomposer le processus de déploiement d'une application.

La première étape est la validation de notre fichier XML décrivant le parc informatique. Pour cela, nous aurons recours à un *parser* qui utilisera le DTD que nous avons créé. Cette étape ne pose aucun problème, il existe à ce jour plusieurs *parsers* qui fonctionnent très bien; nous en verrons dans le chapitre "Outils de développement".

La seconde étape est la création d'un fichier XML décrivant l'application que nous désirons installer. Les informations inportantes sont: le nom des ordinateurs sur lesquels ont veut installer l'application, l'environnement sur lequel tourne notre application, le nom et la version de l'application, le répertoire dans lequel on veut la mettre, et ou elle se trouve actuellement (je suppose qu'elle est sur la machine qui execute notre programme fictif de dépoiement). Imaginons que nous désirions installer l'application 'HelloWorld' sur la machine in1sun21. Ici encore, je simplifie beaucoup, le but étant de voir si cela est possible et non de fournir une solution réelle. Le fichier XML aurait l'allure suivante:

```
<application>
 <applicationName>HelloWorld</applicationName>
 <applicationVersion>1.0</applicationVersion>
 <computerName>in1sun21</computerName>
 <environement>solaris</environement>
 <installDirectory>/home/pjost/bin</installDirectory>
 <initialDirectory>/root/bin</initialDirectory>
</application>
```

Définissons maintenant le DTD:

```
<!DOCTYPE application [
 <!ELEMENT application (applicationName, applicationVersion, com-
puterName, environement, installDirectoy, initialDirectoy) >
 <!ELEMENT applicationName (#PCDATA) >
 <!ELEMENT applicationVersion  (#PCDATA) >
 <!ELEMENT computerName (#PCDATA) >
 <!ELEMENT environement (#PCDATA) >
 <!ELEMENT installDirectory (#PCDATA) >
 <!ELEMENT initialDirectory  (#PCDATA) >
]>
```

La troisième étape est la vérification de la syntaxe de du fichier XML que nous venons de créer.

Maintenant que nous avons deux fichier valides, il nous faut passer aux vérifications. Pour cela il existe des librairies de programmation qui permettent d'accéder au données contenues dans des fichiers XML. Après avoir extrait les informations contenues dans le second fichier, il nous faut faire quelques vérifications:

– L'application n'est elle pas déjà installée sur la machine in1sun21?

– L'application est elle installable: les environnements sont-ils les mêmes?

Pendant la cinquième étape, notre programme doit simplement créer une procédure d'installation en fonction des données contenues dans les fichier XML. Je suppose ici que le programme a les droits nécessaires.

Pour la dernière étape, je suppose que l'installation de notre nouveau programme c'est bien passé. Il reste donc à modifier le fichier XML qui décrit le parc informatique; c'est une oppération facilement réalisable avec les librairies existantes.

Nous avons donc montré, malgré la simplicité de l'exemple, qu'il est possible d'utiliser XML comme outil de déploiement.

3.4 XML et l'état d'un système distribué

Capturer l'état à un instant donné d'un système distribué n'est pas chose facile, stocker les résultats non plus. Nous allons discuter ici de la possiblité d'utiliser XML comme moyen de stckage des données.

Le format XML est auto-descriptif, nous pouvons donc en extraire un schéma et vérifier la validité des informations; cet aspect de XML est intéressant si nous voulons enregistrer des données pour une utilisation future.

Quand des informations sont archivées sous forme binaire, il y a des limitations sur le genre de changements que l'on peut faire si l'on veut assurer la compatibilité avec des versions antérieures. Comme dans le cas de XML comme middleware, l'aspect de l'interopérabilité est très intéressante dans ce cas aussi; de même que l'extensibilité de XML.

Pour distribuer de l'information entre plusieurs systèmes et organisations, nous avons besoin d'un format de données stable, de même qu'un protocole compris par tous les systèmes. Les formats de fichiers doivent être *machine readable*, mais c'est un avantage non négligeable pour une utilisation à long terme d'avoir des fichiers *human readable*.

Nous remarquons, de nos jours, qu'avec les années certains types de fichier tombent en désuetude, les bits peuvent être préservés mais un fichier ne sert a rien si on ne peu plus s'en servir. Qui n'a jamais eu des problèmes pour ouvir un ancien fichier d'un certain traitement de texte très rependu? Les formats *human readable* et auto-déscriptifs comme XML permettent de résoudre ce problème. Comme nous l'avons déjà vu précédemment, XML est un langage extensible. XML est donc un format idéal pour le stckage de données à long terme. Un argument souvent cité par les opposants est qu'en utilisant un format de fichier XML, l'information prend plus de place. Mais, plus nous avençons vers l'avenir, plus le prix des supports de stckage diminue.

Il est très important que pour le stockage de l'information, les formats de fichier ne soient pas proproétaires pour assurer la possibilité d'une réutilisation future. XML répond à ce critère.

Chapitre 4

Autres applications de XML

Ce chapitre est un survol de quelques autres applications intéressantes.

4.1 Mathematical Markup Language (MathML)

Cela fait des années que les mathématiciens essaient de mettre leurs équations *online*. Actuellement, la meilleure solution est d'utiliser TeX-to-HTML qui convertit les équations en images. Cette solution nécessite beaucoup de ressources et n'est pas pleinement satisfaisante. Le W3C à développé MathML, un langage de formatage pour les équations et les formules mathématiques.

MathML décrit des formules mathématiques, il est donc possible de les afficher dans un browser avec une feuille de style XSL. Cette solution est plus rapide que de charger une image par formule.

Il existe déjà des éditeurs graphiques pour MathML. Maple, par exemple, peut importer, exporter des fichier MathML. Techexplorer d'IBM permet d'afficher du MathML. Il existe aussi une collection d'*applets* (et d'API) Java qui affichent des équations en notation standard ou sous forme de graphes.

XML nous offre ici une application vraiment intéressante, nous pouvons écrire des équations en MathML avec des éditeurs graphiques, les utiliser dans des documents classiques, les importer dans notre programme de mathématique favoris, les publier et même développer nos propres *applets*.

4.2 Entity

Entity est une application relativement amusante, XML est utilisé ici pour décrire une interface graphique et des petits programmes. Entity est un *parser* qui vérifie la validité du document et génère l'interface graphique en utilisant GTK+. Pour mieux comprendre, regardons un exemple du tutoriel d'Entity:

```
#!/usr/local/bin/entity

<object>
  <window ondelete="entity:exit" title="Clock">
    <valign>
      <object dragable="true">
        <label font="10x20" name="time" text = "00:00:00"/>
        <timer interval = "1000" action="update_clock"/>
        <perl>
          sub update_clock
```

```
            {
              my ($sec, $min, $hour, $mday) = localtime (time);

              my $str = sprintf ("%02d:%02d:%02d", $hour, $min, $sec);

              my $node = enode("label.time");
              $node->attrib("text" => $str);
            }
          update_clock ();
        </perl>
      </object>
    </valign>
  </window>
</object>
```

Cela nous donne la fenêtre suivante:

Cet exemple nous montre bien qu'XML est un langage de formatage qui permet de décrire à peu près n'importe quoi.

4.3 XML software Autoupdate (XSA)

XSA est une application crée par un étudiant du département d'informatique de l'université d'Oslo pour lui aider à maintenir à jour les mises-à-jour des programmes référencés sur plusieurs sites. Ce procédé est maintenant utilisé par trois sites qui référencent des applications XML: *XML.com Ressource Guide*, *Free XML Software* et *xmltree.com*.

Voici un petit exemple qui décrit la mise à jour du *parser* xmlproc. Après l'execution du programme sur le client, toutes les pages concernant ce produit auront les informations sur la nouvelle version du produit. Cela est très utile dans le cas où il y a beaucoup de site qui le référencent.

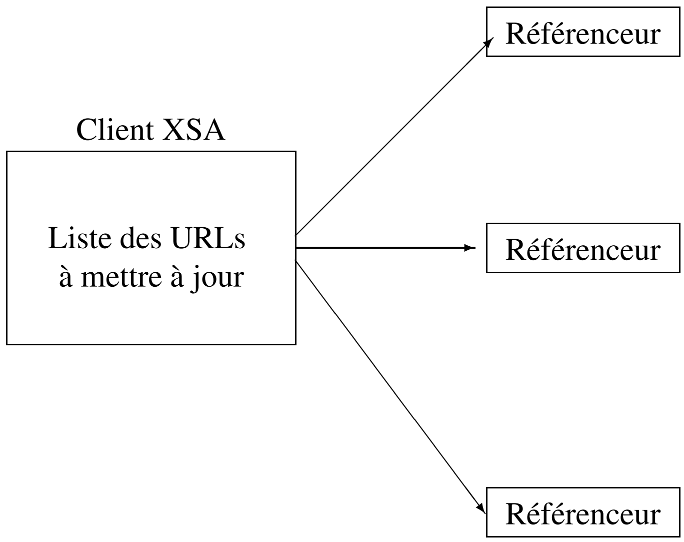
```
<?xml version="1.0"?>
 <!DOCTYPE xsa PUBLIC "-//LM Garshol//DTD XML Software Autoup-
date 1.0//EN//XML"
  "http://birk105.studby.uio.no/www_work/xsa/xsa.dtd">

 <xsa>
  <vendor>
   <name>Lars Marius Garshol</name>
   <email>larsga@ifi.uio.no</email>
   <url>http://www.stud.ifi.uio.no/~larsga/</url>
  </vendor>

  <product id="xmlproc">
   <name>xmlproc</name>
   <version>0.61</version>
   <last-release>19990419</last-release>
   <info-url>
```

```
     http://www.stud.ifi.uio.no/~larsga/download/python/xml/xmlproc.html
   </info-url>
   <changes>
    The parser is now even faster, especially when validating.
   </changes>
 </product>
</xsa>
```

Un programme écrit en Java compare les informations fournies à celle déjà présentes sur le site et les envoie si elle sont plus récentes. La liste des URL à mettre à jour doit encore être fournie. Le shéma suivant illustre le foctionnement de cette application:



Chapitre 5

Outils de développement

Dans ce chapitre, nous allons voir quelques outils de développement disponible actuellement.

5.1 *Parsers*

Un *parser* un un programme qui traite les fichier XML pour une utilisation future; dans la majeure partie des cas, un *parser* est une sorte de pré-processeur qui vérifie la validité de la syntaxe du document. Certains *parsers* permettent d'afficher les documents sous une forme ou une autre.

Tous les *parsers* que nous allons étudier ici sont d'utilisation libre.

5.1.1 XML for Java (xml4j)

Xml4j est un *parser* développé par IBM, comme son nom l'indique, il est écrit en Java. Il est capable de valider un document XML, mais aussi de générer un document XML à partir d'un DTD. Il est aussi capable de générer une vue d'ensemble sous forme d'arbre d'un document XML.

Site Web: http://www.alphaworks.ibm.com/tech/xml4j

5.1.2 XML for C++ (xml4c)

Comme le *parser* ci-dessus, xml4c a été developpé par IBM. Ses possibilités sont les mêmes que pour xml4j. Il s'agit d'un source c++ portable fonctionnant sous AIX, Linux, Solaris, Win32, HP-UX, OS/2 et OS/390.

Site Web: http://www.alphaworks.ibm.com/tech/xml4c

5.2 Librairies

Les librairies de développement ne sont pas toujours utiles, car il est possible de manipuler des fichiers XML avec presque tous les langages de programmation.

5.2.1 XML-RPC

Il existe à ce jour des librairies de développement pour plusieurs langages, le tableau suivant est un résumé:

Langage	**Client**	**Serveur**	**Adresse Internet**
Python	Oui	Non	http://www.pythonware.com/products/xmlrpc/
Java	Oui	Oui	http://helma.at/hannes/xmlrpc/ ou http://www.stonecottage.com/josh/rpcClient.html (Client seulement)
Perl	Oui	Oui	http://bitsko.slc.ut.us/ ken/xml-rpc/
Active sever page (ASP)	Oui	Oui	http://www.wc.cc.va.us/dtod/XMLRPC/
COM	Oui	Non	http://redmonk.editthispage.com/xmlrpccom/
PHP	Oui	Oui	http://usefulinc.com/xmlrpc/php.html

5.2.2 LT XML

LT XML est une collection d'outils de développement incluant une API pour c. LT XML est disponible pour UNIX et WIN32. Ces outils incluent la recherche, l'extraction, la recherche et le tri d'information. LT XML fournit deux façon de voir les fichier XML: comme *stream* d'éléments formatés ou sous forme de texte.

LT XML a été développé par le *Language Technology Group* de l'université d'Edinburgh.

Site Web: http://www.ltg.ed.ac.uk/software/xml/index.htmla

Chapitre 6

Conclusion

6.1 XML est bien adapté pour la description de données

Les exemples cités dans ce rapport sont nombreux et variés; ils utilisent tous XML d'une façon différente, avec un point commun: XML sert a décrire les informations. De plus XML s'adapte très bien a chacun de ces cas. Tous ces faits nous amènent à la conclusion que XML est très bien adapté pour la description des données. Si on regarde chacun de ces exemples en détail, on se rend vite compte que la description est très naturelle, bien organisée mais reste néanmoins très simple.

XML est *human readable*, mais pas seulement l'homme peut bénéficier de la clareté de la systaxe dans ce langage, il existe maintenant de nombreux outils capables de traiter l'information sous forme de XML, d'en extraire les informations, d'en insérer et de les manipuler. Donc XML s'applique bien pour la déscription des infomations mais aussi pour leur traitement informatique.

6.2 XML + HTTP est un middleware

Plusieurs solutions ont été étudiées dans ce rapport et nous avons vu que chacune nous fournissait un *middleware* viable. Cela est la première conclusion, une des plus importantes de cette analyse. La seconde conclusion, que nous pouvons tirer de ces quelques application étudiées, est qu'en plus de fournir un *middleware* intéressant, les solutions basées sur XML répondent à des besoins actuels:

– La communication inter-organisation. En effet une solution XML + HTTP utilise pour les transmission le port 80, il s'agit généralement du seul port qui ne soit pas bloqué par les *firewalls* en place à l'entrée des organisations.

– L'utilisation de structures déjà en place. Ce qui est un atout non négligeable.

– Un *middleware* simple. XML + HTTP nous donne un *middleware* plus simple que CORBA, mais aussi moins complet.

– Une solution robuste. Comme tous les standards du web, XML est *human readable*, donc le débuggage est plus simple et les applications plus robustes.

– Indépendant de la plateforme et du langage. XML est indépendant de la plateforme, de plus étant un fichier ASCII, il est possible de le manipuler avec tous les langages qui sont capables de traiter des *strings*. Cela a permis le développement fulgurant de nombreuses librairies.

6.3 XML + HTTP de peut pas remplacer CORBA

Au vu du paragraphe ci-dessus, on pourrait penser que XML est une solution miracle et révolutionnaire en tout point. Ce n'est pas le cas. XML est effectivement beaucoup plus simple et facile d'implémentation que CORBA mais est aussi moins efficace. CORBA fournit des service plus complets comme la sécurité et l'autentification des utilisateurs. CORBA est une technologie beaucoup plus mature qui bénéficie de nombreuses années d'expérience.

6.4 Suites à ce projet

Ce projet a vraiment été très instructif pour moi, j'ai eu l'occasion de me familariser avec un langage que je ne connaissait pas: XML. Les systèmes distribués étaient aussi nouveaux pour moi. Ce projet est une étude de la faisabilité, il apporte beaucoup de nouvelles notions. Les suites possible à ce projet pourraient être:

– La création d'un programme capable de traduire les requêtes XML d'un vocabulaire en un autre. Un extension à cela pourrait être l'implémentation d'un serveur capable de reconnaître les différents vocabulaires et de traiter les requêtes.

– L'implémentation de l'application vue dans la section *XML comme outil de déploiement*. Ce chapitre pourrait constituer le cahier des charges de ce projet.

– La création d'une nouvelle librairie pour une application comme XML-RPC. Il n'existe pas encore de librairie pour le langage c.

– La nouvelle génération de noyaux linux qui sortira dans quelques mois (2.4) aura un serveur HTTP minimal (GET, POST) inclu. Je pense qu'il pourrait être très intéressant de créer un *middleware* utilisant XML comme message se basant sur ce serveur.

Chapitre 7

Glossaire

Component Object Model (COM)
Modèle d'objets de Microsoft.

Common Gateway Interface (CGI)
API d'extension des serveurs HTTP, il s'agit d'une interface entre le serveur Web et d'autres processus sur la machine.

Common Data Representation (CDR)
Format de représentation des données.

Common Object Request Broker Architecture (CORBA)
Architecture du bus d'objets répartis de l'OMG.

Distributed Component Object Model (DCOM)
Version distribuée du modèle d'objet COM.

Document Object Model (DOM)
Le protocole DOM transforme un document XML en une collection d'objets dans votre programme que vous pouvez ensuite manipuler. Vous pouvez modifier les données, les enlever ou en insérer de nouvelles.

Document Type Definition (DTD)
définit le langage dans lequel sera écrit un document qui devra respecter les règles sémantiques de la DTD associée pour être valide.

HyperText Markup Language (HTML)
HTML est un langage de formatage pour créer des pages Web. HTML définit l'aspect de la page, les polices de caractère, les images à afficher et des liens vers d'autres pages.

Internet Inter-ORB Protocol (IIOP)
Protocole de communication entre bus CORBA sur le réseau Internet.

Namespace
Une façon de définir sans ambiguïté les éléments et les attributs dans un document XML en les associant avec des URI spécifiques.

Object Database Management Group (ODMG)
Consortium de standardisation des bases de données orientées objet.

Object Management Group(OMG)
Consortium pour promouvoir les technologies objets.

Object Request Broker (ORB)
Négociateur de requêtes pour les objets CORBA.

Request For Comments (RFC)
Proposition d'un standard.

Standard Generalized Markup Language (SGML)
Standard de description de documents structurés.

Secure Socket Layer (SSL)

Couche de communication à base de sockets sécurisées.

Uniform Resource Identifier (URI)

Une URI est une chaîne de caractères compacte qui identifie une ressource abstraite ou physique. Une définition plus complète peut être trouvée à l'adresse suivante: http://www.ics.uci.edu/pub/ietf/uri/rfc2396.txt

World Wide Web Consortium (W3C)

Consortium de standardisation du WWW.

Extensible Style Langage (XSL)

Spécifie la présentation et l'apparence d'un document XML.

Bibliographie

[1] Ron Ben-Natan. *Objects on the web*. McGraw-Hill, 1997.

[2] D. Box. Soap: Simple object access protocol. Internet-draft, Microsoft, Novembre 1999.

[3] Dave Carlson. Component interoperability with xml. Technical report, Ontogenics Corp, 1998.

[4] Rachel Chalmers. Does xml need corba? *COMPUTERWIRE*, Septembre 1999. http://www.computerwire.com/cgi-bin/show?article=v2,m,e,h,n,x,e,r,f,m,d,c.

[5] David Chappell. Simple object access protocol. Technical report, Micosoft, Septembre 1999. http://mdsn.microsoft.com/workshop/XML/general/SOAP_White_Paper.asp.

[6] George M. Doss. *CORBA developer's guide with XML*. Wordware Publishing, 1999.

[7] Peter Flynn. Frequently asked questions about the extensible markup language. Technical report, W3C, Juin 1999. http://www.ucc.ie/xml/.

[8] Dana Gardner. Pas de titre. *Infoworld*, Septempbre 1999.

[9] Dana Gardner. Xml promise simple object interoperability. *InfoWorld Electric*, Septembre 1999. http://www.infoworld.com/cgi-bin/displayStory.pl?990927.hnxml.htm.

[10] Lars Marius Garshol. What can we do with xml. Technical report, Université d'Oslo, 1998. http://www.stud.ifi.uio.no/ lmariusg/download/artikler/fin sgml 98.html.

[11] Philippe Merle Jean-Marc Geib, Christophe Gransart. *CORBA, des concepts à la pratique*. InterEditions, masson edition, 1997.

[12] Jon Lavinder John Tigue. Webbroker: Distributed object communication on the web. Technical report, DataChanel, 1998. http://www.w3c.org/TR/1998/NOTE-webbroker/.

[13] World Wide Web Journal. *XML Principles, Tools and Techniques*. Dale Dougherty, o'reilly edition, 1997.

[14] Benoît Marchal. Xml in a three-tier environmen. *www.developer.com*, Septembr 1998. http://www.developer.com/journal/techfocus/092898_xml3tier.html.

[15] Mike Reinertsen Mark Elenko. Xml corba. *OMG*, 1999. http://www.omg.org/library/adt.html.

[16] OMG. Corba and xml: conflict or cooperation. Technical report, OMG, 1999.

[17] OMG. History of corba. Technical report, OMG, 1999. http://www.omg.org/corba/corbahistory.html.

[18] Bryan Pfaffenberger. *Web publishing with XML in 6 easy steps*. AP Professional, academic press edition, 1999.

[19] Alan Radding. Middleware evolution. *www.informationweek.com*, Juillet 1998. http://www.informationweek.com/690/90iumid.htm.

[20] Adam Rifkin Rohit Khare. Capturing the state of distributed systems whith xml. *World Wide Web Journal Special Issue on XML*, 2:207–218, Fall 1997. http://www.cs.caltech.edu/ adam/papers/xml/xml-for-archiving.html.

[21] Simon St.Laurent. *XML a primer*. MIS:Press, 1998.

[22] SUN. A quick introduction to xml. Technical report, SUN, 1998. http://java.sun.com/XML/docs/tutorial/index.html.

[23] W3C. Mathematical markup language. Technical report, W3C, Février 1998. http://www.w3.org/TR/PR-math/.

[24] W3C. Namespaces in xml. Technical report, W3C, Janvier 1999. http://www.w3.org/TR/REC-xml-names/.

[25] Dave Winer. Xml-rpc specification. Technical report, XML-RPC, 1998. http://www.XmlRpc.com.

[26] Dave Winer. Xml-rpc specification. Technical report, XML-RPC, Septembre 1999. http://www.xml-rpc.com/spec.

[27] Wylie Wong. Soap could slip up microsoft rivals. *CNET*, 1999. http://www.omg.org/xml/hpcwire.html.