Table des matières

Applications Web

Avertissement : ceci est une version en cours de révision du document. Je vous l'ai fait imprimer plus rapidement pour faciliter votre étude en vue de l'examen.

Ce document porte plus spécifiquement sur les applications Web. Nous y examinerons, entre autres choses :

- ce qu'est la technologie `XML` et ce que sont quelques-unes de ses applications directes;
- ce que sont les stratégies `DOM` et `SAX` pour consommer des documents `XML`;
- ce qu'est un Servlet;
- ce qu'est une page `JSP`; et
- comment déployer une application Web simple dont le volet client s'affiche dans un fureteur et dont le volet serveur est un servlet. Le servlet en question sera déployé sur le populaire serveur Web Tomcat de Apache.

Les exemples de code seront en Java. Nous présumerons donc une connaissance au moins superficielle de ce langage chez le lectorat, et nous présumerons que vous n'hésiterez pas à poser des questions à votre professeur si le besoin s'en fait sentir.

Nous présumerons aussi d'une compréhension élémentaire des idées de répertoire et de variables d'environnement. Si vous avez de la difficulté avec ces éléments de contenu, n'hésitez pas à poser des questions à votre professeur.

Le sujet à explorer est immense et en mouvance constante. Aucun document papier ne pourra espérer en faire le tour. Il doit donc être clair que la présente se veut descriptive, ouvrant des pistes, illustrant des concepts et démontrant certaines applications simples, mais que le lectorat devra par la suite faire preuve d'éveil et de curiosité pour aller chercher les éléments de connaissance qui lui semblent pertinents en su de ce qui sera exploré ici et dans les documents subséquents de cette série.

XML et l'interopérabilité

Le monde de l'informatique contemporaine est coincé entre un souci croissant envers la sécurité et un besoin d'ouverture sur le monde. La connectivité et l'interopérabilité entre les plateformes, les systèmes et les entreprises sont parmi les principaux pôles d'intérêt de toute entreprise confrontée à l'informatique depuis l'avènement d'Internet comme point focal du commerce et plateforme à part entière.

Le **grand dictionnaire terminologique**[1] (GDT) définit la **connectivité** comme les « [m]oyens techniques dont dispose une personne ou un organisme lui permettant de se relier à un réseau ». Aujourd'hui, la connectivité définie en ces termes ne devrait plus poser de réels problèmes en occident dû à l'ubiquité de l'accès à Internet.

En retour, le GDT définit l'**interopérabilité** comme la « [c]apacité que possèdent des systèmes informatiques hétérogènes à fonctionner conjointement, grâce à l'utilisation de langages et de protocoles communs, et à donner accès à leurs ressources de façon réciproque. ». Dans un cours portant sur les systèmes répartis, l'interopérabilité est une préoccupation incontournable, sinon la principale préoccupation des étudiant(e)s comme du professeur.

Le GDT ajoute quelques notes supplémentaires au sujet de l'interopérabilité, notes qu'il convient de commenter avant d'aller plus loin :

- on y indique que « [l]'interopérabilité implique qu'un programme tournant sur un système ouvert fonctionnera également sur un autre système », ***ce qui est faux***. L'interopérabilité est une démarche qui vise à faciliter la communication de modules hétérogènes sur le plan technologique, ce qui n'implique pas tant la portabilité des sources ou des binaires de ces modules que leur capacité de transiger;
- on y indique que « [l]'interopérabilité a besoin de plus qu'une bonne connectivité technique puisqu'elle nécessite l'utilisation d'éléments comme des interfaces de programmation et des formats de données standardisés », ce qui est absolument vrai. À moins que chaque paire d'homologues dans un système donné ne transige avec exactement la même langue que celle utilisée par toutes les autres paires d'homologues du même système[2];
- enfin, on y indique que « [l]'interopérabilité définie ici est l'interopérabilité technique, mais il en existe d'autres types dont l'interopérabilité sémantique qui est associée à un mode de description de l'information contenue dans une base de données (cette description forme les métadonnées) ».Cette dernière note est vraie, mais trop contraignante : les métadonnées sont la clé de la majorité des protocoles et des modèles pertinents au moment d'écrire ces lignes. Il est inutile de contraindre les métadonnées aux seules BD.

[1] Voir http://www.granddictionnaire.com/

[2] Ce qui se produit à l'occasion, par exemple quand tous les homologues échangent à travers RMI de Java et sont compilés avec précisément la même version du JDK ou quand tous les homologues ont été conçus à l'aide de la même version du CLR de .NET et transigent par *Remoting*. En général, il est sain de ne pas compter sur une telle homogénéité technologique; la variété comporte plus d'avantages que d'inconvénients.

Atteindre l'interopérabilité

Historiquement, l'atteinte de l'interopérabilité a été abordée à l'aide de plusieurs technologies :

- les ***sockets***, qui ont servi à cacher la strate de transport des données de manière à assurer le découpage des messages à transmettre sous forme d'unités nommées **paquets**, la livraison des paquets à l'aide d'un **protocole de livraison** tel `IP` (pour *Internet Protocol*) et de manière à assurer l'intégrité et l'ordonnancement des paquets à l'aide d'un **protocole de transport**. Les protocoles de transport les plus connus sont `UDP` (*User Datagram Protocol*), qui livre les paquets sous forme d'enregistrements de taille connue *a priori* sans exiger que les homologues d'une relation `C/S` ne demeurent connectés l'un à l'autre, et `TCP` (*Transfer Control Protocol*) qui permet le transfert bidirectionnel des flux de données entre deux homologues connectés logiquement l'un à l'autre;
- des **standards d'interopérabilité binaire** tels que `RPC/XDR`, `RPC/NDR` ou `DCE/RPC` qui décrivent comment transformer une invocation de sous-programme en message compact (*marshalling*) et comment transformer un message en invocation de sous-programme (*unmarshalling*). Ces standards permettent d'exprimer chaque transaction entre homologues selon une métaphore d'invocation de méthode plutôt qu'à travers des stratégies de construction de messages manuelles et sujettes à erreur;
- des **moteurs d'interopérabilité** (ou *intergiciels*) tels que `CORBA`, `COM` ou `ICE` (chacun offert selon plusieurs déclinaisons), reposant sur un standard d'interopérabilité binaire et encadrant le développement de systèmes *interopérables* par un ensemble de mécanismes, de systèmes d'annuaires et des registres permettant de nommer et d'identifier les entités, de métaphores de développement, etc.

Les *sockets* ont offert une métaphore universelle de connectivité et permis le développement des mécanismes et infrastructures sophistiquées mentionnées ci-dessus. En retour, développer à l'aide de *sockets* seuls demande beaucoup d'investissement dans la difficile tâche de rédiger des protocoles et le code de communication capable d'utiliser ces protocoles.

Couvrir tous les cas possibles, gérer correctement les erreurs de communication, de domaine, protéger les transactions de tiers hostiles, toutes ces tâches sont difficiles à bien réaliser, ce qui explique que développer des systèmes répartis (à plus forte partie des systèmes offrant une forme d'accès aux données d'un homologue distant) à l'aide de seuls *sockets* ne soit raisonnable, concrètement, que pour les systèmes les plus simples.

Les stratégies de plus haut niveau, qu'on parle standards d'interopérabilité ou de moteurs d'interopérabilité, visent toutes à faciliter l'interopérabilité de produits hétérogènes (au sens de *écrits en plusieurs langages* et de *s'exécutant sur plusieurs plateformes*) dans la mesure où les homologues d'une relation `CS` donnée utilisent la même stratégie pour communiquer. Cela se comprend, le marché des moteurs d'interopérabilité étant vaste, compétitif et lucratif.

Les standards d'interopérabilité ont permis une connectivité à faible coût et le développement plus rapide de systèmes complexes. Les protocoles sont définis à l'aide d'interfaces au sens `OO` du terme, ce qui est accessible à une plus grande proportion des développeurs que ne l'est la rédaction de protocoles robustes et efficaces.

Les moteurs d'interopérabilité ont permis la mise en place de mécanismes de traitement des erreurs, de chiffrement de données, de compression, d'activation à distance des homologues et ainsi de suite. Chaque nouvelle strate permet, en construisant sur les précédentes, des gains de productivité importants.

Un problème technique de la plupart des moteurs d'interopérabilité est qu'ils utilisent tous leurs propres ports pour communiquer, ce qui tend à donner des cauchemars aux administrateurs de réseaux. Concrètement, la plupart des entreprises aimeraient restreindre l'essentiel du trafic entre leur entreprise et le monde extérieur aux échanges Web sur un nombre restreint de ports connus et communiquant à l'aide de protocoles simples.

Cela dit, surtout depuis le début du XXI^e siècle, les besoins d'interopérabilité sont maintenant tels que le but visé est devenu la mise en place et l'exploitation d'une *lingua franca* de la communication entre entités hétérogènes. Une langue commune et digne du rôle de langue officielle des échanges entre entités disparates doit avoir plusieurs propriétés particulières :

- elle doit être suffisamment **générale** pour permettre de décrire tous les protocoles envisageables, *y compris elle-même*;
- elle doit être **extensible**, pour qu'il soit possible d'y ajouter des éléments *a posteriori*; et
- elle doit être **évolutive**, pour qu'il soit possible d'ajouter à un protocole sans briser les programmes existants et en utilisant déjà une version antérieure.

Se baser sur un succès

Un format[3] permettant d'exprimer de manière extensible et évolutive, HTML (pour *Hypertext Markup Language*) avait eu beaucoup de succès et avait pavé la voie à une réflexion sur le principe de **balises**. Les documents HTML visent à représenter de manière textuelle mais flexible une structure documentaire facilitant l'inclusion d'objets sur une surface 2D (la page Web) et des hyperliens, menant vers d'autres documents de nature analogue.

L'idée d'hyperlien est attribuée à ***Douglas Engelbart***, à qui on attribue aussi beaucoup d'autres idées qui ont résulté en ce qu'on perçoit comme l'informatique contemporaine (la souris, les IPM graphiques, la métaphore du bureau de travail virtuel), le tout dans une démonstration publique qu'on nomme encore aujourd'hui *la démo d'entre les démos* (*The Mother of All Demos*)[4].

Le génie du concept de balise que *si un programme ne comprend pas une balise*, par exemple dans le cas où une nouvelle version de la norme entre en vigueur et qu'un fureteur ancien chercherait à consulter un document récent, alors *la balise en question peut être escamotée*, tout simplement, sans que le programme n'ait à planter. La question à savoir si le contenu entre les balises doit alors être escamoté ou non est une question ouverte.

En mêlant données et métadonnées (les balises, qui jouent un rôle de guide descriptif du document) dans un seul et même format, les documents hypertexte en format HTML ont montré par l'exemple qu'un outil de consultation (par exemple un fureteur) pouvait consommer un document correctement balisé tout en omettant les balises qu'il ne comprend pas, réduisant ainsi la fragilité du format face à l'usure du temps.

[3] … et un format de document est une forme de protocole permettant un voyage à travers le temps plutôt qu'à travers l'espace, n'est-ce pas?

[4] Voir http://sloan.stanford.edu/mousesite/1968Demo.html

Décrire un hyperlien

Les hyperliens sont spécifiés par ce qu'on appelle des URL (*Uniform Resource Locators*), des URN (*Uniform Resource Name*s) ou, pour s'exprimer de manière plus générale, des URI (*Uniform Resource Identifiers*). L'idée est simple : à chaque objet sur Internet, peu importe sa nature, correspond une adresse, une URL.

Une URL est composée de la coordonnée de l'ordinateur par laquelle on peut l'obtenir (adresse IP ou équivalent), suivi d'indications (analogues souvent directs du couple fait d'un répertoire et d'un nom de fichier) pour l'y retrouver.

L'idée d'identifier chaque chose pertinente de manière unique et sans ambiguïté est fondamentale à la démarche C/S. Pour qu'une connexion C/S soit possible, le client doit pouvoir nommer et rejoindre son serveur.

La nuance entre URL, URN et URI est surtout intéressante du point de vue technique[5]. Pour la plupart des gens, seuls les URL importent.

Le transport de documents HTML : protocole HTTP

Le protocole HTTP (pour *Hypertext Transfer Protocol*) est fort simple. Composé essentiellement de deux commandes majeures, GET et POST, il vise d'abord à faciliter le transfert de fichiers entre un client et un serveur, normalement (mais pas nécessairement) connectés via le port 80.

Le protocole HTTP repose sur les fondations du couple TCP/IP, ce qui implique l'envoi de flots de données entre le client et le serveur. Une trame HTTP est composée d'un en-tête informatif et d'un corps contenant essentiellement le document transmis.

L'en-tête d'une trame HTTP indique quelques données quant à la nature du document contenu dans le corps de la trame, incluant la taille en octets de ce document[6]. Le document peut être à peu près n'importe quoi, comme on peut le constater en furetant sur Internet.

De par sa simplicité, de par son ubiquité sur Internet[7], et parce qu'il sert essentiellement à transférer des documents[8], le protocole HTTP sert fréquemment de support aux protocoles C/S modernes. Ces protocoles profitent de l'ouverture reconnue à HTTP et s'en servent comme protocole de transport en décrivant à la fois leurs requêtes et leurs réponses sous forme de documents XML.

---

[5] Voir http://www.w3.org/Addressing/ pour en savoir plus. Vous pouvez aussi lire http://www.w3.org/TR/uri-clarification/ qui est plus riche mais plus long. Grossièrement, les identificateurs (URI) peuvent être groupés en deux ensembles que sont les lieux logiques (URL) et les noms logiques (URN), mais il y a des réflexions en cours pour clarifier le propos car la distinction entre lieu et nom n'est pas aussi claire qu'on le souhaiterait.

[6] Sachant ceci, on peut dire qu'un document transmis via HTTP est logiquement équivalent à une chaîne de caractères préfixée.

[7] … ce qui diminue les risques qu'un mur coupe-feu rende son serveur inaccessible sur un hôte donné.

[8] … ce qui le fait apparaître moins risqué aux yeux des administrateurs que les protocoles RPC/XDR ou DCE/RPC, par exemple.

Vers un métalangage général

Le format HTML est un format de présentation et de description de liens; il a évolué depuis sa conception initiale en fonction des besoins du moment, ce qui l'a rendu à la fois très utile et un peu mal foutu sur le plan structurel (certaines balises y nécessitent une balise de fin, d'autres pas, souvent pour des raisons de tradition plutôt que de logique). Il ne se veut pas non plus capable d'autodescription, cette préoccupation étant née de constats récents quant aux besoins d'interopérabilité encore inconnus il y a une vingtaine d'années. En ce sens, il se prête mal au rôle de format général de description de protocoles.

Certains ont toutefois vu l'opportunité de s'inspirer du format HTML pour concevoir une norme plus générale et moins appliquée, XML (pour *eXtensible Markup Language*)[9].

La norme XML est une norme textuelle et lisible reposant sur des balises, tout comme la norme HTML. Alors que HTML sert à décrire des documents liés via hyperliens, la norme XML sert à *décrire des données* (et, en général, à *décrire toute structure sous la forme d'un document*).

Les balises de XML ne sont pas prédéfinies, contrairement à celles de HTML. Une partie du travail avec un document XML est de définir les balises utilisées. On peut valider un document XML en le comparant avec une définition de type de document, en anglais *Document Type Definition* (DTD) ou, parfois, avec un schéma XML.

Un document XML

Imaginons à tout hasard qu'un document XML ait pour mandat de décrire un professeur et qu'un professeur soit fait d'un nom, d'un prénom et d'une liste de cours. Supposons aussi qu'une liste de cours contienne zéro cours ou plus et que chaque cours ait un sigle et un nom.

Un document XML valable et respectueux de ces attentes serait :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<prof>
  <nom>Roy</nom>
  <prenom>Patrice</prenom>
  <liste_cours>
    <cours>
      <nom>Exploitation des bases de données</nom>
      <sigle>INF287</sigle>
    </cours>
  </liste_cours>
</prof>
```

[9] Blague tirée de http://www.w3schools.com/xml/ et traduite librement: quand devriez-vous utiliser XML? Lorsque vous avez besoin d'un *Buzzword* vendeur. Les technologies XML sont très, *très* à la mode.

Remarquez un certain nombre de détails techniques :

- les balises sont en *minuscules*. Il s'agit d'une règle de bienséance mais pas d'une règle du langage; cela dit, XML est sensible à la casse, ce qui signifie que <xyz> et <Xyz> sont deux balises distinctes l'une de l'autre;
- les balises sont **balancées**. Pour chaque balise de début <xyz> on trouvera une balise de fin </xyz>. Seule la **déclaration XML** (placée entre <? et ?>), dont le rôle est de donner à un décodeur XML les indications dont il a besoin pour analyser correctement le document, est exemptée de cette règle. Ici, la déclaration XML sert surtout à donner la langue utilisée pour encoder le document (surtout pour qu'un décodeur traite les accents de la manière appropriée). Ainsi, la séquence de balises <a> est illégale alors que la séquence <a> est correcte;
- une paire de balises décrit un **élément** XML;
- un élément qui ne contient pas d'enfants ne requiert pas de balise de fin séparée de sa balise de début et peut simplement se terminer par />. Ainsi, les notations à droite sont équivalentes;

```
<xyz></xyz>
<xyz />
```

- il est possible d'ajouter des **attributs** à un élément XML. Par exemple, la balise <prof> pourrait s'écrire <prof type="chic"> dans le but de qualifier le professeur qu'elle sert à décrire. Ainsi, les trois notations à droite sont équivalentes;

```
<prof><type>Chic</type></prof>
<prof type="Chic"></prof>
<prof type="Chic" />
```

- la valeur d'un attribut doit être placée entre apostrophes ou être placée entre guillemets, au choix. Définir la valeur d'un attribut sans insérer un tel enrobage est illégal;
- le document est **structuré**. Une balise mère (la **racine**), ici la balise <prof>, débute (et englobe) le document. Des balises enfants (<nom>, <prenom>, <liste_cours>) décrivent des éléments de la balise mère et peuvent à leur tour posséder une liste d'éléments (et ainsi de suite, récursivement);
- un document XML a toujours **une seule racine**, ce qui signifie que si notre document devait décrire tous les professeurs d'un centre de formation, alors il serait préférable que sa racine soit <liste_profs> plutôt que <prof>;
- dans un document XML, les blancs sont importants (un décodeur HTML remplace une séquence de blancs par un seul blanc pour fins d'affichage; un décodeur XML ne le fait pas);
- l'ordre des enfants d'un élément XML n'a pas d'importance;
- un document XML peut contenir des commentaires, placés entre <!-- et -->.

Dans un nom de balise XML, on évitera les symboles **.** (le point), **-** (tiret) et toute forme d'espace. On évitera aussi les **:** qui jouent un rôle structurel et on ne débutera pas un nom de balise par un signe de ponctuation, par un chiffre ou par toute combinaison des lettres xml dans l'ordre en majuscule ou en minuscule.

Étant donné la correspondance fréquente entre noms de balises est champs dans une table de BD relationnelle, il est d'usage de respecter les règles usuelles pour nommer les champs lorsqu'on choisit un nom de balise.

Traduire et formater un document XML : la norme XSLT

Imaginons une forme un peu plus riche du document XML décrivant un professeur et proposée un peu plus haut :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="prof.xslt"?>
<liste_profs>
  <prof>
    <nom>Roy</nom>
    <prenom>Patrice</prenom>
    <liste_cours>
      <cours>
        <nom>Exploitation des bases de données</nom>
        <sigle>INF287</sigle>
      </cours>
      <cours>
        <nom>Rédaction technique pour les TI</nom>
        <sigle>INF705</sigle>
      </cours>
      <cours>
        <nom>Conception orientée-objet avancée</nom>
        <sigle>INF737</sigle>
      </cours>
      <cours>
        <nom>Séminaire en technologie de l'information</nom>
        <sigle>INF770</sigle>
      </cours>
    </liste_cours>
  </prof>
</liste_profs>
```

Imaginons que la manière par laquelle nous souhaitons présenter l'information contenue dans ce document XML soit un titre (balise <h1>) décrivant chaque professeur et une liste (en gras) décrivant chaque cours, avec le nom du cours en italiques.

Obtenir un format de résultat donné à partir d'une description XML structurée initiale est simple si un document décrivant les *transformations requises* est associé au document XML original. Ici, l'hyperlien prof.xsl décrit les règles de transformation en question.

De quoi aurait l'air un tel document? Voici :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" version="1.0" encoding="ISO-8859-1" indent="yes"/>
<xsl:template match="/">
  <html>
  <body>
    <xsl:for-each select="liste_profs/prof">
    <h1>Professeur <xsl:value-of select="nom" />, <xsl:value-of select="prenom" /></h1>
      <xsl:for-each select="liste_cours/cours">
      <xsl:sort select="nom"/>
      <ul>
        <li><strong>Sigle: <xsl:value-of select="sigle" /></strong>;</li>
        <li><strong>Nom: <em><xsl:value-of select="nom" /></em></strong>.</li>
      </ul>
      </xsl:for-each>
    </xsl:for-each>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Les lignes intéressantes sont en caractères gras :

- la balise `<xsl:template match="/">` est un signal dictant à l'analyseur `XML` de consommer tous les éléments (`"/"` est une analogie pour le «`root`» de `UNIX`);
- les balises `<xsl:for-each select="`*règle*`">` où *règle* correspond à une description de balise à partir de la balise racine (nommée `"/"`) équivaut logiquement à un select du langage `SQL`. Ce qu'il faut comprendre ici est qu'entre `<xsl:for-each` et `</xsl:for-each>`, nous décrirons les règles applicables aux éléments sélectionnés par la structure définie dans l'expression accolée au `select` suivant le `for-each`;
- un `select` dans une balise `<xsl:for-each` extrait des éléments d'un document `XML`. Ce faisant, il est possible d'extraire la valeur de chacun (balise `<xsl:value-of select="`*nom*`" />` où *nom* est le nom de l'attribut visé).

Un fureteur Web consommant le fichier `XML` décrivant les profs et examinant les règles de transformation du fichier `XSL` l'accompagnant est en mesure d'afficher l'information du fichier `XML` sous la forme formatée attendue. Le résultat est une pleine séparation du contenu et de l'affichage, chose non négligeable.

Applications de XML

La norme XML trouve aujourd'hui une multitude d'applications, un peu partout dans le monde de l'informatique et pas seulement du côté des systèmes bureautiques ou Internet. La proximité des structures XML avec celles des langages de programmation comme celles des SGBD font de XML une norme de choix pour permettre à un module d'exprimer de manière compréhensible des idées susceptibles d'être comprises par un large éventail d'homologues.

Protocole de communication XML-RPC

Si la notation XML permet de décrire des structures de données de manière générique, il va de soi qu'on peut l'utiliser pour décrire des appels de sous-programmes, et donc s'en servir comme base pour un véritable protocole de communication.

Tout comme RPC/XDR et DCE/RPC, le protocole XML-RPC[10] implémente une stratégie RPC pour réaliser des appels synchrones (sens faible).

La plupart des serveurs Web commerciaux permettent d'implémenter des services Web[11] à l'aide de ce protocole, qui se veut une alternative légère à SOAP (plus bas). SOAP est un protocole poids lourd, mais XML-RPC fait le travail dans les cas où il n'y a pas lieu d'exiger un système de types aussi riche et extensible que ce que SOAP permet de supporter[12].

La principale particularité du protocole XML-RPC est que, plutôt que de transmettre les descriptions d'appels de méthodes selon un standard binaire compact comme le fait RPC/XDR, il les transmet sous forme de description XML. Un humain techniquement alerte peut lire à l'œil nu la plupart des appels de méthodes XML-RPC et leurs résultats. Le concept est simple, mais fonctionne bien et est implanté pour plusieurs infrastructures et plusieurs plateformes distinctes[13].

Tout comme SOAP, XML-RPC utilise HTTP comme protocole transport, profitant des attributs déjà énoncés pour ce protocole (ubiquité, disponibilité, ouverture du port 80, etc.). Un appel XML-RPC apparaît dans une trame HTTP comme un document XML décrivant un appel de méthode ou son résultat.

Lorsqu'une erreur se produit lors d'un appel de méthode XML-RPC, la description de l'erreur est elle aussi retournée sous forme d'une structure de données XML.

---

[10] Voir le site http://www.xmlrpc.com/ pour des données officielles à ce sujet. Voir aussi le site http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html qui est très complet. Un chic site en français : http://www.rebolfrance.net/articles/xmlrpc/tut-rebxr1.html.

[11] Sujet sur lequel nous reviendrons plus tard dans le cours.

[12] La page http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm propose un (bref) examen comparatif de SOAP et de XML-RPC.

[13] Une implantation pour .NET est décrite sur http://www.xml-rpc.net/.Une implémentation pour COM se trouve sur http://redmonk.editthispage.com/xmlrpccom/. Une comparaison des protocoles XML-RPC et SOAP, qui se partagent un même créneau et servent des fins très semblables, se trouve sur http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm.

Le protocole XML-RPC a aussi la particularité d'être véritablement simple. Il ne supporte que les fonctions (méthodes retournant une valeur) et un nombre restreint de types[14] et de structures de données[15]. Ces contraintes de types font de XML-RPC une avenue impraticable pour des langages comme Java et les langages .NET qui veulent à l'occasion envoyer ou retourner des tableaux d'objets de plusieurs types distincts[16], mais rend en retour très alléchant ce protocole pour des systèmes qui n'ont pas besoin d'un support de types et de structures aussi complexes.

Le protocole XML-RPC peut être implémenté manuellement en un temps raisonnable par la plupart des programmeurs, bien qu'il existe plusieurs implémentations commerciales du protocole (C/C++, Java, .NET, COM[17]).

On utilise HTTP comme support pour les transactions XML-RPC existantes, profitant des attributs déjà énoncés pour ce protocole (ubiquité, disponibilité, ouverture du port 80, *etc.*).

Pour que HTTP puisse servir de support à XML-RPC, il faut que les envois XML-RPC soient des envois HTTP valides.

Pour y arriver, on insère les envois XML-RPC dans ce qu'on nomme des enveloppes HTTP (à droite, les cinq premières lignes, grisées).

La requête XML-RPC à droite appelle la fonction ecole.getNomProf en lui passant un seul paramètre, soit un entier 32 bits (4 octets, balise <i4> pour *Integer 4 Bytes*) de valeur 8.

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
   <methodName>ecole.getNomProf</methodName>
   <params>
      <param>
         <value><i4>8</i4></value>
      </param>
   </params>
</methodCall>
```

[14] La liste est courte : on parle d'entiers signés sur 32 bits, de booléens (littéral 0 ou 1), de texte ASCII (car XML-RPC ne supporte pas de manière indigène les caractères étendus de type Unicode avec encodage UTF-8 ou autre), de nombres à virgule flottante (double précision), de dates/ heures ISO 8601 et de données brutes encodées en base64. Il faut retenir que les envois respectent la norme XML, donc que leur contenu est strictement transféré sous forme de texte.

[15] La liste est aussi très courte : on parle d'enregistrements dont les membres peuvent être de types différents et de tableaux dont les éléments doivent tous être de même type. Les enregistrements ne peuvent être nommés; le système de types de XML-RPC se veut simple, pas extensible.

[16] C'est pourquoi ces langages ont plutôt recours à SOAP.

[17] Voir entre autres :

- pour Java, voir http://www.onc-rpc-xdr.com/products/rpc/xml-rpc.asp;
- pour .NET, voir http://www.xml-rpc.net/;
- pour COM, voir http://redmonk.editthispage.com/xmlrpccom/.

Une réponse possible serait celle exprimé à droite, qui n'offre qu'un seul extrant (du texte, contenant le texte `Jean Bon`).

On comprendra que les extrants et les extrants, de même que les types impliqués, doivent être connus des deux homologues.

Les types `XML-RPC` standard ont leurs propres règles (à savoir si les entiers sont signés ou non, par exemple), qu'il faut respecter dans le langage de chaque homologue, qu'il s'agisse ou non du même langage pour chacun.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>Jean Bon</string></value>
      </param>
   </params>
</methodResponse>
```

Tout bon protocole doit inclure une stratégie pour rapporter des erreurs de communication, des erreurs de traitement et des cas d'exception.

C'est le cas pour `XML-RPC`, qui peut retourner des descriptions `XML` standardisées pour les situations hors normes, les fautes et autres erreurs.

L'exemple à droite en donne une illustration.

Le protocole `XML-RPC` est simple, pouvant être implémenté manuellement en un temps raisonnable par la plupart des programmeurs. Il ne s'agit par contre pas du plus complet des protocoles. Un peu comme pour les *sockets*, on utilisera `XML-RPC` lorsqu'on aura à représenter des échanges `CS` synchrones (sens faible) à peu de frais et en peu de temps.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>4</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Prof invalide</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

Pour des projets vastes et complexes, et surtout pour des situations qui sont exigeantes du point de vue du système descriptif de types de données, il est possible qu'on préfère utiliser un protocole du même genre (description `XML` des appels de sous-programmes, utilisant `HTTP` comme véhicule) mais plus riche et plus ardu à comprendre comme à implémenter. Le protocole le plus à la mode en ce sens est `SOAP`.

Protocole de communication SOAP[18]

Tout comme XML-RPC (ci-dessus), SOAP (pour *Simple Object*[19] *Access Protocol*) applique la stratégie XML pour en faire un protocole de communication de type RPC, mais SOAP se distingue de XML-RPC du fait qu'il permet de tirer profit d'une banque de types de données riches et extensible, permettant d'insérer à même le protocole de l'information quant au type des données, et faciliter ainsi la communication entre processus même dans les cas complexes.

SOAP permet, à toutes fins pratiques, l'appel à distance de sous-programmes à travers des trames HTTP, faisant transiger entre les homologues des paquets formatés selon la norme XML. SOAP est donc une application de XML.

On peut voir l'évolution ici :

- la norme HTML décrit des documents hypertextes sous forme de texte balisé;
- la norme XML décrit des structures de données, et donc, à la limite, des normes de format documentaire comme HTML, sous forme de texte balisé;
- le protocole XML-RPC permet des appels de sous-programmes à distance de manière similaire au standard RPC mais en utilisant une description XML; et
- la norme SOAP y ajoute des capacités plus grandes d'extensibilité pour en faire un protocole évolutif. Certaines particularités de SOAP, comme la capacité de grouper des données de types différents dans un même tableau, donnent au protocole SOAP un avantage sur XML-RPC lorsque vient le temps d'appliquer le protocole à des modèles OO.

Que SOAP soit plus évolué que XML-RPC ne signifie pas nécessairement qu'on doive en tout temps préférer, dans la lignée XML, SOAP à XML-RPC. La simplicité a aussi ses vertus.

Les noms des membres d'une telle structure sont significatifs, mais l'ordre (l'organisation interne de la structure une fois transigée) dépend des besoins d'implantation des homologues. SOAP étant un protocole, les homologues peuvent représenter à l'interne les structures comme ils le veulent bien; l'encapsulation de l'implantation est totale.

Le protocole SOAP inclut aussi un standard pour rapporter les erreurs. Des balises offrent de l'information représentative de l'erreur s'étant produite, au niveau de la communication HTTP comme au niveau des données transigées.

Tel que mentionné plus haut, le monde entier semble se standardiser vers SOAP pour les applications commerciales courantes. La plupart des langages utilisés aujourd'hui supportent ce protocole, à divers degrés. Si le mouvement de standardisation vers SOAP aboutit à un véritable standard commun, ce qui reste à faire, alors SOAP aura été un grand succès.

---

[18] Voir http://www.w3schools.com/soap/ pour des exemples de messages SOAP. La syntaxe XML de SOAP est un peu alourdie par des identificateurs arides, qu'on n'a pas vraiment à connaître de fond en comble pour comprendre le principe derrière le protocole et en tirer profit; vous pouvez utiliser ce site Web comme point de lancement si vous désirez faire communiquer vos processus via SOAP.

[19] Attention : on voit le mot *objet* dans la définition de l'acronyme, mais le protocole n'est pas OO, et n'exige pas que les homologues le soient non plus. Cela dit, SOAP n'empêche pas non plus de transiger des objets, ou du moins leurs attributs, comme le permet au fond n'importe quel protocole assez riche.

Norme propriétaire ou norme ouverte?

Le spectre de *Microsoft* plane sur SOAP, au sens où cette compagnie a investi beaucoup d'argent et d'effort pour le faire adopter comme norme *de facto* de communication entre processus. Les appels de sous-programmes faits sur la plateforme .NET reposent sur le protocole SOAP. Certains se demanderont si SOAP, dans ce contexte, est un standard ouvert ou propriétaire.

Il se trouve que SOAP est une application de XML, qui est un standard ouvert, texte et explicite. La maintenance de SOAP est faite, au moment d'écrire ces lignes, par un consortium, ce qui fait qu'on ne parle pas d'une norme propriétaire. Chez W3C, les discussions tournent, *au moment d'écrire ces lignes*, autour de la version 1.2 du protocole.

Honnêtement, SOAP est une bonne idée. Un peu trop compliqué, du moins pour ce qui est de demeurer lisible pour des humains, mais une bonne idée quand même.

Structure de SOAP

SOAP se compose de trois morceaux fondamentaux :

- une ***enveloppe HTTP*** similaire à celle de XML-RPC;
- une ***enveloppe SOAP***, décrite selon les règles de XML, définissant le contenu du message transigé et expliquant comment traiter le message;
- des ***règles***, selon un format décrit par un document XML, décrivant la manière d'encoder les instances des types de données construits à même les messages. C'est ici que SOAP prend beaucoup de sa souplesse; et
- une convention pour représenter des appels de sous-programmes à distance et le résultat de leur exécution, encore une fois de manière analogue à celle utilisée par XML-RPC.

Puisque SOAP applique XML, les données transigées peuvent être aisément structurées; on n'a qu'à penser à des struct de C ou à des RECORD de Pascal. C'est même, à vrai dire, une écriture plutôt naturelle.

Les noms des membres d'une telle structure sont significatifs, mais l'ordre (l'organisation interne de la structure une fois transigée) dépend des besoins d'implantation des homologues; SOAP étant un protocole, les homologues peuvent représenter à l'interne les structures comme ils le veulent bien; l'encapsulation de l'implantation est totale.

Le protocole SOAP inclut aussi un standard pour rapporter les erreurs. Des balises offrent de l'information représentative de l'erreur s'étant produite, au niveau de la communication HTTP comme au niveau des données transigées.

Note :	de nombreux systèmes incluent des murs coupe-feu. En général, HTTP passe quand même ces murailles, étant pratique et relativement inoffensif. SOAP permet donc à des processus d'appeler des fonctions l'un de l'autre malgré la présence d'un mur coupe-feu. Ceci est attirant en soi. Lorsque le besoin se fait sentir de faire utiliser à un serveur SOAP un port différent du port 80 traditionnel, l'usage veut que le port 8080 soit privilégié. Ce n'est pas tant une règle qu'une habitude.

Un petit exemple de transaction SOAP[20]

Ce qui suit présente un petit exemple de transaction, où un client envoie la requête `getStockPrice` à un serveur. On voit d'ailleurs le préfixe `m:` qui évoque l'idée de méthode; c'est au site `http://www.stock.org`, donné dans l'enveloppe du message, qu'on retrouve la description de `m:` comme préfixe nom de fonction.

La balise enfant de la méthode est le paramètre `StockName`. Si on examine la réponse à la requête, on constate que le serveur offre comme résultat la balise `Price` de valeur `34.5`.

Il est clair que le client et le serveur se sont entendus au préalable sur l'idée selon laquelle une demande de prix d'action à l'aide d'un nom de firme en paramètre résulterait en un prix (nombre à virgule flottante). `SOAP` facilite la définition de protocoles de communication entre processus, mais ne remplace pas le design de ces protocoles.

On remarque que, `SOAP` reposant sur http pur et simple, le début du message est une commande `HTTP` régulière. Et que la réponse du serveur inclut un code du succès ou d'erreur (ici : `200 OK`).

La requête `SOAP` aurait l'air de ceci (l'en-tête `HTTP` apparaît en grisé) :

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap; charset=utf-8

<?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 xmlns:m="http://www.stock.org/stock">
    <m:getStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:getStockPrice>
  </soap:Body>
</soap:Envelope>
```

[20] Tiré de `http://www.w3schools.com/soap/soap_example.asp`.

La réponse SOAP à cette requête aurait l'air de :

```
HTTP/1.1 200 OK
Connection: close
Content-Type: application/soap; charset=utf-8
Date: Sat, 12 May 2002 08:09:04 GMT

<?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 xmlns:m="http://www.stock.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Pourquoi tant de protocoles reposant sur HTTP?

Les normes XML-RPC et SOAP ne sont pas les seuls protocoles récents à faire leur entrée dans l'arène des protocoles généraux à vocation commerciale reposant sur HTTP. Il s'agit sûrement d'un aboutissement logique, au moins en esprit, de l'idée d'utiliser un langage pour échanger à la fois instructions et données.

Si on se penche du côté des produits de type C/S de la firme *Microsoft*, les transactions SOAP constituent peut-être le plus grand avantage technologique de l'environnement .NET sur ce qu'on pourrait identifier (par abus de langage) ses prédécesseurs, soit DCOM et COM+.

Ceci parce que l'un des obstacles les plus importants au déploiement via Internet du volet réparti de COM+, c'est-à-dire DCOM, tenait de son utilisation du port RPC (le port 135 pour DCE/RPC), qu'on ne laisse normalement pas ouvert au monde entier, mais qui sert beaucoup à l'intérieur d'un réseau local ou d'un Intranet.

Personnaliser un mur coupe-feu tout en essayant de laisser fonctionner des SC/S créés via DCOM peut être un cauchemar, surtout dans un environnement se voulant sécurisé, comme les environnements militaires, et ce, malgré le chiffrement des données et les différentes strates de sécurité utilisées par cette technologie.

Consommer un document XML : un aperçu

Deux grandes familles de stratégies reconnues par le W3C existent pour consommer des documents XML : les stratégies de la famille **SAX** et les stratégies de la famille **DOM**.

L'acronyme SAX signifie *Simple API for XML*. La stratégie de traitement préconisée par SAX est simple, comme son nom l'indique : un client d'abonne aux services du processeur SAX et lui indique quels éléments l'intéressent dans un document donné en lui donnant suffisamment d'information pour que SAX le rappelle.

Lorsque les éléments en question sont rencontrés, le moteur SAX invoque les méthodes de ses abonnés et leur permet de réagir à leur manière. *SAX est donc une stratégie implémentée sous forme du schéma de conception observateur*.

L'avantage principal de SAX est qu'il est peu coûteux en ressources : le document XML est consommé en une seule traversée et n'est pas conservé en mémoire pour fins de navigation ultérieure et arbitrairement riche. En retour, SAX est peu flexible au sens où un élément est perdu s'il n'est pas traité lorsqu'il est rencontré la première fois.

L'acronyme DOM signifie *Document Object Model*. Consommer un document avec DOM signifie le charger au complet en mémoire puis naviguer à travers à l'aide des divers services que le modèle OO du document propose.

L'approche DOM est très flexible. Les éléments sont consommés au gré du programme et peuvent être navigués et traversés plusieurs fois si cela semble pertinent. En retour, DOM coûte beaucoup plus cher en ressources que SAX du fait qu'il doit charger (et représenter de manière OO!) le document XML au complet en mémoire.

Règle générale, l'approche DOM est un peu plus fragile que l'approche SAX. Un tiers hostile désireux d'attaquer un consommateur de document XML (ce qui implique **beaucoup** de cibles potentielles!) pourrait livrer un document XML généré automatiquement et contenant une infinité d'éléments inintéressants sans jamais fermer la balise racine du document. Une consommation selon l'approche SAX serait peu sensible à une telle attaque alors qu'une consommation selon l'approche DOM risquerait de voir sa mémoire s'épuiser.

Consommer un document XML

Un document XML décrit une structure de données. En tant que tel, il s'agit donc d'un outil descriptif intéressant, souple et extensible. En retour, il est clair que sa pertinence dépend de notre capacité d'en automatiser la consommation et le traitement.

Il existe deux grandes métaphores recommandées par le W3C[21] pour consommer des documents XML, soit DOM[22] et SAX[23]. Le W3C offre même un bref texte décrivant la relation entre ces deux stratégies[24], mais pour bien comprendre chacune d'entre elles, avec ses avantages et ses inconvénients, il vaut mieux mettre la main à la pâte et explorer ses tenants et ses aboutissants.

Consommation d'un document XML avec DOM

La philosophie DOM est de représenter un document XML par une structure arborescente en mémoire. Le document est lu en entier par un bâtisseur puis représenté par un objet qu'il est possible de traverser à loisir. C'est à la fois simple à coder, flexible et coûteux en ressources.

Consommer un document XML avec DOM suit une syntaxe de base assez stable d'un projet à l'autre, au point où il est possible de définir le comportement de base dans une classe générique et de spécialiser simplement ce qui doit être fait avec le document lui-même.

En premier lieu, certaines classes standard doivent être importées :

- quelques classes et interfaces standard du *Java XML Parser* (JAXP), qui implémentent un schéma de conception fabrique;
- quelques exceptions de consommation de documents XML (prises du paquetage SAX);
- quelques classes standard d'entrées/sorties pour l'accès aux fichiers; et
- quelques classes propres à DOM et provenant du W3C.

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
```

Notre petite classe générale de consommation de documents XML se nommera ConsommateurDOM. Son rôle sera de mettre en place tout le code propre à la consommation d'un document XML avec DOM jusqu'au moment de la consommation à proprement dit à l'aide d'outils standard.

[21] Voir http://www.w3.org/

[22] Voir http://www.w3.org/DOM/

[23] Voir http://www.saxproject.org/

[24] Voir http://www.w3.org/DOM/faq.html#SAXandDOM

La classe `ConsommateurDOM` se chargera donc de la création de la structure de document `DOM` à partir d'un fichier `XML` mais pas de son analyse.

Elle sera abstraite parce qu'elle invoquera une méthode (`traiter()`, prenant en paramètre un `Document` correctement créé) que ses enfants devront implémenter.

Sa principale méthode sera `consommer()` et prendra en paramètre un nom de fichier `XML`. Cette méthode construira le document `XML` à consommer selon les règles du standard DOM puis invoquera la méthode `traiter()` pour que celle-ci procède à l'analyse du document.

Pour construire un document, on a recours à un `DocumentBuilder` construit à l'aide d'une fabrique. Notez que la validation de la structure du document `XML` est inactive par défaut mais peut être activée en appelant la méthode `setValidating()` de la fabrique et en lui passant la valeur `true` en paramètre. Il en va d'ailleurs de même pour le support aux espaces nommés `XML`.

```
public abstract class ConsommateurDOM {
   public ConsommateurDOM() {
   }
   protected abstract void traiter (Document doc);

   public void consommer(String nomFich) {
      DocumentBuilderFactory Fabrique =
         DocumentBuilderFactory.newInstance();
      // Fabrique.setValidating(true);
      Fabrique.setNamespaceAware(true);
      try {
         DocumentBuilder bâtisseur =
            Fabrique.newDocumentBuilder();
         Document doc =
            bâtisseur.parse(new File(nomFich));
         traiter (doc);
      } catch (SAXException sxe) {
         Exception ex =
            (sxe.getException() != null)?
               sxe.getException() : sxe;
         ex.printStackTrace();
      } catch (ParserConfigurationException pce) {
         pce.printStackTrace();
      } catch (IOException ioe) {
         ioe.printStackTrace();
      }
   }
}
```

La méthode `parse()` d'un bâtisseur de document reçoit en paramètre un nom de fichier `XML`, analyse le document correspondant et en crée une représentation en mémoire (un document). Une fois le document bâti, ne reste plus qu'à le traiter.

Le code de `ConsommateurDOM` ne varie pas vraiment d'une classe de consommation de documents `XML` avec `DOM` à une autre. Le code des enfants est entièrement dépendant de l'application souhaitée. Vous trouverez à l'annexe `00` un exemple opérationnel consommant un document `XML` et l'affichant dans un `JTree`.

Consommation d'un document XML avec SAX

La philosophie SAX est de traverser un document XML en reconnaissant certaines structures pertinentes (éléments, attributs) et d'invoquer des méthodes en réaction à cette reconnaissance. C'est à la fois simple à coder et peu coûteux en ressources mais il n'est pas possible de traverser le document entier à loisir. Il faut que le code utilisant SAX soit en mesure de réagir correctement lors de la rencontre des éléments et attributs auxquels il s'intéresse.

Un exemple simple pour montrer le fonctionnement de SAX est de consommer un fichier XML et d'en reproduire la structure à la console[25].

En premier lieu, certaines classes standard doivent être importées :

- les diverses classes et interfaces standard du *Java XML Parser* (JAXP), qui implémentent un schéma de conception fabrique;
- quelques exceptions de consommation de documents XML (prises du paquetage SAX); et
- quelques classes standard d'entrées/sorties pour l'accès aux fichiers.

```
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

import org.xml.sax.*;

import java.io.*;
```

Notez en particulier DefaultHandler, qui est un adaptateur implémentant une version vide pour les diverses méthodes qu'un analyste SAX est susceptible de rappeler. En dérivant de cet adaptateur, notre code n'aura plus qu'à implémenter les méthodes réagissant aux parties d'un document qui nous intéressent vraiment.

Notre classe ConsommateurSAX dérivera donc de DefaultHandler, ce qui en simplifiera de beaucoup l'implémentation.

La méthode main() de cette classe sera un simple programme qui prendra une liste de noms de fichiers XML et créera un ConsommateurSAX pour chacun de ces fichiers.

```
public class ConsommateurSAX extends DefaultHandler {
   public static void main(String [] args) {
      if (args.length == 0) {
         System.err.println
            ("Usage: java AfficheurDOM fichierXML...");
      } else {
         for (String s : args) {
            ConsommateurSAX cs = new ConsommateurSAX();
            cs.consommer (s);
         }
      }
   }
```

La méthode consommer() d'un ConsommateurSAX sera le point de départ de l'analyse d'un fichier XML donné.

[25] Je me suis fortement inspiré du démonstrateur détaillé sur le site à l'adresse suivante pour cet exemple : http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/sax/2a_echo.html

Les attributs de notre petite classe seront au nombre de quatre, soit deux attributs de classe et deux attributs d'instance :

- un attribut de classe, nommé `ENCODAGE`, représentera le standard d'encodage de caractères choisi pour la projection sur le flux en sortie;
- un attribut de classe, nommé `ENTÊTE_XML`, représentera un en-tête `XML` conforme;
- un attribut d'instance, `niveau_`, représentera le niveau d'imbrication où nous en sommes rendus dans l'analyse du document `XML` (pour faciliter un affichage correctement indenté); et
- un attribut d'instance, `écrivain_`, sera notre outil standard pour projeter le fruit de l'analyse sur un flux en sortie.

```
private static final String
   ENCODAGE = "ISO-8859-1";
private static final String
   ENTÊTE_XML = "<?xml " +
                "version='1.0' " +
                 "encoding='" +
                 ENCODAGE +
                 "'?>";
private int niveau_;
private Writer écrivain_;
```

Il faut comprendre ici que nous allons analyser un document `XML` et en générer une forme structurée sur un flux de sortie (ici; la console) ce qui implique de tenir à jour un outil permettant d'écrire à la fin de ce flux. Un attribut est donc un choix raisonnable.

Le niveau d'imbrication augmentera à la rencontre d'un début d'élément et décrémentera à la rencontre de la fin d'un élément. Ainsi, l'affichage sera indenté convenablement.

Le constructeur d'une instance de `ConsommateurSAX` initialisera, comme il se doit, les attributs d'instance de l'objet en cours de construction.

Par défaut, le niveau d'imbrication sera nul. Pour fins de démonstration, le flux en sortie sera la console.

```
public ConsommateurSAX () {
   niveau_ = 0;
   try {
      écrivain_ =
         new OutputStreamWriter(System.out, ENCODAGE);
   } catch (IOException ioe) {
      ioe.printStackTrace();
   }
}
```

Pour générer une `String` contenant l'indentation propre au niveau d'imbrication courant, nous utiliserons un algorithme naïf et inefficace (mais je vous invite à faire mieux; c'est impératif dans un projet réel!).

```
private String générerIndentation() {
   final int NB_ESPACES_NIVEAU = 3;
   String ind = "";
   for (int i = 0; i < niveau_ * NB_ESPACES_NIVEAU; ++i) {
      ind = ind + " ";
   }
   return ind;
}
```

Puisque nous allons projeter des données (du texte!) sur un flux à tout moment, le code des diverses méthodes risque d'être parsemé de blocs `try` et de blocs `catch`. Pour éviter d'alourdir le tout, nous mettrons en place une méthode `émettre()`, responsable de projeter une `String` sur le flux en sortie, et une méthode `nouvelleLigne()`, responsable de changer de ligne dans le respect du standard d'encodage de caractères choisi.

Remarquez que ces méthodes lèvent, au besoin, des `SAXException` enrobant une autre sorte d'exception pour que le code de traitement se concentre sur un seul type d'exception.

Remarquez aussi la jolie manière d'exprimer un saut de ligne (oui, un simple `"\n"` aurait fait l'affaire).

```
private void émettre(String s) throws SAXException {
   try {
      écrivain_.write(s);
   } catch (IOException e) {
      throw new SAXException
         ("Erreur d'entrée/ sortie: ", e);
   }
}
private void nouvelleLigne() throws SAXException {
   émettre(System.getProperty("line.separator"));
}
```

La méthode `consommer()` en soi ressemble fortement à du code vu précédemment (voir ***Consommation d'un document XML avec DOM***).

Ce n'est pas accidentel : les schémas de conception sont les mêmes et le traitement sous-jacent est semblable (en fait, il est fort probable que `DOM` soit, en Java, construit par-dessus `SAX`). Conséquemment, les exceptions susceptibles d'être levées sont, pour l'essentiel, les mêmes.

La méthode `parse()` d'un `SAXParser` analyse un fichier `XML`. Le deuxième paramètre est un gestionnaire, ce que notre `ConsommateurSAX` est du fait qu'il dérive de `DefaultHandler`.

```
public void consommer(String nomFich) {
   try {
      écrivain_.émettre
         ("Décodage du fichier " + nomFich + "\n");
      écrivain_.flush ();
      SAXParserFactory Fabrique =
         SAXParserFactory.newInstance();
      SAXParser AnalysteSAX = Fabrique .newSAXParser();
      AnalysteSAX.parse (new File(nomFich), this);
   } catch (SAXException sxe) {
      Exception ex = (sxe.getException() != null)?
         sxe.getException() : sxe;
      ex.printStackTrace();
   } catch (ParserConfigurationException pce) {
      pce.printStackTrace();
   } catch (IOException ioe) {
      ioe.printStackTrace();
   }
}
```

Ceci explique que `this` soit passé en tant que 2e paramètre à `parse()` : c'est notre propre objet qui sera rappelé par le `SAXParser` lorsque celui-ci rencontrera des éléments et des attributs qu'il considère pertinents.

Pour le reste, le code de notre classe `ConsommateurSAX` sera constitué de méthode réagissant à la rencontre d'éléments précis d'un document `XML`.

Rencontrer un début de document `XML` entraîne l'invocation de la méthode `startDocument()` du gestionnaire du `SAXParser`. De même, une fin de document `XML` entraîne l'invocation de sa méthode `endDocument()`.

Dans notre cas, le texte à projeter sur le flux en sortie est relativement simple dans chaque cas.

```
public void startDocument() throws SAXException {
   émettre (ENTÊTE_XML);
   nouvelleLigne();
}
public void endDocument() throws SAXException {
   try {
      nouvelleLigne ();
      écrivain_.flush();
   } catch (IOException e) {
      throw new SAXException
         ("Erreur d'entrée/ sortie: ", e);
   }
}
```

À la rencontre d'un début d'élément, le `SAXParser` invoque la méthode `startElement()` de son gestionnaire.

L'essentiel du code inséré dans cette méthode pour notre petite classe a trait à l'identification de ce qui doit apparaître dans la balise projetée sur le flux en sortie : certains documents `XML` utilisent des espaces nommés qualifiant les noms, d'autres ne le font pas; de même, certains éléments sont munis d'attributs, d'autres non.

```
public void startElement
   (String namespaceURI, String nomSimple,
    String nomQualifié, Attributes attrs) throws SAXException {
   émettre (générerIndentation ());
   ++niveau_;
   String nomÉlément = nomSimple;
   if ("".equals(nomÉlément)) { // pas de namespace
      nomÉlément = nomQualifié;
   }
   émettre("<" + nomÉlément);
   if (attrs != null) {
      for (int i = 0; i < attrs.getLength(); i++) {
          String nomAttribut = attrs.getLocalName(i);
          if ("".equals(nomAttribut)) {
             nomAttribut = attrs.getQName(i);
          }
          émettre (" " + nomAttribut + "=\"" +
                   attrs.getValue (i)+ "\"");
      }
   }
   émettre(">");
   nouvelleLigne();
}
```

Petite technique charmante : pour tester si une `String` nommée `s` est vide ou non, le code utilise `"".equals(s)` plutôt que `s.equals("")` ou `s.length()==0`. La raison de cette manœuvre est que `""` n'est certainement pas `null` alors que s peut l'être et que tout bon `equals()` devrait gérer correctement le cas où son paramètre vaut `null`. Il s'agit donc d'un test plus léger que `s==null || s.equals("")`.

La méthode pour réagir à la fin d'un élément, nommée `endElement()`, est elle aussi assez simple.

Enfin, la méthode servant à gérer le texte rencontré entre le début et la fin d'un élément se nomme `characters()`.

Clairement, un analyseur `SAX` est une chose *a priori* simple.

Si nous voulions être attentifs à certains éléments seulement, il ne nous resterait plus qu'à analyser le nom des éléments et des attributs qui nous intéressent et à invoquer le code nous permettant de réagir de la manière nous semblant la plus appropriée.

```
    public void endElement
       (String namespaceURI, String nomSimple,
        String nomQualifié) throws SAXException {
       --niveau_;
       émettre(générerIndentation () +
               "</" + nomQualifié + ">");
       nouvelleLigne ();
    }
    public void characters
       (char buf[], int offset, int len) throws SAXException {
       String s = new String(buf, offset, len).trim();
       if (! "".equals (s)) {
          émettre (générerIndentation () + s);
          nouvelleLigne();
       }
    }
}
```

Exercices—Série 00

EX00—Inspirez-vous de la classe `AfficheurDOM` pour concevoir la classe `EchoDOM` qui générera un écho texte structuré à la console, semblable à celui généré par `AfficheurSAX`, plutôt qu'un affichage graphique à l'aide d'un arbre.

EX01—Inspirez-vous de la classe `AfficheurSAX` pour concevoir la classe `ArbreSAX` qui générera un affichage graphique à l'aide d'un arbre, semblable à celui généré par `AfficheurSAX`, plutôt qu'un écho texte structuré à la console.

EX02—Utilisez `JDBC` pour réaliser une requête `SQL`. Utilisez les données et les métadonnées obtenues par les `ResultSet` générés suite à votre requête pour construire un document `XML` correspondant à ce `ResultSet`. Examinez ce document `XML` à l'aide d'un outil de visualisation `XML` de votre choix (pourquoi pas l'un de ceux produits en réponse aux exercices `EX00` et `EX01` de cette série?).

Cet exercice est important : règle générale, dans un système réparti consultant des données à distance, les échanges se font aujourd'hui à travers des documents `XML`. Le code que vous obtiendrez ici, s'il est bien écrit, sera général et fortement réutilisable.

Servlets et JSP

Il existe plusieurs manières d'offrir des services à travers Internet. L'une des plus connues est d'exposer des services Java (des *Servlets*) à travers un serveur Web comme le populaire Tomcat du projet Jakarta de Apache.

Nous développerons ici une approche opérationnelle pour faire le pont entre le Web, le code Java et un SGBD.

Le projet Jakarta, comme la plupart des projets de la fondation Apache[26], est un projet (ou un groupe de projets) à code ouvert[27], mené de front par des volontaires enthousiastes.

Tomcat est, dans les circonstances, un produit d'une qualité étonnante et sa position enviable sur le marché n'en est que plus fascinante.

Installer Tomcat

Nous ne couvrirons pas ici l'installation du serveur Tomcat. Sachez toutefois que cette installation, bien que légèrement complexe, n'est pas difficile à réaliser. Vous trouverez les fichiers à installer, de même que les instructions détaillées pour procéder à l'installation, sur le site http://jakarta.apache.org/.

Une fois Tomcat installé et démarré, vous devriez être en mesure d'ouvrir la page http://localhost:8080 ou encore (à votre convenance) http://127.0.0.1:8080 et voir la page d'accueil par défaut de Tomcat[28].

Le nom réel de la page est index.jsp, donc http::/localhost:8080/index.jsp et http://127.0.0.1:8080/index.jsp devraient toutes deux ouvrir aussi la ^page d'accueil par défaut de Tomcat.

Si vous rencontrez des ennuis au démarrage de Tomcat, il est possible qu'un autre processus (peut-être un serveur Web tel IIS) soit déjà lancé sur votre poste de travail et se soit déjà approprié le port 8080. Dans ce cas, il vous faudra arrêter l'autre serveur pour pouvoir ensuite lancer Tomcat; en tout temps, sur un ordinateur donné, un seul processus peut être propriétaire d'un port logique de communication.

[26] Voir http://www.apache.org/foundation/ pour des détails.

[27] Notez que plusieurs projets de la fondation Apache (incluant Tomcat) sont des projets à code ouvert et rédigés en Java, mais que Java lui-même n'est pas (au moment d'écrire ces lignes, soit en février 2007) un produit à code ouvert. Il y a souvent confusion sur le sujet : Java est un produit qui appartient à une entreprise (Sun Microsystems) mais qui est soutenu en partie par une communauté et qui est utilisé (comme plusieurs autres langages d'ailleurs) pour le développement de projets à code ouvert. Ainsi, Tomcat est à code ouvert et Tomcat est écrit en Java, mais Java n'est pas, en soi, un produit à code ouvert. Cela dit, Sun a accepté d'ouvrir les sources de Java et, en théorie, le tout devrait être ouvert à la fin de l'année 2007. L'appellation Java, elle, demeurera une appellation propriétaire.

[28] Le nom de domaine localhost et l'adresse 127.0.0.1 sont tous deux réservés, selon la tradition, pour ce qu'on nomme le *loopback*, c'est-à-dire une adresse pour communiquer sans devoir passer à travers Internet.

Ce qu'est un servlet[29]

Un servlet est une classe Java prise en charge par un serveur Web et réagissant à des requêtes selon un protocole donné.

Typiquement, un servlet est invoqué en réaction à une action dans une page Web, portant souvent l'extension `JSP` pour *Java Server Page*, et générant en sortie une autre page `JSP`. L'idée derrière une page `JSP` est d'avoir une mise en forme gérée par balises `HTML` et un contenu généré par programmation (souvent à travers un accès à une `BD`).

La plupart des servlets répondent à des requêtes `HTTP` de type `GET` et `POST`; conséquemment, la plupart des servlets sont des classes dérivant de `HttpServlet` et offrant un petit nombre de services (en particulier les méthodes `init()`, `destroy()`, `doGet()` et `doPost()`). Un peu comme un fureteur charge et initialise un applet, un serveur Web tel Tomcat chargera un servlet et l'utilisera de manière polymorphique.

Règle générale, toutefois, un serveur Web chargera en mémoire beaucoup de servlets alors qu'un fureteur n'aura à gérer qu'un petit nombre d'applets à la fois. Pour des raisons de performance, le serveur Tomcat charge les servlets au démarrage, ce qui implique qu'injecter de nouveaux servlets dans sa configuration puisse entraîner un redémarrage du serveur.

Dans ce document, j'utiliserai **application Web** pour un groupe de servlets exposés à travers une interface Web. Je n'aime pas ce terme plutôt galvaudé mais il est consacré par la tradition.

Java et ses déclinaisons standard et entreprise

Le `JDK` de Java se décline en deux grandes versions, soit la version standard (`J2SE`, pour *Java Standard Edition*) et la version entreprise (`J2EE`, pour *Java Entreprise Edition*). La version entreprise est beaucoup plus volumineuse que la version standard. Pour utiliser des servlets, il est nécessaire d'avoir accès à `J2EE` (ou, au minimum, à l'archive `j2ee.jar` qui contient la classe `HttpServlet` et ses associées).

Soyez prudents si vous développez des classes Java pour la version entreprise du `JDK` car celle-ci tend à être un peu en retard sur la version standard (une simple question de volume).

[29] Je vous invite à lire `http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets2.html` pour une couverture très détaillée des servlets.

Cycle de vie d'un servlet

Les servlets et les applets vivent tous deux dans des conteneurs. Le conteneur d'un servlet est habituellement un serveur Web qui le chargera, en invoquera les méthodes et, éventuellement, s'en débarrassera.

Un servlet aura une opportunité de préparer le terrain pour que son exécution soit fluide lorsque son conteneur invoquera sa méthode `init()`. De même, un servlet pourra procéder à une dernière étape de nettoyage lorsque son conteneur invoquera sa méthode `destroy()`. Pour le reste, les méthodes d'un servlet seront surtout invoquées en réponse à des requêtes propres au protocole pour lequel il a été conçu.

Le rôle du `ServletConfig`

La méthode `init()` d'un servlet reçoit en paramètre une référence à une instance de `ServletConfig`. Il se trouve que `ServletConfig` est une interface derrière laquelle se cache un objet capable de décrire la situation ayant mené à l'invocation du servlet.

À travers une référence à un `ServletConfig`, un servlet pourra connaître les divers paramètres lui ayant été passés au démarrage et pourra obtenir une référence sur son contexte (décrit par une instance de `ServletContext`).

Le rôle du `ServletContext`

Derrière l'interface `ServletContext` se cache une représentation opérationnelle du conteneur d'un servlet. Pour chaque `JVM`, on peut trouver plusieurs applications Web, et pour chaque application Web d'une `JVM` donnée on trouvera un `ServletContext`.

Un `ServletContext` offre divers services à ses servlets. C'est donc à travers un `ServletContext` qu'un servlet parvient à communiquer avec son conteneur et avec la plateforme sur laquelle ce conteneur s'exécute.

L'un des services les plus utiles offerts par une instance de `ServletContext` est la méthode `getRequestDispatcher()`, qui retourne une référence sur un `RequestDispatcher`. Nous y reviendrons un peu plus loin.

Protocole HTTP, en bref

Le protocole HTTP est un simple protocole de transfert de documents. Les serveurs Web communiquent naturellement à l'aide de ce protocole (entre autres) mais il est relativement simple de développer un système transigeant des trames HTTP.

Une trame HTTP est faite d'un en-tête et d'un corps (aussi nommé *Payload*). L'en-tête peut contenir un vaste éventail de données descriptives[30], incluant la version de HTTP utilisée, la taille du corps, le comportement des homologues quant au maintien de la connexion, la technologie et la plateforme de l'émetteur, la langue du fureteur ou du document contenu dans la trame, le type de contenu, *etc*.

Caractéristique importante de HTTP : **il s'agit d'un protocole sans état**. C'est là à la fois une des raisons de son succès et la source de divers maux de tête chez les gens développant des SC/S ou transactionnels reposant sur ce protocole. En effet, un protocole sans état ne présume pas que les homologues demeurent connectés l'un à l'autre entre deux échanges et n'exige pas du code serveur qu'il maintienne une représentation de l'état de ses échanges avec son client.

Cela implique qu'une trame HTTP doit être pleinement autodescriptive. Le client est responsable de faire suivre, à travers les paramètres apposés à la fin de l'URL indiquant le lieu du serveur ou à partir de données session telles celles entreposées dans des *Cookies*, la totalité des états pertinents à l'état courant de son échange avec le serveur.

Le bon côté des protocoles sans état est qu'ils sont fortement *échelonnables* (il s'agit d'un néologisme mais c'est-ce que j'ai trouvé de mieux pour l'anglais *Scalable*). En effet, en laissant reposer le poids de la conservation des états transactionnels du côté client, les protocoles sans état permettent la conception de serveurs moins sujets à être écrasés en période de pointe par une masse trop importante de clients.

Les protocoles sans état facilitent aussi la mise en place de procédures pour remplacer dynamiquement un serveur déficient ou sujet à une période d'entretien : puisque chaque requête provenant d'un client est autonome et pleinement autodescriptive, il est possible de mettre en place un groupe de serveurs tels que les uns soient aussi qualifiés que les autres pour traiter n'importe quelle requête HTTP.

Une **requête HTTP** débute par le type de commande (souvent GET ou POST) suivi d'une gamme de paramètres descriptifs (nom de domaine du serveur, version de HTTP utilisée, document demandé dans le cas d'une requête GET, langue, *etc.*).

Une **réponse HTTP** débute par un code de succès ou d'erreur[31] suivi d'un ensemble de données descriptives (langue, encodage, taille du corps du message, *etc.*). Le code de succès typique pour un GET ou un POST est 200, et l'erreur la plus fréquemment rencontrée par un fureteur est sans doute l'erreur 404 (le document à l'URL spécifiée par la requête est introuvable).

---

[30] Voir http://www.http.header.free.fr/httpattributs.html pour quelques exemples typiques.

[31] Voir http://en.wikipedia.org/wiki/List_of_HTTP_status_codes pour une liste des codes de succès ou d'erreur HTTP possibles.

Une **requête `GET`** permet à un client de demander à un serveur l'information correspondant à l'`URI` de la requête (`URI` susceptible d'être une `URL` suivie de paramètres). Cette information peut être un document tel quel ou généré par programmation. L'en-tête d'une trame `HTTP` pour une requête `GET` peut contenir de l'information ayant pour but de guider la réaction du serveur suite à l'analyse des paramètres (l'équivalent d'alternatives en fonction de la présence ou de l'absence de certains paramètres, par exemple).

D'un point de vue performance, les requêtes `GET` peuvent être placées en antémémoire dans le fureteur si certaines conditions sont respectées. D'un point de vue sécurité, le fait que les paramètres d'une requête `GET` apparaissent dans l'`URI` rend ce type de requête vulnérable à une interception par un tiers hostile. Pour en savoir plus, mieux vaut consulter la documentation du protocole[32].

Une **requête `POST`** est le moyen privilégié de soumettre à un serveur des données potentiellement sensibles à partir d'un poste client. Les divers champs d'une requête `POST` sont imbriqués dans le contenu du message plutôt que dans l'`URI` de destination. En retour, la plupart des clients n'entreposent pas les requêtes `POST` en antémémoire.

Marche à suivre

La partie la plus complexe des applications Web faites à l'aide de servlets (et de la plupart des services orientés vers le Web, tous langages confondus) n'est pas le volet programmation, qui est presque toujours fortement simplifié de par les stratégies polymorphiques rendues possibles par l'approche `OO`.

La majorité des approches orientées Web impliquent en effet, d'un point de vue programmation, qu'on dérive d'une classe donnée et qu'on implémente certaines méthodes *a priori* abstraites, ce qui est plutôt routinier pour les programmeuses et les programmeurs `OO`.

Ce qui est complexe, en fait, est la liste d'étapes à suivre pour intégrer nos classes à un serveur Web ou l'autre. Cette démarche, qui n'est pas du ressort de la programmation, est très dépendante de la technologie de serveur Web utilisée.

[32] Voir `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

Identifier le répertoire racine

Pour intégrer des applications Web à Tomcat, donc, il faut tout d'abord **identifier le répertoire racine de Tomcat**. Chez moi, ce répertoire est

```
C:\Program Files\Apache Software Foundation\Tomcat 5.0\
```

Pour le reste de cette marche à suivre, j'identifierai ce répertoire par le nom logique **`$CATALINA_HOME`** (la notation de variable d'environnement n'est pas accidentelle).

Créer l'arborescence de base pour une application Web

Tomcat a des attentes envers la manière dont sera organisée l'arborescence de répertoires pour les applications Web placées sous sa gouverne. Pour être en mesure de déployer nos applications Web avec Tomcat, il nous faudra donc combler ces attentes.

Il faut tout d'abord savoir que les applications Web sous Tomcat se retrouvent (par défaut) toutes sous **`$CATALINA_HOME/webapps/`** de manière à ce que chaque application ait droit à son propre répertoire, où elle pourra placer ses interfaces, ses servlets, ses fichiers accessoires et ainsi de suite. Par défaut, un Tomcat fraîchement installé contient un certain nombre d'applications de démonstration placées sous ce répertoire.

Imaginons donc que nous ayons une application Web en vue et que nous ayons fait le choix de lui donner le nom **`petits_servlets`**. L'arborescence que nous devrons mettre en place ira comme suit.

Répertoire	Rôle
`$CATALINA_HOME/webapps/petits_servlets/`	On y déposera les pages `JSP` et `HTML` racines de `petits_servlets`. Si nous souhaitons créer des répertoires pour contenir divers médias associés à nos pages Web, c'est dans ce répertoire que nous les créerons[33]
`$CATALINA_HOME/webapps/petits_servlets/WEB-INF/`	On y déposera le fichier de configuration Web de notre application Web, fichier qui sera nommé **`web.xml`** (nous y reviendrons sous peu).
`$CATALINA_HOME/webapps/petits_servlets/WEB-INF/classes/`	On y déposera les fichiers .class de nos servlets. Notez que les fichiers seront en fait dans des sous-répertoires de ce répertoire et que les noms de sous-répertoires utilisés dépendront des noms de paquetages utilisés pour les servlets (nous y reviendrons).
`$CATALINA_HOME/webapps/petits_servlets/WEB-INF/lib/`	On y déposera les archives `WAR` (pour *Web Archives*), qui sont en fait des fichiers `JAR` mais orientés vers le Web[34].

Le nom **`WEB-INF`** et les noms **`WEB-INF/classes`** et **`WEB-INF/lib`** doivent être respectés à la lettre, incluant le respect des majuscules et des minuscules.

---

[33] Par exemple, `$CATALINA_HOME/webapps/petits_servlets/images/` pourrait contenir les images des pages Web placées dans `$CATALINA_HOME/webapps/petits_servlets/`

[34] Voir `http://java.sun.com/developer/technicalArticles/Servlets/servletapi/index.html` pour des détails. Les archives `WAR` et les archives `JAR` sont toutes deux créées avec `jar -cvf` ce qui démontre qu'il s'agit en fait d'une seule et même chose.

Contextualiser une application Web dans Tomcat

La configuration au démarrage de Tomcat est décrite dans le document XML nommé **$CATALINA_HOME/conf/server.xml**. Pour intégrer notre application Web nommée petits_servlets dans Tomcat, il est nécessaire d'insérer dans ce fichier de configuration un élément tel que :

```
<Context path="/petits_servlets" docBase="petits_servlets" debug="0" reloadable="true" />
```

La documentation de Tomcat n'est pas claire pour ce qui est de l'endroit où insérer cet élément, mais chez moi j'ai choisi de le placer à l'intérieur de l'élément <Server> et le tout a fonctionné sans problème.

Si Tomcat ne connaît pas le contexte de votre application Web, alors il fera comme si cette application n'existe pas (car, pour lui, c'est effectivement le cas).

Je n'ai pas fait de tests exhaustifs à ce sujet, mais la documentation indique qu'on pourrait placer une application Web n'importe où sur disque en changeant la valeur de l'attribut docBase de l'élément Context ci-dessus (par exemple, remplacer "petits_servlets" par "c:\AppWeb\petits_servlets"). Je doute fort que ce soit une pratique recommandable, en toute honnêteté, mais ça demeure une option.

Penser une application Web

Maintenant que la structure de base est en place, nous devrons penser notre application Web. Je construirai cet exemple en me basant sur un petit exemple d'introduction du site OnJava.com de O'Reilly[35].

Notre application permettra à un usager d'entre un nom et un mot de passe. Lorsque l'usager aura sélectionné le bouton Envoyer (ou Submit si votre fureteur est de langue anglaise), nous invoquerons un servlet nommé **ServiceCoucou** qui recevra le nom et le mot de passe. En réponse à cette action de l'usager, dans le but de garder le tout aussi simple que possible, nous afficherons dans le fureteur une nouvelle page Web dans laquelle nous injecterons du texte généré par programmation dans le servlet.

Les fichiers de code que nous rédigerons seront la page par laquelle nous accueillerons l'usager (nommée **accueil.jsp**), la page qui sera produite suite à l'action du servlet (**coucou.jsp**) et le servlet en tant que tel (**ServiceCoucou.java**). À ceci s'ajoutera un fichier de configuration de l'application Web (fichier **web.xml**, placé dans le répertoire WEB-INF de notre petite application Web).

Cette application montera comment il est possible de faire travailler en collaboration un volet présentation (les pages JSP) et un volet code en arrière-plan (les servlets).

[35] Voir http://www.onjava.com/pub/a/onjava/2001/03/15/tomcat.html pour examiner l'exemple sur lequel je me suis basé.

Page d'accueil `accueil.jsp`

La page d'accueil, nommée `accueil.jsp`, sera telle que proposé à droite.

Les éléments pertinents sont en gras. Remarquez :

- les noms attribués aux champs en entrée dans la page Web (balises `<input>`). Ces noms (`"username"` et `"password"`) seront utilisés par le servlet pour comprendre la requête émise par la page Web;
- le contrôle permettant d'envoyer la requête (balises `<input>` de type `"submit"`); et, surtout
- l'attribut `action` du formulaire, qui indique le nom qualifié du servlet à invoquer lors de la sélection du contrôle `"Submit"`.

Le lien entre la page `JSP` invoquant le servlet et le servlet lui-même se fera à l'aide de ces quelques champs. Les valeurs des zones de texte seront transmises au servlet correspondant à l'attribut `action` du formulaire.

```
<html>
  <head>
    <title>Petite application Web</title>
    <meta http-equiv="Content-Type"
          content="text/html; charset=iso-8859-1" />
  </head>
  <body bgcolor="#FFFFFF"
        onLoad="document.loginForm.username.focus()">
    <div align="center">
      <img src="images/H-Deb_Logo.png" />
    </div>
      <form name="loginForm" method="post"
            action="com.service_coucou.ServiceCoucou">
        <table align="center" border="0"
               cellspacing="5" cellpadding="5">
          <tr>
            <td>Nom d'usager:</td>
            <td><input type="text"
                 name="username"></td>
          </tr>
          <tr>
            <td>Mot de passe:</td>
            <td><input type="password"
                       name="password"></td>
          </tr>
          <tr><td colspan="2"><div align="center">
             <input type="Submit" name="Submit">
          </div></td></tr>
        </table>
      </form>
  </body>
</html>
```

Remarquez que cette page contient une image (balise `<img>` menant vers l'image à l'adresse `"images/H-Deb_Logo.png"`). Le répertoire où doit se trouver le répertoire `images/` dans ce cas-ci est `$CATALINA_HOME/webapps/petits_servlets/WEB-INF` qui est, implicitement, le répertoire courant de notre application Web.

Nous reviendrons un peu plus loin sur la notion de paquetages dans du code Java, la notation **`com.service_coucou.ServiceCoucou`** et les liens entre ces deux idées et la structure de répertoires de notre application Web.

Page de réponse `coucou.jsp`

La page de réponse à la requête, nommée `coucou.jsp`, aura la forme proposée dans l'exemple à droite.

L'élément pertinent est en gras. Le servlet générera des données nommées. L'une d'entre elles sera nommée `USAGER` et contiendra le nom par lequel l'usager sera connu. Remarquez la syntaxe qui place entre **`<%=`** et **`%>`** une expression Java (ici, le résultat de l'invocation de la méthode `getAttribute()` de l'objet `request` associé à la page)[36].

```
<html>
  <head>
    <title>R&eacute;ponse (coucou!)</title>
    <meta http-equiv="Content-Type"
          content="text/html; charset=iso-8859-1" />
  </head>
  <body bgcolor="#FF00FF">
    <div align="center">
      <strong>
        Coucou! Nous t'appellerons <em>
        <%= request.getAttribute("USAGER") %>
        </em>
      </strong>
    </div>
  </body>
</html>
```

Comme vous pouvez le constater, dans une page `JSP`, il est possible d'intercaler code `HTML` et contenu généré par programmation. Ceci permet donc, tel qu'annoncé plus haut, de séparer la forme (présentation, responsabilité de la page `JSP`) du contenu (tâche du servlet).

Le servlet `ServiceCoucou.java`

Un servlet est une classe Java dérivant au moins de `GenericServlet`. La plupart des servlets dérivent en fait de son enfant `HttpServlet` et réagissent à une requête `HTTP`.

Il est de bon aloi d'insérer un sevrlet dans un paquetage. Cela permet de réduire les risques de conflits de nom lors de déploiement à grande échelle ou sur Internet.

Ici, le nom de paquetage choisi sera `com.service_coucou`. Vous y reconnaîtrez d'ailleurs le nom utilisé dans l'attribut `action` du formulaire dans la page `accueil.jsp` (voir un peu plus haut).

Les paquetages dont sont extraites les classes et les interfaces pertinentes aux servlets sont `javax.servlet` et (pour nous) `javax.servlet.http`.

```
package com.service_coucou;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
```

[36] Pour d'autres types de balises `JSP`, voir `http://fr.wikipedia.org/wiki/JavaServer_Pages`

Notre servlet sera une classe nommée `ServiceCoucou`. Elle dérivera de la classe `HttpServlet`.

La page où sera projetée l'information générée par le servlet sera la page `coucou.jsp` située à la racine de notre arborescence d'application Web (d'où le `"/"` précédant son nom).

Les méthodes `init()` et `destroy()` permettent respectivement de prendre en charge le code d'initialisation et de nettoyage associé au servlet.

Remarquez le paramètre à `init()`, de type `ServletConfig` (voir section ***Cycle de vie d'un servlet***, plus haut, pour des détails).

```
public class ServiceCoucou
   extends HttpServlet {
   private static final String
      JSP_DESTINATION = "/coucou.jsp";


   public void init (ServletConfig config)
      throws ServletException {
      super.init(config);
      // code d'initialisation...
   }
   public void destroy() {
      // code de nettoyage...
   }
}
```

Ce servlet est un peu simpliste. Il ne s'occupe que des requêtes `POST` et traite d'éventuels `GET` comme des `POST` par souci de simplicité.

Les paramètres à une méthode `doGet()` sont au nombre de deux :

- une référence sur une instance de `HttpServletRequest`, décrivant la requête `HTTP` soumise au servlet; et
- une référence sur une instance de `HttpServletResponse`, décrivant le résultat du traitement effectué par le servlet et émis vers le demandeur.

```
public void doGet
   (HttpServletRequest request,
    HttpServletResponse response)
   throws ServletException,
          IOException {
   doPost(request, response);
}
```

Notez que la technique appliquée dans le code à droite est une pratique simple, répandue mais non recommandable. Discutez-en avec votre professeur.

Le code de la méthode `doPost()` a le même profil que celui de la méthode `doGet()`. Dans ce servlet, c'est ici que nous placerons l'essentiel du traitement de la requête `HTTP`.

Tout d'abord, nous extrayons les paramètres de la requête à l'aide de la méthode `getParameter()` de notre `HttpServletRequest`.

Nous réalisons ensuite un traitement sur ces paramètres (invocation de la méthode `quiEstCe()`). Dans ce cas, on parle d'un traitement bidon, mais peu importe.

S'ensuit une préparation des données qui seront émises vers la page `JSP` résultante, puis les données résultantes sont émises vers cette page.

```
public void doPost
    (HttpServletRequest request,
     HttpServletResponse response)
    throws ServletException, IOException {
    // Extraire des paramètres de la requête HTTP
    String NomUsager =
       request.getParameter("username");
    String MotDePasse =
       request.getParameter("password");
    // Traitement de la requête
    String Usager = quiEstCe(NomUsager, MotDePasse);
    // Préparation du résultat
    request.setAttribute("USAGER", Usager);
    // Compléter une page JSP avec cette requête
    ServletContext Contexte = getServletContext();
    RequestDispatcher Répartiteur =
       Contexte.getRequestDispatcher(JSP_DESTINATION);
    Répartiteur.forward (request, response);
}
```

Enfin, la petite méthode interne nommée `quiEstCe()` remplace, pour ce servlet de démonstration, une requête à une `BD` par le retour d'une valeur fixée *a priori*.

C'est ici que le code de travail réel (à prendre au sens *d'indépendant de l'infrastructure*) d'un servlet commencerait.

```
   private String quiEstCe
      (String NomUsager, String MotDePasse) {
      return "Zygomar";
   }
}
```

Le rôle du `RequestDispatcher`

L'interface `RequestDispatcher` enrobe une ressource (une page `HTML`, une page `JSP`, un servlet, peu importe) et offre des services pour communiquer avec elle. On y trouve deux services, nommés `forward()` (pour faire suivre une paire requête/ réponse vers la ressource) et `include()` (pour intégrer à une paire requête/ réponse le contenu d'une ressource).

Dans notre exemple, le répartiteur fait suivre la paire requête/ réponse vers la page `JSP` de destination. Cette dernière récupère l'attribut nommé `"USAGER"` de la requête et s'en sert en tant que contenu affichable (voir ***Page de réponse `coucou.jsp`***, plus haut, pour des détails).

La classe `HttpServletRequest`

La classe `HttpServletRequest` représente une requête `HTTP`. Entre autres choses, elle permet de consulter un `Cookie` préalablement déposé côté client et susceptible de décrire des données propres à la session de furetage en cours.

À travers une instance de `HttpServletRequest`, il est possible de consulter des valeurs dans l'en-tête de la requête `http` ou de consommer des paramètres insérés à même une `URL`. Aussi, une instance de `HttpServletRequest` permet à un servlet de connaître le détail de la session en cours. Le méthode pour ce faire est `getSession()`, qui retourne une référence sur une instance de `HttpSession`. Une session `HTTP` permet de décrire le comportement courant de furetage du client à travers plusieurs pages.

La classe `HttpServletResponse`

La classe `HttpServletResponse` représente une réponse `HTTP`. Entre autres choses, elle permet de déposer un `Cookie` côté client pour tenir à jour des données propres à la session de furetage en cours.

À travers une instance de `HttpServletResponse`, il est possible d'insérer des valeurs dans l'en-tête de la réponse `HTTP`, de rediriger un client vers une nouvelle `URL` ou encore d'obtenir un flux en sortie et d'y écrire du contenu arbitraire (ce qui permet de générer des pages Web purement par programmation).

Compiler un servlet

N'oubliez pas, lors de la compilation d'un servlet, de joindre l'archive `JAR` nommée `j2ee.jar` à la compilation. Par exemple, si cette archive est dans `c:/j2ee/`, vous pouvez inscrire lors de la compilation

```
javac -extdirs "c:/j2ee/" monservlet.java
```

La configuration `WEB-INF/web.xml`

Une fois l'arborescence de répertoires de l'application Web construite, l'interface personne/machine (à partir de pages `JSP`) bien en place et le code de notre servlet correctement compilé, il nous faut intégrer un descripteur `XML` de note application Web à son répertoire `WEB-INF`.

Le fichier XML de configuration d'une application Web Java déployée sous Tomcat doit être nommés **web.xml** et se trouver dans son répertoire WEB-INF.

Un fichier web.xml correct pour notre petite application Web de démonstration apparaît à droite. Portez attention aux sections en caractères gras.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
  <display-name>Un petit servlet</display-name>
  <session-timeout>30</session-timeout>
  <servlet>
    <servlet-name>ServiceCoucou</servlet-name>
    <servlet-class>com.service_coucou.ServiceCoucou</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServiceCoucou</servlet-name>
    <url-pattern>/com.service_coucou.ServiceCoucou</url-pattern>
  </servlet-mapping>
</web-app>
```

L'élément racine de ce document XML est **<web-app>**. L'élément <display-name> est surtout décoratif mais permet au serveur Web d'offrir une interface administrative plus descriptive pour notre application.

Pour chaque servlet, on devrait trouver un élément **<servlet>** en décrivant les paramètres. Vous retrouverez évidemment le nom bref et le nom pleinement qualifié (basé sur son nom de paquetage) de notre petit servlet. L'élément <load-on-startup> permet de charger le servlet au démarrage du serveur Web; s'il est absent ou s'il vaut zéro, alors le servlet ne sera chargé que sur demande d'un client. D'une manière ou d'une autre, le servlet, une fois chargé en mémoire, le restera.

Pour chaque servlet, on devrait aussi trouver un élément **<servlet-mapping>** pour faciliter au serveur Web la tâche de retrouver correctement le fichier .class du servlet. En théorie, ce n'est pas nécessaire, mais en pratique on en a presque toujours besoin.

Déposer le servlet au bon endroit

Le répertoire où doit être déposé un servlet dépend de la racine de l'application Web à laquelle il contribue, mais aussi de son paquetage.

Dans notre cas, au moment du développement, il fut choisi de déposer le servlet dans le paquetage **com.service_coucou** ce qui signifie que le répertoire où sera déposé le fichier .class de ServiceCoucou sera :

```
$CATALINA_HOME/webapps/petits_servlets/WEB-INF/classes/com/service_coucou/
```

Une fois le servlet au bon endroit, il ne reste plus qu'à redémarrer Tomcat.

Exercices—Série 01

EX00—Présenter dans une page Web le résultat d'une requête SQL réalisée sur le poste serveur par un servlet à partir de paramètres soumis par un formulaire JSP.

EX01—Modifier EX00 pour que les données générées soient en format XML et que la présentation dans la page générée soit formatée à l'aide de règles de formatage XSLT.

Langages de scripts

Il existe plusieurs guerres de clochers d'un point de vue philosophique comme d'un point de vue technique entre des tenants dogmatiques de diverses approches de programmation :

- les *aficionados* des langages portables sur le plan des sources (par exemple C++) et ceux des langages portables au niveau des binaires (langages de scripts, Java, langages `.NET`);
- les *aficionados* des langages surtout fonctionnels (Haskell, Lisp, JavaScript, *etc.*) et les amants des langages surtout `OO` (C++, Java, C#);
- les *aficionados* des langages surtout dynamiques (Ruby, Python, plusieurs langages fonctionnels) et ceux qui privilégient les langages statiques (C, C++, les langages semi interprétés comme Java et les langages `.NET`);
- les *aficionados* de dialectes (langages à accolades, langages à parenthèses, langages semblables à ceux de Niklaus Wirth);
- les *aficionados* des langages `OO` à héritage simple seulement (Java, langages `.NET`), ceux qui préfèrent l'option de l'héritage multiple (C++, Eiffel) et ceux qui expriment l'héritage comme un idiome de programmation (JavaScript, Lisp);
- les *aficionados* de langages à collecte automatique d'ordures et ceux qui privilégient les comportements déterministes;
- les *aficionados* de langages riches et mots clés et en objets déjà faits (langages `.NET`, Java, et en général les langages qui viennent avec un *Framework*) et ceux des langages qui privilégient moins de mots clés, plus d'expressivité et les bibliothèques (C/C++); *etc.*

Pourtant, le monde des `TI` a besoin de plusieurs outils parce que ses défis sont nombreux et diversifiés. En effet :

- quand le problème exige vitesse et clarté ou quand il sort des sentiers battus, nous sommes heureux d'avoir accès à C++ et à la métaprogrammation;
- quand le problème demande une solution réalisée rapidement et s'exécutant à une vitesse raisonnable, ou quand il ressemble à des problèmes types, les langages à *Framework* comme Java et C# permettent de relever le défi à pied levé;
- quand un problème demande principalement de la souplesse et du dynamisme, les langages dynamiques et les langages de scripts sont fréquemment la solution la plus pertinente.

Même dans une situation où le souci de performance est omniprésent (simulation, réalité virtuelle, jeu vidéo, systèmes en temps réel), on voudra souvent combiner plus d'une approche pour arriver à nos fins. La trame narrative d'un jeu s'exprime bien avec un langage de scripts alors que le moteur se prête à de la programmation C++ très serrée. Un service Web construit en `VB.NET` se développe sans peine et peut cacher à la fois un composant C++ pour les calculs les plus exigeants et un relais vers une application Java interfaçant avec des technologies pour lesquelles `VB.NET` est moins utile (tout ce qui sort du monde *Windows*).

N'adhérons donc pas aux dogmes et aux chicanes de basse-cour et explorons brièvement les langages de scripts qui permettent de dynamiser (principalement) la partie cliente (souvent dans un fureteur) des applications Web.

Les langages de scripts sont nombreux[37]. Parmi les plus populaires, on trouve PHP, Perl, Python, Ruby, JavaScript, Tcl/Tk et bien d'autres (Adobe Flash[38], dans une certaine mesure, avec ActionScript[39]). Chacun a sa raison d'être et ses adhérent(e)s.

JavaScript/ ECMAScript

Le langage ECMAScript, connu sous plusieurs noms, en particulier sous sont nom d'origine qu'est JavaScript, est l'un des **langages de scripts**[40] les plus populaires dans le monde merveilleux du Web.

À peu près tous les fureteurs commercialement pertinents supportent au moins un dialecte de ce langage.

Dans ce document, j'utiliserai surtout la dénomination JavaScript, mais retenez que ECMAScript est le fruit de la standardisation de JavaScript (et de JScript et de Mocha et de LiveScript et…)[41].

Un langage de script est habituellement un langage interprété, donc traduit ligne par ligne ou instruction par instruction en code exécutable plutôt que compilé une fois pour être exécuté sur une plateforme donnée.

Dans les cas qui nous intéressent, les interpréteurs sont les fureteurs Web ou des modules qui leur sont ajoutés.

Malgré son nom, JavaScript n'est pas un dialecte de Java ou un sous-ensemble de Java. Les similitudes syntaxiques de ce langage avec Java ressemblent à celles qui existent entre Java et C ou entre C# et Java. Les deux langages ne viennent même pas d'une souche commune, Java étant un produit de *Sun Microsystems* alors que JavaScript fut, à la base, mis au point par Brendan Eich de Netscape[42] (aujourd'hui chez Mozilla[43]).

Le suffixe *Script* tend à faire croire que JavaScript (que nous devrions plutôt nommer ECMAScript aujourd'hui) est un langage jouet, mais il se trouve en fait que ce langage entre dans le créneau de ce qu'on nomme les **langages dynamiques**[44], créneau où on retrouve aussi des langages comme Lisp, Smalltalk, Python, Ruby ou PHP.

Tout comme Lisp, JavaScript entre aussi dans la catégorie des **langages fonctionnels**[45]. À plusieurs égards, d'ailleurs, JavaScript est plus près de Lisp que de Java.

---

[37] Voir http://www.devsource.com/print_article2/0,2533,a=148207,00.asp pour une entrevue avec les concepteurs de certains de ces langages.

[38] Voir http://en.wikipedia.org/wiki/Macromedia_flash

[39] Voir http://en.wikipedia.org/wiki/ActionScript

[40] Voir http://en.wikipedia.org/wiki/Scripting_language

[41] Voir http://en.wikipedia.org/wiki/ECMAScript

[42] Voir http://en.wikipedia.org/wiki/JavaScript ou encore le site fortement pro-JavaScript à l'adresse http://www.crockford.com/javascript/

[43] Pour la vision qu'avait Brendan Eich (en mai 2006) du futur du Web et du rôle de JavaScript (JS2, en fait) dans ce futur, voir http://developer.mozilla.org/presentations/xtech2006/javascript/

[44] Voir http://en.wikipedia.org/wiki/Dynamic_language ou l'article relativement détaille à l'adresse http://activestate.com/Company/NewsRoom/whitepapers_ADL.plex

[45] Voir http://en.wikipedia.org/wiki/Functional_programming

Les langages dynamiques ont ceci de particulier que leur structure tend à être fluide et changeante, même une fois qu'un programme a été lancé. Selon les langages, il peut être possible de modifier le système de types, ajouter des classes, ajouter des fonctions et ainsi de suite. En général, pour être en mesure de permettre de telles manœuvres, les langages de scripts offrent aussi un mécanisme d'introspection permettant de décrire le langage dans ses propres termes (les programmeuses et les programmeurs Java et `.NET` reconnaîtront une similitude avec la réflexion dans ces langages).

Le caractère interprété et dynamique de JavaScript le rend particulièrement propice à l'injection de comportements dynamiques dans une page Web, donc à l'accroissement de l'expérience de navigation sur Internet. Le fait que les langages dynamiques et, de manière générale, les langages de scripts soient pour la plupart interprétés implique aussi que les programmes écrits dans ces langages soient portables, une qualité indéniable pour le déploiement à travers Internet.

Java, dans sa déclinaison `1.6`, offrira un paquetage (`javax.script`) permettant d'interagir avec le code JavaScript sur une page Web. Plusieurs outils de Google procèdent déjà ainsi. JavaScript sert aussi hors des pages Web mais c'est à la lueur du monde Internet que ce langage est le plus connu.

Déverminer du code JavaScript

JavaScript a mauvaise réputation en partie parce qu'il est réputé difficile à déverminer. Ce n'est pas tout à fait faux, historiquement du moins, mais les outils à notre disposition aujourd'hui font en sorte que déverminer un programme JavaScript soit une tâche beaucoup plus raisonnable et beaucoup moins pénible qu'auparavant.

Certains outils en ligne[46] permettent de valider le code JavaScript tel quel : on copie le code dans une fenêtre et une liste de messages d'erreurs est produite en retour. Autre outil pratique : l'explorateur `DOM` et la console d'erreurs de certains fureteurs qui donnent des messages significatifs et font des liens avec les éléments pertinents de la page Web.

Comment fonctionne JavaScript dans une page Web

Dans une page Web, le code JavaScript peut être intégré directement dans le texte de la page et est en mesure d'interagir avec le modèle `DOM` du document qu'elle décrit. Son côté script est mis en valeur par le fait que le code dans la page, exécuté par le fureteur, peut réagir aux éléments de la page et les modifier.

Certains individus désactivent JavaScript dans leur fureteur, souvent à cause de problèmes de sécurité détectés avec leur technologie de prédilection. Règle générale, il est préférable d'avoir un plan `B` lorsqu'on a recours à JavaScript dans une page Web pour que celle-ci demeure utilisable même si cette particularité a été désactivée.

Les cas les plus simples de recours à JavaScript, traditionnellement, incluent la génération de fenêtres modales d'avertissement (des *popups*), les changements d'états de contrôles sur la page alors que la souris se déplace sur eux ou la validation des entrées de l'usager dans un formulaire.

[46] Pensez à `http://www.jslint.com/` ou à `FireBug` (`http://getfirebug.com/`)

Cela dit, JavaScript est un véritable langage de programmation, très polyvalent, ce qui rend la tâche de sécuriser les fureteurs foncièrement plus complexe qu'il n'y paraît *a priori*. Certaines attaques (comme les célèbres manœuvres de *Cross-Site Scripting*[47], ou XSS) qui permettent, surtout avec certains fureteurs plus fragiles, la réalisation d'opérations dangereuses comme des accès au disque local d'un ordinateur ou la capture d'informations privilégiées[48].

Le code JavaScript est simplement du code intégré à une page Web à l'intérieur d'une balise <script> correctement construite.

Ce faisant, la page Web devient un volet présentation dans lequel s'insère un programme sujet à être interprété (donc exécuté) par une machine virtuelle : le fureteur.

Étant intégré à même la page Web, le script a accès à la description DOM dans le fureteur du document qu'elle représente et peut agir sur celle-ci, en lecture et en écriture.

```
<html>
  <head>
    <title>Titre</title>
    <script type='text/javascript'>
      // le code va ici
    </script>
  </head>
  <body>
    <!-- le code de la page Web -->
  </body>
</html>
```

Nous verrons plus loin, dans la section ***JavaScript discret***, une technique en vogue pour séparer le code du script du code de la page Web en tant que tel.

Accéder à un élément de la représentation DOM d'un document[49]

Dans un document HTML, les éléments peuvent être qualifiés par un nom, un identifiant, une classe[50] et ainsi de suite.

Par exemple, dans l'extrait proposé à droite, on trouve entre autres une balise input dont la classe est text (standard une zone de texte) et dont l'identifiant est resultat.

On trouve aussi un bouton de commande sur lequel apparaît le texte Additionner qui, lorsqu'il aura été sélectionné, invoquera la fonction JavaScript nommée calculerSomme().

```
<input class="text" id="val_a" />
<input class="text" id="val_b" />
<button onclick="calculerSomme();">
  Additionner
</button>
<input class="text" id="resultat" />
```

[47] Voir http://en.wikipedia.org/wiki/Cross-site_scripting qui inclut entre autres la description de quelques cas réels d'assauts du genre (avec liens). Pour un cas vécu d'assaut par XSS, voir http://www.webmonkey.com/webmonkey/00/18/index3a.html

[48] Souvenons-nous que JavaScript interagit avec le DOM d'un document, alors si une manœuvre de programmation parvient à injecter du code JavaScript discrètement dans une page conçue imprudemment, il peut devenir possible d'envoyer des données confidentielles à un tiers hostile sans que l'humain devant le fureteur n'en soit conscient.

[49] Voir http://www.yourhtmlsource.com/javascript/objectsproperties.html pour plus d'information à ce sujet.

[50] Nous reviendrons sur les classes quand nous discuterons de CSS, ce sera plus approprié.

Comment la fonction JavaScript pourra-t-elle savoir quelles sont les valeurs dans les zones de texte portant les identifiants `val_a` et `val_b` et comment sera-t-elle en mesure d'écrire dans la zone de texte portant l'identifiant `resultat`?

La réponse est somme toute simple :

- la représentation `DOM` du document est (au sens de JavaScript) un objet nommé `document`; et
- la méthode `getElementById()` permet de retrouver un élément par son identifiant (présumé unique dans la page); ainsi
- il suffit à la fonction de saisir les valeurs de ces deux éléments, de les manipuler et de déposer le résultat dans la valeur de l'élément portant l'identifiant `resultat`.

```
function calculerSomme () {
  var x = new Number(
    document.getElementById ("val_a").value
  );
  var y = new Number(
    document.getElementById ("val_b").value
  );
  var somme = x + y;
  document.getElementById("resultat").value = somme;
}
```

JavaScript et les événements

Au préalable, le principal rôle de JavaScript était de valider le contenu des formulaires Web avant émission. Cette validation ne remplace pas la validation côté serveur (nécessaire parce que le monde demeure un milieu hostile) mais peut par exemple permettre d'éviter l'envoi d'un formulaire dont certains champs obligatoires n'ont pas été remplis.

Présumant l'ébauche de code à droite, l'envoi des données contenues entre les balises `<form>` et `</form>` impliquera au préalable l'invocation de la fonction JavaScript `valider()` qu'on présumera booléenne et dont le rôle sera de valider les données avant envoi.

```
<form action="operationCoteServeur"
      onsubmit="return valider()">
  <!-- peu importe -->
</form>
```

Si la fonction retourne `false` ou quelque chose qui, en JavaScript, équivaut au concept de *faux*, alors il n'y aura pas du tout d'envoi de données vers le serveur, économisant ainsi temps de traitement et bande passante.

La fonction `onsubmit` est la réaction à un événement de soumission de données du formulaire, tout comme la fonction `onclick` d'un bouton de commande est la réaction à un clic d'un bouton de commande. Pour la validation de données textuelles, il est fréquent que la méthode `onchange` soit remplacée par du code maison.

Règle générale, JavaScript permet de remplacer le code par défaut des divers événements susceptibles de se produire sur une page Web suite à une interaction avec un usager par du code développé et choisi par les conceptrices et les concepteurs de la page.

Un petit mais intéressant exemple[51] de recours à JavaScript pour dynamiser le contenu d'une page Web serait celui proposé à droite.

On y voit un exemple de style `CSS` nommé `exemple` qui a la particularité d'avoir un attribut nommé `display` dont la valeur initiale est `hidden`.

La principale conséquence de ce choix est que, dans la page représentée, tout élément dont la classe est `exemple` ne sera pas affiché.

La fonction JavaScript nommée `modifierAffichage()` fera passer l'état de l'attribut `display` de `none` à `block` (dans une boîte) et inversement.

Nous obtenons ainsi une page Web dont certains éléments seront affichés ou non selon les actions (et les besoins) d'un usager.

```
<html>
  <head>
    <title>Petite d&eacute;mo</title>
    <style type="text/css">
      #exemple {
        display: none; margin: 3%; padding: 4%;
        background-color: limegreen
      }
    </style>
    <script type="text/javascript">
      function modifierAffichage (id) {
        d=document.getElementById("cacher-ou-non");
        e=document.getElementById(id);
        if (e.style.display == 'none' ||
            e.style.display == "") {
          e.style.display = 'block';
        } else {
          e.style.display = 'none';
        }
      }
    </script>
  </head>
  <body>
    <div>
      <a id="cacher-ou-non"
         href="javascript:modifierAffichage('exemple')">
           Petit exemple chouette
      </a>
    </div>
    <div id="exemple">
      Un exemple pourrait aller ici
    </div>
    <p>Et ainsi de suite...</p>
  </body>
</html>
```

[51] Inspiré de `http://en.wikipedia.org/wiki/Dynamic_HTML`

Bases syntaxiques[52]

Ce qui suit s'applique à la version 1 de JavaScript (JS1). D'ici la fin de l'an 2007, une standardisation de la version 2 du langage (JS2) devrait être rendue disponible par ECMA.

Le langage JavaScript a plusieurs particularités. Tout d'abord, à l'instar de plusieurs langages dynamiques, *ce n'est pas un langage fortement typé* ce qui implique que ses variables sont nommées puis utilisées et que c'est le moteur du langage qui cherche à comprendre pour nous ce que nous cherchons à faire.

Ceci implique aussi qu'il n'y a pas, en JavaScript, de déclaration de variable à proprement dit. Dans ce langage, une variable existe dès qu'on la nomme et dès qu'on commence à l'utiliser. De plus, toutes les variables y sont globales sauf si elles sont préfixées de var lors de leur première utilisation (dans ce cas, elles sont locales au bloc où elles apparaissent). Cet énoncé demeure vrai que la variable soit initialisée dans une fonction ou hors de toute fonction.

Ainsi, le code à droite est légal du fait que val y est globale et est donc accessible aux deux fonctions.

```
function coucou() {
   val = 3;
   alert('Coucou!' + val);
}
function bonjour() {
   alert('Bonjour!' + val);
}
```

Un bémol, toutefois : si ce code invoque bonjour() avant la première invocation de coucou(), alors une erreur surviendra car val ne sera pas initialisée et l'appel à alert() (qui affiche une fenêtre modale d'avertissement) échouera.

Personnellement, je recommande des saines pratiques de programmation : réduisez les globales au minimum et utilisez var chaque fois que cela s'avère possible.

Comme dans bien des *langages à accolades*, chaque instruction et chaque déclaration est terminée par un ; et chaque bloc logique d'instructions est délimité par une paire d'accolades. Les commentaires suivent la même forme que ceux en C++ ou en Java. Le langage est sensible aux majuscules et aux minuscules.

Vrai ou faux

Le langage JavaScript étant dynamique, une convention est requise pour les opérations reposant sur l'évaluation d'un booléen.

Ainsi, sont considérés faux le littéral false, une chaîne vide, un NaN[53] et les constantes null et undefined (cette dernière étant la valeur des variables déclarées mais non initialisées). Toutes les autres valeurs sont considérées vraies au sens de l'algèbre booléenne.

Le code JavaScript est analysé de haut en bas par son interpréteur. Le code qui ne se situe pas dans une fonction est du code global qui sera exécuté dans l'ordre.

[52] Pour en savoir plus, voir http://en.wikipedia.org/wiki/JavaScript_syntax ou le (plus riche) http://developer.mozilla.org/en/docs/A_re-introduction_to_JavaScript, alors que de nombreux autres sites (par exemple : http://home.cogeco.ca/~ve3ll/jstutor0.htm et http://home.earthlink.net/~mafriedman/jscript.html, mais il y en a beaucoup d'autres) offrent des masses d'information et de trucs que je ne couvrirai pas dans cette introduction.

[53] Le mot NaN signifie *Not a Number*. Certaines séquences de bits ne peuvent correspondre à un nombre à virgule flottante dans un encodage comme celui d'IEEE dont tout le monde se sert, alors ces représentations sont considérées des NaN.

Un coût pas si caché des langages de script

Le fait que le texte (incluant les blancs entre les mots!) du code JavaScript accompagne la page Web à laquelle il s'applique peut ralentir les téléchargements de cette page. Plusieurs techniques sont possibles pour compresser le texte `HTML` transmis entre le serveur Web et le fureteur, incluant un transfert compressé (voir la documentation de votre serveur) ou le recours à des outils de transformation des sources comme par exemple `JSMin`, disponible à l'adresse `http://www.crockford.com/javascript/jsmin.html`.

Opérateurs

L'opérateur `+` sert à la fois aux additions numériques (`var x = 2 + 3;` dépose `5` dans `x`) et à la concaténation de texte (`var x = 2 + '3';` dépose `'23'` dans `x`).

Les opérateurs arithmétiques usuels (+, évidemment, de même que `-`, `*`, `/` et `%`) se comportent comme en Java ou en C++. L'affectation requiert l'opérateur `=`. Les opérateurs `++` et `--` (en version préfixée ou postfixée) et les opérateurs `+=`, `-=`, `*=`, `/=` et `%=` se comportent aussi comme en Java ou en C++.

Les opérateurs de comparaison habituels dans les langages à accolades (==, `!=`, <, <=, >, >=) fonctionnent selon les usages. Deux opérateurs s'ajoutent, soit `===` pour identiques (même valeur, même type) et `!==` (non identiques).

Les opérateurs logiques `&&`, `||` et `!` ont le même comportement qu'en C++ pour la majorité des cas (je ne couvrirai pas les subtilités ici mais il y a une subtilité autour de `&&` et `||`). Les opérateurs bit à bit <<, >>, `&` et `|` ont aussi le comportement attendu (et l'opérateur >>> sort complètement de notre champ d'intérêt). L'opérateur ternaire `?:` fonctionne tel qu'attendu.

Scalaires

Les nombres de JavaScript sont tous structurellement équivalent aux nombres à virgule flottante (double précision… des variables de type `double`) de C++ ou de Java. Ceci implique des risques d'imprécision lors d'affichages et lors de comparaisons (prudence!). Les notations habituelles en C++ ou en Java pour les littéraux numériques sont supportées (`3`, `3.14159`, `2.12e4`, `0xff`, `0177`) mais les variables contenant des nombres demeurent de format `double` peu importe la forme des littéraux qu'on leur affecte.

Le caractère non typé des variables entraîne des surprises. Conséquemment, soyez prudentes et prudents. Le code proposé à droite affichera `Coucou!34`, pas `Coucou!7`, du fait que `val` contient le texte `"3"` plutôt que la valeur `3`.

```
function coucou() {
   var val = "3";
   alert('Coucou!' + (val + 4));
}
```

Pour résoudre de tels imbroglios, on peut remplacer `val` par **`new Number(val)`** dans l'opération `(val + 4)`. Ainsi, `alert('Coucou!' + (new Number (val) + 4))` affichera `Coucou!7`. Le recours à l'instanciation d'un `Number` clarifie notre intention.

Pour la concaténation de chaînes, les formes `x = x + "texte"` et `x += "texte"` sont toutes deux absolument identiques.

Vous remarquerez que les notations `'allo'` et `"allo"` sont toutes deux supportées pour dénoter des chaînes de caractères. Dans une page Web, il arrive que l'une soit plus utile que l'autre selon les circonstances.

En JavaScript comme en Java, une chaîne de caractères est immuable et ne peut donc pas être modifiée une fois instanciée. Il est possible de lire les caractères un par un (en utilisant la notation indicée des tableaux) mais il est impossible de les remplacer.

L'exemple à droite affichera deux fois le texte `Coucou commence par un C`.

```
function coucou() {
   var Texte = "Coucou!";
   var PremièreLettre = Texte[0];
   alert(Texte + " commence par un " +
         PremièreLettre);
   Texte[0] = "W"; // illégal
   alert(Texte + " commence par un " +
         PremièreLettre);
}
```

Structures de contrôle

Les alternatives (`if`), les répétitives (`while`, `do ... while` et `for`) et les sélectives (`switch`) ont le même comportement qu'en C++ ou en Java, à ceci près qu'une sélective peut travailler avec des chaînes de caractères comme avec des nombres.

Une forme un peu particulière (`for (x in E) { ... }`) permet d'itérer à travers les clés d'un tableau.

Les exceptions ont la même forme qu'en Java (`try`, `catch`, `finally`).

On peut lever toutes sortes d'objets en JavaScript. Il est d'ailleurs fréquent que de simples chaînes de caractères soient levées en tant qu'exceptions.

L'exemple de `invoquerDiviser()` proposé à droite sert de paravent à la fonction `diviser()` qui est susceptible de lever une exception si son dénominateur est nul. Bien que le code puisse être simplifié (et de beaucoup), il montre comment :

- lever une exception (dans la fonction `diviser()`);
- mettre en place des blocs `try`, `catch` et `finally` (à l'intérieur de la fonction `invoquerDiviser()`); et
- utiliser l'exception attrapée par un bloc `catch`.

Certains fureteurs permettent de spécialiser les cas d'exceptions mais, à ma connaissance, cette tolérance ne fait pas partie du standard du langage.

```
<html>
   <head>
      <title>Division</title>
      <script type='text/javascript'>
function diviser(num, denom) {
  if (denom == 0) {
    throw "tentative de division par zéro!";
  }
  return num / denom;
}
function invoquerDiviser() {
  var numérateur = new Number(
    document.getElementById ("numerateur").value
  );
  var dénominateur = new Number(
    document.getElementById ("denominateur").value
  );
  var quotient;
  var résultat;
  try {
    quotient = diviser(numérateur, dénominateur);
    résultat = quotient;
  } catch (e) {
    résultat = e;
  } finally {
    document.getElementById("quotient").value =
       résultat;
  }
}
      </script>
   </head>
<body>
  <input class="text" id="numerateur"></input>
  <input class="text" id="denominateur"></input>
  <button onclick="invoquerDiviser();">
    Diviser
  </button>
  <input class="text" id="quotient"></input>
</body>
</html>
```

Tableaux

Les tableaux de JavaScript sont des tableaux associatifs, un peu comme le sont les `std::map` de C++[54]. Cela implique qu'on puisse attribuer des valeurs à des positions logiques représentant des clés.

L'exemple à droite utilise des clés numériques, un peu comme dans un tableau classique.

```
function coucou () {
   var Ménagerie = new Array(3);
   Ménagerie[0] = "chats";
   Ménagerie[1] = "lapins";
   Ménagerie[2] = "poissons";
   if (Ménagerie.length > 0) {
      var Texte = "Dans ma maison, il y a: ";
      for (i in Ménagerie) {
         Texte = " " + Texte + "des " + Ménagerie[i];
      }
      alert(Texte);
   }
}
```

Remarquez au passage la boucle `for` qui itère sur les clés (pas sur les valeurs) du tableau. Une notation plus compacte pour initialiser des tableaux serait celle proposée à droite.

```
function coucou() {
   var Ménagerie = ["chats", "lapins", "poissons"];
   var Texte = "Dans ma maison, il y a: ";
   for (i in Ménagerie) {
      Texte = " " + Texte + "des " + Ménagerie[i];
   }
   alert(Texte);
}
```

L'exemple à droite utilise quant à lui des clés textuelles et fonctionne précisément de la même manière (et avec le même effet) que le code de l'exemple précédent.

On remarquera qu'en JavaScript, itérer sur des clés texte ou sur des clés numériques revient à réaliser la même opération.

Les notations possibles pour opérer sur des tableaux sont nombreuses; référez-vous à la documentation en ligne pour des exemples plus riches que ce que permet ce bref survol.

```
function coucou () {
   var Ménagerie = new Array(3);
   Ménagerie["félins"] = "chats";
   Ménagerie["lagomorphes"] = "lapins";
   Ménagerie["marins"] = "poissons";
   if (Ménagerie.length > 0) {
      var Texte = "Dans ma maison, il y a: ";
      for (i in Ménagerie) {
         Texte = " " + Texte + "des " +
                 Ménagerie[i];
      }
      alert(Texte);
   }
}
```

[54] En fait, les tableaux JavaScript sont implémentés sous la forme de tables de hashage.

Fonctions

JavaScript est un langage dynamique et fonctionnel, au sens où les fonctions y sont des entités à part entière.

On peut déposer une fonction dans une variable (ce qui n'est *pas* la même chose que déposer le résultat de l'invocation d'une fonction dans une variable), par exemple, ou écrire une fonction qui retourne des fonctions.

L'exemple à droite montre une page Web (vraiment très simple) dans laquelle est définie la fonction `factorielle()` utilisant des éléments du document tel que perçu par le fureteur en tant qu'intrants et en tant qu'extrants.

Par souci d'esthétique (et non pas par obligation), j'ai choisi d'écrire une fonction `factorielle()` réutilisable et une autre fonction, liée plus spécifiquement au document, pour l'invoquer.

```
<html>
  <head>
    <title>Factorielle</title>
    <script type='text/javascript'>
      function factorielle(n) {
        var Facto = 1;
        while (n > 0) {
          Facto *= n;
          --n;
        }
        return Facto;
      }
      function invoquerFactorielle() {
        var n = new Number(
          document.getElementById("valeur_n").value
        );
        var Facto = factorielle (n);
        document.getElementById("resultat").value = Facto;
      }
    </script>
  </head>
  <body>
    <input class="text" id="valeur_n"></input>
    <button onclick="invoquerFactorielle();">
      Factorielle
    </button>
    <input class="text" id="resultat"></input>
  </body>
</html>
```

On remarquera que le caractère dynamique du langage est mis en relief par la notation de la fonction : elle n'a pas de type (sinon celui de `function`), ses variables non plus (bien qu'on remarque le recours à une conversion explicite en `Number` pour éviter toute ambigüité[55]).

[55] Avec des multiplications comme dans le cas d'une factorielle, c'est probablement une opération superflue. Cela dit, sachant que l'opérateur `+` peut vouloir dire concaténer ou additionner selon le type de ses opérandes (type tel que perçu par le moteur exécutant le script, ne l'oublions pas), nous pourrions avoir de vilaines surprises si nous cherchions plutôt à faire une sommation.

Il est possible, tel qu'indiqué plus haut, d'affecter une fonction à une variable. Ainsi, l'exemple de code JavaScript à droite se lit comme suit :

- trois fonctions *nommées* (`choixParDéfaut()`, `choisirA()` et `choisirB()`) retournent chacune une fonction susceptible d'être exécutée éventuellement;
- la variable globale `Choix` reçoit initialement la valeur de `choixParDéfaut()` (qui est une fonction);
- invoquer `choixA()` ou `choixB()` dépose dans `Choix` une fonction *anonyme*, elle aussi susceptible d'être exécutée;
- la fonction `executer()`, enfin, invoque la fonction dans la variable `Choix` en en affiche le résultat dans une petite fenêtre.

Une conséquence de cette stratégie : les boutons `Choisir A` et `Choisir B` déterminent le plus récent choix fait mais le bouton `Éxécuter`, lui, invoque véritablement la fonction choisie.

```
<html>
   <head>
      <title>Test&mdash;00</title>
      <script type='text/javascript'>
        function choixParDéfaut() {
          return function() {
            return "Vous n'avez pas encore choisi!";
          }
        }
        function choisirA() {
          Choix = function() {
            return "Vous avez choisi A";
          }
        }
        function choisirB() {
          Choix = function() {
            return "Vous avez choisi B";
          }
        }
        Choix = choixParDéfaut();
        function executer() {
          alert(Choix());
        }
      </script>
   </head>
<body>
  <button onclick="choisirA();">
    Choisir A
  </button>
  <button onclick="choisirB();">
    Choisir B
  </button>
  <button onclick="executer();">
    &Eacute;x&eacute;cuter!
  </button>
</body>
</html>
```

Notez que comme en Java, les paramètres de fonctions en JavaScript sont passés par valeur mais les objets étant toujours manipulés indirectement, le passage en paramètre d'un objet revient dans la majorité des cas à un passage par référence.

Objets

```
function Coucou() {
   var ChicProf = {
      nom : "Patrice Roy",
      cours : {
         nom : "Applications Internet",
         sigle : "INF777"
      },
   };
   alert(ChicProf.nom + " est un chic prof de " +
         ChicProf.cours.sigle);
}
```

Les objets JavaScript possèdent des attributs nommés. Dans l'exemple à droite, l'objet `ChicProf` contient un `nom` et un `cours`, ce dernier contenant quant à lui un `nom` et un `sigle`.

On remarquera que le nom d'un attribut est séparé de sa valeur par un `:` et que les attributs sont séparés les uns des autres par des virgules.

La notation des tableaux est aussi applicable aux objets : dans l'exemple ci-dessus, `ChicProf.nom` et `ChicProf['nom']` sont équivalents. L'inverse n'est pas vrai (on ne peut pas appliquer la notation `.nom` à un tableau indicé par le littéral texte `'nom'`).

```
function Bête (nom, espèce, poids) {
  this.nom = nom;
  this.espèce = espèce;
  this.poids = poids;
}
function créerDragon (nom, couleur, poids) {
  return new Bête(
    nom, "Dragon " + couleur, poids
  );
}
function surprise () {
  var Zak = créerDragon("Zak", "rouge", 3000);
  alert (Zak.nom + " est un " + Zak.espèce +
         " de " + Zak.poids + " Kg.");
}
```

L'exemple à droite montre :

- un constructeur de `Bête` (notez le nom de la méthode et le recours explicite à `this`) prenant en paramètre un `nom`, une `espèce` et un `poids`;
- une fonction de fabrication de dragons nommée `créerDragon()` (qui utilise le constructeur de `Bête`... Notez le recours à new); et
- une fonction `surprise()` qui crée un dragon rouge nommé `Zak` et le décrit dans une petite fenêtre modale.

Le caractère dynamique de JavaScript mène à une fluidité structurelle qui pourra surprendre certaines et certains parmi vous. En effet, il est possible d'ajouter ou d'enlever dynamiquement des attributs à un objet existant.

Présumant que le constructeur de `Bête` et la fonction `créerDragon()` du code ci-dessus soient , invoquer `surprise()` dans le code à droite a l'effet suivant :

- créer le dragon `Zak` et en afficher le pédigrée;
- chercher à le faire voler (un échec puisqu'il n'a pas encore d'ailes);
- lui ajouter des ailes;
- chercher à le faire voler (un succès puisqu'il a maintenant de grandes ailes);
- lui couper les ailes (remarquez l'opérateur `delete` sur un attribut); et
- chercher à le faire voler (un échec puisqu'il n'a plus d'ailes).

Bien que JavaScript soit pourvu d'un ramasse-miettes, il est possible d'y détruire explicitement des objets ou des membres d'objets à l'aide de l'opérateur `delete` et de tester la présence d'un membre en le comparant avec la constante `null`.

```
// constructeur de Bête (inchangé)
// créerDragon() (inchangé)
function surprise() {
  var Zak = créerDragon(
    "Zak", "rouge", 3000
  );
  alert (Zak.nom + " est un " +
         Zak.espèce + " de " +
         Zak.poids + " Kg.");
  envol (Zak);
  ajouterAiles(Zak, 'grandes');
  envol (Zak);
  couperAiles (Zak);
  envol (Zak);
}
function ajouterAiles(bête, valeur) {
  bête.ailes = valeur;
}
function couperAiles(bête) {
  if (bête.ailes != null) {
    delete bête.ailes;
  }
}
function envol(bête) {
  if (bête.ailes != null) {
    alert(bête.nom + " peut voler");
  } else {
    alert(bête.nom + " n'a pas d'ailes!");
  }
}
```

Encapsulation[56]

Les membres attributs proposés plus haut étaient tous publics. Ceci ne suffit pas, en pratique, à assurer une forme stricte d'encapsulation dans un objet.

Pour déclarer un membre d'instance privé, il suffit de le spécifier avec `var` dans son constructeur. Ceci s'applique aux attributs comme aux méthodes.

L'exemple (simplifié) à droite montre comment placer un accesseur public simple pour le nom d'une `Bête` à même sa construction.

```
function Bête(unNom) {
  var nom = unNom;
  this.getNom = function() {
    return nom;
  }
  // ...
}
function surprise() {
  var Zak = new Bête("Zak");
  alert(Zak.getNom()); // etc.
}
```

[56] Voir `http://www.crockford.com/javascript/private.html` pour plus de détails.

Notez qu'en JavaScript, une méthode d'instance privée est une méthode déclarée dans le constructeur de l'objet auquel elle appartient (et donc invisible de l'extérieur).

Selon l'usage en JavaScript, la méthode `getNom()` d'une `Bête` est ce qu'on nomme une méthode privilégiée, en ce sens qu'elle a accès à la fois aux membres privés d'une `Bête` et qu'elle est accessible de l'extérieur (ayant été ajoutée à la `Bête` en question suivant la forme `this.getNom = function { ... }`).

Ce que JavaScript considère comme une méthode publique est une méthode ajoutée au prototype d'une classe.

Ainsi, la méthode `grogner()` d'une `Bête` dans l'exemple à droite est publique. On remarquera que les méthodes publiques au sens entendu en JavaScript sont ajoutées suite à la construction de l'objet et sont donc des compléments externes à cet objet. En particulier, elles n'ont pas accès aux membres privés.

Remarque le recours à `this` devant `getNom()` dans grogner, qui est nécessaire en JavaScript d'un point de vue syntaxique.

```
function Bête(unNom) {
  var nom = unNom;
  this.getNom = function() {
    return nom;
  }
  // ...
}
Bête.prototype.grogner = function(texte) {
  return this.getNom() + " fait " + texte;
}
function surprise() {
  var Zak = new Bête("Zak");
  alert(Zak.grogner("grrr")); // etc.
}
```

Encapsulation et immuabilité sans attributs

Une forme compacte (prise directement d'un exemple proposé par le concepteur du langage) de construction d'une instance immuable d'une classe `Point` serait celle visible à droite.

Notez qu'il s'agit d'un objet sans attributs à proprement dit, représenté par une fonction contenant des fonctions et qui est pleinement encapsulé.

```
function Point (x, y) {
  return {
    getX() : { return x; }
    getY() : { return y; }
  }
}
```

Cette stratégie a le défaut de coûter cher (en JavaScript, une représentation sous forme de fonctions comme celle-ci coûte environ le triple de la représentation équivalente sous forme d'attributs *pour chaque instance de* `Point`), mais cela ne devient vraiment important que si le nombre d'objets devient grand.

Héritage

JavaScript ne supporte pas l'héritage au sens classique du terme et ne supporte pas non plus le polymorphisme en ce sens. Cependant, son caractère dynamique fait en sorte qu'un peu comme dans les langages permettant la programmation générique, il soit possible d'implémenter en JavaScript des algorithmes génériques complexes sur la base des signatures des objets.

Le site `http://www.crockford.com/javascript/inheritance.html` explore en détail quelques approches pour développer des classes JavaScript se comportant à l'usage comme si elles avaient été implémentées à l'aide d'héritage et de polymorphisme (et même plus!). Le sujet, bien que très divertissant, dépasse largement les attentes du cours.

JavaScript discret[57]

Intégrer du code JavaScript directement à une page Web pose problème à plusieurs égards :

- la réutilisation du code devient une opération de copier/ coller;
- la séparation de la forme et du fond est compromise;
- le code est visible à tout individu examinant les sources de la page; *etc*.

La vision du Web comme plateforme de développement a plusieurs impacts sur les pratiques de développement qu'on y retrouve, incluant une standardisation des usages et une formalisation de certains vieux trucs sous forme de schémas de conception ou d'idiomes de programmation.

Plusieurs plateformes existent aujourd'hui pour formaliser une démarche plus programmatique que celle rencontrée traditionnellement dans les pages Web. Indiquer, par exemple, que tous les éléments du document qui sont munis d'une même classe ou d'un même nom soient soumis aux mêmes règles de validation (ce qui évite les accidents qui surviennent lorsqu'un humain doit insérer ces indications à la pièce).

La chose la plus simple que nous puissions faire, cela dit, pour séparer code JavaScript et présentation HTML est, tout simplement, de les séparer en fichiers distincts.

Ainsi, alors que la tradition suggère d'insérer le code JavaScript directement entre les balises <script> et </script> dans la page Web proposée à droite...

```
<html>
   <head>
      <title>Test</title>
      <script type='text/javascript'>
/* votre code JavaScript va ici */
      </script>
   </head>
<body>
  <!-- peu importe -->
</body>
</html>
```

… il est aussi possible de procéder comme proposé ici et de référer simplement à un fichier source JavaScript externe au code HTML.

On comprendra immédiatement l'intérêt de cette démarche : plusieurs pages Web peuvent partager un même support programmatique en référant aux mêmes fichiers de scripts, et la page Web elle-même en est allégée d'une masse de texte qui n'est pas essentielle à son rôle de présentation d'information.

```
<html>
   <head>
      <title>Test</title>
      <script type='text/javascript'
              language="javascript"
              src="moncode.js" />
   </head>
<body>
  <!-- peu importe -->
</body>
</html>
```

Il est possible de référer à plusieurs fichiers JavaScript dans une même page Web (il suffit simplement d'insérer plusieurs balises <script>, qui seront lues dans l'ordre).

[57] Les idées de cette section sont décrites plus en détail sur le site Wiki à l'adresse suivante : http://en.wikipedia.org/wiki/Unobtrusive_JavaScript

Exercices—Série 02

EX00—Insérez dans une page Web une procédure de validation assurant qu'un champ dans un formulaire soit bel et bien rempli avant envoi vers le serveur.

EX01—Insérez dans une page Web une procédure de validation assurant que tous les champs d'un formulaire soient remplis avant envoi vers le serveur (truc : construisez un tableau des id des contrôles à valider avant d'invoquer la méthode de validation).

EX02—Insérez dans une page Web une procédure remplaçant le texte de chaque zone de texte par son équivalent en minuscules avant validation.

EX03—Insérez dans une page Web une procédure faisant en sorte que les données numériques d'un champ soient inclusivement situées entre 0 et 100.

EX04—Insérez dans une page Web une procédure faisant en sorte que les champs devant obligatoirement être remplis soient identifiés par une couleur différente s'ils n'ont pas été remplis avant une tentative de soumission.

EX05—Réalisez un exercice dans la liste allant de EX00 à EX04 inclusivement sur une page générée (au moins en partie) par un Servlet.

Approche SOA

Poursuivons notre examen de divers aspects du monde des applications Web.

Modèle SOA

Il est fréquent qu'une entreprise, surtout si elle a un solide vécu dans le monde des TI, possède une gamme important de services existants, services à la fois efficaces, solides et rodés.

Typiquement, une gamme de services bien testés vaut de l'or à qui sait bien s'en servir. Parmi les options à la disposition des entreprises possédant de tels atouts, on trouve :

- étendre l'offre de services à une clientèle interne à l'entreprise (par exemple, offrir un service bien rodé à d'autres groupes ou à d'autres départements[58]);
- étendre l'offre de services à des partenaires commerciaux de l'entreprise, que ce soit en tant que *plus-value* dans la relation commerciale existante ou comme source de revenus supplémentaire; ou encore
- étendre l'offre de services au monde entier, que ce soit gratuitement (par exemple pour gagner en renommée) où à peu de frais (les *micropaiements* peuvent être une option intéressante si le volume transactionnel est élevé[59]).

Dans tous ces cas, on remarquera que le coût initial de développement est *a priori* payé. L'entreprise possède un atout testé et opérationnel, mais cherche maintenant à en amortir le coût ou à transformer cet atout en source de revenus, peu importe la forme.

Il est présumé ici que l'entreprise veut offrir ses services à d'autres, mais le passage d'un système pour usage interne à un système ouvert sur le monde peut simplement apparaître comme une étape normale de son évolution dans un processus s'apparentant à celui de l'urbanisation[60].

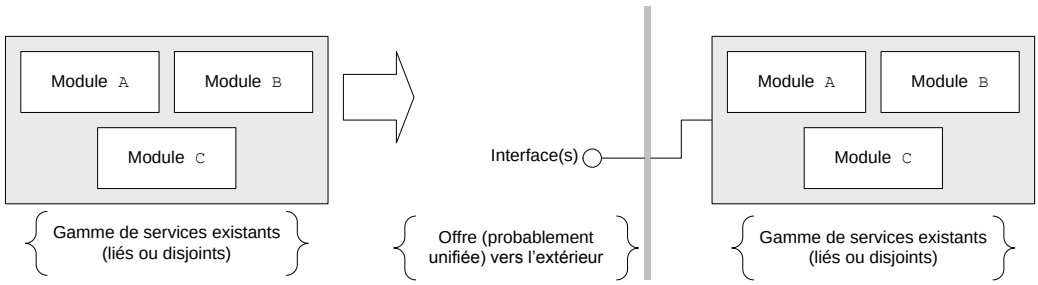
La question devient : comment offrir *le mieux possible* ces services aux demandeurs? L'entreprise qui choisit d'offrir des services existants sous une nouvelle forme a devant elle un investissement à faire pour la mise en place d'une nouvelle interface mais la présomption est que les modules derrière cette interface sont solides et que leur coût n'est plus un obstacle.

---

[58] Un étudiant de la cohorte 10 du DTI, monsieur ***Benoît Springuel***, m'a fait remarquer avec justesse qu'en français, une entreprise n'a pas de départements mais bien des services. Le recours à *départements* ici est un anglicisme. Le problème, dans ce cours, est que le mot *service* est porteur de plusieurs sens. J'ai donc choisi de conserver cet anglicisme mais en le reconnaissant comme tel.

[59] À titre d'exemple simpliste, quelques grandes entreprises (par exemple *Sun* et IBM) et quelques universités (McGill est un cas, sûrement pas isolé) sous-louent les services de leur parc informatique hors des périodes de pointe à faible coût (par exemple un dollar par processeur par heure)… ce qui peut résulter un des gains faramineux si les locataires ont d'importants besoins en temps de calcul.

[60] Voir http://fr.wikipedia.org/wiki/Urbanisation_%28informatique%29



Plusieurs façons de faire existent pour livrer des services, mais du point de vue du patron d'interaction entre le client (ceux pour qui les services sont exposés) et le serveur (celui qui expose les services), deux grandes familles prédominent :

- offrir des interfaces par composant (conceptuel). Selon cette stratégie, les utilisateurs (les clients de la relation) sont écrits un peu comme s'ils transigeaient avec des objets qui leur appartiennent, que ce soit vrai ou non; et

Le schéma de conception qui intervient habituellement à ce niveau est *proxy* (intermédiaire).

- offrir une[61] interface regroupant une gamme de services apparaissant un peu comme une `API` et permettant aux clients de transiger avec le système exposé à travers `cette API` comme s'ils invoquaient des sous-programmes.

Le schéma de conception qui intervient habituellement à ce niveau est *façade*.

Les **`SOA`**, ou **Architectures orientées services** (*Service-Oriented Architecture*), sont des systèmes client/ serveur au sens large. L'approche `SOA` ne présume pas d'intermédiaire particulier entre le client et le serveur et ne présume presque rien sur le plan des plateformes. Habituellement, les `SOA` sont présumés sans états (donc chaque requête faite par un client est présumée complète en elle-même et ne dépend pas d'une connaissance *a priori* du serveur).

Dans un `SOA`, les services sont habituellement présumés indépendants les uns des autres (chose mise en relief lorsque les transactions ne présument pas d'états côté serveur) et doivent pouvoir être accédés sans que le client n'ait connaissance de leur implémentation ou de la plateforme où les services résident.

L'effort d'intégration qui prédomine dans les `SOA` est donc un effort d'ouverture. Les `SOA` visent typiquement les standards les plus ouverts possibles pour faire en sorte que les services soient rendus disponibles au plus grand nombre de clients possibles.

Démystifier les `SOA`

Avec la publicité faite autour des `SOA`, on penserait qu'il s'agit d'une technologie novatrice ou d'une architecture d'intégration avancée, mais la réalité est bien plus simple : l'approche `SOA` est une philosophie, qui peut être implémentée à partir de plusieurs technologies distinctes.

Un `SOA` expose des services existants, faisant probablement partie de la gamme logicielle existante de l'entreprise, de manière potentiellement disjointe (les services offerts ne sont pas fortement couplé du point de vue des clients) et habituellement sans états (on ne présume pas que le serveur est le même à chaque invocation d'un même service).

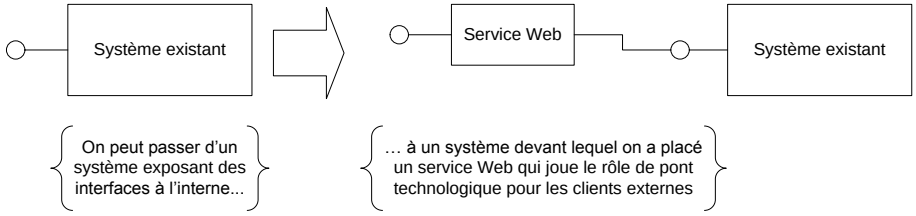
[61] …ou plusieurs interfaces, mais habituellement un petit nombre.

L'*invocation* d'un SOA est présumée faite à travers des technologies ouvertes, et qui dit technologies ouvertes aujourd'hui dit technologies Internet, ce qui explique le lien direct avec notre cours.

Clairement, les SOA visent le Graal de l'interopérabilité.

L'*implémentation* d'un SOA est quant à elle dépendante de ce qu'utilise l'entreprise à l'interne, qu'il s'agisse d'intergiciels comme COM ou CORBA, de services Web avec ou sans approche REST[62] ou autre chose.

La mise en relation des services existants avec les clients potentiels est le focus des SOA. On comprendra que le langage des applications Web y joue un rôle clé.



La standardisation de l'offre de services externe dans un SOA tend à mener les services à être exposés sous forme WSDL (nous y revenons plus bas), donc habituellement sous forme de services Web. Il n'est donc pas rare qu'un client interagisse avec un service Web qui, lui, fait le pont avec le système existant et les services déployés dans la compagnie selon les standards locaux.

Introduire un intermédiaire entre le service offert et le client est un cas particulier du schéma de conception ***proxy***.

Le coût du *marshalling*

On comprendra tout de suite que l'ajout d'un tiers intermédiaire entre le client et les services offerts, intrinsèque au modèle SOA, implique presque inévitablement un coût à l'invocation de chaque méthode.

En effet, le code client doit d'abord interfacer avec un service Web, ce qui implique formater sa requête de manière à ce que le service Web la comprenne (protocole SOAP ou autre), opération en soi coûteuse, puis transmettre cette demande sur un lien réseau. Le service Web lui-même doit être invoqué (démarré par le serveur Web si ce n'est fait), comprendre le message reçu et relayer, peut-être sous une autre forme, une requête équivalente à un service interne à la compagnie.

Chaque transformation de requête en message et chaque transformation de message en requête est ce qu'on nomme une étape de *marshalling*. L'introduction de *marshalling* dans une communication est un mal nécessaire mais ralentit considérablement les échanges (habituellement par un facteur avoisinant 10 quand les opérations sont faites à l'intérieur d'un même ordinateur et par un facteur bien pire quand les opérations transitent sur un lien réseau).

[62] Voir http://fr.wikipedia.org/wiki/Representational_state_transfer pour des détails mais, essentiellement, l'approche REST chante les louanges des services sans états. Certains l'opposent à SOAP mais les deux ne sont pas vraiment en opposition : REST est une philosophie qui peut être mise en application avec SOAP alors que SOAP est un protocole selon lequel le serveur peut avoir ou non des états.

Si les services exposés au monde extérieur ont une granularité très fine, alors l'enrobage de chaque petite requête impliquant du *marshalling* constituera l'essentiel de la bande passante consommée. Les services seront coûteux en ressources et le code client en contact avec le SOA sera lent au point de ne pas être utilisable concrètement.

La pratique veut donc que les interfaces des SOA offrent des **services à granularité grossière** : plutôt que d'offrir à un client des données à la pièce (par exemple des petits services exposant une donnée à la fois comme `getNom()`, `getÂge()` ou `getNip()`), on offrira souvent des données par blocs (le descriptif entier d'une personne par exemple). Ce faisant, le coût du *marshalling* d'une trame d'un protocole lourd comme SOAP sera amorti par la présence d'un contenu (un *Payload*) plus riche.

Grands principes des SOA

Pour réaliser un bon SOA, la recette prétend qu'un service doit être :

- sans états (nous y reviendrons);
- modulaire (ce qui va de soi);
- réutilisable (ce qui est naturel pour un service sans états);
- composable/ *orchestrable* (pour faciliter la construction de services complexes à partir de services plus simples);
- exposé de manière conforme aux standards d'interopérabilité en vigueur;
- localisables et découvrables (d'où les annuaires de services, voir plus loin);
- clairement documentés;
- indépendants de toute plateforme et de toute technologie (du moins du point de vue de son invocation);
- autonome (le service contrôle la logique qu'il encapsule); et
- performant (car soyons honnêtes : qui voudrait d'un service lent si un service plus efficace est disponible à coût équivalent?).

On remarquera que plusieurs des principes sous-jacents à la POO apparaissent aussi comme des principes sous-jacents aux services exposés selon une approche SOA.

Approche par composants

Si le service à exposer est *a priori* implémenté selon une approche par composants (technologie COM, CORBA ou EJB par exemple), il peut arriver que l'exposition de ces services au monde extérieur respecte en bonne partie le découpage des interfaces déjà existantes.

Imaginons donc un service permettant de calculer certains paramètres de la physique des fluides et utilisé dans un centre de recherche où sont réalisés des tests sur le plasma au cœur de la foudre[63]. Imaginons que ce service soit offert à travers un objet programmé en C++ par des spécialistes du domaine et exécuté sur un ordinateur à plusieurs processeurs très puissant.

[63] Exemple tout à fait hypothétique, bien entendu.

Il est probable qu'il y ait des périodes creuses dans l'utilisation de ce service et que le centre de recherche envisage l'exposer au monde extérieur sous forme locative pour rentabiliser en partie son effort de développement.

Ce cas est intéressant pour un `SOA` du fait que le système existe et que son exposition vise à amortir le coût de développement déjà encouru.

Il existe assurément une méthodologie pour invoquer le service à l'intérieur du centre de recherche et le rôle d'une interface standardisée placée au-dessus sera de déléguer correctement le travail d'un client externe vers un composant existant et opérationnel.

> J'escamote ici la question, plus complexe qu'il n'y paraît, de la synchronisation des services. En effet, certains services peuvent être offerts à tous, en tout temps. D'autres, comme celui-ci, perdent de leur pertinence si la demande est telle que le calcul devient long et lourd à réaliser.
>
> Notez que la solution à ce problème peut être logicielle mais est souvent humaine : les clients prospectifs louent des plages horaires et invoquent les services lorsqu'ils y sont autorisés.

Ici, il est probable que l'interface de l'objet réalisant les calculs convienne pleinement aux besoins d'éventuels clients scientifiques. Des ajustements quant à la granularité des paramètres d'invocation du service sont possibles, mais la nature des intrants et des extrants risque d'être facile à déterminer.

L'interface exposée au monde extérieur risque donc fortement de ressembler à l'interface interne. L'interface externe jouera surtout un rôle de clarification d'intention : là où l'objet interne a probablement tiré profit des technologies connues dans les groupes ayant développé le serveur à exposer, l'interface externe guidera les éventuels programmes clients pour leur permettre d'invoquer le service peu importe leurs technologies ou leurs plateformes de prédilection.

Approche par façade

Procéder par composants tend à révéler en partie la structure des systèmes de l'entreprise. Si une entreprise souhaite exposer plusieurs petits objets, il peut arriver que la gestion de ces objets obscurcisse l'offre de services. Dans ce cas, il est commun que l'on regroupe les services apparentés dans un agrégat sous une seule et même interface. C'est ce qu'on nomme une **façade**, schéma de conception bien connu.

La façade a plusieurs particularités, qui peuvent être des avantages si elles rejoignent les objectifs de l'entreprise exposant des services :

- elle dissimule les détails de structure interne de l'entreprise, du fait qu'une même façade peut cacher plusieurs objets distincts et que deux (ou plus) façades distinctes peuvent donner accès à des services distincts d'un même objet;
- elle donne une cohérence apparente à des services qui peuvent ne pas en avoir au préalable (par exemple à plusieurs services ayant été développés de manière indépendante à divers endroits dans l'entreprise), même si ces services résident sur des plateformes distinctes et ont été développés à l'aide de technologies *a priori* hétérogènes;
- elle offre un point de vue sur une gamme de services. Ce faisant, elle fait ressortir un service manifeste à partir d'un agrégat de services effectifs.

Exposer efficacement des services

En appliquant une approche SOA, l'entreprise veut exposer ses services au monde extérieur de façon technologiquement agnostique et de manière à leur donner une identité propre.

La gamme de services offerts à l'externe n'a pas à être conforme à la gamme de services utilisés dans l'entreprise. Certains services utilisés à l'interne peuvent simplement rester exposés à l'interne, rien de plus. Les services visibles à l'externe peuvent être une réorganisation, un réaménagement des services existants.

Certains services supplémentaires peuvent n'être présents que dans la couche exposant les services à l'externe (qu'il s'agisse de services comptables, de sécurisation, d'orchestration ou d'autres services qui ne prennent leur sens que dans le visage exposé par l'entreprise hors de son propre système).

Les principaux éléments d'une offre de services SOA sont l'*annuaire de services*, le *bus de services*, le *protocole d'accès aux services* et le *service* en soi. Dans une approche SOA, le service est presque pris pour acquis puisqu'il existe déjà mais certaines préoccupations quant au modèle d'interaction avec le service doivent être considérées.

Annuaire de services

Un service qui n'est pas connu, documenté et décrit convenablement ne sera pas utilisé. Une API qui n'est pas documentée ne sera pas utilisée. Un service Web dont les URI sont inconnues est inutile.

> Dans tous les cas, le facteur hasard demeure : on peut tomber sur ces services par accident, mais il n'est pas sage de compter sur ces accidents pour construire un succès commercial.

Un **annuaire de services** est un lieu qui référence et décrit à la fois des services et des contrats[64] d'utilisation pour ces services. Il n'est pas rare qu'un annuaire de services décrive aussi l'entreprise offrant les services et permette les contacts humains en sus des contacts logiciels, ceci tenant du fait que les relations contractuelles reposent beaucoup sur des relations de confiance.

Le modèle UDDI (nous y reviendrons) est probablement le plus connu des annuaires de services et est celui vers lequel tous les autres convergent. Son ascendant commercial est considérable et il est moussé par la majorité des grands joueurs dans le monde des TI.

Bus de services

Au sens des SOA, le bus de services est une entité (la plupart du temps logicielle) donnant accès aux services dans l'entreprise. Les bus sont des moteurs d'interopérabilité (des intergiciels) et visent à permettre l'interaction performante de composants logiciels. Par opposition, l'exposition externe des services dans un SOA visent l'interopérabilité maximale mais introduisent une strate d'interface supplémentaire, donc entraînent un coût (en temps) à l'invocation.

Dans un intergiciel, les objets sont développés selon une ou plusieurs technologies supportées par le moteur d'interopérabilité. Ce dernier propose une métaphore d'invocation à distance et un protocole (habituellement très performant) de communication entre les utilisateurs des objets et les objets eux-mêmes. Le moteur d'interopérabilité décrit aussi les règles d'invocation des objets et les règles de contrôle de leur durée de vie.

---

[64] Prenez le mot *contrat* au sens large ici, incluant à la fois les règles d'invocation, les coûts, les plages horaires disponibles, les paramètres légaux et les particularités techniques.

Certains bus de services offrent aussi des services supplémentaires pour combiner divers services, les répartir sur un parc informatique complexe, assurer l'intégrité transactionnelle (au sens entendu dans le monde des SGBD), la surveillance de l'état et du fonctionnement des services, le contrôle des versions, *etc.*

Un SOA ne vise pas à remplacer un bus de services mais bien à l'enrober pour l'exposer sous une forme convenant aux besoins externes. Une même entreprise peut utiliser divers bus de services et chacun de ces outils est une plateforme complète en soi. Dupliquer ou remplacer ces infrastructures est la plupart du temps déraisonnable d'un point de vue technique et financier.

Protocole d'accès aux services

Les services à l'interne sont déployés et utilisés selon les règles propres au bus de service sur lesquels ils sont offerts. Cependant, l'offre de services à l'externe est régie par des règles d'interopérabilité qu'on veut maximale, ce qui explique que l'exposition au monde extérieur est le plus souvent réalisée avec interfaces WSDL, protocole SOAP et données XML.

Les technologies en question étant toutes des technologies Web, les services exposés selon une approche SOA sont habituellement invoqués à travers SOAP sur HTTP. La majorité des technologies qu'implique cette démarche sont des recommandations du W3C.

Pour sécuriser l'accès aux services, plusieurs technologies existent :

- le *Transport Layer Security*[65], ou TLS, autrefois nommé *Secure Sockets Layer*, SSL pour sécuriser les échanges dès le tout début;
- les signatures électroniques XML[66] et le chiffrement XML[67] pour, respectivement, authentifier l'émetteur d'une requête sous forme XML et représenter le résultat d'un chiffrement de données sous forme XML[68];
- une forme alternative et relativement récente pour authentifier un émetteur à travers un tiers se portant garant de son identité est le format SAML, pour *Security Assertion Markup Language*[69], qui est un autre format XML.

[65] Voir http://en.wikipedia.org/wiki/Secure_Sockets_Layer

[66] Voir http://en.wikipedia.org/wiki/XML_Signature ou, pour un document plus complet et plus officiel, http://www.w3.org/TR/xmldsig-core/

[67] Voir http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[68] Pensez-y : chiffrer un document, peut-être un document XML, et s'assurer que le résultat du chiffrement soit aussi en format XML. Pour en savoir plus sur le format XMLKS, qui décrit un format XML de gestion des clés publiques de chiffrement : http://en.wikipedia.org/wiki/XML_Key_Management_Specification

[69] Voir http://en.wikipedia.org/wiki/SAML

Services

Évidemment, les services sont au cœur d'une démarche `SOA`. Un bon service :

- expose une fonctionnalité claire;
- est essentiellement autonome par rapport à d'autres services et sait bien réaliser une tâche à partir d'un groupe d'intrants donné. On souhaite habituellement qu'un service soit sans états, pour des raisons d'*échelonnabilité* (*Scalability*)[70] et d'invocation à travers des protocoles Web tels que `HTTP`;
- notons qu'un service sans états est susceptible d'être implémenté selon le schéma de conception ***singleton*** sans que cela n'encoure de problèmes de synchronisation particuliers lorsque plusieurs clients ont recours au même service en même temps;
- certains exigent d'un service qu'il soit indépendant de tout service externe mais cette exigence constitue un bris d'encapsulation. Cependant, ajouter des dépendances externes à un service implique lui ajouter des délais de transmission supplémentaires (*marshalling* entre lui et les services dont il dépend) ce qui le ralentit et réduit son utilité[71].

Les contraintes d'indépendance relative entre les services et de découplage (d'autonomie) sont des contraintes aux yeux du monde extérieur. Dans son implémentation, par exemple, un service peut servir à en coordonner d'autres.

Tel que mentionné précédemment, pour réduire les coûts du *marshalling*, un bon service s'exprime habituellement par des requêtes dont la granularité est grossière. Les services à granularité fine sont une bonne approche quand le serveur est un objet dans l'espace adressable du processus du client, mais sont beaucoup trop coûteux dans un système réparti.

[70] Si un service ne tient pas d'états à jour, alors son besoin en ressources est plus faible et il lui est plus facile d'accepter les demandes d'un grand nombre de clients.

[71] Notons aussi qu'ajouter des dépendances entre les services tend à réduire la probabilité que chacun soit disponible à un moment ou l'autre. Notons aussi qu'il est important d'éviter une circularité de dépendance entre les services.

Enfin, un bon service sera implémenté selon des stratégies d'invocation convenant aux besoins de ses clients. Ceci peut impliquer :

- des services **synchrones au sens faible**[72], où le *thread* client est mis en attente pendant que le serveur exécute la tâche qui lui est soumise;
- des services **asynchrones par sondage**, où le client démarre une tâche chez le serveur et obtient en échange un jeton identifiant cette tâche. Le démarrage de la tâche est non bloquant. Le client conserve le jeton et vérifie périodiquement l'état du traitement de la tâche. Quand la tâche est complétée, le client en récupère les résultats et se débarrasse du jeton;

Un système asynchrone par sondage ou par rappel implique le maintien à jour de certains états côté serveur.

- des services **asynchrones par rappel**, où le client démarre une tâche chez le serveur tout en lui donnant suffisamment d'information pour que le serveur puisse lui signaler le résultat du traitement de cette tâche. Selon cette approche, le serveur rappelle le client lorsque le traitement de la tâche est terminé, ce qui implique une certaines synchronisation du code client pour éviter un conflit lors du rappel, qui est indéterministe.

Il est possible de construire un service sans états autour d'un service avec états. Par exemple, si un composant est accessible à travers le bus de services et offre un service long à traiter selon une interface asynchrone par rappel, l'interface exposée à l'externe peut, de son côté :

- invoquer le service asynchrone;
- bloquer en attente du résultat; et
- retourner le résultat au client qui, lui, est alors soumis à des contraintes d'invocation synchrones au sens faible.

Notez qu'une approche asynchrone est en général beaucoup plus efficace qu'une approche synchrone au sens faible, mais que le souci d'offrir un service sans états à un client peut être tel que le prix (important) à payer côté performance rapporte, au global, lorsque vient le moment de commercialiser un service.

[72] Être synchrone au sens strict implique de pouvoir placer un plafond au temps requis pour réaliser une tâche. Nous ne pouvons pas appliquer une contrainte aussi forte dans l'exposition de services selon une approche SOA à cause des nombreux ralentissements potentiels entre l'invocation d'un service et l'obtention de son résultat.

Orchestration des services

Coordonner et orchestrer[73] des services est un problème passionnant en soi[74]. Un langage bien connu pour orchestrer des processus d'entreprises est le *Business Process Execution Language*, ou BPEL[75], étudié dans d'autres cours du DTI.

Orchestrer des services peut vouloir dire assurer une forme d'intégrité transactionnelle (mettre en place une sémantique *Commit*/ *Rollback* sur des services représentant chacun une partie d'une tâche complexe) ou faire en sorte que deux services développés indépendamment l'un de l'autre puissent être combinés sans se nuire mutuellement.

Certains protocoles[76] et certains outils (les *Workflows*[77]) cherchent à offrir des solutions à grande échelle, presque universelles, au problème de l'orchestration pris au sens large. Une approche à haut niveau permet d'approcher le problème de l'orchestration dans son ensemble, mais il est inévitable qu'on doive éventuellement en arriver à un examen des particularités techniques des divers services.

Exposer des services selon une approche SOA nécessite une combinaison de qualités et d'aptitudes, une vision à haut niveau et une compréhension des détails. Un projet pour une équipe, pas une tâche individuelle.

---

[73] Certains parlent même de chorégraphier des services lorsque l'action d'orchestration se fait dans une relation un à un, mais c'est une distinction de sémantique.

[74] À ce titre, voir l'essai de ***Sébastien Demers***, étudiant au DTI, nommé ***L'orchestration de services de téléphonie IP*** et portant sur ce problème dans un cadre très concret.

[75] Voir http://en.wikipedia.org/wiki/BPEL

[76] Par exemple le *Business Transaction Protocol*, ou BTP, le protocole *WS-Transaction*, le protocole WS-BPEL pour les services Web, le *Web Services Choregraphy Interface* (WSCI) ou le *Web Services Choregraphy Description Language* (WS-CDL).

[77] Voir http://en.wikipedia.org/wiki/Workflows

L'approche SOA (synthétisé)

Le mot SOA désigne une approche, des façons de faire. Pas une technologie, ni une panacée. Contrairement à ce que certains textes ont prétendu, appliquer SOA ne signifie pas utiliser des services Web, bien que l'approche SOA soit un domaine d'application intéressant pour les services Web.

Le cycle SOA se présente à peu près comme suit :

- procéder à partir de l'existant;
- y aller par petits pas, ne pas tout renverser;
- s'assurer de la valeur de chaque geste pour l'entreprise;
- communiquer à l'aide de types de données simples;
- viser des contrats de services relativement extensibles;
- planifier un cycle de vie pour les services. Ceux-ci naissent, évoluent et sont éventuellement remplacés; et surtout
- ne pas oublier les autres participants à l'approche dans la validation de l'offre de services (incluant lors de leur évolution), dans les tests des services (incluant les données à manipuler) et dans la gestion des coûts.

Quelques considérations supplémentaires :

- il faut assurer en tout temps la traçabilité des systèmes (prévoir de la journalisation et des outils de suivi);
- il faut s'assurer que le bus de services (l'infrastructure qui permettra d'assurer la communication entre les clients et les services) soit à la hauteur des attentes;
- il faut déterminer les modalités d'orchestration ou de chorégraphie des services.

Lorsqu'on développe des SCS à partir d'interfaces IDL, on vise habituellement à développer le client et le serveur en parallèle. Lorsqu'on développe des SCS à partir d'interfaces WSDL, on développe habituellement le serveur, puis on expose son interface, et enfin on développe des clients en fonction de cette interface. ***Lorsqu'on prend un virage SOA, les services existent déjà, les clients existent déjà, mais un effort de* refactorisation*, d'aménagement, d'adaptation et d'intégration doit être fait***.

Services Web

Le monde des systèmes répartis est un monde en forte mouvance, et constitue sûrement l'un des champs de bataille les plus importants de l'informatique commerciale contemporaine.

Après un essor modeste grâce au modèle simple et universel des *sockets*, puis grâce au modèle RPC dans ses nombreuses déclinaisons, grâce au raffinement du modèle RPC pour que celui-ci satisfasse aux contraintes identitaires du modèle OO, et enfin grâce à la croissance considérable du monde interconnecté suite à l'adoption par le grand public du réseau Internet, le monde CS est devenu une zone cruciale pour laquelle se battent tous les géants de l'industrie, chacun avec son propre modèle se voulant ouvert mais privilégié seulement par certains.

Le grand rêve d'une standardisation complète des protocoles et des modèles de communication demeure un objectif plein de promesses, mais il est très ardu de faire s'entendre les principaux protagonistes sur une seule et même façon de communiquer, sur un format cohérent et convenant à tous et chacun, sur un modèle permettant une indépendance suffisante de pensée pour chaque extrémité de chaque communication CS tout en offrant le support suffisant pour les langages OO commerciaux récents (C+, Java, C#) et les infrastructures complexes qui les accompagnent (CLR de .NET, JVM de Java, la gamme CORBA, *etc.*).

L'interconnexion grandissante de tous ces composants du monde Internet que sont nos postes de travail a aussi amené des problèmes de sécurité importants, qu'on parle de pourriel, d'espionnage industriel, de virus de toute sorte ou de tout autre irritant du genre, ce qui[78] a entraîné un souci croissant de la sécurité chez les gens et les organisations. Les ordinateurs sur Internet ouvrent habituellement des ports logiques, auxquels écoutent des programmes serveurs, aptes à répondre à des requêtes respectant un certain protocole. Plusieurs de ces programmes, même parmi les plus répandus, ont été à la base construits de manière naïve[79].

Pour protéger les ordinateurs comme les entreprises, on en est venu à les isoler du monde à l'aide de coupe-feu, murs virtuels refusant l'entrée (et parfois la sortie) à travers certains ports logiques.

Les entreprises sont donc prises entre souhait d'interopérabilité accrue et souci de sécurité; entre démarche de commerce électronique unifiée et choix d'outils spécifiques pris dans une pluralité immense pour développer les homologues qui permettront de mettre en place l'architecture commerciale tant souhaitée.

C'est dans cette logique de souhait de standardisation et de sécurisation qu'apparaissent les services Web, paradigme CS sinon dominant, du moins sur le point de le devenir.

---

[78] … avec bien sûr la crainte du terrorisme, exacerbée depuis l'attentat du 11 septembre 2001.

[79] Selon les standards de sécurité en vigueur aujourd'hui, du moins; à l'époque où ces systèmes ont été mis au point, on se préoccupait beaucoup plus de faire se parler ensembles les homologues que de garantir à fond la sécurité transactionnelle qui les lie, du côté protocolaire comme du côté de l'implémentation. Il ne faut pas mettre tous les programmes dans le même panier, mais force est d'admettre que plusieurs clients et que plusieurs serveurs commerciaux se sont avérés, face à des attaques concertées ou planifiées, plus fragiles qu'on l'aurait cru *a priori*.

Tout d'abord : XML

Il est rapidement devenu clair que la standardisation des protocoles de communication se ferait éventuellement autour d'une norme à la fois claire et simple[80].

Le coût du développement et celui du déverminage ont depuis assez longtemps dépassés le coût de l'achat de nouveau matériel. Il semble y avoir consensus dans l'industrie que, dans la majorité des cas[81], il est maintenant préférable de pêcher du côté de la lisibilité et de l'emploi d'outils commerciaux et standardisés d'analyse de protocoles plutôt que de développer des protocoles selon un format maison et de développer des outils d'analyse et de déverminage à la pièce, devant eux aussi être validés individuellement.

La norme XML, on l'a vu, s'est imposée comme l'option la plus plausible en ce sens. Sa notation, évoquant celle de HTML, est pour l'essentiel accessible au commun des mortels. Son caractère pleinement balisé en fait un candidat très sérieux pour permettre la construction d'outils analytiques, et il est relativement simple de concevoir un outil capable d'extraire la structure d'un document XML sans même connaître le protocole sous-jacent.

La norme XML permet d'analyser la syntaxe de manière indépendante de la sémantique.

[80] La norme XML est simple, mais il est possible d'écrire des documents respectant cette norme et qui sont, eux, terriblement difficiles à lire pour un humain.

[81] On parle évidemment ici des cas typiques; il existe des cas plus industriels qui sont soumis à des contraintes importantes de performance et de bande passante et pour lesquels un protocole basé XML, dont les paquets tendent à être clairs mais volumineux, pourrait ne pas être approprié.

Principales stratégies analytiques

Deux standards analytiques globaux de documents XML dominent le marché. Il s'agit du *Document Object Model* (**DOM**) et du *Simple API for XML* (**SAX**). En gros :

- une **analyse DOM** charge le document analysé en mémoire, en construit une structure hiérarchique et permet, une fois le document construit, d'aller y chercher des éléments sur demande. Sa principale force est la souplesse: DOM permet de naviguer à loisir dans l'arbre ainsi construit. Sa principale faiblesse est son coût en ressources: DOM nécessite de charger la totalité du document XML en mémoire avant de permettre d'y accéder[82];
- une **analyse SAX** a comme caractéristique d'analyser le document au fur et à mesure qu'il est traversé par un processeur de langage. Pour utiliser SAX, il faut y abonner des composants en indiquant au processeur les balises à propos desquelles chaque composant doit être rappelé. Quand un processeur rencontre une balise XML dans un document, il avertit tous les observateurs qui se sont abonnés pour cette balise, et ceux-ci peuvent alors réagir à la donnée lue[83]. Sa principale force est la vitesse: le traitement d'un document XML s'y fait au fur et à mesure que le document est lu. Sa principale faiblesse est sa rigidité: SAX ne permet pas de revenir en arrière une fois une balise traitée.

Bien que tous soient conscients du danger inhérent aux structures XML, la standardisation des procédés d'analyse de ces documents s'est avérée si précieuse et si utile que les démarches d'interopérabilité se sont malgré tout orientées autour de ce format.

Les services Web, l'accès aux bases de données, les fichiers décrivant les règles de déploiement de composants, les protocoles de communication en vogue … Toutes ces choses sont maintenant basées XML, ou sur le point de l'être.

---

[82] DOM comporte une faiblesse structurelle fondamentale, d'ailleurs, puisqu'il est possible de construire des documents XML arbitrairement gros. Si on utilise DOM pour analyser des documents XML reçus de sources quelconques, un vandale pourrait en tirer profit en émettant par un client hostile un document sans fin, brisant ainsi le code du serveur (et inversement). Les architectures DOM commerciales ne sont pas toutes capables de bien retomber sur leurs pattes face à une telle attaque.

[83] Remarquez que cela blinde SAX contre les attaques de documents formés de manière hostile, mais déplace la problématique de l'analyse de ces documents vers les composants abonnés au processeur SAX. Une balise formée de manière hostile pourra faire planter SAX *indirectement*, par le biais d'un abonné fragile.

Les services Web

Les ***services Web***, que certains nomment *services Web XML*[84], ont plusieurs particularités qui permettent de les reconnaître. En général, un service Web :

- expose des services sous forme de méthodes accessibles à des clients sur le Web (incluant des fureteurs) à l'aide d'un protocole standard. On parlera souvent de **SOAP**, mais d'autres options existent (en particulier XML-RPC pour des services légers);
- décrit ses services à l'aide d'une spécification d'interface standardisée, inspirée des notations IDL. La plupart du temps, la norme *Web Services Definition Language* (**WSDL**) sera utilisée car elle permet d'ajouter aux spécifications opérationnelles[85] de l'information sémantique, ce qui est utile dans une démarche commerciale où on peut, par exemple, vouloir automatiser la négociation des modes de paiement; et
- inscrit ses services dans un registre pour faciliter la tâche d'éventuels clients qui souhaiteront transiger avec lui. Les registres de type *Universal Discovery Description and Integration* (**UDDI**) existent précisément pour offrir ce type de service.

On le voit, les services Web s'inscrivent dans une démarche commerciale autant que dans une démarche de développement.

Les services Web ont une autre caractéristique: ils sont pensés pour être transigés en utilisant comme protocole de transport un protocole Web standard, souvent—mais pas exclusivement—le protocole HTTP. Pour cette raison, les services Web peuvent aisément être exposés à travers un serveur Web prenant en charge la question de la sécurité.

Rendre le serveur Web imputable de la sécurité est généralement perçu comme étant une attitude raisonnable.

Pris dans une acception plus large, un service Web est, bêtement, un service logiciel offert sur Internet, la plupart du temps suivant une approche RPC. Le serveur est identifié par une URL, et le cycle typique d'invocation d'un service Web dans une démarche synchrone au sens faible inclut la transmission d'une requête du client au serveur, le traitement de la requête par le serveur, et l'émission de la réponse du serveur vers le client.

Le modèle RPC n'est pas le seul possible. Sous Java, par exemple, les services Web apparaissent souvent comme des systèmes requête/ réponse de messagerie. Des rappels et des transactions asynchrones sont possibles avec des services Web comme avec d'autres composants.

Un service Web peut donc être simplement vu comme un composant sans état, et permet de réaliser pratiquement les mêmes tâches que d'autres composants sans état, dans la mesure où un serveur Web compatible est disponible sur l'ordinateur hôte dudit composant.

---

[84] Rien ne force, en fait, à utiliser XML pour exposer des services Web, mais la tendance est extrêmement forte en ce sens, et il serait difficile aujourd'hui de pencher pour une approche divergente de celles reposant sur une notation XML tout en offrant un plaidoyer pour l'interopérabilité. Nous utiliserons le vocable services Web sans y imposer la mention XML pour conserver une forme de généralité; qui sait, après tout, si des pirates ne forceront pas le monde, par une action hostile concertée, à adopter éventuellement un standard protocolaire plus robuste?

[85] Noms des méthodes, paramètres, types, intrants et extrants, gestion des exceptions, *etc.*

Le protocole SOAP

L'emploi du protocole SOAP n'est pas une obligation pour exposer des services Web, mais c'est la manière typique de procéder. Notez que SOAP n'est qu'un protocole, et ne constitue pas un moteur d'interopérabilité complet.

Le protocole SOAP respecte la norme XML, mais selon un format quelque peu complexe, principalement parce que SOAP offre un système de types très riche. On ne programme pas le protocole SOAP manuellement, règle générale, préférant utiliser des produits commerciaux qui, comme Java avec Apache et les langages .NET avec leur propre moteur, se servent de SOAP et de ses règles pour assurer leur interopérabilité et implémenter leur *marshalling*.

Cela signifie que l'analyse automatisée des trames SOAP transigées se fera selon le modèle privilégié sur le produit choisi (souvent SAX avec Java et DOM avec .NET, quoique les deux plateformes supportent les deux modèles). Les homologues de part et d'autre ne verront pas SOAP directement, car ce sont leurs intermédiaires respectifs (*proxy* et *stub*) qui s'en serviront pour transiger entre eux.

Nous avons utilisé SOAP comme protocole de communication dans chaque exemple utilisant le *Remoting* sur .NET, et ce de manière transparente. L'utilisation de SOAP n'est donc pas quelque chose de douloureux.

La description WSDL

Tel qu'indiqué plus haut, une spécification WSDL est l'équivalent type des IDL pour les services Web, à ceci près que WSDL suit une notation basée XML et que WSDL peut ainsi inclure plus d'information que les fichiers IDL sans que cela n'affecte outre mesure les consommateurs de descriptions WSDL[86].

Ainsi, un client potentiel qui ne serait intéressé que par les caractéristiques techniques d'une interface décrites en format WSDL (ce qu'on trouve habituellement dans une spécification IDL) peut s'y limiter sans souffrir outre mesure, alors qu'un autre client pourrait vouloir connaître les modalités de paiement ou l'identité de la firme offrant le service décrit par la spécification WSDL pourrait aussi s'y limiter, au besoin.

Il y a une différence philosophique de fond entre une approche basée IDL et une approche basée WSDL.

En effet, il est tout à fait envisageable de faire développer une spécification IDL par un humain, donc de concevoir l'interface avant les homologues qui communiqueront à travers elle. En retour, à cause de la complexité de la tâche, il est très difficile de générer une spécification WSDL à la main.

Ainsi, pour un service Web, on développera habituellement le serveur, puis on en extraira une spécification WSDL, et les clients seront développés par la suite. Cela simplifie le développement d'un serveur destiné à une multitude de clients (et en général de serveurs *grand public*) mais complique le développement en parallèle des divers homologues d'un SCS complexe.

En résumé, la plupart du temps, des outils informatisés (programmes Java, programmes tiers C++ ou outils intégrés à *Visual Studio*) généreront les spécifications WSDL d'un serveur à partir de son code. La philosophie sous-jacente de cette démarche est que les services seront pensés, codés puis exposés à des masses de clients potentiels[87] plutôt que pensés pour un SCS cohérent. Les services Web sont philosophiquement pensés dans le but très commercial de vendre des services à grande échelle.

[86] Les notations balisées ont cet avantage qu'une balise incomprise ou ne présentant pas d'intérêt particulier pour le consommateur d'un document peut souvent être escamotée.

[87] …qui découvriront la description WSDL en temps et lieu, en *magasinant* des services.

Les registres UDDI

Puisque les services Web sont a priori pensés pour diffusion au grand public, il importe de les rendre disponibles à travers des canaux contrôlés, connus et efficaces.

Le modèle développé pour ce faire sur nomme *Universal Discovery Description and Integration* (UDDI) est se base sur le principe des pages jaunes. Les services Web, décrits sous forme WSDL, peuvent être inscrits dans des registres UDDI qui faciliteront leur recherche sur des bases syntaxiques et sémantiques.

Les registres UDDI savent des choses sur les services, sur leur provenance, sur l'entreprise qui les expose, et ainsi de suite. On peut donc entreprendre des recherches sur UDDI pour des services de certains types et respectant certains modalités de paiement, respectant aussi certaines contraintes quant à la firme offrant le service (excluant les fournisseurs qui ne se sont pas montrés dignes de confiance dans le passé, par exemple), et ainsi de suite.

Dans UDDI, on trouve des pages blanches (décrivant les entreprises), des pages jaunes (décrivant des catégories d'entreprises et de taxonomies) et des pages vertes (listant des services). On peut donc travailler à plusieurs niveaux avec ce registre. Parce qu'UDDI est en soi un service Web, il peut servir à des humains procédant à une recherche par page Web comme à des programmes, et peut automatiser des recherches à partir de critères formels.

Rôle du serveur Web

Chaque service Web est un composant qui apparaît au monde Internet sous la forme d'une `URL`. Cela signifie que l'invocation d'un service Web passe par une requête `HTTP` prise en charge par un serveur Web, qui comprendra dans cette `URL` une demande d'invocation et qui :

- créera le composant;
- invoquera la méthode sur demande; et
- assurera, en temps et lieu, la destruction du composant.

Il faut donc, pour tester des services Web, avoir au préalable installé un serveur Web. Vous comprendrez que cette tâche et un peu trop volumineuse à décrire pour que nous le fassions en ces pages. Ainsi, si vous désirez faire des tests, je vous invite à procéder comme suit :

- si vous désirez développer des services Web avec Java, téléchargez *Apache Tomcat*[88] et installez-le. Des instructions sur l'installation et la personnalisation de *Tomcat* seront disponibles à même le site Web et vous permettront de procéder;
- si vous désirez développer des services Web avec `.NET`, installez *Visual Studio* `.NET` au complet, incluant `IIS` et les extensions *FrontPage*, et vous aurez accès à un outil de développement assez bien intégré.

D'une manière ou d'une autre, vous comprendrez que je ne puisse pas assurer le support pour l'installation et la personnalisation de serveurs Web, ou prendre la responsabilité pour les considérations de sécurité pour l'installation d'un serveur Web sur votre poste de travail.

Agissez prudemment!

Vous pouvez bien sûr choisir un autre serveur Web si le cœur vous en dit[89]

Parce que les serveurs Web jouent un rôle important dans l'exposition de services Web, il est fréquent que des pages *Java Server Pages* (`JSP`) pour Java ou *Active Server Pages* (`ASP`) pour les outils `.NET` soient générées par les outils de développement et soient placées à l'avant-plan pour contrôler l'accès à vos services.

[88] `http://jakarta.apache.org/tomcat/`

[89] Pour des raisons d'espace, vous comprendrez aussi que nous n'irons en profondeur dans aucune de ces technologies, nous limitant au nécessaire pour que vous puissiez expérimenter par vous-mêmes. N'hésitez donc pas à poursuivre vos lectures dans un sens comme dans l'autre si le sujet vous intéresse.

Services Web comme devanture

Lorsque Java a fait son apparition sur le marché, dans les années '90, l'une des niches envisagées pour ce produit était celle où, à l'aide d'applets, on offrirait des strates programmées au-dessus de systèmes anciens pour dissimuler et interfacer avec des sections moins *sexy* d'un parc informatique.

Le cours des événements a mené Java vers un créneau de programmation générale avec une couleur plus serveur que client, mais l'intention d'ériger des façades modernes au-dessus de systèmes plus anciens demeure. Les raisons sont multiples: maintenir les systèmes existants en les rendant plus conviviaux; exposer leurs services à distance; impliquer graduellement de nouveaux développeurs dans des systèmes qui leur sont moins familiers, etc.

Les services Web, visiblement, représentent une technologie faite sur mesure pour remplir ce créneau: accessibles via Internet, aussi attrayants visuellement qu'une page Web peut l'être, aussi dynamiques qu'un programme, encadrés de manière rigoureuse et pensés pour la connectivité et l'interopérabilité. Les services Web sont des candidats très sérieux au titre de devanture pour du code qui vieillit mal.

Activité—Générer un service Web avec un servlet Java

En rédigeant ces notes de cours, j'ai eu un dilemme. En fait, j'en ai eu plusieurs, en particulier quant à la place à accorder aux plateformes Java et .NET et quant à la place à accorder à CORBA et à COM.

Si vous avez des interrogations sur ces sujets, il est hautement probable que je me sois posé les mêmes questions auparavant et que j'aie vécu les mêmes dilemmes.

Dans la plupart des cas, l'approche que je me suis résolu à mettre en application fut une approche pragmatique, un peu pour vous et beaucoup pour moi.

Qu'est-ce qu'un servlet?

En bref, un servlet est une classe Java dérivant de la classe HttpServlet et capable de réagir à des commandes HTTP de type GET ou POST, à l'aide de méthodes doGet() et doPost() respectivement.

Ces deux méthodes reçoivent toutes deux en paramètre une instance de HttpServletRequest et une instance de HttpServletResponse.

Le servlet extrait les paramètres de la requête à partir de l'instance de HttpServletRequest et détermine le contexte de la page Web résultante à travers l'instance de HttpServletResponse.

Le résultat est habituellement une page .jsp tirant une part de son contenu des données générées par le servlet.

Le cas CORBA est frappant en ce sens : je suis philosophiquement plus proche de CORBA que de COM à plusieurs égards, mais gérer des notes de cours de la complexité de celles que je produis pour ce cours dans le monde CORBA est une excellente manière de devenir fou.

Un nouveau dilemme se présente pour ce qui est de mettre sur pied une activité montrant comment générer un service Web avec Java. En effet, le volet programmation est extrêmement minime, mais le volet configuration l'est beaucoup moins.

J'ai donc choisi de vous diriger vers le didacticiel[90] officiel de *Sun* pour les services Web avec J2EE (*Java 2 Entreprise Edition*) à l'URL suivante :

```
http://java.sun.com/j2ee/1.4/docs/tutorial/doc/
```

Je vous conseille de consulter le chapitre 8, portant sur la conception et sur le déploiement de services Web simples avec JAX-RPC.

Pour utiliser ce didacticiel, il est nécessaire de télécharger les exemples, à l'URL

```
http://java.sun.com/j2ee/1.4/download.html#tutorial
```

et d'installer le SDK de *Sun* que vous trouverez à

```
http://java.sun.com/j2ee/1.4/download.html#sdk
```

Ce SDK comprend des outils précieux comme wscompile (un outil pour transformer une classe Java conforme à certaines règles en service Web utilisant SOAP) et ant (un gestionnaire de configurations semblable à make mais basé sur des fichiers XML).

[90] J'utilise le mot *didacticiel* ici en sachant très bien qu'il s'agit d'un abus de langage. Le document en question est plus un volumineux ensemble de pas à pas et d'exemples qu'un logiciel pour apprendre un logiciel.

Vous remarquerez la nomenclature particulière pour des concepts auxquels vous êtes maintenant habitué(e)s, comme des *stubs* et des *ties* au lieu des *proxies* et de *stubs*; les fichiers `WSDL` plutôt que `IDL`; le serveur Web comme outil de prise en charge des composants; *etc*.

À la fin de cette section, je présenterai la structure de répertoires et une stratégie possible pour exposer un service Web à travers le très populaire serveur Web *Apache Tomcat `5.0`*, pour offrir au moins un exemple opérationnel. Étant donné que ce service mêlera `HTML` et Java, il sera présumé que vous aurez une connaissance minimale de ces deux technologies (ou que vous aurez accès au support nécessaire pour pallier tout manque de ce côté).

Les grandes lignes de la démarche

Le didacticiel propose de créer un service Web qui prend en paramètre une chaîne de caractères et retourne une chaîne de caractères. La chaîne retournée par le composant sera la concaténation de `"Hello "` et de la chaîne reçue en paramètre.

À noter que les contraintes sont essentiellement les mêmes pour un service Web que pour un composant utilisant `RMI` (l'interface dérive de `Remote` et les méthodes lèvent un `RemoteException`).

```
package helloservice;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface HelloIF extends Remote
{
   public String concou(String s)
      throws RemoteException;
}
```

Les différentes étapes du didacticiel suggéré plus haut font en sorte d'extraire une description `WSDL` de l'interface `HelloIF` ci-dessus. On voit encore une fois que la tendance semble être de développer les serveurs d'abord et de concevoir les clients en conséquence. La qualité des outils récents et la standardisation des descriptions `XML` comme celles offertes par `WSDL` font de ce virage un moindre mal.

La manière dont le composant en soi expose l'interface `HelloIF` est aussi simple qu'elle le devrait: le composant est une classe dérivant de cette interface.

Comme tout service Web qui se respecte, un composant `HelloImpl` est sans état (il ne possède aucun attribut d'instance).

```
package helloservice;
public class HelloImpl implements HelloIF
{
   private static final String MSG ="Hello";
   public String coucou(String s) {
      return MSG + s;
   }
}
```

Une fois le composant créé, en faire un service Web et le déployer devient affaire de configuration, et dépend largement des outils utilisés. Ceux-ci étant nombreux et volatiles (les étapes changent selon les versions), vous fier sur un didacticiel souvent mis à jour et adapté par une horde de gens payés pour le faire est sûrement mieux que d'avoir des notes de cours promptes à tomber en désuétude avec chaque nouvelle version des outils.

Offrir un service Web à travers Apache Tomcat 5.0[91]

Le serveur Web *Apache Tomcat 5.0* cartographie les services Web à sa disposition lors de son démarrage. Pour les reconnaître, il explore une hiérarchie de répertoires relativement stricte.

Une partie de l'exposition d'un service Web à travers *Tomcat* tient donc à une répartition des fichiers pertinents dans les répertoires prévus à cet effet.

Tout service Web Java exposé sous Apache doit être contenu dans un répertoire précis (et dans ses sous-répertoires). Ce répertoire doit être placé sous `RACINE/webapps`.

Question de notation

Chaque compagnie aura sa propre structure de répertoires, dont on ne peut présumer ici, mais la structure des répertoires d'*Apache Tomcat* devrait, elle, être stable d'une plateforme à l'autre.

Ainsi, j'utiliserai `RACINE` comme notation pour le répertoire racine de *Tomcat*, et j'utiliserai la notation `/` plutôt que la notation `\` pour séparer les répertoires. Notez donc que la notation `RACINE/webapps`, par exemple, devrait être remplacée par la notation correspondante mais convenant à votre configuration[92].

⇒ Ainsi, le service Web `PetitService` devrait être placé en totalité à l'intérieur du répertoire `RACINE/webapps/PetitService/`

On placera la page d'accueil du service Web (une page Web, évidemment) dans le répertoire du service Web en question.

⇒ Ainsi, la page d'accueil du service Web `PetitService` devrait porter un nom complet semblable à `RACINE/webapps/PetitService/index.html` ou à `RACINE/webapps/PetitService/index.htm`.

S'il y a des répertoires de travail accessoires mais utiles au bon fonctionnement du service Web, on les positionnera aussi sous le répertoire du service Web.

⇒ Par exemple, les images utilisées par les pages propres au service Web `PetitService` pourraient être placées sous `RACINE/webapps/PetitService/images/`

[91] Notez qu'il existe plusieurs manières d'exposer un service Web à travers *Tomcat*. Je propose une stratégie ici, à titre d'exemple, mais il se peut que des spécialistes de cet outil aient des préférences ou des habitudes autres. Sachant que nous ne voulons que démontrer la possibilité et exposer la base du modèle, nous ne chercherons pas à proposer la manière optimale d'y arriver, et nous resterons simples.

[92] Chez moi: `C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\`, du fait que la racine de *Tomcat* y est `C:\Program Files\Apache Software Foundation\Tomcat 5.0\`.

Le répertoire `WEB-INF`

Le répertoire fondamental au bon fonctionnement de tout service Web sous *Apache Tomcat* est le répertoire `WEB-INF` du service.

⇒ Pour nous: `RACINE/webapps/PetitService/WEB-INF/`

Ce répertoire doit contenir principalement trois choses :

- le fichier `web.xml` décrivant les règles de déploiement du service;
- un répertoire `classes` contenant les binaires Java (`.class`) du service. Dans ce cas, un simple servlet suffit; et
- un répertoire `lib` contenant les archives `JAR` du service.

Habituellement, on développera le service Web Java dans un environnement de test isolé et on déposera les fichiers `.class` compilés et testés dans le répertoire classes du service Web lorsque le tout sera prêt.

Le descripteur `web.xml` du service Web

Le fichier descripteur `web.xml` d'un service Web sous *Apache Tomcat* est celui dont se sert le serveur Web pour comprendre la configuration dudit service.

Un fichier typique pour un service Web très simple représenté par un servlet Java nommé `login.class` et appartenant au service Web `PetitService` serait :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
  <display-name>Petit service Web</display-name>
  <session-timeout>30</session-timeout>
  <servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>login</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/PetitService</url-pattern>
  </servlet-mapping>
</web-app>
```

Les lignes en caractères gras sont les seules pertinentes à notre propos (les autres peuvent, pour nos fins, être considérées comme obligatoires et hors de notre contrôle). Vous reconnaîtrez, aux endroits appropriés, le nom du binaire Java et celui du dossier dans lequel réside le service Web.

La page Web sollicitant le service Web

Dans notre exemple simplet, la page d'accueil invoquera le service Web. Pour ce faire, nous définirons une page Web avec formulaire muni d'un bouton de commande invoquant l'action `PetitService`, qui est le nom de notre service Web:

```
<html>
  <head>
    <title>Service Web de test</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
   </head>
  <body bgcolor="White" onLoad="document.loginForm.username.focus()">
    <table width="500" border="0" cellspacing="0" cellpadding="0">
      <tr>
        <td>
          <table width="500" border="0" cellspacing="0" cellpadding="0">
            <form name="loginForm" method="post" action="PetitService">
            <tr>
              <td width="401"><div align="right">Nom: </div></td>
              <td width="399"><input type="text" name="Nom"></td>
            </tr>
            <tr>
              <td width="401"><div align="right">Mot de passe: </div></td>
              <td width="399"><input type="password" name="MotDePasse"></td>
            </tr>
            <tr>
              <td width="401"> </td>
              <td width="399"><br><input type="Submit" name="Soumettre"></td>
            </tr>
            </form>
          </table>
          </td>
      </tr>
    </table>
  </body>
</html>
```

Le service Web en soi

Dans ce cas simpliste, le service Web est implanté par un servlet, qui reçoit une paire nom d'usager/ mot de passe saisie d'une page Web lors de son invocation, qui fouille dans une BD (action seulement simulée ici) pour trouver l'usager correspondant à cette paire, et qui rend cette donnée disponible de par le nom de variable USAGER à la page JSP résultante.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class login extends HttpServlet {
  private String pageCible_ = "/welcome.jsp";
  // normalement, on examinerait la paire nom/ mot de passe et on ferait
  // une recherche dans une BD pour obtenir le nom d'usager désiré
  private String getUsager(String nom, String motDePasse) {
     return "Nom générique";
  }
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException { // simpliste
    doPost(request, response);
  }
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // extraire le nom d'usager et le mot de passe de la requête
    final String nom = request.getParameter("Nom");
    final String motPasse = request.getParameter("MotDePasse");
    final String usager = getUsager(nom, MotPasse);
    // Pour informer la page JSP résultante du nom de l'usager obtenu
    request.setAttribute("USAGER", usager);
    // rendre l'information disponible
    ServletContext ctx = getServletContext();
    RequestDispatcher répartiteur = ctx.getRequestDispatcher(pageCible_);
    répartiteur.forward(request, response);
  }
  public void destroy() {
  }
}
```

La page JSP saisie du résultat de l'exécution du service Web

Une page JSP est en fait une page Web contenant de l'information de formatage, comme une page HTML, et suffisamment d'information pour générer du code Java capable de prendre en charge une part du traitement d'une paire requête/ réponse HTTP.

La page JSP proposée ici sera générée en réponse à l'invocation de notre service Web, et utilisera le contenu de l'attribut USAGER de la requête HTTP ayant mené à son apparition pour afficher le nom de l'usager sur la page Web en question.

```
<html>
  <head>
    <title>Service Web de Test</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>
  <body color="White">
    <table width="500" border="0" cellspacing="0" cellpadding="0">
      <tr>
        <td>
          <b>Bonjour, usager <%= request.getAttribute("USAGER")
%></b>
        </td>
      </tr>
    </table>
  </body>
</html>
```

Pour tester le tout, suffit d'ouvrir la page JSP à l'aide d'un fureteur.

Activité—Générer un service Web brut avec Java

Notes préliminaires : références Web

Les ressources suivantes ont été utilisées pour concevoir et déployer les services Web proposés dans cette section. La liste n'est pas exhaustive.

- Le site `http://java.sun.com/webservices/index.jsp`, qui sert de référence principale pour les services Web sous Java.
- Le site `http://java.sun.com/xml/jaxrpc/index.jsp`, qui décrit l'`API` Java d'appels `RPC` via `XML`.
- Le site `http://www.javaworld.com/javaworld/jw-06-2003/jw-0620-webservices.html`, qui propose un survol bref de la technologie.
- Les sites `http://developers.sun.com/techtopics/webservices/index.html`, `http://docs.sun.com/source/816-7152-10/wsgjaxrpc.htm`, qui listent des références techniques.
- Le site `http://java.sun.com/developer/technicalArticles/J2EE/j2ee_ws/`, qui propose un pas à pas fort raisonnable.
- Le site `http://java.sun.com/webservices/tutorial.html`, qui propose un autre pas à pas.
- Le site `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXRPC3.html`, qui fait partie d'un autre pas à pas.

Notes préliminaires : choix d'outils

Le choix des outils sur lesquels seront déployés des services Web influencera de beaucoup la démarche de déploiement, au sens de *ce qui doit être fait* comme au sens de *comment cela devra être fait*.

Pour les services Web proposés ici, j'ai utilisé le serveur Web ***Tomcat 5.0*** de *Apache*[93]; le serveur d'applications ***Sun Application Server***[94]; la plateforme Java industrielle **`J2EE 1.4.2`**[95]; et le ***Java Naming Directory Interface*** (**`JNDI`**)[96].

Les services Web sous Java sont considérés comme faisant partie des technologies de calibre industriel, devant être déployés à travers un service Web et présumant donc une connaissance technique plus approfondie que la moyenne (ou un support technique équivalent) et l'accès au matériel et au logiciel rendant pertinents l'emploi de services Web. Sachant cela, il est important d'utiliser le `JDK` le plus récent de `J2EE` (*Java 2 Entreprise Edition*) plutôt que le `JDK` le plus récent de `JSE` (*Java Standard Edition*). Au moment d'écrire ces lignes, la version entreprise en était à la version `1.4.2` alors que la version standard en était à la version `1.5`, ce qui implique que des classes Java compilées avec le `JDK` standard ne pourront être prises en charge par la `JVM` de la version entreprise, qui est plus ancienne. Heureusement, les deux `JDK` peuvent coexister sur un même ordinateur, ce qui simplifie de beaucoup les choses.

Nous allons, dans cette section, expliquer à la fois comment créer un service Web en Java; comment le déployer; et comment produire des clients capables de s'en servir.

Il est présumé que la lectrice ou le lecteur a une connaissance du langage de programmation Java ou a accès à des ressources capables de l'encadrer en ce sens. Il est aussi présumé que la lectrice ou le lecteur est en mesure de télécharger et d'installer les produits mentionnés dans la section ***Notes préliminaires : choix d'outils***, ci-dessus.

---

[93] Voir `http://jakarta.apache.org/tomcat/`

[94] Voir `http://www.sun.com/software/products/appsrvr/index.xml` mais notez que des versions corrompues par virus existent (Java n'est pas immunisé contre les infections). Appliquez un antiviral aux fichiers téléchargés avant de les utiliser!

[95] Voir `http://java.sun.com/j2ee/1.4/index.jsp`

[96] Voir `http://java.sun.com/products/jndi/`

Mêler Java et XML : les principaux outils

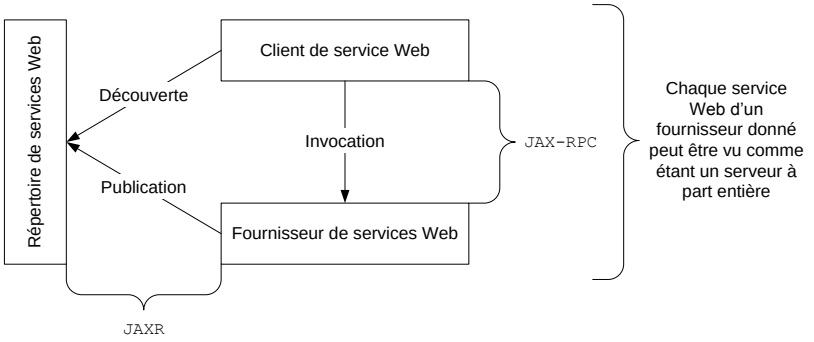
Les principales API de développement Java orientées vers XML, et regroupées en bloc sous l'égide JAX (*Java API for XML*) peuvent être groupées en deux familles, soit celle des outils de traitement des documents XML (API dites *Document-Oriented*) au sens large et celle des outils de traitement d'appels de sous-programmes (API dites *Procedure-Oriented*). Selon les besoins des projets, on installera et on déploiera les API dont on a véritablement besoin.

Du côté du traitement de documents XML, on trouve principalement :

- l'API nommée **JAXP**, pour *Java API for XML Processing*, dont le rôle est de faciliter le traitement et l'analyse de documents XML; et
- l'API nommée **JAXB**, pour *Java Architecture for XML Binding*, qui fait une tâche similaire à celle accomplie par JAXP mais le fait à partir de composants Java (des *Beans*) représentant des schémas XML.

Du côté du traitement d'appels de sous-programmes XML, on trouve principalement :

- l'API nommée **JAX-RPC**, pour *Java API for XML-Based RPC*, dont le rôle est de permettre les échanges RPC via SOAP;
- l'API nommée **SAAJ**, pour *Java API for XML Messaging*[97], qui fait une tâche similaire à celle accomplie par JAX-RPC mais selon une stratégie de messagerie conventionnelle plutôt que par appels RPC, décrivant les appels de sous-programmes comme des documents XML dans lesquels apparaissent les descriptions SOAP; et
- l'API nommée **JAXR**, pour *Java API for XML Registries*, qui facilite l'accès aux répertoires de services Web, ce qui facilite entre autres le repérage de services selon certains critères.



[97] Oui, je sais que l'acronyme ne convient pas…

Offrir un serveur Java sous forme de service Web

Pour offrir un serveur Java sous forme de service Web, il convient de suivre les étapes suivantes :

- concevoir l'interface du service (ce qu'on fait habituellement en Java directement, selon la philosophie en vigueur dans le monde des services Web);
- implémenter le service, ce par quoi on entend rédiger une classe Java qui implémentera les méthodes de cette interface;
- rédiger un fichier `XML` de configuration;
- générer les fichiers requis pour le déploiement du service; et
- procéder au déploiement en tant que tel.

Concevoir l'interface du service

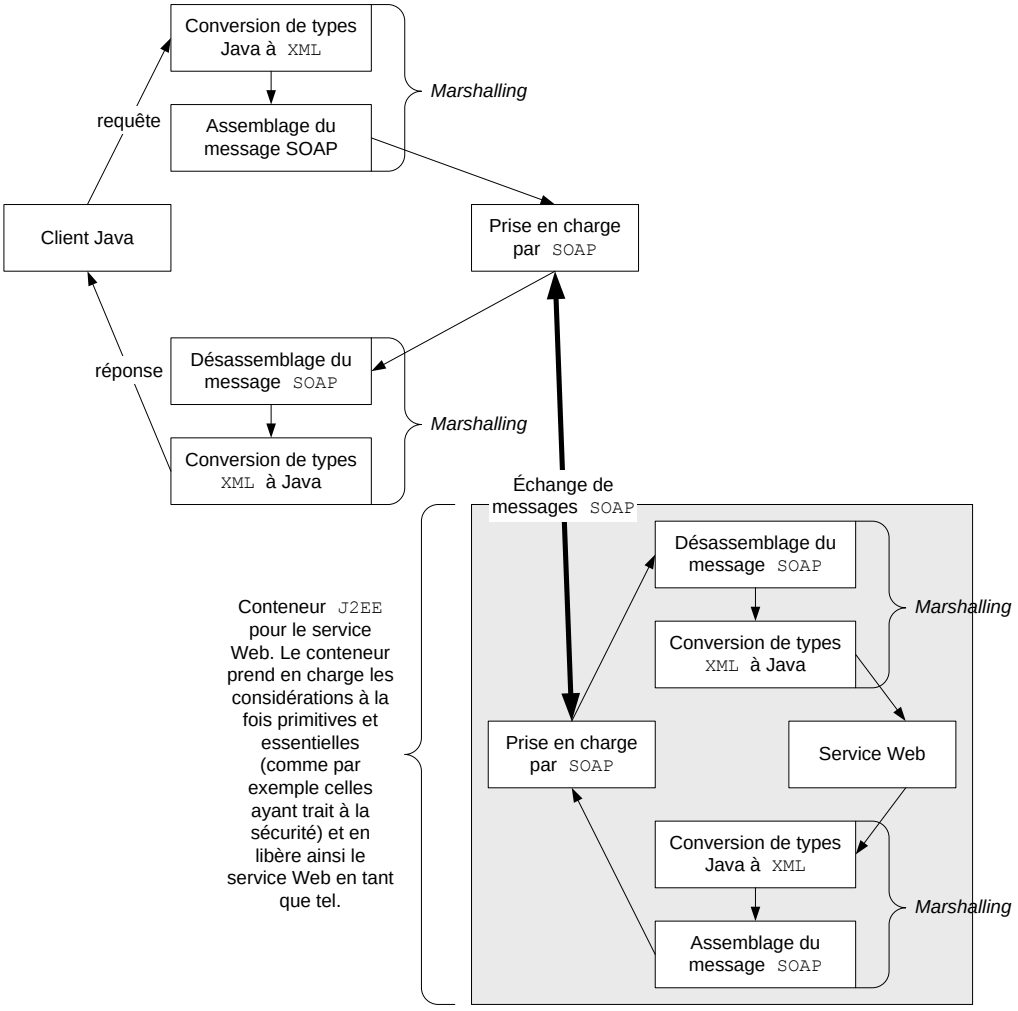
Une interface de service Web Java est une interface `OO` normale selon les règles du langage Java, mais respectant en plus quelques règles propres au modèle des services Web.

Ces règles sont :

- que l'interface doit être une spécialisation de `java.rmi.Remote`;
- que l'interface ne peut pas contenir de membres de classe (rien qui soit spécifié `static`);
- que les méthodes de l'interface doivent toutes *potentiellement* lever `java.rmi.RemoteException` ou l'une de ses classes dérivées; et
- que les paramètres et le type des méthodes de l'interface doivent tous être faits de types supportés par `JAX-RPC`.

Les types en question sont les types primitifs de Java (p. ex. : `boolean`, `float`, `short`, …); les classes enrobant ces types primitifs (p. ex. : `Boolean`, `Float`, `Short`, …); le type `String`; quelques classes relativement communes (`java.util.Date`; `java.util.URI`; `java.math.BigInteger`; …). `JAX-RPC` supporte aussi les *Beans*; si ça vous intéresse, je vous invite à lire par vous-mêmes sur ce sujet.

Le moteur JAX-RPC supporte aussi des types qu'on qualifie de *Value Type*, qui sont des classes exposant un constructeur par défaut public, qui n'implémentent ***pas*** java.rmi.Remote et qui ne sont eux-mêmes composés que de types supportés par JAX-RPC[98].



Les tableaux de Java sont des objets conformes à la définition de *Value Type* et sont pris en charge par JAX-RPC eux aussi. Les types que JAX-RPC prend en charge sont traduits en arrière-plan dans des types conformes au standard SOAP.

[98] Notez que les attributs de toute classe qualifiée de *Value Type* ne doivent pas exposer d'attribut public qui soit final (donc constant) ou transient (ce qui a trait à la sérialisation des objets). Celles et ceux parmi vous qui se souviennent de leur cours de POO n'auront pas le réflexe, je l'espère, d'avoir recours à des attributs publics qui ne soient pas constants. Les attributs privés doivent quant à eux exposer une paire accesseur/ manipulateur conforme aux standards Java (get et set en minuscules, suivi du nom de l'attribut en question).

Marche à suivre pour concevoir le service Web

Nous allons concevoir un service Web représenté par une interface nommée `IEvaluateur`, et n'exposant qu'une seule méthode nommée `plusPetit()`, prenant en paramètre trois entiers signés sur `32` bits chacun (des `int` en Java) et retournant le plus petit d'entre eux.

```
// Fichier Evaluateur.java
package trucs;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface IEvaluateur extends Remote {
   public int plusPetit(int a, int b, int c)
      throws RemoteException;
}
```

Présumant que vous désiriez loger ce service Web avec d'autres services Web dans un lieu standard et présumant que votre ayez installé et lancé *Sun Application Server* :

(a) créez tout d'abord un répertoire pour les sources (p. ex. : **`apps`** comme dans `c:\sun\AppServer\`**`apps`**); puis

(b) créez aussi un sous-répertoire de ce répertoire pour les binaires qui seront générés (p. ex. : **`build`** comme dans `c:\sun\AppServer\`**`apps\build`**).

Implémenter le service

Un service Web en Java est, hormis les quelques règles énoncées ci-dessus, une classe comme toutes les autres. J'ai choisi d'utiliser un paquetage nommé `trucs`; le choix de ce nom est arbitraire.

Implémenter le service proposé ici est une tâche triviale. Compiler l'interface et l'implémentation du service, présumant le code dans le répertoire courant et les binaires destinés au répertoire `build/trucs` :

```
javac -d build *.java
```

```
package trucs;
import java.rmi.Remote;
import java.rmi.RemoteException;
public class EvaluateurImpl
   implements IEvaluateur {
   public int plusPetit(int a, int b, int c)
      throws RemoteException {
      int ppetit = a;
      if (b < ppetit) {
         ppetit = b;
      }
      if (c < ppetit) {
         ppetit = c;
      }
      return ppetit;
   }
}
```

Le développement avec Java pour des applications courantes est souvent fait à l'aide de la version la plus récente du *Java Development Kit* (`JDK`), version *Java 2 Standard Edition* (`J2SE`). Lorsqu'on développe des composants côté serveur comme ceux proposés ici, par contre, il importe de s'en tenir au `JDK` version *Java 2 Enterprise Edition* (`J2EE`), pour que les binaires des diverses classes utilisées soient pleinement compatibles entre eux. Assurez-vous donc d'utiliser un `JDK` conforme à la norme Java implémentée par votre version de `J2EE`, *pas* votre version de `J2SE`.

Rédiger un fichier de configuration

Dans le répertoire où se trouvent les sources[99], il importe de rédiger un fichier de configuration nommé **`config.xml`**. Le rôle de ce fichier est d'expliquer l'organisation du service Web.

Je vous propose, pour notre service Web, le fichier suivant. Seules les lignes en caractères gras nous importeront, les autres étant des balises qui resteront telles quelles d'un projet à l'autre :

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service name="PetitService"
           targetNamespace="urn:TiService"
           typeNamespace="urn:TiService"
           packageName="trucs">
      <interface name="trucs.IEvaluateur"/>
  </service>
</configuration>
```

On remarquera en particulier l'indication du nom de paquetage (**`trucs`**) et de l'interface (**`trucs.IEvaluateur`**). Le service Web sera nommé **`PetitService`**, dans l'espace nommé **`urn:TiService`**. Ces deux noms sont arbitraires.

Nous utiliserons le fichier `config.xml` en invoquant l'outil **`wscompile`** pour mener à la génération d'un fichier `WSDL` qui décrira que :

- le service se nommera `PetitService`;
- son espace nommé `WSDL` sera `urn:TiService`;
- les classes (binaires) du service seront dans le répertoire `build/trucs`; et
- l'interface du service sera `trucs/IEvaluateur`.

[99] Présumant que vous respectiez les noms de répertoires proposés plus haut, on parle ici du répertoire `apps`.

Générer les fichiers requis

Présumant que nous soyons dans le répertoire `apps`, l'invocation de `wscompile` pour générer le service Web à partir du fichier `config.xml` se fera comme suit :

```
wscompile -define -mapping build/mapping.xml -d build -nd build -classpath build config.xml
```

Cette commande génère la description `WSDL` du service `PetitService`, qui sera logée dans un fichier nommé **`PetitService.wsdl`**, de même que le fichier **`mapping.xml`**. Tous deux seront placés dans le répertoire `build`.

On remarquera quelques options dans l'invocation de `wscompile` ci-dessus :

- l'option **`-define`** demande de lire une interface Java et de générer la description `WSDL` correspondante;
- l'option **`-mapping`** indique le nom du fichier de *mapping* à générer[100]; et
- les options **`-d`** et **`-nd`** servent respectivement à indiquer où générer les fichiers binaires et les autres fichiers produits par l'exécution de `wscompile`.

Une fois l'exécution de `wscompile` réalisée, la génération de notre service Web est complète.

Les fichiers générés

Veuillez examiner l'annexe `00` pour voir le contenu du fichier `PetitService.wsdl`, de même que l'annexe `01` pour voir le contenu du fichier `mapping.xml`.

Procéder au déploiement

Un service Web rédigé en Java sera éventuellement transformé en servlet, qu'il en ait été un ou non à l'origine.

Chaque serveur Web aura sa propre stratégie de déploiement, du dépôt des fichiers sources ou binaires dans un répertoire spécial[101] jusqu'à un ensemble de configurations manuelles complexes utilisant une série de fichiers `XML`. Personnellement, j'ai utilisé `deploytool`, un assistant au déploiement muni d'une interface personne/ machine graphique et livré avec *Sun Application Server*[102], ce qui fut raisonnable sans être aussi simple que le déploiement avec `.NET`.

Une fois le déploiement réalisé, le service Web sera accessible à travers une `URL` reconnue par le serveur Web de votre choix sur son ordinateur hôte.

---

[100] Ce fichier décrit la correspondance entre les types `JAX-RPC` de la plateforme Java et ceux de `SOAP` pour l'interface du service Web.

[101] Ces fichiers seront alors compilés ou transformés en servlets au démarrage.

[102]. Voir `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXRPC3.html#wp124102`.

Rédiger un client Java pour un service Web

Présumant notre service Web `PetitService` décrit ci-dessus, plusieurs stratégies s'offrent à nous pour rédiger un client approprié :

- produire un **intermédiaire fixe** pour le service, ce qu'on nomme dans la terminologie Java un *Static Stub*[103]. On a alors la rapidité propre à l'utilisation de code compilé et la simplicité du modèle `RPC` brut, mais on obtient aussi un client lié à une spécification d'interface précise—il faut pouvoir compter sur la stabilité de l'interface `WSDL` du service Web[104];
- générer un **intermédiaire dynamique**, ce qu'on nomme dans la terminologie Java un *Dynamic Proxy*. Dans ce cas, on ne met le client en contact avec son intermédiaire qu'au moment où le client s'exécute. Cette stratégie est plus proche de celle utilisée dans le cours, avec le *marshalling* pris en charge par un *proxy* généré à partir de la spécification `IDL` mais inséré dans une `DLL` générée à part du client;
- générer des **invocations complètement dynamiques**, ce qu'on nomme en terminologie Java le *Dynamic Interface Invocation* (ou `DII`[105]). Ceci consiste à décrire chaque appel de service et ses paramètres dynamiquement, passant par un *proxy* générique[106]. Cette stratégie a l'avantage d'être la plus souple des trois (aucun *proxy* n'est compilé ou associé directement à une interface), mais a l'inconvénient d'être la plus lente et la plus complexe à utiliser: tout[107] doit alors être décrit à l'exécution;
- générer un client qui vivra dans un **fureteur Web**; *etc*.

Un exemple de client avec intermédiaire fixe et un exemple de client avec intermédiaire dynamique suivront (le temps aura bêtement manqué pour générer les autres).

---

[103] Il faut prendre ici le mot *stub* au sens du terme *proxy* dans le monde `COM`. Les *proxies* de `COM` sont des *stubs* en Java, et les *stubs* de `COM` sont des *ties* en Java. Sans commentaire.

[104] Cette stabilité d'interface devrait aller de soi dans un monde `CS` ou `OO` rigoureux, mais sachez que le monde des services Web est un monde `CS` plus … démocratique, disons, et où les règles de bienséance quant à la stabilité des interfaces publiques tendent, malheureusement, à être moins respectées que ce qui vous est enseigné en classe. Il se peut que, pour certains clients faisant face à des services aux interfaces plus mouvantes (ou à des stratégies de facturation changeantes, ce qui est bien possible), des stratégies à la fois plus lentes à l'exécution et plus flexibles soient à privilégier.

[105] …à ne pas confondre avec `DLL`!

[106] En Java, l'interface pour ce *proxy* générique est décrite par la classe `javax.xml.rpc.Call`.

[107] Par tout, on entend le nom de l'opération, le nom et le type de chaque paramètre, le port, le mode d'invocation de la méthode, et ainsi de suite.

Générer un intermédiaire fixe

Un intermédiaire fixe doit être généré avant que le client ne soit compilé.

Cette génération se fait habituellement à l'aide de l'outil `wscompile` et d'un descripteur `XML` indiquant l'endroit où l'intermédiaire pourra trouver la description `WSDL` du service en question.

Présumant la structure de répertoires décrite plus haut pour notre service Web, ajoutez le répertoire **`static-stub`** sous `apps`, où nous déposerons le code de l'intermédiaire en question et le fichier `XML` susmentionné, et un répertoire **`build`** sous **`apps/static-stub`** qui contiendra les binaires de l'intermédiaire.

Un fichier `XML` correct pour l'intermédiaire fixe vers notre service serait le fichier suivant, nommé **`config-wsdl.xml`**, que je vous propose de déposer dans **`apps/static-stub`** :

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location="http://localhost:8080/PetitService/TiService?wsdl"
       packageName="sstub"/>
</configuration>
```

La génération de l'intermédiaire fixe se fait ensuite en compilant ce descriptif `XML` à l'aide de `wscompile`, invoqué ici à partir du répertoire **`apps/static-stub`** :

```
wscompile -gen:client -d build -classpath build config-wsdl.xml
```

Cette invocation a pour effet de consulter le fichier de configuration `config-wsdl.xml`, d'en extraire le lieu où se trouve la description `WSDL` du service pour lequel on cherche à générer un intermédiaire côté client, et de générer des fichiers en fonction des diverses options passées à la ligne de commande.

Ici, l'option **`-gen:client`** demande la génération de l'intermédiaire côté client, prenant en charge la sérialisation et le *marshalling*. Les options **`-d`** et **`-classpath`** déterminent respectivement le répertoire de destination des fichiers générés et la valeur de la variable d'environnement `CLASSPATH` de Java pour la durée de la compilation.

Exemple de client avec intermédiaire fixe

Un exemple de client Java viable pour notre service irait comme suit. Il est présumé que le code source de ce client se trouve dans le répertoire `apps/static_stub` :

```
package sstub;
import javax.xml.rpc.Stub;
public class ClientEvaluateur {
   private String m_AdresseIntermédiaire;
   public static void main(String [] args) {
      try {
         // Création d'un intermédiaire côté client (nomenclature Java)
         Stub stub = créerIntermédiaire (); // voir plus bas dans la classe
         // Indiquer son adresse
         stub._setProperty
            (javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
             "http://localhost:8080/PetitService/TiService");
         // Conversions explicite de types (magie!)
         IEvaluateur pEval = (IEvaluateur) stub;
         // Invocation des méthodes qui constituent les services Web désirés
         int a = 3, b = 7, c = 4;
         System.out.println(pEval.plusPetit(a, b, c));
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
   private static Stub créerIntermédiaire() {
      // Création de l'intermédiaire. PetitService_Impl et getIEvaluateurPort
      // sont des noms générés par wscompile, et changeront d'un service à l'autre
      return (Stub) (new PetitService_Impl().getIEvaluateurPort());
   }
}
```

Une fois l'intermédiaire généré, ne reste plus qu'à générer le client. L'appel au compilateur Java va comme suit :

```
javac -classpath build -d build ClientEvaluateur.java
```

Ne reste plus qu'à exécuter le client, opération présumée ici comme étant faite à partir du répertoire `apps/static-stub` :

```
java -classpath build sstub.ClientEvaluateur
```

Tel que prévu, ce programme devrait afficher `3`, soit le plus petit des trois entiers passés en paramètre lors de l'invocation du service Web caché derrière l'interface `IEvaluateur`.

S'il manque des paquetages

Si des erreurs laissent croire à l'absence de certains paquetages, vous pouvez corriger le problème en modifiant l'invocation du compilateur de la manière suivante[108] :

```
javac -classpath "build;C:\Sun\AppServer\lib\jaxrpc-impl.jar;C:\Sun\AppServer\lib\jaxrpc-
api.jar" -d build ClientEvaluateur.java
```

…puis en modifiant l'invocation de la JVM de la manière suivante[109] :

```
java -classpath "build;C:\Sun\AppServer\lib\jaxrpc-impl.jar;C:\Sun\AppServer\lib\jaxrpc-
api.jar;C:\Sun\AppServer\lib\jax-qname.jar;C:\Sun\AppServer\lib\saaj-
api.jar;C:\Sun\AppServer\lib\saaj-
impl.jar;C:\Sun\AppServer\lib\endorsed\xercesImpl.jar;C:\Sun\AppServer\lib\endorsed\dom.jar"
sstub.ClientEvaluateur
```

Dans les deux cas, il est présumé que les paquetages de *Sun Application Server* sont utilisés et que les archives JAR qui l'accompagnent sont localisées dans les répertoires `c:\Sun\AppServer\lib` et `c:\Sun\AppServer\lib\endorsed`.

Certains des paquetages requis pour compiler un client ou invoquer des services Web en Java ne sont pas livrés avec une installation J2EE standard et doivent être pris à même les bibliothèques du serveur Web ou du serveur d'applications retenus, ce qui explique qu'on doive parfois[110] les spécifier individuellement.

[108] … le tout sur une seule ligne, malgré les apparences.

[109] … le tout sur une seule ligne ici aussi.

[110] Surtout si certaines variables d'environnement ne sont pas configurées convenablement.

Générer un intermédiaire dynamique

Un intermédiaire dynamique est généré lors de l'exécution du client, pas avant.

Les intermédiaires dynamiques sont générés au moment de l'exécution du client, mais demandent quand même un peu de préparation pour la production d'interfaces.

Comme dans le cas de l'intermédiaire fixe, on utilisera `wscompile` et un descripteur `XML` indiquant l'endroit où l'intermédiaire pourra trouver la description `WSDL` du service.

Présumant la structure de répertoires décrite plus haut pour notre service Web, ajoutez le répertoire **`dynamic-proxy`** sous `apps`, où nous déposerons le code de l'intermédiaire en question et le fichier `XML` susmentionné, et un répertoire **`build`** sous **`apps/dynamic-proxy`** qui contiendra les binaires de l'intermédiaire.

Un fichier `XML` correct pour l'intermédiaire dynamique vers notre service serait le fichier suivant, nommé ici **`config-wsdl.xml`**, que je vous propose de déposer dans **`apps/dynamic-proxy`** :

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location="http://localhost:8080/PetitService/TiService?wsdl"
       packageName="dynamicproxy"/>
</configuration>
```

La génération de l'intermédiaire fixe se fait ensuite en compilant ce descriptif `XML` à l'aide de `wscompile`, invoqué ici à partir du répertoire **`apps/dynamic-proxy`** :

```
wscompile -import -d build -nd build -f:norpcstructures -classpath build config-wsdl.xml
```

Les options à remarquer ici sont **`-d`** et **`-nd`**, pour indiquer respectivement le répertoire où seront déposés les fichiers binaires générés et celui où seront déposés les autres fichiers générés; de même que les options **`-import`** et **`-f:norpcstructures`** pour indiquer le souhait de ne voir générées *que* les interfaces abstraites.

Exemple de client avec intermédiaire dynamique

Un client viable exploitant un intermédiaire dynamique pour notre service `IEvaluateur` aura d'abord recours à une fabrique de services, qui servira à créer un intermédiaire pour ce service.

La méthode **`createService()`**, qui servira à créer l'intermédiaire, prendra en paramètre l'`URL` de la description `WSDL` du service et un **`QName`**, pour *nom qualifié*, qui est en fait un conteneur dans lequel sont répertoriés l'`URI` du service et son nom.

Une fois l'intermédiaire dynamique créé, il ne reste plus qu'à en invoquer les méthodes.

```
package dynamicproxy;
import java.net.URL;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;

public class ClientEvaluateur {
   public static void main(String [] args) {
      final String uriEspaceNommé = "urn:TiService";
      final String nomService = "PetitService";
      final String nomPort = "IEvaluateurPort";
      try {
         // Indiquer le lieu du fichier WSDL
         URL url = new URL("http://localhost:8080/PetitService/TiService?wsdl");
         // Instancier une fabrique de services
         ServiceFactory fabriqueServices = ServiceFactory.newInstance();
         // Instancier un service qui servira de fabrique d'intermédiaires
         Service srvEvaluateur = fabriqueServices.createService
           (url, new QName(uriEspaceNommé, nomService));
         // Créer un intermédiaire (utilise la réflexion)
         dynamicproxy.IEvaluateur pEval =
            (dynamicproxy.IEvaluateur) srvEvaluateur.getPort
               (new QName(uriEspaceNommé, nomPort),
                dynamicproxy.IEvaluateur.class);
         // Invocation des méthodes qui constituent les services Web désirés
         int a = 3, b = 7, c = 4;
         System.out.println(pEval.plusPetit(a, b, c));
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

Ne reste plus qu'à générer le client. L'appel au compilateur Java va comme suit :

```
javac -classpath build -d build ClientEvaluateur.java
```

Ne reste plus qu'à exécuter le client, opération présumée ici comme étant faite à partir du répertoire `apps/dynamic-proxy` :

```
java -classpath build dynamicproxy.ClientEvaluateur
```

Tel que prévu, ce programme devrait afficher `3`, le plus petit des trois entiers passés en paramètre lors de l'invocation du service Web caché derrière l'interface `IEvaluateur`. Vous devriez remarquer un délai significatif au démarrage, dû en grande partie à la génération dynamique de l'intermédiaire.

S'il manque des paquetages

Si des erreurs laissent croire à l'absence de certains paquetages, vous pouvez corriger le problème en modifiant l'invocation du compilateur de la manière suivante :

```
javac -classpath "build;C:\Sun\AppServer\lib\jaxrpc-impl.jar;C:\Sun\AppServer\lib\jaxrpc-
api.jar;C:\Sun\AppServer\lib\jax-qname.jar" -d build ClientEvaluateur.java
```

…puis en modifiant l'invocation de la `JVM` de la manière suivante:

```
java -classpath "build;C:\Sun\AppServer\lib\jaxrpc-impl.jar;C:\Sun\AppServer\lib\jaxrpc-
api.jar;C:\Sun\AppServer\lib\jax-qname.jar;C:\Sun\AppServer\lib\saaj-
api.jar;C:\Sun\AppServer\lib\saaj-
impl.jar;C:\Sun\AppServer\lib\endorsed\xercesImpl.jar;C:\Sun\AppServer\lib\endorsed\dom.jar"
dynamicproxy.ClientEvaluateur
```

Activité—Générer un service Web avec C#

Pour réaliser l'activité ci-dessous, il vous faut avoir au préalable installé *Visual Studio .NET* et IIS avec extensions *FrontPage*. La simplicité gagnée au développement dépend du support de ces outils.

Créer un service Web de somme d'entiers avec C#

Les services Web avec C# (et avec VB.NET, mais nous limiterons notre discours à C# pour fins de brièveté) sont très faciles à générer :

- démarrez *Visual Studio .NET*, et créez un nouveau projet, section C#, type ***Service Web ASP.NET***[111], que vous nommerez **ServiceSomme**;
- par défaut, le projet s'installera dans http://localhost/ServiceSomme/[112];
- la fenêtre Design qui se propose à nous par défaut est sans intérêt pour le service simple que nous construisons. Cliquez sur le lien nommé **cliquez ici pour passer en mode code**;
- par défaut, un projet bidon (mais opérationnel) existe dans l'espace nommé **ServiceSomme**. Un premier service par défaut y est créé et se nomme **Service1**. Renommez ce service pour qu'il porte un nom significatif (disons **ServiceSommeImpl**);
- une méthode bidon de type Hello World existe déjà mais est en commentaire. Je vous invite à enlever les commentaires et à tester ce service, c'est-à-dire à le compiler et à « l'exécuter »;

Prenez note qu'un service Web n'est exécuté que si un client fait affaire avec lui; par défaut, le client de *Visual Studio .NET* sera le fureteur par défaut sur votre poste (des versions antérieures utilisaient Internet Explorer par défaut, sans égard aux préférences des usagers).

- tester le programme bidon à travers la page Web générique signifie appuyer sur le bouton Appeler. Vous devriez recevoir le message Hello world et le voir s'afficher dans la page Web résultante.

Pour insérer votre propre service de somme, remplacez la méthode HelloWorld() par la méthode suivante (ou encore ajoutez la méthode suivante dans la classe ServiceSommeImpl, au choix).

```
[WebMethod]
public int Somme(int x, int y)
{
   return x + y;
}
```

Compilez et testez le tout. Bingo! Voilà le service Web offrant un service de somme d'entiers généré et utilisable.

[111] Les services Web sous .NET fonctionnent grâce au support de ASP.NET, et ne peuvent pour l'instant s'exécuter que sur des serveurs Web en mesure de gérer cette technologie.

[112] http://localhost/ correspond normalement, avec IIS, au répertoire C:\Inetpub\wwwroot\.

Ce qui prend en charge la magie de l'interopérabilité

L'attribut `[WebMethod]` fait en sorte (entre autres choses) de pousser le compilateur à générer un *stub* pour les appels `SOAP` en fonction de la méthode à laquelle l'attribut en question s'applique. C'est aussi simple que ça.

Les `JDK` les plus récents procèdent de manière semblable, avec des annotations comme `@WebMethod` qui allègent de beaucoup la rédaction du code à proprement dit.

Voyant les intrants et les extrants de la méthode, le compilateur est capable de décrire sous forme `WSDL` le service correspondant. Tout en générant une description `WSDL` complète, le compilateur en profite pour générer le code pour procéder au *marshalling* des intrants et des extrants à travers un *stub* qui effectuera l'appel à proprement dit de la méthode.

À notre insu, nous venons de générer un composant sans état sous le mode d'un service Web. La magie du moteur `.NET` pour les services Web tient à l'automatisation de cette mécanique.

Le service Web selon différents fureteurs

Vous pouvez tester ce service avec un autre fureteur (par exemple Firefox), pour voir que le tout fonctionne, quoique le résultat de l'appel de service ne soit pas traité de la même manière par les deux fureteurs.

La raison pour cette différence dans le traitement de la réponse tient au fait que la norme `SOAP` n'est pas implantée entièrement de la même manière par toutes les entreprises. Certains champs `SOAP` sont jugés obligatoires par certains outils et optionnels par d'autres, ce qui introduit, tristement, des incompréhensions entre les différentes plateformes.

Les problèmes causés par ces différences sont la plupart du temps minimes, mais existent tout de même et demandent de nous de la prudence. `SOAP` est encore un standard assez jeune.

Espaces nommés uniques

On déploie normalement des services Web à des fins commerciales. Chaque service Web devrait être offert dans un espace nommé spécifique à la compagnie le déployant.

Par défaut, un service Web généré avec *Visual Studio .NET* utilisera un espace nommé générique (nommé `tempuri.org`) qui est convenable pour déverminer le service, mais beaucoup moins pratique pour une utilisation commerciale. `IIS` se plaindra en voyant apparaître cet espace de nom générique dans la description `WSDL` du service.

La clé pour offrir un service valide est d'insérer un attribut indiquant un nom légal juste avant la classe `ServiceSommeImpl`. Par exemple :

```
[WebService(Namespace="http://macompagnie.com/WebServices/")]
public class ServiceSommeImpl : System.Web.Services.WebService
{
   // ...
```

Le nom utilisé peut être à peu près n'importe quoi de compréhensible, significatif et unique. Cela dit, respecter des standards est une bonne chose, et ici on a un cas légèrement problématique.

En effet, si `MSDN` suggère d'utiliser une `URL`, la norme `W3C` est d'utiliser, de manière plus large, un ***URN*** (*Uniform Resource Name*) pour servir d'espace nommé.

⇒ Un **URN** doit être à la fois unique et persistant. Il ne doit pas changer avec le temps, sinon il faudrait le redécouvrir à chaque utilisation, ce qui serait inacceptable pour des services à vocation commerciale.

Un `URN` typique aurait plutôt la forme `urn:`*`NID`*`:`*`NSS`* où `NID` est un identificateur d'espace nommé et où `NSS` est une chaîne de caractères identifiant de manière unique la ressource dans l'espace nommé en question.

Pour notre service, un `URN` acceptable aurait pu ressembler à quelque chose comme `urn:servicesdemacompagnie:somme`.

Créer un client pour ce service Web de somme d'entiers avec C#

Pour ce qui est de présenter un client C# pour le service Web `ServiceSomme`, nous irons au plus simple et ferons un programme console capable de calculer la somme de deux entiers à l'aide de notre service et d'afficher cette somme.

La manière la plus simple de procéder pour y arriver est sans doute :

- de créer un nouveau projet C# de type application console;
- d'y insérer ce qu'on nomme une référence Web (***Projet➔Ajouter une référence Web***) au service désiré. Sous `.NET`, l'ajout d'une référence Web se fait par une recherche sur un ordinateur donné ou sous `UDDI`, et extrait de l'information `WSDL` trouvée la description opérationnelle des services disponibles. Si vous avez installé votre service Web sur l'ordinateur local (`http://localhost/`), procédez à une recherche sur l'ordinateur local et votre service devrait apparaître dans la liste de ceux disponibles. Sélectionnez le service désiré et sélectionnez **`Ajouter cette référence`**;
- d'insérer, dans la méthode `Main()` de la classe créée par défaut dans votre projet (pour faire simple, car on pourrait utiliser le service Web comme on utilise n'importe quelle méthode), le code requis pour instancier un *proxy* pour le service et réaliser un appel viable à ce service.

Ce qui prend en charge la magie de l'interopérabilité

Lorsque la référence Web aura été ajoutée à votre projet, l'explorateur de projet connaîtra une représentation `OO` d'un *proxy* local pour le service correspondant. Le moteur `.NET` est capable de construire automatiquement un *proxy* pour le service Web à partir de seule sa description `WSDL`. Visiblement, pour les services Web, `WSDL` joue le rôle d'un `IDL`.

Dans mon projet client[113] de démonstration, le *proxy* à la référence Web nommée `localhost.ServiceSommeImpl` apparaît sous le nom (local mais fort compréhensible) **`ClientSommeCDièse.localhost.ServiceSommeImpl`**.

[113] Espace nommé `ClientCDièseSomme`, classe `ClientSomme`.

Invoquer le service Web

Ainsi, le code suivant est un appel correct au service Web :

```
ClientSommeCDièse.localhost.ServiceSommeImpl pSomme =
   new ClientSommeCDièse.localhost.ServiceSommeImpl();
int Résultat = pSomme.Somme (2, 3);
Console.WriteLine ("Somme de 2 et 3 selon le service Web: {0}", Résultat);
```

Ça ressemble beaucoup à l'utilisation d'un objet COM représenté par une coclasse, n'est-ce pas? Effectivement, c'est pratiquement la même chose.

On gagne, avec un service Web, la souplesse d'exposer nos composants à travers un serveur Web sous la forme d'une URL, avec gestion du composant et de la sécurité pleinement prises en charge par ledit serveur, de même que la possibilité pour d'autres de les découvrir à distance.

Si vous explorez un peu la référence Web telle que perçue par votre client, vous verrez entre autres que le service Web offre une interface synchrone et une interface asynchrone, et ce de manière automatique à cause de la génération dynamique de *proxy* rendue possible par la description WSDL du service.

On perd de la vitesse au lancement, en retour, à cause de l'invocation devant passer par HTTP, SOAP et les règles du serveur. L'opinion générale est que c'est là, dans l'ensemble, un gain net.

Créer un client pour ce service Web de somme d'entiers avec Java

La section présente dans les versions antérieures de ces notes de cours et traitant de la rédaction de clients pour services Web (et de services Web à proprement dit) doit être réécrite à court terme dû à des modifications majeures de stratégie du côté de Sun.

Malheureusement, pour cette version-ci, le temps m'aura manqué.

Je vous invite donc entre-temps à examiner ce très chouette didacticiel qui devrait vous permettre de vous orienter :

```
http://developers.sun.com/prodtech/javatools/jsenterprise/reference/techart/jse8/websvcs_accessing.html
```

Si vous avez pour objectif de faire interagir un client Java avec un service Web .NET de manière asynchrone, je vous propose l'article disponible à l'URL suivante :

```
http://developers.sun.com/sw/building/tech_articles/async_paper.html
```

Vous allez reconnaître en grande partie, dans le modèle proposé par .NET, l'approche proposée pour les interfaces asynchrones sous COM. Ça ne devrait donc pas être trop dépaysant.

Un mot sur WSDL et UDDI

Ce qui suit n'est pas une introduction technique à WSDL et à UDDI. Pour une telle introduction, référez-vous aux sections sur la mise en place et sur l'utilisation de services Web, plus haut.

Le monde des systèmes répartis contemporains est un monde très vivant, bien sûr, mais aussi extrêmement volatile.

On peut sans peine prétendre que ce monde est à la croisée des chemins, pris entre plusieurs technologies en pleine compétition, entre plusieurs contraintes et préoccupations conflictuelles, presque paradoxales lorsque prises deux à deux :

- plateformes Java et .NET;
- technologies programmatiques et technologies documentaires basées XML, prisées par les entreprises exploitant beaucoup le commerce électronique;
- réseaux de composants avec états et sans états;
- centraliser les données dans une BD ou les répartir à travers le réseau;
- ouverture des marchés et considérations de sécurité; *etc*.

Les systèmes répartis et l'entreprise

Le volet ouverture des marchés est peut-être le plus important pour l'ensemble des entreprises, qu'elles soient versées ou non dans les technologies de l'information.

Dans un monde où les corridors de libre-échange et de libre circulation des biens, toute entreprise veut être en droit et en mesure d'offrir ses services à l'ensemble du marché. L'ensemble du marché n'est plus local, il faut bien l'avouer : il est mondial.

Pour l'entreprise désireuse de vendre des produits et des services (ou d'en acheter), que ces produits soient logiciels ou non, il ne suffit plus d'acheter un espace publicitaire dans le bottin des pages jaunes. On a alors recours à la mise sur pied d'un site Web pour rendre accessible de l'information sur notre entreprise et sur ce qu'elle offre, et à la recherche sur Internet pour trouver ce dont nous avons besoin.

Un processus qui demande une intervention humaine?

Ce processus est, étrangement, *manuel*. À l'aide d'outils logiciels extrêmement puissants (pensez au moteur de recherche *Google*) et de l'immense réseau de réseaux qu'est Internet, des intervenants humains procèdent à la présentation pour des humains de sites Web attrayants pour d'autres humains, alors que des humains procèdent à des recherches qui les mèneront, si leurs choix de mots clés sont adéquats et si les *bons* sites Web pour *leurs* fins sont bien annoncés, à trouver la perle rare.

La prise de contact entre vendeur et acheteur se fera par la suite souvent par courriel ou par téléphone, mis à part peut-être dans des cas de vente au détail, comme avec *Amazon.ca* ou avec *eBay*, où l'acheteur insère dans un panier virtuel des objets virtuels qu'il paie par crédit et reçoit éventuellement par courrier postal sans jamais vraiment interagir de manière directe avec un homologue humain.

Le caractère manuel du processus demeure aussi dans le domaine plus large de vente au plus offrant, qu'on parle d'un appel d'offres (ou **RFP**, pour *Request For Proposal*), par lequel on demande à des vendeurs d'offrir une solution à un problème et où l'on choisit parmi les propositions soumises celle qui nous convient le mieux, ou d'un **RFQ** (*Request For Quote*) par lequel un vendeur offre ouvertement ses services aux clients potentiels et choisit parmi ceux qui sollicitent ses services.

Plusieurs gros projets, tous domaines confondus, sont attribués par l'une ou l'autre des ces mécaniques. On comprendra donc pourquoi il est préoccupant pour les entreprises d'être essentiellement à la merci d'une mécanique manuelle.

Le rôle de UDDI

Sachant ceci, on est plus en mesure de comprendre l'intention derrière l'initiative UDDI, pour *Universal Description, Discovery and Integration*, et le rôle que ce standard est appelé à jouer.

Un consortium[114] imposant incluant les plus grands joueurs du monde des technologies de l'information (*Hewlett Packard*, IBM, *Microsoft*, OMG, SAP, *Sun Microsystems*, et ainsi de suite) travaille de manière conjointe à son évolution, à sa diffusion et à son acceptation dans le marché global.

Présenté simplement, UDDI est un bottin intelligent des pages jaunes. Utilisé de manière naïve, un site UDDI permet une interaction simplifiée et standardisée, à l'aide de moteurs de recherche, pour retrouver des entreprises offrant ou cherchant des produits et des services. Les entreprises y sont classées selon des critères qu'elles fixent elles-mêmes, ce qui permet à chacune de s'annoncer comme elle le juge convenable.

Sachant ceci, on peut sûrement voir poindre l'intérêt commercial de UDDI, mais pas nécessairement percevoir en quoi utiliser UDDI sera préférable à procéder à une recherche normale sur Internet.

En effet, pour voir les avantages d'UDDI, et pour comprendre en quoi UDDI connecte aux services Web et au modèle CS, il faut examiner comment se dynamise UDDI et comment on peut y offrir ou y choisir des *services*.

[114] Le site officiel du consortium est http://www.uddi.org, mais des sites imposants sont aussi maintenus par les géants que sont IBM (http://uddi.ibm.com), *Microsoft* (http://uddi.microsoft.com) et SAP (http://uddi.sap.com).

Le rôle d'un UBR

Chaque entreprise est décrite sous UDDI dans un **UBR**, pour *Universal Business Registry*. Les différents UBR dans le monde sont des miroirs régulièrement mis à jour les uns des autres.

Dans un UBR, on peut, en gros :

- inscrire une entreprise;
- inscrire un service pour une entreprise donnée;
- inscrire une taxonomie (un modèle d'interaction) pour un service donné;
- inscrire un *Service Binding* (une manière d'utiliser un service, surtout pour des services électroniques comme des fonctions ou des interfaces) pour un service donné; et
- découvrir une entreprise ou un service à partir d'un ensemble de critères de recherche normalisés et standardisés.

Les interactions avec un UBR se font par transactions respectant la norme XML et transportées sur protocole HTTP ou HTTPS, ce qui les rend aptes à traverser la plupart des murs coupe-feu et les colle aux standards transactionnels les plus communs et les plus courants sur Internet. Un avantage simple mais non négligeable de ce choix est qu'il devient aisé de mettre sur pied des pages Web permettant à des humains d'inscrire des entreprises à un UBR[115].

Facturation et paiement?

Notez que les questions comme *Comment facturer?* ou *Comment indiquer les modes de paiement qui nous sont acceptables?* sont adressées par le modèle UDDI, et font partie des inscriptions descriptives des compagnies et des services.

Si vous désirez en savoir plus à ce sujet, je vous suggère de procéder à vos propres recherches et de vous documenter par vous-mêmes. La question de la facturation est à la fois importante et complexe, mais je ne suis pas un comptable ou un administrateur, et je crains de ne pas aller chercher pour vous l'information qui vous serait véritablement utile.

Pages jaunes?

Un registre UDDI offre trois catégorisations distinctes pour les entreprises et les services qui y sont inscrits :

- les **pages blanches** évoquent leur équivalent dans un bottin téléphonique. On trouve dans les pages blanches d'un registre UDDI des choses comme le nom d'une compagnie, son numéro de téléphone, son adresse postale, *etc.*;
- les **pages jaunes** sont aussi semblables à leur équivalent téléphonique. Dans les pages jaunes UDDI, on trouve diverses offres de services pour chaque compagnie inscrite; enfin
- les **pages vertes** UDDI n'ont pas d'équivalent téléphonique direct. Elles servent à décrire l'information technique quant aux services inscrits au registre UDDI, qu'il s'agisse de services désirés ou offerts. Les modèles *métadescriptifs* de UDDI pour les services (ce qu'UDDI nomme les ***tModel***s) se trouvent ici.

[115] Voir entre autre https://uddi.microsoft.com/register.aspx pour un exemple opérationnel.

La place des UUID dans UDDI

Comme c'est souvent le cas dans le monde des systèmes répartis, et en particulier des SCS, l'architecture UDDI dépend en partie de la capacité d'identifier de manière unique et sans ambiguïté les entreprises, les services et ainsi de suite.

Vous serez peut-être surpris(e) de savoir que le consortium œuvrant sur UDDI a choisi d'identifier les objets UDDI par des UUID, exactement selon la stratégie de COM[116]. Cette approche a été jugée à la fois simple et élégante par le consortium.

Nommer les choses de manière unique

Le souci d'identifier de manière unique les objets apparaît depuis longtemps sur Internet avec les URL et (de manière plus générale) les URI[117]. On le voit aussi dans des infrastructures par composants comme COM avec ses UUID sous plusieurs déclinaisons (LIBID, GUID, CLSID, etc.) et CORBA avec ses IOR[118].

La capacité de nommer de manière unique les choses sert aussi beaucoup dans des stratégies de sérialisation pour mettre en place des stratégies de persistance d'objets.

Chercher dans UDDI

Chercher dans un registre UDDI se fait par une requête HTTP encapsulant la description XML de la recherche à réaliser. Ceci permet de générer des interfaces Web aux moteurs de recherche UDDI, et permet aussi de créer des programmes capables de générer leurs propres requêtes de recherche et de décoder les résultats de ces recherches.

Le volet programmatique permettant d'automatiser les accès à UDDI peut se faire en n'importe quel langage permettant de manipuler des chaînes de caractères, mais Java et .NET sont les deux plateformes pour lesquelles il est le plus facile de développer de tels outils.

En effet, .NET est livré avec un UDDI SDK complet, et offre l'environnement de développement le plus convivial qui soit (à ce jour) pour accéder à UDDI, mais impose un prix quand même important pour ce faire: développer et tester avec les outils *Microsoft*, ce qui impose de souscrire au programme Passport pour procéder à des tests… Un prix que certains ne voudront peut-être pas payer.

Avec Java, de bons outils sont fournis pour interagir avec UDDI, mais ceux-ci demandent plus de programmation que ne le font les outils *Microsoft* destinés aux mêmes fins. Le gain de liberté qu'offre ici Java entraîne un plus grand effort de la part des développeurs.

---

[116] Pour en savoir plus sur les UUID, voir http://kruithof.xs4all.nl/guid-uuid-make.html

[117] Rappel: URL signifie *Uniform Resource Locator*. Chaque URL est un lieu logique unique pour tout objet. Une URL débute par une désignation de protocole typique comme http:// ou ftp://. Une URI, pour *Uniform Resource Identifier*, est une idée plus générale de point dans l'espace d'information qu'est Internet. On considère maintenant URI comme plus adéquat, techniquement, que URL, mais tout URL est aussi un URI (l'inverse n'est pas vrai). Pour en savoir plus, voir http://www.w3.org/Addressing/.

[118] Par IOR, on entend *Interoperable Object Reference*. Un IOR sous CORBA est l'équivalent opérationnel d'un UUID avec COM, à ceci près qu'un UUID représente un gros entier alors qu'un IOR représente une longue chaîne de caractères. Des utilitaires sont disponibles pour générer des IOR, similaires à ceux qui permettent de générer des UUID.

Les registres UDDI est les services Web

Du côté CS, on sera principalement intéressé à la possibilité d'offrir ou d'acheter automatiquement des services Web via un registre UDDI, ce qui peut équivaloir, dans un monde idéal, la possibilité de laisser agir un phénomène de libre concurrence entre divers serveurs offrant la même gamme de services, et ce de manière automatique.

En effet :

- un client potentiel pourrait, périodiquement, consulter un registre UDDI dans le but de connaître les fournisseurs potentiels pour un service dont il a besoin;
- ayant accès directement aux termes possibles d'entente et aux frais associés à chacun des fournisseurs potentiels, le client pourrait procéder à une sélection en fonction de critères qui lui sont propres (durée de l'offre de service; termes de suivi et d'entretien; coût brut; valeur ajoutée; réputation de la firme offrant le service; *etc*.);
- selon les critères et les situations, on pourrait créer ainsi un marché de location ou d'achat de services à la carte. Des logiciels clients pourraient être écrits de manière à exploiter des services *a priori* inconnus (mis à part leur interface) à partir d'un budget d'achat, et des services pourraient être élaborés et être mis à la disposition de clients *a priori* inconnus (quoiqu'on présume ici que les fournisseurs de services auront la sagesse de procéder à une étude de marché avant d'investir temps et argent dans le développement de services Web).

Le modèle de négociation de services souhaité par le consortium UDDI, en ce sens, n'est pas sans rappeler le modèle transactionnel qu'on connaît dans les marchés boursiers.

Présumant une liste claire de critères et un formalisme pour décrire une offre de services, la place des humains ne serait pas au sein de la négociation du contrat mais bien aux étapes où l'on fixe les critères ou l'offre, et (probablement) à l'étape de validation de la négociation.

Décrire un service Web

Décrire clairement un service ou un groupe de services Web demande à la fois l'équivalent d'une ou plusieurs description IDL[119], pour que les services soient utilisables sans qu'on ait à connaître les technologies exploitées par les clients potentiels, de même que d'une description dans un métalangage[120] du ou des services de manière à ce qu'on puisse décider, pour chaque service, ce à quoi il peut bien nous servir. Les services Web se prêtent bien à de tels efforts taxonomiques pour plusieurs raisons :

- en premier lieu, les services Web utilisent pour la plupart le protocole **HTTP** comme support protocolaire. Avec HTTP, la communication se fait part transfert de documents;
- les règles de notation **XML** ont été choisies comme base syntaxique pour décrire les services, au sens où elles permettent une description textuelle, donc accessible à tous les langages de programmation, des services et des appels de services. Avec XML, on gagne une grande latitude pour décrire des documents de manière à la fois neutre, souple et très expressive;
- il se trouve que XML est un guide pour décrire des protocoles (et des structures de données en général), pas un protocole en soi. Le protocole **SOAP** est l'application de XML la plus souvent choisie pour décrire les services Web, ce qui permet à plusieurs outils existants (.NET, bien sûr, mais aussi certaines implantations de COM, de CORBA et de Java) d'offrir des services Web avec un minimum d'effort.;

Pour en donner une description brève, disons qu'un document SOAP permet d'envelopper de manière standardisée et compréhensible une description XML d'un appel de fonction. Sous SOAP, on représente un appel de fonction par un document XML.

- la notation **WSDL** est une notation XML enrobant les services SOAP pour en faire des services Web à part entière. Un document WSDL décrit à la fois le lieu où se trouve un service Web et l'interface de ce service. Avec la description WSDL d'un service, on sait entre autres si le mode transactionnel associé à une taxonomie donnée de service doit être par passage de document ou par une implantation Web de RPC. On sait aussi quel port utilise le service; quels types de messages il transige; quelles particularités techniques lui sont associées (langage précis, ordinateur spécifique, ce genre de truc… s'il y a lieu, évidemment). ***WSDL est l'équivalent de IDL pour les services Web***;
- enfin, **UDDI** sert de registre pour lister, par entreprise offrant ou consommant des services, des descriptions WSDL de services possibles, et permet d'associer offre et demande.

En voyant cet échafaudage, on comprend mieux pourquoi le modèle UDDI dépasse en capacité et en intérêt la simple recherche manuelle sur Internet.

---

[119] Pour utiliser un service, comme pour appeler une fonction ou une méthode, il faut en connaître au minimum l'équivalent d'un prototype: nom, type et nombre des intrants et des extrants; cas d'erreurs ou d'exceptions possibles; restrictions quant aux valeurs d'intrants jugées valides; etc. Notez qu'un même service peut exposer plusieurs taxonomies distinctes, ce qui équivaut à offrir plusieurs fonctions différentes pour réaliser une même tâche, selon les besoins (on peut parler par exemple d'une version synchrone et d'une version asynchrone d'un même service, par exemple, pour permettre aux clients cherchant la souplesse et à ceux cherchant la simplicité d'y trouver tous deux leur compte à l'usage).

[120] On entend ici une description semblable en fonction à des commentaires clairs, mais destinée à des homologues logiciels plutôt qu'à des humains: rôle du service; dépendance entre services (doit-on appeler celui-ci avant d'appeler celui-là?); comportement synchrone ou asynchrone, bloquant ou non; *etc.*

En résumé

Le modèle UDDI offre un système de registres intelligents d'entreprises et de services.

Pour utiliser un registre UDDI, une entreprise doit s'y enregistrer. Ceci peut être fait de manière manuelle ou automatisée, à l'aide du protocole de transport HTTP ou HTTPS dans un cas comme dans l'autre.

Pour offrir un service via UDDI, une entreprise doit l'y inscrire et en décrire la taxonomie de manière formelle et standardisée. Pour solliciter un service via UDDI, une entreprise doit effectuer une recherche manuelle ou automatisée. Meilleure sera la rencontre entre les critères de recherche et la taxonomie descriptive d'un service, et plus rapidement le service sera identifié.

Plusieurs services peuvent être offerts (ou sollicités) par une même entreprise. Plusieurs taxonomies distinctes peuvent être utilisées pour décrire un même service.

Lorsqu'un service est offert selon le modèle CS (présenté comme service Web), le service doit inclure un ou plusieurs *Bindings* à même sa taxonomie. Chaque *Binding* explique comment interagir, par programmation, avec ledit service.

La notation XML y est utilisée sous UDDI pour décrire entreprises et services.

Le support HTTP y est privilégié pour la communication et les offres de services, ce qui simplifie énormément la gestion des murs coupe-feu.

Le protocole SOAP, sur HTTP, y est privilégié pour décrire les interfaces aux services Web.

La norme WSDL associe aux services une description, un lieu et offre un métalangage pour faciliter la compréhension automatisée de ce que font, effectivement, les services.

On décrit un service Web avec WSDL comme on décrit un service COM ou CORBA avec IDL. On peut construire un *proxy* et un *stub* à partir de cette description, dynamiquement (si on a besoin de souplesse plus de que performance) ou à la compilation (si on a au contraire d'abord besoin de vitesse).

L'approche AJAX

Internet est un milieu chic, où se côtoient les modes et les vagues plus ou moins éphémères d'excitation. Parmi les modes tenaces et un peu plus *enthousiasmantes* que la moyenne, on trouve celles autour des termes **AJAX**[121], que nous examinerons ici[122].

Qui dit *mode* dit (trop souvent) surenchère et débordements. Ainsi, démystifions un peu la chose. En quelques mots, donc :

- AJAX n'est pas une technologie; c'est une technique, une approche. L'acronyme AJAX ne désigne pas un produit mais bien *Asynchronous JavaScript And XML*, donc une combinaison de technologies adjointes à un mode d'invocation (asynchrone)[123];
- AJAX n'est pas une panacée mais bien un raffinement de certaines stratégies applicables aux IPM de type Web;
- AJAX n'est pas un modèle CS mais peut servir de complément au volet client d'un SCS avec IPM présentée par un fureteur.

Nous allons, dans cette section, explorer à la fois la philosophie derrière l'approche AJAX, des exemples en tirant profit tout en prenant soin de réfléchir aux limites et aux contraintes particulières de cette approche.

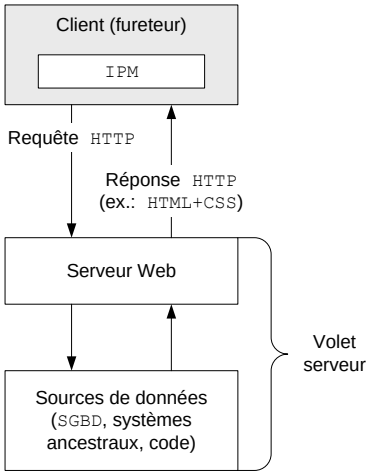
---

[121] Notez que certains des schémas présentés ici sont des inspirations directes du texte original de définition du terme AJAX: http://www.adaptivepath.com/publications/essays/archives/000385.php, qui date du 18 février 2005. C'est un choix volontaire.

[122] On trouve aussi dans cette catégorie ce qu'on nomme le **Web 2.0**, chose sur laquelle nous reviendrons dans une section ultérieure.

[123] Pour être bien honnête, même si les technologies JavaScript et XML font habituellement partie de l'approche, suffisamment pour faire partie du nom de l'approche, même ces deux technologies ne sont pas véritablement nécessaires à la mise en place d'une solution de type AJAX. On pourrait implémenter la même approche autrement : bien le recours à XMLHttpRequest force en soi le recours à XML, un fureteur pourrait offrir une stratégie asynchrone d'invocation qui ne serait pas basée sur des documents XML. Ce serait contreproductif, soit, mais ce ne serait pas impossible.

Approcher le Web de manière traditionnelle



Le développement Web a explosé depuis la fin des années '90. Traditionnellement, une IPM exposée sur le Web suivait l'un des patrons suivants (tel qu'explicité par le schéma à droite) :

- présenter une page Web sous forme statique, décrite sous forme HTML;
- présenter une page Web sous forme statique mais avec des éléments plus dynamiques, comme par exemple des applets, du code JavaScript ou JScript ou (horreur![124]) des contrôles *ActiveX*, qui sont des objets COM, le tout dans une stratégie de dynamisation du volet client de l'expérience de navigation;
- présenter une page Web sous forme statique mais avec une génération dynamique de contenu, comme par exemple des scripts PHP, des servlets Java ou des pages JSP ou ASP, des scripts CGI écrits en PERL, *etc*., le tout dans une stratégie de dynamisation du volet serveur de l'expérience de navigation
- la page Web se veut légèrement dynamique mais avec des balises interactives comme <form> et <input> qui, combinées avec des scripts côté serveur, mènent à la génération de nouvelles pages en réponse à des requêtes décrites (typiquement) par les contenu des champs sur la page Web au moment de l'invocation de la requête par pression sur le contrôle représentant la contrepartie visuelle de la balise <input>; et ainsi de suite.

On remarquera que la mise à jour des pages Web dépend dans chaque cas d'une action de navigation à proprement dit : rafraîchir la page, suivre un hyperlien, provoquer un rafraîchissement de manière indirecte par l'invocation d'une requête HTTP de type GET ou POST suite à une interaction avec un formulaire, *etc.*

Les systèmes installés sur un poste de travail offrent donc, traditionnellement, une expérience plus riche à leurs usagers. Le temps de réponse tend à y être meilleur, ce qui se comprend un peu étant donné la localité des éléments constitutifs du système en question, et il est possible de ne rafraîchir que certains éléments à la fois parmi ceux présentés à travers l'IPM (chose bien moins lourde que de rafraîchir chaque fois l'interface toute entière).

[124] Les *ActiveX* dans les pages Web sont une très, *très* mauvaise idée. On parle de laisser un contrôle provenant d'un site extérieur s'installer sur un poste de travail et y avoir des droits d'accès arbitrairement complexes, incluant ceux d'écrire sur les disques locaux. Ne faites pas ça!

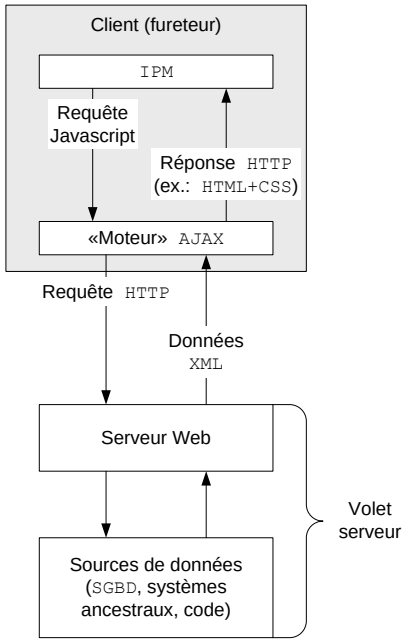
Redéfinir le paradigme[125] de l'expérience Web

Ce qui empêchait, de l'avis de plusieurs, l'avènement d'un véritable paradigme d'interface dynamique aux applications à travers un fureteur, donc l'élévation vers ce qu'on tend à appeler le ***Web 2.0***[126], était **l'interactivité limitée** du modèle Web de base.

⇒ Le mot **AJAX**[127] désigne une technique de programmation servant entre autres à pallier le caractère limité de l'expérience client sur le Web.

⇒ La désignation AJAX est de Jesse James Garrett[128]. Comme c'est souvent le cas, nommer quelque chose lui donne une existence propre et tend à faire exploser les idées.

⇒ Le mot AJAX ne désigne pas une technologie. Cette technique peut être implémentée à l'aide d'un tas d'outils et de langages.



L'approche AJAX repose en partie sur une classe JavaScript nommée XMLHttpRequest[129], développée en 1998 par *Microsoft* pour IE5 et implémentée presque aussitôt par la plupart des fureteurs.

Cette requête a la particularité de ne pas résulter en un réaffichage éventuel de la page Web complète. C'est cette particularité toute simple qui est, en grande partie, la clé du bonheur (et des critiques) dans le monde AJAX.

La commande XMLHttpRequest est devenue la source de quelques *Killer Apps* d'Internet grâce à Google, en particulier les très impressionnants[130] *Google Suggests*[131] et *Google Maps*[132] que tous cherchent à imiter depuis.

---

[125] J'essaie d'éviter d'utiliser des mots aussi galvaudés que le mot *paradigme*, mais que voulez-vous…

[126] Voir http://en.wikipedia.org/wiki/Web_2.0, mais notez que Web 2.0 porte plusieurs sens dont celui du Web sémantique—voir http://en.wikipedia.org/wiki/Semantic_Web—doublé de balises sémantiques de type RDF (description de ressources) et de OWL (balises ontologiques). L'insertion de balises porteuses de sens dans HTML (au lieu de <b>, au lieu de <i>) s'inscrit dans la même veine mais d'une manière plus immédiate. Le milieu Web se rapproche d'une forme de maturité, en partie grâce à des efforts de linguistes, par une mouvance rappelant celle des compilateurs formels ayant suivi FORTRAN et COBOL.

[127] Voir http://en.wikipedia.org/wiki/AJAX

[128] Voir http://www.adaptivepath.com/publications/essays/archives/000385.php, 18 février 2005, tel que mentionné précédemment.

[129] Voir http://en.wikipedia.org/wiki/XMLHttpRequest

[130] En particulier à l'époque (!) où ils ont été rendus publics.

[131] Voir http://www.google.com/webhp?complete=1&hl=en

[132] Voir http://maps.google.com (pour les imitations, qui sont heureusement des imitations de qualité, voir http://www.virtualearth.com/).

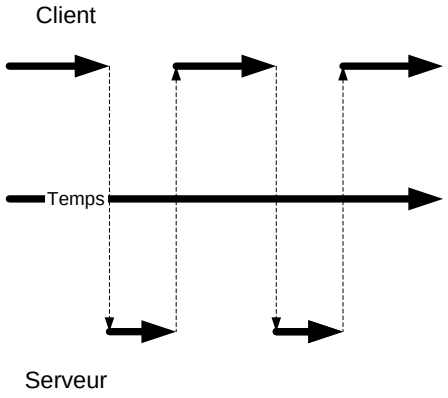
En toute honnêteté, on pourrait dire que AJAX est venu combler l'une des niches initialement visées (et manquée) par la technologie Java, soit celle de la dynamisation de l'expérience client sur le Web. La vision était là, mais la solution universelle a pris un peu plus de dix ans avant de se répandre. L'ubiquité des liens à haute vitesse, honnêtement, n'a pas nui.

Le schéma ci-dessus indique *Moteur AJAX*. Encore une fois, notons qu'il s'agit de la vision de l'auteur de l'article séminal derrière la formalisation de l'approche AJAX. En réalité, le *Moteur AJAX* est une simple technique de programmation.

Une vision alternative mais complémentaire du rôle de l'approche AJAX tient à la construction graduelle des interfaces présentées aux usagers.

Une construction graduelle est plus facile à réaliser s'il est possible d'éviter des ralentissements récurrents résultant d'un client en attente du serveur (comportement typique des systèmes synchrones au sens faible comme les systèmes RPC simples).

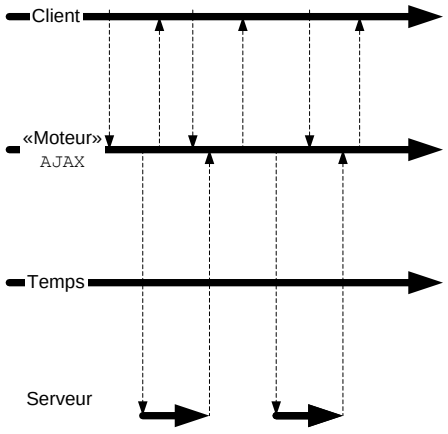
Cette stratégie permet, lorsqu'elle est bien appliquée, de réduire les délais d'affichage des éléments essentiels de pages riches et complexes. L'expérience de l'usager commence alors plus rapidement et paraît plus fluide.



On remarquera qu'une approche AJAX tend à mieux tirer profit de plusieurs des techniques que nous avons explorées dans ce cours :

- le caractère asynchrone des transactions permet une forme d'injection d'intelligence protocolaire dans la portion client du système;
- la gestion du volet IPM devient proactive;
- la capacité de traitement des deux homologues est mieux exploitée et la charge de traitement peut être mieux répartie;
- le découplage entre affichage et traitement est nettement plus clair que dans un modèle Web traditionnel; *etc*.

Tous ces avantages sont potentiels, bien entendu, et pas nécessairement effectifs.



On voit ici une application très claire du vieil adage informatique selon lequel *la plupart des problèmes se règlent en ajoutant un niveau d'indirection* : comment dynamiser une page Web? En ajoutant une couche entre la présentation et l'accès au serveur et en dissociant les données reçues par l'une des données émises par l'autre.

Le pour et le contre de l'approche AJAX

L'approche AJAX, avec son objectif avoué d'insuffler de la vie dans le modèle Web, est présentement l'un des principaux champs commerciaux effervescents dans le monde de l'informatique. Elle a ses détracteurs et ses *aficionados*, mais même les plus ardents critiques de cette approche lui reconnaissent des mérites lorsque bien appliquée.

La question maintenant ne semble donc plus être de savoir si l'approche AJAX a sa place sur le Web mais bien de savoir quels seront les outils qui permettront le mieux d'exploiter cette démarche à bon escient. *Microsoft* a offert en 2006 la technologie *Atlas*[133] au grand public, mais ce n'est que la pointe de l'iceberg, un indicateur que la pression du marché est forte.

L'approche AJAX propose une manière novatrice (pour le Web, du moins) d'interagir, une métaphore à part entière si on la compare à la métaphore traditionnelle où la navigation suit un modèle proposé par le fureteur (avancer, reculer, placer un signet, *etc*.) et où le serveur Web est responsable de l'offre de contenu.

Avec une approche AJAX, le client est en partie responsable de la métaphore d'utilisation alors que la métaphore proposée par le fureteur perd un peu son sens. En effet, *l'expérience AJAX se passe essentiellement à même la page Web*, et recharger la page la ramène à son état d'origine.

Revenir vers la page précédente implique, si cette page avait été modifiée par une interaction avec l'usager, un retour à sa version d'origine plutôt qu'à sa version à jour.

Une page développée selon une approche AJAX est plus proche, sur le plan conceptuel, d'une application que d'une unité de navigation.

La question qui se pose alors est : comment présenter une application englobée (la page utilisant une stratégie AJAX) dans une application *englobante* (le fureteur) sans briser la métaphore proposée par l'application *englobante*?

Les IPM Web appliquant une stratégie AJAX répondent plus localement et plus rapidement qu'une page Web traditionnelle. L'expérience associée à une telle IPM gagne considérablement du point de vue de la fluidité, dans la mesure où le lien réseau du client est de bonne qualité.

Construire une expérience asynchrone de qualité n'est pas une chose aussi simple qu'il n'y paraît; c'est une tâche habituellement dévolue à des programmeurs d'expérience, or plusieurs[134] développeurs Web (du moins ceux qui ont fait un bout de chemin sur le Web traditionnel) sont peu expérimentés ou peu versés côté programmation.

La gestion des délais sur un lien réseau est une chose complexe, et une interface appliquant une stratégie AJAX tout en gérant mal cet aspect systémique peut sembler déficiente ou inopérante à l'œil de l'utilisateur.

[133] Maintenant connue sous le nom de ASP.NET AJAX.

[134] Pas tous, évidemment: évitons les généralisations indues! Mais l'époque où on pouvait s'improviser développeur Web est probablement révolue. Les modèles de programmation sur ce média deviennent aussi riches et aussi complexes que les autres. Fait croire qu'il n'y a pas de réel substitut pour la formation.

Revenant à la question de la métaphore de navigation, exposée plus haut, rappelons que l'usager perd, avec l'approche `AJAX`, la possibilité d'exploiter les métaphores devenues universelles des signets et des contrôles pour avancer et reculer dans la liste de pages récemment visitées. La raison de ce problème est que l'approche `AJAX` ne génère pas, à proprement dit, une nouvelle page avec une `URL` distincte lors de chaque mise à jour importante. L'approche `AJAX` est plutôt de modifier une page existante.

Avec une approche `AJAX`, il est préférable de réfléchir l'expérience de l'usager en terme applicatif plutôt qu'en terme Web. **L'approche `AJAX` ne convient pas à toutes les interfaces**.

D'un point de vue plus technique, notez que le code JavaScript d'une page Web est interprété, pas nécessairement à la vitesse de l'éclair, par le fureteur, ce dernier jouant alors le rôle d'une machine virtuelle. Surcharger imprudemment une page Web de code JavaScript entraîne un risque : celui de dégrader l'expérience client plutôt que de l'améliorer. Il faut donc s'assurer de tester une page appliquant une stratégie `AJAX` dans un environnement proche de l'environnement cible chez la clientèle avant de la livrer au grand public.

Un autre détail à noter : un peu comme les pages à fort contenu multimédia (les pages contenant des modules *Flash* par exemple), l'approche `AJAX` ne donne pas de très bonnes pages d'accueil pour fins d'indexage par les moteurs de recherche, faute d'offrir une adresse distincte par élément pertinent de contenu.

Enfin, d'un point de vue sécurité, notez que le dynamisme de l'approche `AJAX` en fait aussi un excellent candidat pour construire de tous nouveaux stratagèmes de hameçonnage (en anglais : *Phishing*[135]), au sens où elle rend beaucoup plus simple l'acte de simuler, par une page frauduleuse, l'interface d'un autre site et de capturer en arrière-plan des mots de passe[136].

De même, par défaut, les requêtes `XMLHttpRequest` sont transférées en clair. Prudence avec l'information confidentielle! Notez aussi que, sur un réseau local du moins, un objet `XMLHttpRequest` a de la difficulté à gérer les requêtes faites sur une `URL` ne donnant pas dans le même domaine que celui du poste client.

Sous Firefox, un script JavaScript dans page Web locale ne peut ouvrir une `URL` à travers un `XMLHttpRequest` sans qu'un usager ne l'ait explicitement permis.

[135] Le *Phishing* est une technique leurrant un usager de manière à ce qu'il ait un comportement qui le désavantage (p. ex. : un courriel ou une page Web frauduleuse demandant de cliquer sur un lien et d'entrer un `NIP`).

[136] Voir par exemple `http://ajaxian.com/archives/2005/12`

Exemple concret interrogeant un script PHP

Imaginons une petite page Web tout à fait standard qui souhaite permettre d'interroger un script côté serveur capable de donner des infos sur un sujet choisi par l'usager.

```
<!DOCTYPE html PUBLIC "-//W3C//D/D XHTML 1.0  Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
      Petit test avec strat&eacute;gie Ajax
    </title>
```

Pour que cet exemple reste simple, le script JavaScript s'insérera dans l'en-tête, à l'intérieur de la section `<head>` de la page Web.

De manière simple, une variable globale générique (ce que permet Javascript) nommée `rq` sera initialisée, dans du code global, avec un objet permettant une requête asynchrone, donc une instance de `XMLHttpRequest`).

Il y a plusieurs cas possibles à couvrir à cause des incompatibilités entre fureteurs. C'est le cauchemar des développeurs Web.

```
<script language="javascript" type="text/javascript">
var rq = false;
try {
        rq = new XMLHttpRequest();
} catch (trymicrosoft) {
  try {
    rq = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (othermicrosoft) {
    try {
      rq = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (failed) {
      rq = false;
    }
  }
}
```

⇒ Notez que le code JavaScript global, donc ce code d'instanciation, sera exécuté à chaque fois que la page sera chargée.

La fonction JavaScript utilitaire `getInfos()` génère une requête de type `GET` pour interroger un programme sur une `URL` précise. Remarquez le recours à la fonction `escape()`[137] pour encoder le paramètre à la requête. Le 3ième paramètre de la fonction `open()` de `rq` vaut `true` pour que la requête soit asynchrone. Le `send(null)` signifie *envoi sans paramètres additionnels*.

De son côté, La fonction `miseAJourPage()` sera rappelée à chaque fois que l'état de complétion de la requête représentée par `rq` changera[138].

```
function getInfos() {
  var sujet = document.getElementById("sujet").value;
  var url = "http://pollux.pav.clg.qc.ca/bouffe2.php?aliment="
            + escape(sujet);
  rq.open("GET", url, true);
  rq.onreadystatechange = miseAJourPage;
  rq.send(null);
}
```

[137] Cette fonction transforme une chaîne de caractères pour qu'elle soit lisible sur tous les ordinateurs. L'opération inverse se nomme `unescape()`. Voir `http://www.w3schools.com/jsref/jsref_escape.asp`

[138] La progression de la valeur de cet état nous permettra, au besoin, de détecter les erreurs et, le cas échéant, de les analyser convenablement.

Sur réception du résultat de la requête (lors d'un rappel), l'état de la requête sera analysé et, si aucune erreur ne s'est produite, la représentation `DOM` de la page Web affichée dans le fureteur sera partiellement mise à jour.

Lors d'un rappel, le `readyState` de la requête passera graduellement de `0` (pas encore initialisé) à `4` (traitement complet). La fonction de rappel sera donc rappelée plusieurs fois, mais les données ne sont (dans notre cas) utiles que lorsque cet état indiquera la complétion du traitement.

Le `status` représente un code `HTTP` de succès (`200`) ou d'erreur (incluant le classique `404` pour les pages introuvables.

```
function miseAJourPage() {
  if (rq.readyState == 4) {
    if (rq.status == 200 || rq.status == 0) {
      var rep = rq.responseText;
      document.getElementById("resultats").value = rep;
    } else if (rq.status == 404) {
      alert("Erreur sur l'URL demandee" + rq.url);
    } else {
      alert("Erreur: code " + rq.status);
    }
  }
}
    </script>
  </head>
```

Le volet présentation de la page propose un formulaire à l'usager. Le déclenchement de la requête asynchrone sera provoqué par une modification de la zone de texte `"sujet"` suivie d'une perte de focus de sa part.

L'idée ici est que la mise à jour de cette `IPM` telle que proposée dans le fureteur ne sera pas une nouvelle page mais bien la même page avec une légère modification de contenu.

```
    <body>
      <h1>Petit test avec strat&eacute;gie Ajax</h1>
      <p>Petit formulaire full cool</p>
      <form action="POST">
        <p>
          Votre sujet?
          <input type="text" size="14"
                 name="sujet" id="sujet"
                 onChange="getInfos();" />
        </p>
        <p>Résultats de la recherche: </p>
        <p>
          <textarea name="resultats" rows="6" cols="50"
                    id="resultats"></textarea>
        </p>
      </form>
    </body>
</html>
```

Exemple concret utilisant une simple requête HTTP

Un exemple encore plus simple serait celui d'une page Web capable d'aller chercher le code source d'une autre page Web et de déposer le texte ainsi obtenu dans une zone de texte pour qu'on puisse le lire ou le copier pour s'en inspirer.

Le code à droite est un script JavaScript nommé `testAjax.js` qui permet de faire précisément une telle chose.

La fonction `chercherContenu()` sera invoquée par la page Web suite à la pression d'un bouton. Elle demandera au fureteur (voir le code en gras, qui fonctionne dans Firefox; je n'ai pas cherché l'équivalent pour Internet Explorer) la permission de démarrer une requête `HTTP` asynchrone de type `GET` et invoquera `miseAJourPage()` lorsque l'état de la requête changera.

L'`URL` dont le script obtiendra le contenu proviendra d'un contrôle portant l'identifiant `petite_url` dans la page Web alors que le contrôle qui recevra le texte du document obtenu sera celui affublé de l'identifiant `aRemplir` sur le même page.

```
var rq = false;
try {
  rq = new XMLHttpRequest();
} catch (trymicrosoft) {
  try {
    rq = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (othermicrosoft) {
    try {
      rq = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (failed) {
      rq = false;
    }
  }
}
if (!rq) {
  alert ("incapable d'appliquer une approche AJAX");
}
function miseAJourPage() {
  if (rq.readyState == 4) {
    if (rq.status == 200 || rq.status == 0) {
      var rep = rq.responseText;
      document.getElementById("aRemplir").value = rep;
    }  else if (rq.status == 404) {
      alert("Erreur sur l'URL demandee" + rq.url);
    } else {
      alert("Erreur: code " + rq.status);
    }
  }
}
function chercherContenu () {
  try {
    netscape.security.PrivilegeManager.enablePrivilege
      ("UniversalBrowserRead");
  } catch (e) {
    alert("Permission refusée.");
  }
  var url = document.getElementById("petite_url").value;
  rq.open("GET", url, true);
  rq.onreadystatechange = miseAJourPage;
  rq.send(null);
}
```

Pour ce qui est de la page Web elle-même, pas de grande surprise.

```
<!DOCTYPE html PUBLIC "-//W3C//D/D XHTML 1.0  Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Petite d&eacute;mo avec AJAX</title>
    <script type="text/javascript" src="testAjax.js" />
  </head>
  <body>
    <h1>Petite d&eacute;mo avec AJAX</h1>
    <p>
      La section qui suit devrait changer de contenu...
    </p>
    <button onclick="chercherContenu();">
      Changer le contenu
    </button>
    <table align="center">
      <tr>
        <td>URL:</td>
        <td><input id="petite_url" size="40"
                   value="Entrez une URL ici" /></td>
      </tr>
      <tr>
        <td colspan="2">
          <textarea id="aRemplir" rows="10" cols="40">
            Ce texte va changer...
          </textarea>
        </td>
      </tr>
    </table>
  </body>
</html>
```

Le code pertinent y est proposé en caractères gras. On note à la fois le lien vers le code JavaScript en tant que tel et les contrôles sur la page Web permettant d'interagir avec le code JavaScript en question.

Cette page utilise une balise `<textarea>` pour recevoir le texte de l'`URL` demandée puisque ce texte peut être arbitrairement long.

En retour, l'`URL` elle-même est décrite par une simple balise `<input>` de type `"text"` du fait qu'elle tiendra nécessairement sur une seule ligne.

Développer selon une approche AJAX : quelques pistes[139]

Puisque l'approche AJAX est une métaphore, et une métaphore à la mode par dessus le marché, il existe plusieurs outils qui cherchent à faciliter la vie des développeurs Web intéressés à un tirer profit sans pour autant se rendre misérables.

Voici quelques exemples de tels produits. La liste n'est pas exhaustive.

Le développement AJAX avec ASP.NET, autrefois connu sous le nom *Atlas*, est décrit ici :

```
http://ajax.asp.net/Default.aspx?tabid=47
```

Le *Google Web Toolkit* est décrit sur :

```
http://code.google.com/webtoolkit/
http://www-128.ibm.com/developerworks/library/j-ajax4/index.html?ca=dgr-lnxw01GWT4Ajax
```

Une infrastructure nommée *AvancedAjax* 1.1 est décrite ici :

```
http://advajax.anakin.us/index-en.htm
```

Le très sympathique projet à code ouvert COWSAjax est décrit ici :

```
http://cows-ajax.sourceforge.net/
```

Autre produit à code ouvert, le *AJAX Framework* est décrit ici :

```
http://glm-ajax.sourceforge.net/
```

Un autre projet sympathique, FlapJax, sous licence BSD, est décrit ici :

```
http://www.flapjax-lang.org/
```

Un produit se nommant SAJAX, le *Simple AJAX Toolkit*, est décrit ici :

```
http://www.modernmethod.com/sajax/
```

Un produit visant à simplifier la mise en place d'une approche AJAX avec Java est :

```
http://sourceforge.net/projects/ajax-simple
```

Pédagogie à propos de l'approche AJAX

Quelques sites visent à offrir de l'information (et de la publicité) gravitant autour de l'approche AJAX. Parmi eux, on trouve (liste non exhaustive encore une fois) :

- http://www.ajaxmatters.com/
- http://www.openajax.org/
- http://groups.google.com/group/ajax-web-technology

[139] J'en ai des centaines d'autres; des liens seront incessamment ajoutés au site Web du cours.

Quelques exemples de l'approche AJAX appliquée concrètement

L'approche AJAX cache-t-elle de réels produits hormis quelques canons plus connus comme **Google Maps** (http://maps.google.com), **GMail** (http://gmail.com) et ***Microsoft* Live** (http://www.live.com/)?

Effectivement, les produits Web appliquant une approche AJAX sont de plus en plus nombreux. Je vous suggère d'examiner entre autres d'examiner les suivants, classés par catégorie. Remarquez la prépondérance de produits de type *Office*…

Voir http://h-deb.clg.qc.ca/Liens/Applications-Internet-Liens.html#ajax pour une liste qui ne sera jamais à jour mais risque d'être plus riche en contenu que tout document papier pourrait l'être. Au moment d'écrire ces lignes, j'ai sous forme de signets au moins cinq fois le nombre d'applications listées ci-dessous.

Chiffriers électroniques

Nom	URL
NumSum	http://numsum.com/

Courriel et messagerie électronique

Nom	URL
RoundCube	http://www.roundcube.net/
YahooMail	http://mail.yahoo.com/
Meebo	http://www2.meebo.com/

Diaporamas électroniques

Nom	URL
S5	http://www.meyerweb.com/eric/tools/s5/

Éditeurs de documents

Nom	URL
AjaxWrite	http://www.ajaxwrite.com/
Writely[140]	http://www2.writely.com/info/WritelyOverflowWelcome.htm

[140] Ce produit appartient maintenant à Google.

Organisation du travail

Nom	URL
WebNote[141]	http://www.aypwip.org/webnote/
Kiko[142]	http://www.kiko.com/
Basecamp	http://basecamphq.com/

Outils de développement

Nom	URL
FckEditor	http://www.fckeditor.net/
Atlas	http://atlas.asp.net/
OpenAjax	http://www.openajax.ca/
Yahoo Design Pattern Library	http://developer.yahoo.com/ypatterns/
Yahoo User Interface Library	http://developer.yahoo.com/yui/

Suite Office

Nom	URL
gOffice	http://www.goffice.com/
ThinkFree	http://online.thinkfree.com/
Zimbra	http://www.zimbra.com/index.html

Systèmes de fichiers

Nom	URL
openomy	http://www.openomy.com/

Divers

Nom	URL
WuFoo[143]	http://wufoo.com/
Dictionary.hm[144]	http://www.dictionary.hm/
gliffy[145]	http://www.gliffy.com/

[141] Un système de notes *Post-It* virtuel.

[142] Un calendrier Web.

[143] Un générateur de formulaires Web.

[144] Un dictionnaire en ligne.

[145] Un éditeur de graphes et de diagrammes.

La mouvance Web 2.0

Le cours Applications Internet est rempli de *buzzwords* et de thématiques à la mode. En plus des termes AJAX, XML, services Web, SOA et autres, on trouve le très présent et très critiqué vocable de Web 2.0.

Ce qu'on nomme Web 2.0[146]

Le terme Web 2.0 est un sujet controversé[147]. À la fois *buzzword* un peu vide pour les uns et un symbole d'espoir pour les autres, on compte sur Internet (à l'œil) énormément d'articles constructifs et plusieurs articles violemment critiques à son sujet.

L'un des membres fondateurs du mouvement Web 2.0, Tim O'Reilly (de *O'Reilly Publishing*) a écrit[148] que, lorsque le terme Web 2.0 a été lancé pour la première fois dans une rencontre entre *aficionados* du Web, la définition que ses auteurs ont cherché à en donner était une définition par exemples[149].

J'ai reproduit ces exemples à droite pour vous éviter la recherche, mais ils sont tous tirés intégralement du site original.

Web 1.0	**Web 2.0**
DoubleClick	Google AdSense
Ofoto	Flickr
Akamai	BitTorrent
mp3.com	Napster
Britannica Online	Wikipedia
personal websites	blogging
evite	upcoming.org and EVDB
domain name speculation	search engine optimization
page views	cost per click
screen scraping	Web services
publishing	participation
content management systems	wikis
directories (taxonomy)	tagging ("folksonomy")[150]
stickiness	syndication

La liste a sûrement beaucoup évoluée depuis lors, du fait que le Web 2.0 attire investisseurs et développeurs à un rythme fou. Plusieurs y voient une espèce d'eldorado technologique du début du XXI[e] siècle.

---

[146] Voir entre autres un (toujours utile) site Wiki: http://en.wikipedia.org/wiki/Web_2.0, qui prend la peine de mentionner que la définition de Web 2.0 est encore aujourd'hui fluctuante et tend à incorporer chaque nouveau concept relié au Web. À la décharge des *aficionados* de ce terme, par contre, une définition mouvante et en constante mutation du terme Web 2.0 sied bien au concept même de Web 2.0.

[147] Voir http://web2.wsj2.com/review_of_the_years_best_web_20_explanations.htm par exemple.

[148] Voir http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

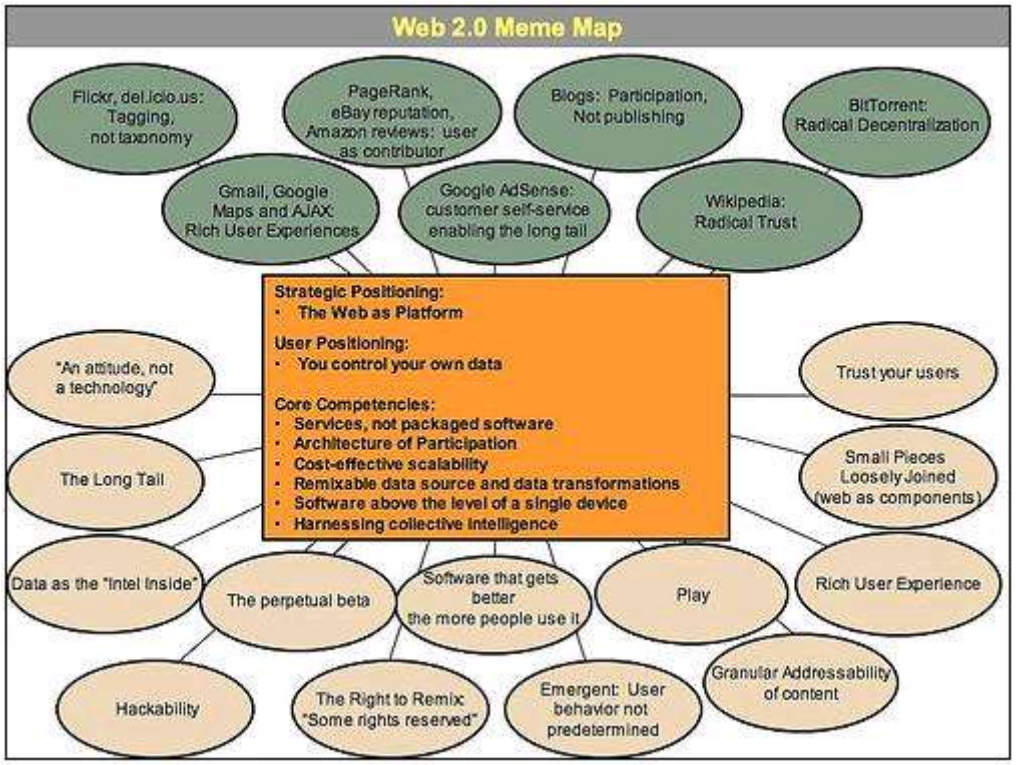
[149] Ceci explique en partie l'une des plus grandes critiques du Web 2.0 : le concept est commercial, vendeur, s'illustre bien mais n'est pas clairement défini et résiste même aux tentatives de le définir.

[150] L'idée derrière le terme *Folksonomy* est de construire des dénominations sur une base sociale. Les multitudes peuvent qualifier et nommer les objets de manière publiquement accessible est construisent collectivement un vocabulaire qui leur sied. Voir http://en.wikipedia.org/wiki/Folksonomy

Au cœur de Web 2.0, on trouve un noyau philosophique :

- **concevoir le Web comme une plateforme**, ce terme étant pris selon une acception large incluant celle de *lieu de l'expérience humaine*;
- **accroître le contrôle qu'a l'usager sur son expérience** (ce qui explique en partie qu'on associe l'approche AJAX à l'idée de Web 2.0);
- **orienter le développement vers les services** plutôt que vers le logiciel vendu sur support physique (le Web 2.0 est un Web à haute vitesse);
- **mettre l'accent sur la participation des multitudes** (pensez Wiki[151], *Collaboraware*, développement à code ouvert…); pour
- **profiter de l'intelligence collective**.

Une cartographie de ce que comporte l'idée de Web 2.0 (une carte des *memes*) a été proposée par Tim O'Reilly[152] et reprise par plusieurs autres depuis. Je me permets d'en insérer moi aussi une copie ici pour éclaircir le propos; après tout, on dit bien que la collaboration est une pierre d'assise du Web 2.0, non?



[151] Voir http://en.wikipedia.org/wiki/Wiki

[152] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

Vous remarquerez sur cette cartographie plusieurs concepts choquants d'un point de vue traditionnel. Par exemple :

- l'idée de perpétuel bêta. Le logiciel qui évolue toujours et qui n'est jamais terminé. Évidemment, les individus pragmatiques comprendront que cela ne veut pas dire un logiciel mal construit ou plein de bugs (personne n'en voudrait) mais bien un logiciel mouvant, en apparence comme en fonctionnalité;
- l'idée de *Hackability*, qui ne veut pas tant signifier *attaquabilité* mais bien « *gossabilité* », si vous me permettez l'écart de langage, au sens intuitif et très québécois du terme qui voudrait dire assujetti aux manipulations des tiers;
- l'idée de reposant sur les données (*Data as the "Intel Inside"*), approche qui amène beaucoup de flexibilité en pratique mais longtemps décriée dans les universités. Le succès de XML n'est pas étranger à cette tangente;
- le logiciel qui s'améliore au fur et à mesure qu'il est utilisé, les comportements émergents, deux aspects qui peuvent à la fois résulter de l'implication d'agents électroniques autonomes qui apprennent les comportements des usagers comme de l'implication de la communauté qui entraîne un enrichissement continu du contenu.

L'usager comme agent de sa propre expérience, bien sûr, mais aussi comme agent de la construction collective de l'expérience de tous.

L'effet Wiki est très présent derrière la philosophie Web 2.0 : mettre sur pied un système par lequel un service d'accès à l'information est offert et à travers lequel les utilisateurs peuvent accroître l'offre du service à travers l'utilisation qu'ils en font.

Du concret

Combien y a-t-il d'applications qui méritent d'être qualifiées comme faisant partie de la mouvance Web 2.0? La seule manière philosophiquement légitime de les répertorier est de procéder par annotation collaborative.

Voir http://www.web2list.com/

Simplicité et réutilisation

Pour que le Web 2.0 se réalise, plusieurs facteurs techniques doivent être rassemblés :

On remarquera la jonction directe du Web 2.0 avec la philosophie SOA d'exposition vers l'extérieur de l'entreprise des services qui y existent déjà et l'approche par services Web sans états.

- les interfaces d'offre de services doivent être à la fois simples et fortement découplées, de manière à maximiser la réutilisation de services existants et la combinaison de services simples du côté client;
- une approche CS du début à la fin, mais surtout avec des composants sans états;
- un accent mis sur le code contrôlé par les données (*Data Driven*), ce qui implique à la fois l'acceptation que les données soient *ailleurs* (et entraîne fondamentalement les questions *où sont-elles?* et *qui les possède?*) et y met en relief l'importance des BD;
- une expérience accrue pour l'usager, car le concept de Web comme plateforme ne fonctionnera réellement que si l'usager l'accepte, ce qui ne se produira que si l'expérience humaine y est signifiante.

Dans ce que les spécialistes qualifient de produits Web 2.0, on trouver un point commun, un aspect libérateur, délocalisant : si le Web est une plateforme en soi, la mouvance Web 2.0 libère l'usager de son ordinateur en tant que lieu.

Ses signets peuvent être en ligne, partagés. Son environnement de travail le suit partout. Le lieu physique perd son intérêt. Le monde entier devient un prétexte au télétravail.

Des traits communs

Certains s'interrogent à voir ce qu'ont en commun (en surface) les premières applications sur la plateforme Web 2.0. Une réflexion à ce sujet peut être lue sur http://mittermayr.wordpress.com/2006/02/03/20-culture/

Définir le terme Web 2.0 est matière à débats, mais il demeure un consensus sur certaines caractéristiques de cette mouvance :

- les usagers sont impliqués (on pense aux Wiki, évidemment, mais aussi à de grands succès comme le répertoire digg.com);
- les usagers se révèlent (on pense à del.icio.us et à ses signets publics, le phénomène du *Social Bookmarking*, qui favorisent la rencontre entre gens partageant un ensemble d'intérêts communs);
- les usagers s'y sentent bien. On favorise une expérience client riche, souvent (mais pas nécessairement) mise en place par une approche AJAX.

On pourrait sûrement faire des parallèles entre cette approche des humains envers la technologie et le recours à Internet pour organiser des événements subits (les célèbres *Flash Mobs*), ou avec la vogue des blogues. Le Web 2.0, c'est un peu la démocratisation de la technologie, l'humain qui s'approprie ce média qu'est Internet. C'est un concept éminemment social.

Grandeurs et misère du communautarisme

Pour un individu, le Web 2.0 implique un virage philosophique. Sur un Wiki, par exemple, la multitude construit et tient à jour un dépôt de savoir. Plusieurs approches sont possibles :

- anonymat absolu des auteurs. Aucune responsabilité individuelle, savoir pleinement collectif. Une approche radicalement communiste (ou communautariste), aimée des puristes du modèle collaboratif. Les risques : il devient facile de polluer l'espace collaboratif et de vandaliser une page de l'encyclopédie, et il devient ardu de valider les sources (et d'éviter le plagiat). La solution : régulation par les pairs. Le problème : les conflits d'experts, qui entraînent des changements fréquents et violents du contenu accessible. Notez l'aspect philosophique *Trust the Users* du Web 2.0;
- enregistrement des auteurs. Responsabilité individuelle quant au contenu, potentiel de médiation entre experts, identification des auteurs. Une approche plus restrictive, donc;
- restriction des droits d'écriture aux experts seuls. Approche contraire à l'esprit du Web 2.0. Mène à un modèle plus traditionnel de rédaction.

L'approche collaborative permet la mise en place plus rapide d'une banque de savoir par des bénévoles et des individus intéressés. La question du contrôle de la qualité demeure ouverte.

En pratique, doit-on faire confiance aux Wikis? Les études[153] tendent à montrer que le nombre d'erreurs factuelles dans une page moyenne de Wikipédia tend à ressembler au nombre d'erreurs factuelles sur une page d'encyclopédie traditionnelle (l'Encyclopaedia Britannica sert souvent de comparatif). Comme pour tout média, il est préférable de puiser son savoir à plusieurs sources pour réduire, statistiquement, le péril d'une information incorrecte.

Considérations éthiques et légales

Qui possède les données entreposées chez un tiers? Quels droits confère-t-on à un tiers qui entrepose nos données? Nos droits sont-ils affectés par le lieu physique d'entreposage (le pays) ou par les lois en vigueur du pays où se situe le siège social de la firme qui entrepose nos données? Les règles peuvent-elles changer à notre insu si la firme change de statut?

Ce ne sont là que quelques questions fondamentales liées au Web collaboratif et aux services en ligne. La confiance est la clé d'un Web collaboratif.

[153] J'ai plusieurs liens mais pas le temps de les citer ici. Google sera votre ami dans ce cas bien précis.

*Un monde en qu*ête de sens : le Web sémantique

Le terme Web 2.0 porte aussi un autre sens : celui d'un combat pour accroître le sens sur Internet et pour transformer ce réseau de contenu un tant soit peu désorganisé en une entité redéfinie sous le nom de **Web sémantique**[154].

Le Web sémantique est d'abord une philosophie, ensuite une gamme d'outils et d'applications. On décline le Web sémantique sous plusieurs angles :

- la formalisation du **Web comme véhicule de contenu**, découplant de manière stricte présentation (CSS, HTML) et présenté (servlets, ASP, JSP). En soi, ceci est un *a priori* du Web sémantique, pas une de ses conséquences;
- une mouvance privilégiant les balises porteuses de sens. Par exemple: ... pour mettre l'accent sur plutôt que ... pour mettre un passage en caractères italiques. Le même effet visuel est obtenu[155] dans chaque cas mais la première balises est porteuse de sens, sémantiquement riche, alors que la seconde est un élément de formatage;
- l'insertion de **balises ontologiques** pour ajouter au contenu sur le Web, déjà lisible et décodable sur les plans lexical et syntaxiques, une strate sémantique.

En prenant le virage sémantique, le Web devient non seulement une plateforme mais aussi une immense source de données d'une richesse presque infinie, du moins du point de vue de ces pauvres mortel(le)s que nous sommes.

[154] Voir http://en.wikipedia.org/wiki/Semantic_Web

[155] Par défaut, du moins.

Générer du sens

Au moment d'écrire ces lignes, le Web sémantique est principalement construit par des humains. L'insertion de balises ontologiques est une tâche humaine, ce qui implique un effort, une expertise et un biais humain sur le sens attribué à un énoncé ou à un objet Web quelconque.

Philosophiquement parlant, l'insertion manuelle d'une strate ontologique sur le Web est un palliatif *imparfait* mais *nécessaire*.

Imparfait parce que porteur du biais de l'individu ayant annoté l'information, mais nécessaire parce qu'il ramène le problème de *comprendre automatiquement* le Web à une dimension pour laquelle on pourra éventuellement envisager une solution.

Le biais est-il un mal?

Il se peut qu'insérer un biais humain à chaque élément de la strate ontologique du Web soit, au fond, une bonne chose. Donner à l'information le sens souhaité est un choix (et un droit) d'auteur, après tout.

Cela dit, rien n'empêche la production de logiciel chargé d'insérer une ontologie à même un flux de données sur le Web. Qu'un humain puisse le faire ne signifie pas qu'un logiciel en soit incapable…

Le Web sémantique est teinté de ses auteurs. Intégré conceptuellement à la mouvance Web 2.0, le Web sémantique fait confiance aux usagers. Toute prétention d'objectivité de l'information est discréditée *a priori*. Le seul véritable accès à une objectivité apparente est de multiplier les sources d'information, de générer une objectivité statistique.

Chercher du sens

Le problème d'extraire du sens d'un flux de données est un problème d'une complexité considérable, du moins si on le prend dans son acception la plus large. Considérant la masse d'information déjà disponible sur le Web, le problème est peut-être déjà insoluble.

L'accès à l'information sur le Web est déjà fortement teinté par la présence d'une multitude d'agents. Qui pourrait véritablement fonctionner sur Internet sans un moteur de recherche aujourd'hui? Une première sélection est faite dans l'information pour qui procède à travers Google ou Live.com pour obtenir des données selon ses critères. Une partie du filtrage sert à des fins commerciales (Google AdSense, par exemple, qui oriente les publicités proposées à l'usager en fonction de ses intérêts apparents), une autre en fonction de stratégies politiques (présenter un article Wikipédia avant d'autres sources).

Chercher sur le Web sémantique implique être en mesure de faire des inférences logiques sur le sens apparent et sur les relations entre les entités sémantiques qui y apparaissent. Voir le Web comme une immense `BD` et voir les balises ontologiques comme des métadonnées.

Remarquez le lien direct entre la mouvance sémantique et la description `WSDL` des services Web. Codifier le sens des interfaces et formaliser leur description est le premier pas en direction d'une opérabilité automatisée du Web et de son contenu, puisque cela ouvre la porte au développement raisonnablement efficient d'agents analytiques capables de reconnaître l'applicabilité commerciale d'un service et d'en négocier les termes.

Construire le Web sémantique : les technologies

Pour faciliter l'avènement d'un Web sémantique, il faut à la fois pouvoir *décrire le sens* des éléments d'un document et permettre l'intervention d'agents qui *extrairont du sens* de ces documents. Un formalisme est nécessaire.

Les technologies au cœur du Web sémantique sont (évidemment) la notation XML, mais aussi :

- un modèle de métadonnées (**RDF**[156], pour *Resource Description Framework*), par lequel il est possible de construire des graphes décrivant des relations entre concepts, et
- un langage pour décrire des ontologies (**OWL**[157], pour *Web Ontology Language*[158], ou **RDF Schema**[159]).

La mouvance vers les métadonnées est un mouvement énorme, considérable, qui touche toutes les sphères des sciences fondamentales comme appliquées, du monde académique à celui de l'industrie, du monde alternatif au monde commercial.

La codification sémantique du Web (et comment en tirer profit) est un sujet à suivre de près, d'un point de vue scientifique, culturel et commercial.

Une balise RDF est un triplet XML fait d'une **ressource** (souvent une URI), d'une **propriété** et d'une **valeur**. La valeur d'une balise RDF peut être un littéral ou une autre ressource.

Un exemple simple de balise RDF possible serait celle apparaissant à droite. En soi, le format se décrit presque de lui-même.

```
<rdf:Description
    rdf:about='http://h-deb.clg.qc.ca/UdeS/AIN/index.html'>
  <titre>Applications Internet</titre>
  <enseignant>Patrice Roy</enseignant>
</rdf:Description>
```

On utilise RDF pour annoter le contenu d'un document et lui attribuer des propriétés descriptives, porteuses de sens, qu'un programme pourra ensuite exploiter pour mieux en comprendre la teneur, le classer, le formater, l'intégrer dans des modèles plus complexes, *etc*. C'est par une **ontologie** qu'on construit des liens entre des documents *sémantiquement augmentés* par RDF.

Les spécifications ontologiques RDF Schema permettent de décrire des classes de manière hiérarchique et munies de propriétés. Les classes décrites par schémas RDF sont extensibles et ouvertes (plus comme les « classes » JavaScript que comme les classes C++ ou Java) et peuvent être décrites partiellement par plusieurs documents XML distincts.

Un exemple de schéma RDF décrivant des relations hiérarchiques simples entre des cours du CeFTI est proposé à droite. Rien de bien sophistiqué. Notez que les schémas RDF supportent sans peine l'héritage multiple.

```
<rdfs:Class rdf:ID='Cours' />
<rdfs:Class rdf:ID='INF777'>
  <rdfs:subClassOf rdf:resource='#Cours' />
</rdfs:Class>
<rdfs:Class rdf:ID='INF756'>
  <rdfs:subClassOf rdf:resource='#Cours' />
</rdfs:Class>
```

---

[156] Voir http://en.wikipedia.org/wiki/Resource_Description_Framework

[157] Voir http://en.wikipedia.org/wiki/Web_Ontology_Language

[158] Je sais que l'ordre des lettres ne concorde pas mais ce n'est pas ma faute!

[159] Voir http://www.w3.org/TR/rdf-schema/

Le langage de description d'ontologies OWL[160] va plus loin que RDF Schema et introduit des concepts ensemblistes tels que la cardinalité, l'énumération, la symétrie des relations et ainsi de suite. Dans un cas comme dans l'autre, l'objectif des langages de description d'ontologies est d'établir des relations entre les entités sémantiques.

Par rapport à une BD relationnelle :

- un nœud RDF correspond conceptuellement à un enregistrement dans une table;
- un type de propriété RDF s'apparente à un champ (une colonne) de la table; et
- la valeur d'un nœud RDF correspond à une cellule de la table.

On peut décrire des propriétés RDF comme dans l'exemple proposé ci-dessous.

```
<rdf:Property rdf:ID='titre'>
  <rdfs:domain  rdf:resource='#Cours'/>
  <rdfs:range rdf:resource= '&rdfs;Literal'/>
</rdf:Property>
<rdf:Property rdf:ID='enseignant'>
  <rdfs:domain rdf:resource='#Cours'/>
  <rdfs:range rdf:resource='#Personne'/>
</rdf:Property>
<rdf:Property rdf:ID='nom'>
  <rdfs:domain rdf:resource='#Personne'/>
  <rdfs:range rdf:resource= '&rdfs;Literal'/>
</rdf:Property>
```

Vous remarquerez les liens relatifs entre les concepts de cours, d'enseignant, et les exemples proposés un peu plus haut de schémas et de balises RDF.

Notez que ces exemples sont des adaptations d'exemples classiques; je dispose dans le moment de peu d'outils pour valider la conformité de ces descriptifs RDF avec l'état le plus récent du standard (du moins, au moment d'écrire ces lignes).

À partir de descriptifs RDF, on peut procéder à des inférences logiques semblables à celles qu'on pourrait générer sur un SGBD : combien existe-t-il d'enseignants au CeFTI (faudrait ajouter des balises en conséquence)? Quels sont les cours enseignés par votre chic professeur? Y a-t-il d'autres professeurs qui enseignent le même cours?

```
<rdf:Property rdf:ID='name'>
  <rdfs:domain rdf:resource='#Person'/>
  <rdfs:range  rdf:resource='&rdfs;Literal'/>
  <rdfs:label xml:lang='fr'>nom</rdfs:label>
  <rdfs:label xml:lang='en'>name</rdfs:label>
</rdf:Property>
```

Il existe des langages d'interrogation de documents sémantiquement annotés, certains (on pense à OQL) étant même syntaxiquement proches de SQL.

On peut construire des entités sémantiquement riches à partir de stratégies d composition ou d'agrégation, grouper les entités RDF par des espaces nommés, et même utiliser des synonymes pour un même terme (voir à droite, en caractères gras).

Des conteneurs RDF peuvent être définis pour grouper des entités (on pense à un <rdf:Bag> par exemple). Il est très clair que le double objectif de représenter le monde comme immense BD chargée de sens et de décrire les entités du monde sous forme OO est visé.

[160] Voir http://www.w3.org/TR/owl-features/

Éviter les exagérations

Le W3C a mis en ligne un article intéressant[161] décrivant ce que le Web sémantique n'est pas. On y remarque que :

- construire un réseau de données sémantiques de la taille du Web (au sens figuré) ne signifie pas construire une intelligence artificielle. Cela dit, une multitude d'agents œuvrant en parallèle sur une masse mouvante de données réparties évoque clairement l'image d'une intelligence émergente et on comprend les gens de faire des parallèles entre les deux;
- intégrer des balises RDF au document n'est pas la même chose que transformer le Web en un immense modèle entité/ relation, même s'il est possible de construire un modèle entité/ relation à partir de balises RDF. Rien n'empêche un moteur consommant un document contenant des modèles décrits à l'aide de balises RDF de présumer de l'existence de certaines données qui seraient absentes de ce document : de faire ce que les anglophones nommeraient des *Educated Guesses*, disons. Si on examine divers pays et s'il nous manque le PIB d'une instance particulière, il peut être possible de construire la donnée manquante à partir d'autres indicateurs ou de présumer une valeur par défaut;
- l'intégration de balises sémantiques (balises RDF) à des documents Web n'est pas la même chose que la fusion du monde entier sous un même modèle de données. L'approche par balises est plus découplée que cela.

Pour un article fouillé sur le Web sémantique, je vous propose le fort sympathique http://www.w3.org/DesignIssues/Semantic.html.

[161] Voir http://www.w3.org/DesignIssues/RDFnot.html

Critiques de la mouvance Web 2.0

Le Web 2.0 peut être perçu comme la rencontre de trois mouvances : le Web en tant que plateforme, le Web en tant que milieu de vie et le Web en tant que source de données. Ces tendances peuvent être complémentaires comme elles peuvent être antagonistes.

Pensons par exemple à une stratégie AJAX par laquelle des balises <script>...</script> sont insérées dans le corps des pages Web servant de descriptions d'applications.

Ces balises tendent à proliférer dans les applications Web dont le volet client se veut riche mais ne permettent pas d'extraire un sens précis ni des balises, ni de ce qu'elles délimitent (à moins d'essayer d'analyser le sens du programme décrit par le script).

Les tenants du mouvement sémantique nomment ce trait des pages appliquant une stratégie AJAX une *Tag Soup*, ou soupe à la balise, pour laisser transparaître l'image d'une masse informe de balises sémantiquement vides.

Une autre critique, qui perdure et risque de perdurer un certain temps encore, est qu'une définition en constante mouvance n'est pas vraiment une définition.

Au sujet de la confusion autour de ce que signifie vraiment Web 2.0, voir cet amusant mais sarcastique site qu'est http://www.cerado.com/web20quiz.htm

Certains auteurs[162] parlent de la mouvance Web 2.0 comme d'une mise à jour technologique, dont ils reconnaissent la valeur intrinsèque mais dont ils critiquent la qualification en tant que révolution conceptuelle. C'est une critique légitime : décrire une mouvance comme s'il s'agissait d'une révolution est une stratégie pour le moins optimiste. La théorie des révolutions scientifiques de Thomas Kuhn[163] nous laisse entendre que le mot révolution s'applique mieux *a posteriori* qu'*a priori*.

Une critique particulièrement sympathique de la mouvance Web 2.0 et de l'approche AJAX peut être lue sur le site http://www.alistapart.com/articles/web3point0 où l'auteur prend soin d'en faire ressortir le côté mode, le côté m'as-tu vu.

Toutes critiques confondues, la mouvance vers un Web en tant que plateforme est réelle, les applications embryonnaires existent, la possibilité de pondre des applications capables de capturer un créneau du marché a été démontrée concrètement, et l'insertion d'une strate ontologique croissante sur le Web est un bulldozer. On comprendra en ce sens l'émergence de cours sur les applications Web…

[162] Voir par exemple le texte de Paul Boutin sur http://www.slate.com/id/2138951/

[163] Voir http://fr.wikipedia.org/wiki/Thomas_Samuel_Kuhn

Gestion des états

L'adage veut que le Web soit sans états. Cette formule grossière signifie que, reposant en majeure partie sur des protocoles sans états tels que HTTP, les technologies Web font reposer le problème de tenir à jour au moins certains des états d'une transaction en cours sur les épaules du poste client de la transaction.

Cependant, pour des raisons de sécurité, les volets clients des applications Web (scripts JavaScript, pages HTML ou JSP, applets Java ou autres) ne sont habituellement pas autorisés à utiliser les ressources au sens large du poste client.

L'idée est qu'il serait risqué de permettre à du code pris sur un serveur distant et auquel on ne fait pas nécessairement confiance d'accéder au disque rigide (en lecture ou en écriture) ou de réaliser de l'arithmétique de pointeurs pour se promener allégrement dans la mémoire des divers processus en exécution sur l'ordinateur d'un pauvre humain qui se pensait en sécurité en examinant le cours de la bourse de son salon.

Ceci ne signifie pas que le client doit conserver tous les états sur le poste client. Le serveur peut presque tout conserver si le client est en mesure de s'identifier ou d'identifier sa transaction lors de chaque requête.

Pour cette raison, le problème des états peut des résumer à celui de connaître, côté client, un identifiant de session (souvent un vulgaire entier). Ceci permet à un serveur sans états muni d'une BD d'inférer, au besoin, le reste des données propres à la transaction représentée par cet identifiant.

Pour diverses raisons (incluant la confidentialité, la sécurité et l'*échelonnabilité*), il est fréquent que les postes clients conservent un peu plus qu'un numéro de session, mais on parle alors d'un choix politique, pas d'une contrainte technique.

Variables session

Pour une application Web, créer une BD sur le poste client est donc hors de question. Y entreposer des fichiers arbitraires l'est tout autant. Tenir à jour des variables en mémoire du serveur peut être une option[164] mais peut nuire à l'*échelonnabilité* du système en ajoutant un poids sur le serveur.

En gros, une variable session existera dans le serveur et aura une durée de vie délimitée dans le temps, chose nécessaire du fait que la navigation Web ne permet pas de savoir en tout temps quand un usager a cessé de transiger avec le serveur.

Les technologies comme ASP ou JSP permettent aussi de transiger avec un objet session[165], ce qui simplifie l'utilisation des variables session. Les données relatives à la session existent côté serveur mais sont accessibles par la page côté client et ont, de ce fait, la particularité de pouvoir être créées par une session de travail dans un fureteur puis de pouvoir être accédées par une session de travail dans un autre fureteur (si leur durée de vie n'est pas expirée, du moins).

[164] Voir http://www.asp-php.net/tutorial/asp-php/sessions.php pour un exemple avec ASP et avec PHP. Prudence car ce site se comporte mal et fait apparaître des popups désagréables.

[165] Voir http://www.w3schools.com/asp/asp_ref_session.asp pour des détails avec ASP et voir http://www.jsptut.com/Sessions.jsp pour des détails avec JSP.

Petit exemple JSP

Imaginons la page Web toute simple ci-dessous (nommons-la `legume.html`). Cette petite page demande à un usager d'entrer un nom de légume, puis soumet à une page nommée `Wow.jsp` le texte entré dans une variable nommée… `legume`.

La variable `legume` n'est pas une variable session mais bien un paramètre de la requête `POST` soumise pour obtenir `Wow.jsp`. Son contenu permet à `legume.html` de soumettre à `Wow.jsp` une donnée sur laquelle cette dernière pourra opérer.

En termes fonctionnels, `legume.html` est un peu l'appelant d'un sous-programme et `Wow.jsp` est un peu l'appelé.

```
<html>
  <body>
    <form method="post" action="Wow.jsp">
      Entrez un nom de l&eacute;gume
      <input type="text" name="legume" size=32>
      <p><input type="submit"></p>
    </form>
  </body>
</html>
```

La page `Wow.jsp`, dans cet exemple, saisira le paramètre `"legume"` et créera une variable session du même contenu en invoquant la méthode `setAttribute()` de l'objet `session`.

Puisqu'il faut lui donner un nom et qu'il s'agit du légume choisi par l'usager, j'ai choisi de nommer cette variable `"zeLegume"`.

```
<%
   String s=request.getParameter("legume");
   session.setAttribute("zeLegume", s);
%>
<html>
  <body>
    <a href="VoirZeLegume.jsp">
      Je r&eacute;fl&eacute;chis...
    </a>
  </body>
</html>
```

La page intermédiaire `Wow.jsp` aurait-elle pu être évitée en intégrant un peu de JavaScript à la page précédente?

Enfin, la page `VoirZeLegume.jsp` récupère la valeur de la variable session `zeLegume` et en affiche fièrement le contenu à l'usager sur le poste client, sûrement ébahi par tant de magie.

```
<html>
  <body>
    <p>
      Vous avez ent&eacute;
      <%=
        session.getAttribute("zeLegume")
      %>
    </p>
  </body>
</html>
```

Moyens maison[166]

Il est possible de faire transiter des données entre deux pages sans utiliser de code côté serveur en ayant recours à des scripts JavaScript et, au choix :

- en insérant des données dans l'`URL` invoquée. mais cette option est contraignante puisque, au-delà de `2083` caractères dans une `URL` (précédées par un `?`) le comportement ne peut plus être garanti. En pratique, cette technique semble utilisée si les données sont de petite taille et si leur visibilité dans l'`URL` ne pose pas de problème de sécurité;
- utiliser des champs cachés à même une page Web pour y insérer des valeurs, ce qui à dégager l'`URL` et est un peu plus discret (l'usager standard peut ne pas s'en apercevoir), sans que ce soit vraiment plus sécuritaire; ou, ce qui est franchement à déconseiller
- d'insérer les données dans l'attribut `name` de l'objet `window`. Même si cela fonctionne en pratique, *ne le faites pas* puisqu'il devient banal pour un tiers hostile d'accéder à vos données.

[166] Voir `http://www.boutell.com/newfaq/creating/scriptpass.html` pour quelques exemples concrets, code à l'appui.

*Témoins (*Cookies*)*[167]

Les variables session font porter le poids de la gestion des états sur les épaules du serveur, ce qui a des avantages et des inconvénients. Si le souhait est de faire porter une partie du poids sur les épaules du client, alors il faut avoir recours à une stratégie permettant :

- de conserver quelques données en mémoire sur le poste client, de manière à ce que ces données transcendent la page où elles ont été générées; ou
- d'entreposer de petits éléments de données sur le disque, de manière limitée et encadrée (pour éviter les débordements et les abus).

Pour le moment, l'option la plus crédible est d'entreposer des données sur disque. L'alternative représenterait une transformation trop importante du modèle de navigation établi.

⇒ Les fichiers contenant de petites quantités de données que les fureteurs permettent de générer sont ce qu'on nomme des ***Cookies*** (le mot français est **témoin**, ou **mouchard** pour qui préfère leur attribuer une connotation négative).

Le rôle d'un témoin est de conserver des données. Certains les utilisent pour conserver les préférences de navigation d'un individu (nombre d'options à afficher, couleurs, polices, adresse postale), d'autres pour connaître les habitudes d'un individu (pages visitées, par exemple, et dans quel ordre) ou pour diversifier la publicité proposée.

Lorsqu'un témoin est associé à une `URL`, il est envoyé en accompagnement à chaque requête soumise à cette `URL`. Ceci grossit légèrement chaque requête, mais les témoins sont habituellement très petits[168] alors leur impact sur le trafic reste mineur.

Sur le plan historique, les témoins ont été développés chez Netscape pour résoudre des problèmes de gestion des états dans les transactions Web. La définition préliminaire originale peut être consultée sur `http://wp.netscape.com/newsref/std/cookie_spec.html`

[167] Le `RFC` des témoins est disponible sur `http://www.ietf.org/rfc/rfc2109.txt`. Le site `http://www.commentcamarche.net/securite/cookies.php3` propose une discussion assez claire sur les témoins et leur fonctionnement. Le site `http://en.wikipedia.org/wiki/HTTP_cookie` est aussi une source détaillée d'information sur le sujet. Le site `http://www.cookiecentral.com/faq/` est aussi pertinent et plutôt complet.

[168] La taille maximale d'un témoin et de `4 Ko`. On s'attend d'un fureteur à ce qu'il puisse entrepose un minimum de `20` témoins par site et de `300` témoins au total.

Durée de vie des témoins

Les témoins ont une durée de vie, variable selon le site et les besoins, et certains sont extrêmement transitoires (expirant à la fermeture du fureteur). Les fureteurs commerciaux permettent pour la plupart d'accepter automatiquement les témoins, de les rejeter automatiquement ou de les accepter ou de les rejeter à la pièce[169]. Il est possible sur certains fureteurs de dresser une liste noire de sites d'où refuser les témoins puis d'accepter automatiquement les témoins d'autres sites.

Certains sites, surtout des sites de commerce électronique, dépendent de la présence de témoins sur le poste client. Rejeter systématiquement les témoins tend à être un peu trop agressif pour les opérations normales de navigation. En retour, les accepter ou les rejeter à la pièce est un peu irritant du fait que plusieurs sites abusent (énormément!) des témoins[170].

Il est essentiel qu'un témoin ne contienne aucune information confidentielle et que sa durée de vie soit aussi brève que possible.

Les données d'un témoin sont habituellement entreposées sous forme de texte en clair sur le disque, dans un ou plusieurs fichiers selon les fureteurs. Si un ordinateur est utilisé par un individu hostile, par exemple suite à un cambriolage (je parle en connaissance de cause ici), n'importe qui peut ouvrir le ou les fichiers témoins d'un disque avec un éditeur conventionnel et en examiner le contenu.

Si des données confidentielles vous semblent devoir être entreposées dans un témoin, assurez-vous que la durée de vie du témoin soit la plus proche possible de la durée de vie de la session de navigation d'un usager.

Étant donné les risques que posent les témoins pour la confidentialité de l'expérience de navigation[171], mieux vaut concevoir des pages Web n'ayant qu'un usage restreint et transitoire de cet outil. Ne courrez pas le risque d'être tenus responsables d'une fraude si votre application choisit de conserver de l'information confidentielle en clair sur le disque rigide d'un poste client.

[169] Mon option de prédilection.

[170] Un exemple parmi plusieurs : une simple connexion au site de Bell Canada, entreprise de stature importante dans le monde des télécommunications s'il en est une, peut facilement installer une douzaine de témoins, dont plusieurs proviennent de tiers. C'est une expérience frustrante que de constater combien de données sont entreposées pour si peu de navigation; je doute sincèrement qu'ils soient tous nécessaires.

[171] Voir `http://www.foruminternet.org/texte/documents/general/lire.phtml?id=304&` (un peu vieillissant), `http://www.junkbusters.com/cookies.html` (un peu alarmiste), `http://www.ciac.org/ciac/bulletins/i-034.shtml` (vieillissant mais moins alarmiste); `http://www.microsoft.com/info/cookies.mspx` (un peu gentil envers la technologie); ou encore `http://www.securiteinfo.com/conseils/cookies.shtml`

Utiliser des témoins[172]

Examinons un exemple simple de recours à un témoin en JavaScript.

La page Web proposée à droite offre une zone de texte pour fins d'entrée de données (`id="nom"`) de même que trois boutons de commande, servant respectivement à créer un témoin, examiner ce témoin et détruire ce témoin.

La valeur du témoin sera le nom entré par l'usager dans la zone de texte. Pour fins de simplicité, j'utiliserai la même page du début à la fin plutôt que d'avoir recours à une interaction avec un serveur Web quelconque.

```
<!DOCTYPE html
  PUBLIC "-//W3C//D/D XHTML 1.0  Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Test de t&eacute;moin</title>
    <script type='text/javascript'
            language="javascript" src="temoins.js" />
  </head>
  <body>
    <input type="text" id="nom" />
    <button
onclick="créerTémoin(document.getElementById('nom').value);">
      Accepter
    </button>
    <button onclick="voirTémoin();">
      Examiner
    </button>
    <button onclick="détruireTémoin();">
      D&eacute;truire
    </button>
  </body>
</html>
```

[172] Cet exemple simple est rédigé en JavaScript, mais il est possible de gérer des témoins dans la plupart des langages de programmation axés sur le Web. Pour des exemples en `ASP`, par exemple, vous pouvez consulter le site `http://www.w3schools.com/asp/asp_cookies.asp`

Le code JavaScript de gestion des témoins sera placé dans le fichier `temoins.js` auquel la page Web en question fait référence.

La fonction `créerTémoin()` prend un paramètre et crée un témoin associant le texte `"nom_bidon"` et la valeur de ce paramètre. On peut mettre tout ce qu'on souhaite dans un témoin, jusqu'à concurrence de `4 Ko` de données. Le format est "*nom=valeur*" et on sépare les paires {nom, valeur} par des points-virgules.

La fonction `voirTémoin()` affiche la valeur du témoin de ce document.

Pour détruire un témoin, il suffit de lui attribuer une date d'expiration (`expires=`*`date`*) qui soit valide mais antérieure à maintenant.

```
function créerTémoin(arg) {
  if (arg) {
    document.cookie = "nom_bidon=" + arg;
  }
}
function voirTémoin() {
  alert (document.cookie);
}
function détruireTémoin() {
  va expiration="Fri, 13 Jul 2001 05:28:21 UTC";
  document.cookie =
    "nom_bidon=bidon;expires="+expiration;
}
```

Le site `http://www.yourhtmlsource.com/javascript/cookies.html` propose du code plus général et plus joli pour gérer les témoins en JavaScript.

Notez que sauvegarder le nom d'un usager ou le contenu des champs d'un formulaire dans un témoin n'est pas recommandable (à tout le moins, assurez-vous que le témoin expire rapidement si vous procédez ainsi!).

Faiblesses des témoins

La première faiblesse du modèle des témoins est qu'ils entreposent puis émettent de l'information sur le profil d'utilisation d'un individu. Par définition, un témoin constitue une forme d'intrusion dans la vie privée, une forme de bris de confidentialité. En conséquence, il est raisonnable de limiter les témoins sur un poste de travail donné à ceux dont on a véritablement besoin et à ceux provenant de serveurs jugés dignes de confiance.

Autre faiblesse des témoins : ils sont locaux à un fureteur, ce qui réduit leur potentiel pour identifier une session ou un individu si celui-ci utilise plusieurs fureteurs distincts.

Plusieurs attaques sont possibles sur le mécanisme des témoins[173] :

- il est possible pour un tiers hostile de capturer un témoin transigé entre un fureteur et un serveur, ce qui peut poser problème si le témoin contient de l'information privilégiée (chiffrer les communications avec `HTTPS` peut aider à résoudre ce problème);
- il est aussi possible de corrompre un témoin en modifiant son contenu alors qu'il est véhiculé entre le client et le serveur .Un truc permettant de réduire l'impact de cette attaque est de limiter au minimum l'information contenue dans le témoin (n'utiliser qu'un numéro de session généré aléatoirement par le serveur, par exemple);
- très irritant et indépendant des tiers hostiles : l'état d'un témoin peut être corrompu par le simple fait de naviguer (par exemple : revenir à la page précédente dans une séance de navigation peut avoir comme effet secondaire la modification des valeurs du témoin à partir d'un état de navigation antérieur).

Les sites qui ont recours aux témoins indiquent habituellement leurs raisons et leur politique à cet effet sur une page de notice légale. Il est sage de prendre connaissance de ces considérations avant de faire le choix d'accepter un témoin.

Étant donné que plusieurs utilisateurs refusent les témoins, tout comme plusieurs refusent les applets ou bloquent le code JavaScript, il est sage, si possible, de concevoir une interface Web de manière à ce que celle-ci fonctionne correctement avec un sans témoins (quitte à ce qu'elle soit plus agréable à utiliser quand les témoins sont acceptés).

Aller de l'avant : la technologie `DOM`:Storage

Certains fureteurs (en particulier les versions récentes de Firefox[174]) implémentent un standard émergent du `W3C` nommé *`DOM:Storage`*. L'idée derrière cette spécification est de proposer aux applications Web un espace d'entreposage local au poste client pour des données applicatives, de manière à dépasser les limites des témoins sans compromettre la sécurité du poste client.

Le recours à des espaces d'entreposage plus importants découle de la mouvance *Web `2.0`*. En effet, les attentes des usagers des applications Web contemporaines sont d'avoir, à travers leurs interfaces Web, des outils aussi riches que ceux déployés localement sur leurs postes clients.

---

[173] Voir `http://en.wikipedia.org/wiki/HTTP_cookie#Drawbacks_of_cookies`

[174] Voir `http://developer.mozilla.org/en/docs/DOM:Storage`

La capacité d'entreposer des données localement permet, entre autres choses, de ne pas perdre de données pendant une session de travail même si le poste de travail rencontre des ennuis pour ce qui est de l'accès à Internet. Une implémentation correcte du standard devrait permettre à une application Web de récupérer ses données même suite à un plantage du fureteur dans lequel s'exécutait cette application.

> On remarquera que le standard *DOM:Storage* a pour effet de mettre en place l'équivalent d'une base de données locale pour applications Web. Le fureteur est de plus en plus un système d'exploitation en soi.
>
> Pour un exemple d'application utilisant *DOM:Storage*, je vous invite à essayer `http://aaronboodman.com/halfnote/`

Le standard *DOM:Storage* permet d'entreposer des données propres à une session (objet `sessionStorage`) ou des données persistantes à plus long terme (objet `globalStorage`). On parle ici d'un standard sujet à devenir extrêmement important du fait qu'il s'intègre dans un ensemble plus large qu'on nomme le standard Web Applications `1.0`[175]. Les applications Web se standardisent et convergent vers un modèle de plus en plus universel; vous ne perdez pas votre temps en étudiant ce sujet!

[175] Voir `http://www.whatwg.org/specs/web-apps/current-work/` pour le document entier et `http://www.whatwg.org/specs/web-apps/current-work/#storage` pour la section portant sur le standard *DOM:Storage* à proprement dit.

Questions de sécurité

Les systèmes utilisant des services Web reposent sur des protocoles de communication très standardisés et très lisibles (`SOAP` et `XML-RPC`, principalement), ce qui, joint au fait que ces systèmes sont connectés au monde entier et accessible par des outils aussi répandus que des fureteurs Web, les rend particulièrement vulnérables à certains types d'attaques.

Les services Web tendent à être de petits objets, habituellement sans états, pris en charge par des outils très complexes que sont les serveurs Web comme Apache et `IIS`.

Une partie de la sécurité propre à ces systèmes repose sur les serveurs Web eux-mêmes, mais une partie revient aussi aux composants qui implémentent les services Web en tant que tels.

On considère particulièrement dangereuses les menaces qui tombent dans les catégories suivantes :

- invasions (tentative d'obtention d'un accès non autorisé au serveur);
- transformation des messages;
- espionnage sur la ligne;
- exposition de données de configuration; et
- l'interception de messages dans le but de les envoyer plus d'une fois à un serveur.

Vocabulaire

Commençons par présenter le vocabulaire de base dont nous aurons besoin pour nous exprimer clairement sur le sujet.

Dans la transmission d'un message, l'***émetteur*** est celui qui envoie le message alors que le ***destinataire*** est celui à qui le message est destiné. On identifiera comme ***tiers hostile*** un intervenant autre que l'émetteur et le destinataire qui tente de lire et de comprendre le message transmis entre ces deux homologues.

Un message ***en clair*** est un message n'ayant pas été transformé, conservant sa lisibilité normale. On parle de ***message transmis en clair*** lorsqu'on veut indiquer qu'un message passe de l'émetteur au récepteur sans quelque transformation supplémentaire que ce soit. Convenons tout de suite que ce n'est pas là une stratégie sécuritaire de communication.

On parle de ***chiffrement***[176] pour identifier l'opération transformant un message pour le rendre incompréhensible à un (hypothétique) tiers. Le message résultant d'un chiffrement est dit ***chiffré***.

On parle de ***déchiffrement*** pour identifier l'opération transformant un message chiffré de manière à ce que l'opération de chiffrement correspondante soir renversée. Un message une fois ***déchiffré*** devrait avoir la forme qui était la sienne avant chiffrement.

Par **chiffrement**, on entend une transformation réversible d'un texte[177] en un texte différent et qui n'est pas porteur du même sens (qui peut, en fait, ne pas avoir de sens pour qui ne sait pas le ***déchiffrer***). Il existe même un standard de chiffrement `XML`[178].

La ***cryptographie*** est la science qui étudie les techniques de chiffrement et de déchiffrement pour les rendre les plus sécuritaires et les plus efficaces possibles. La ***cryptanalyse*** est la science qui étudie les techniques pour briser des messages chiffrés.

---

[176] Ce qu'on nomme parfois ***encryption*** est un anglicisme pour chiffrement.

[177] Le mot ***texte*** doit être pris ici dans le sens très large de message, et pas nécessairement un message dont le contenu serait alphabétique.

[178] Voir `http://www.w3.org/TR/xml-encryption-req`

Standards de transformation à des fins non sécurisées

Le chiffrement n'est pas la seule transformation applicable (ou appliquée) à des transactions entre homologues. On trouve aussi :

- des transformations de ***compression*** (inverse : ***décompression***) servant à comprimer un message de manière réversible dans le but de diminuer la bande passante requise pour son transfert ou l'espace requis ou pour son entreposage. Pensez aux standards `ACE`, `ZIP` et `RAR`, pour n'en nommer que deux des plus populaires[179];
- des transformations de ***binaire à texte***, qui permettent de transférer n'importe quel fichier à travers un protocole *a priori* destiné au transfert de messages textuels. Remplacer, entre autres, les valeurs non affichables comme `'\b'` ou `L'\0'` par des équivalents affichables, un peu comme la norme `HTML` demande de remplacer `<` par `<`. Pensez à `BINHEX`, à `UUENCODE` ou à `MIME` pour les messages envoyés par courriel et comprenant des fichiers joints;

> Certaines transformations servent précisément à sécuriser le texte transmis. On utilise pour cette fin le terme anglais *sanitization*, qu'on pourrait traduire librement par nettoyage.
>
> Ce terme décrit ce que font les fonctions qui consomment du texte susceptible de contenir des balises (p. ex. : des balises `HTML`) et qui remplacent les symboles jugés à risque par un équivalent qui n'en contient pas[180].

- des transformations protocolaires, incontournables lors de transactions par *sockets*, par lesquelles le message est inséré dans des trames standards (`TCP`, `UDP`, `SMTP`, `FTP`, et ainsi de suite, selon les protocoles) et segmenté en plusieurs paquets.

Mis à part la compression, qu'on applique souvent manuellement sur les données dans un souci d'économie, les transformations indiquées ci-dessus se font généralement de manière automatique et à l'insu des humains.

---

[179] Pour une étude de la question plus spécifique de documents ou de flux contenant des données en format `XML`, voir `http://www.w3.org/2003/08/binary-interchange-workshop/33-HiT-W3C-Workshop-2003.pdf`

[180] En JavaScript, la fonction `escape()` joue un rôle semblable et transforme de manière inversible un texte en texte `XML` légal.

Chiffrement

L'un des aspects primitifs de la sécurité est le chiffrement des communications. Le chiffrement se décline en plusieurs familles. On compte :

- la ***stéganographie***, par laquelle l'émetteur du message s'efforce de le dissimuler dans autre chose (la proverbiale aiguille dans une botte de foin[181]);
- le ***chiffrement à clé secrète***, qui repose sur la connaissance partagée par les homologues (chiffreur et déchiffreur) d'un secret, habituellement un nombre premier de grande taille et choisi avec soin (ou le fruit d'un calcul sur de gros nombres premiers). La technique menant au chiffrement des données doit être publique (les attaquants prospectifs doivent avoir droit à un accès plein et entier de l'algorithme de chiffrement et de déchiffrement) et seul la clé de chiffrement et de déchiffrement doit être secrète[182]; et
- le ***chiffrement à clé publique***, qui repose sur une paire de clés par homologue, soit une clé publique et une clé privée. La clé publique d'un homologue sert pour permettre à d'autres de chiffrer le texte qui lui sera envoyé, mais seule la clé privée du même homologue permet de déchiffrer ledit texte. Ce chiffrement dépend de deux clés plutôt que d'une seule. Encore une fois, il faut que la stratégie repose entièrement sur le secret du chiffre (la clé privée) plutôt que sur la non divulgation d'un algorithme[183].

Une stratégie de chiffrement doit être examinée par des experts et doit être mise à l'épreuve de manière très rigoureuse et très agressive pour être considérée sécuritaire. C'est pour cette raison que les algorithmes doivent toujours être divulgués : on présume que les attaquants vont obtenir l'algorithme, alors il doit être arbitrairement difficile de déchiffrer un message codé avec un algorithme donné qu'on ait ou non l'algorithme de chiffrement entre les mains.

Le texte chiffré doit survivre à une attaque directe (donc être difficile à déchiffrer dans l'immédiat) mais doit aussi survivre à une attaque prolongée (donc offrir une forme de protection longue durée). Habituellement, il est considéré souhaitable qu'un secret chiffré puisse être protégée pour une vingtaine d'années, et ce malgré les progrès matériels et les estimés obtenus à l'aide de la loi de Moore[184].

Les clés sont habituellement des chiffres à `128` ou (ce qui est nettement préférable) à `256` bits. En `2007`, on suggérait de ne pas utiliser de clés d'au moins `256` bits. Les problèmes de calcul sur des clés de cette taille tiennent à ce que l'arithmétique sur de grands entiers est une algorithmique complexe et coûteuse en temps (trouver le reste de la division entière par un entier sur `256` bits d'une exponentiation entre deux entiers de `256` bits chacun est une opération demandant beaucoup, *beaucoup* de calcul).

---

[181] Voir `http://www.onlamp.com/pub/a/bsd/2003/12/04/FreeBSD_Basics.html` ou encore `http://www.cs.cmu.edu/~dst/DeCSS/Gallery/` pour des détails.

[182] Les cas les plus connus d'algorithmes de chiffrement à clé secrète sont probablement le (aujourd'hui désuet) `DES` et le (plus récent) `AES`.

[183] Les cas les plus célèbres sont les algorithmes `DH` et `RSA`.

[184] Voir `http://fr.wikipedia.org/wiki/Loi_de_Moore`

Le chiffrement est une opération visant à transformer un message de manière à le rendre illisible à un intervenant hostile. Il s'agit d'une transformation appliquée au *contenu*[185], qui doit pouvoir être inversée (opération de déchiffrement) par le récepteur autorisé du message mais pas par autrui. Le contenu devrait être porteur du sens attendu sans nécessairement contenir le texte précis par lequel on l'aurait exprimé normalement—un langage secret où *Les patates sont cuites* aurait un sens particulier est un langage admissible pour le texte chiffré.

La qualité du chiffrement dépend de quelques facteurs :

- la rapidité du calcul menant au chiffrement et au déchiffrement est une considération importante pour les transactions courantes. Lorsqu'on envoie un courriel à un destinataire, ou lorsqu'on désire procéder à un achat via Internet, il est rare qu'on soit en mesure d'accepter des délais importants;
- la robustesse de l'algorithme utilisé : à quel point les messages ainsi chiffrés résisteront-ils à des tentatives de déchiffrement hostiles?
- la catégorie de clé utilisée : l'algorithme est-il un algorithme à clé secrète ou à clé publique?
- la fréquence à laquelle les clés doivent être mises à jour, ce qui équivaut souvent à parler de la taille des clés utilisées;
- la capacité que nous offre l'algorithme de remarquer les tentatives de modification ou de bris du message; *etc*.

Règle générale, on pensera les algorithmes de chiffrement de manière pessimiste : la présomption de base sera qu'il est peine perdue de garder le secret quant à une clé secrète ou quant à un algorithme sophistiqué de chiffrement des données, du fait que tôt ou tard, il se produira une fuite humaine ou autre qui compromettra le secret.

[185] Ceci parce que les homologues sont responsables de chiffrement avant envoi et du déchiffrement après réception. Les informations servant à la construction des trames pour le protocole de transport sur lequel transitent les paquets (p. ex. : `TCP/IP`), si elles sont chiffrées, rendent le transport impossible (à moins que les routeurs entre l'émetteur et le récepteur du paquet chiffré ne sachent comment déchiffrer cette information, comme c'est le cas avec `IPSec`).

Modes de chiffrement répandus sur le Web et pour le courriel

Les algorithmes phares de la cryptographie à clé publique sont RSA[186] et DH, tous deux des abréviations construites à partir de la première lettre du nom de leurs auteurs.

Il existe, on s'en doute, des outils cryptographiques disponibles à tous et répandus sur le Web. Le plus connu du grand public est sans doute le logiciel (autrefois gratuit) PGP[187], pour *Pretty Good Privacy*, qui chiffre le message, authentifie son origine et protège le message contre les modifications.

De manière plus générale, les standards PKCS[188], pour *Public Key Cryptography Standards*, sont ceux qui sont implantés par les fureteurs[189] pour sécuriser les transactions électroniques.

La convention de sécurité n'est pas aussi claire pour les homologues d'une transaction SMTP (transmission de courriel). Les encodages typiquement reconnus par des homologues SMTP ne sont pas des encodages voués à la sécurisation des transactions. On parle habituellement, avec le courriel, d'encodage MIME, BINHEX ou UUENCODE.

Chiffrer les courriels

Par défaut, les courriels SMTP ne sont pas chiffrés outre mesure[190]. Cela n'empêche pas l'émetteur et le destinataire de transiger des courriels chiffrés, mais demande de leur part de chiffrer et de déchiffrer eux-mêmes leurs messages, à l'aide d'outils comme PGP, par exemple.

Certains outils de courriel[191] permettent d'utiliser SSL comme support aux transmissions de messages, ce qui leur donne un niveau de sécurité intéressant. La possibilité pour les clients de courriel d'utiliser des certificats d'authentification existe, mais cette pratique n'est pas universelle pour le moment.

La guerre aux pourriels

Il y a présentement un mouvement (dont *Microsoft* se veut le porte-étendard) pour modifier quelque peu la forme par laquelle se transigent les courriels.

Dans le modèle préconisé, un destinataire potentiel enverrait un puzzle à l'émetteur, que l'émetteur devrait pouvoir résoudre dans la mesure où il déploierait une certaine capacité de calcul.

Ceci impliquerait un coût plus élevé si un message est envoyé à plusieurs destinataires; la présomption sous-jacente est que cela découragerait la production de pourriels en la rendant plus coûteuse.

---

[186] Voir par exemple sur http://www.di-mgt.com.au/rsa_alg.html, ou encore sur http://www.rsasecurity.com/rsalabs/rsa_algorithm/

[187] Voir http://pgp.com/

[188] Voir http://www.rsasecurity.com/rsalabs/pkcs/

[189] ***Firefox*** utilisait, au moment d'écrire ces lignes (version ***2.0.0.3***), **PCKS #11**, que vous pouvez consulter sur le Web à l'adresse http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html

[190] À ce sujet, lire http://luxsci.com/extranet/articles/email-security.html

[191] Comme par exemple *Pegasus Mail* : http://www.pmail.com/index.cfm; *Hushmail* : https://www.hushmail.com/; ou encore *Eudora :* http://www.eudora.com/, pour n'en nommer que quelques unes des options disponibles. *Outlook*, l'outil de courriel de *Microsoft*, supporte aussi SSL si les options requises y sont activées.

Authentification et de certification

Le chiffrement sert à protéger le texte transmis dans l'espace (sur un lien de communication) ou dans le temps (entreposé sur un média pour être récupéré ultérieurement).

Cela dit, il est possible pour un attaquant se logeant ente deux homologues de nuire à l'un ou à l'autre sans déchiffrer le message transmis. Par exemple, un tiers hostile pourrait intercepter un message, l'endommager et le laisser poursuivre sa route vers son destinataire. Il pourrait aussi chercher à supprimer une trame du texte ou rejouer plusieurs fois la même trame.

Ce sont là des raisons pour lesquelles il est important que le destinataire d'un message puisse établir avec confiance l'identité de son émetteur. C'est ce qu'on nomme l'**authentification**, qu'on implémentera souvent à l'aide d'une forme de ***signature électronique*** (insérer dans le flux d'une trame une valeur disant *Ce message vient de…*). Il importe que le destinataire puisse reconnaître les tentatives de fraude, donc que la stratégie d'authentification soit robuste, alors la signature fera habituellement partie du message chiffré.

L'authentification comporte toutefois son lot de problèmes dû au nombre d'identités que chaque individu (et chaque programme) se doit de maintenir. Pensez simplement au nombre d'adresses de courriel que vous possédez! De même, on ne peut s'attendre à ce que *chaque* homologue de *chaque* relation `CS` dans le monde puisse authentifier de lui-même chaque homologue potentiel. Ce problème, dans le cas des fureteurs Web en particulier, serait d'un niveau de complexité incontrôlable.

C'est pourquoi certaines entités s'offrent aussi comme **autorités de certification**. Les homologues désirant s'authentifier auprès d'une autorité de certification autorisent cette dernière à se porter garante de leur identité auprès de tiers, un peu comme le font les firmes de crédit. Évidemment, une autorité de certification doit être un organisme de confiance[192].

Une autorité de certification offre ce qu'on nomme un ***certificat***, soit un document électronique décrivant à la fois l'autorité de certification et l'entité certifiée. Un fureteur sécuritaire fera en sorte, avant d'entreprendre une transaction devant être sécurisée, de valider, auprès d'une autorité de certification, le certificat offert par le site en question.

L'authentification et la certification sont des concepts complémentaires. Visiter un site muni d'un certificat permet de connaître la confiance accordée au site par les autorités de certification reconnues, et permet aussi de connaître les standards de sécurité par lesquels il est possible de transiger avec lui.

[192] Le plus connu en Amérique du Nord et ***Verisign***.

Emploi de SSL

La plupart des sites Web offrent des URL selon le protocole HTTP, qui implique des transactions par sockets typiquement faites suite à une connexion sur les ports 80 ou (surtout pour les services Web) les ports 8080, 8087 et 8088.

Les sites voués à des transactions sécurisées offriront plutôt des URL selon le protocole HTTPS, utilisant TLS, basé sur le *Secure Sockets Layer*[193] (SSL) développé par *Netscape*. Les transactions SSL sont chiffrées selon un protocole à clé publique.

C'est à travers SSL qu'un fureteur prendra conscience du certificat d'un site Web, verra si les transactions y sont chiffrées à l'aide de clés chiffrées sur 40 bits ou 128 bits (les standards de taille de clés pour SSL). On comprendra que plus la taille de la clé sera grande et plus grande sera la sécurité obtenue.

Le fureteur et les protocoles

Le fureteur est un outil de téléchargement et d'affichage de fichiers de différents types.

Il encapsule le support de plusieurs protocoles standards construits sur TCP/IP, comme GOPHER, HTTP, HTTPS, TFTP, NEWS, et ainsi de suite. Le début d'une URL dénote le protocole à employer pour communiquer avec l'objet qu'elle désigne (p. ex. : http://).

Lorsqu'une URL préfixée par https:// est consultée par un fureteur, ce dernier sait devoir vérifier un certificat. Le protocole de vérification sous-jacent est implicitement compris par le fureteur de par la nature d'une URL donnée.

Le support de SSL est offert sur pratiquement tous les fureteurs Web, ce qui signifie que l'accès à des sites https:// est possible peu importe la plateforme ou l'outil.

Emploi de SSH

Les transactions de type telnet, rlogin ou rsh, par lesquelles on utilise sur un ordinateur un terminal virtuel sur un autre, peuvent aussi être réalisées de manière relativement sécurisée si chaque échange de données est chiffré depuis le tout début. Depuis la transmission du nom et du mot de passe du client se connectant par ce protocole.

Le protocole *Secure Shell* (SSH) offre précisément ce genre de support, et permet une partie de la fonctionnalité du protocole FTP. Ce protocole constitue une alternative propre et sécuritaire aux telnet traditionnels. Le prix à payer : des délais de transmission résultant du chiffrement et du déchiffrement des messages.

Rôle d'IPSec

Le protocole IPSec est un protocole IP sécurisé.

Là où HTTPS offre une version chiffrée de HTTP et où SSL offre des *sockets* par canal chiffré, IPSec assure que la strate IP elle-même soit sécurisée.

Dans tous les cas, le chiffrement est un pas dans la bonne direction mais pas la solution à tous les maux. Encore faut-il l'utiliser correctement.

[193] **URL**: http://wp.netscape.com/security/techbriefs/ssl.html

Catégories d'attaques

Les problèmes de sécurité listés ici et les suggestions faites pour répondre à ces situations problématiques s'appliquent à autre chose qu'à des services Web, évidemment.

Invasions

Les services Web servent souvent de façade pour des accès circonscrits à des SGBD, mais de manière générale tout service Web susceptible de permettre un accès à un média de masse ou à d'autres processus devrait authentifier son client et valider les droits devant être accordés et reconnus à ce client.

Évidemment, les transactions pour lesquelles le serveur est un service Web utilisent presque nécessairement XML, peut-être sous forme SOAP ou XML-RPC, à travers HTTP, et tendent donc à transporter des données en clair, visibles à tout intervenant hostile s'intercalant entre les homologues.

On comprendra qu'il est important de ne pas faire circuler un mot de passe en clair sur une trame http; chiffrez les données importantes avant envoi, ou assurez-vous que vos échanges sont véhiculés par HTTPS plutôt que par HTTP.

Transformation des messages

On a en tête ici un intervenant hostile qui s'interpose entre un service Web et son client et qui transforme un message au passage (changeant un mot ou une valeur numérique par exemple). Ceci peut avoir des effets arbitrairement graves, allant du banal (mauvaise réponse à un calcul bénin) au très sérieux (fraude financière, destruction d'une BD, bris matériel suite à une valeur numérique incorrecte, *etc.*).

Les principales options s'offrant aux homologues pour se protéger contre des modifications hostiles aux messages échangés entre deux homologues sont de *chiffrer* chaque message et de *signer* chaque message :

- chiffrer un message le rend difficile à comprendre et à altérer; alors que
- signer un message numériquement permet de valider que chaque message s'est rendu intact à destination.

Espionnage sur la ligne

L'espionnage est une tactique d'écoute (idéalement non intrusive, pour éviter les transformations de messages susmentionnées) sur un lien de communication par laquelle un intrus cherche à obtenir de l'information pouvant être précieuse, compromettante ou confidentielle.

La protection de base contre ce type d'attaque est le chiffrement, que ce soit celui du message (insertion d'une strate de chiffrement entre la construction du message et son envoi, et entre la réception du message et son traitement) ou du protocole de transport (HTTPS plutôt que HTTP).

Exposition des données de configuration

On a ici en tête ce qui peut se produire lorsqu'un serveur Web ou une mécanique côté serveur génère des descriptions `WSDL` sur demande, ce qui risque d'exposer au monde des fonctionnalités qui auraient dû rester connues seulement du serveur lui-même.

Une manière bête par laquelle on peut obtenir des données descriptives de la configuration d'un serveur est simplement en s'assurant de générer des exceptions lors d'appels de méthodes de services Web. Les exceptions réparties tendent à contenir de l'information descriptive des postes sur lesquelles elles se produisent.

La tactique est perverse mais répandue : créer des erreurs pour essayer de décoder le contexte dans lequel elles se produisent. C'est une technique plus répandue et plus facile à implémenter qu'on pourrait le croire.

Un truc simple pour empêcher que la confidentialité de la configuration du serveur Web ne soit compromise par la génération automatique de fichiers `WSDL` est de placer les fichiers `WSDL` sur des ordinateurs autres que celui où réside le serveur Web lui-même.

Un autre truc est de capturer la plupart des exceptions possibles côté serveur et de ne laisser sortir que des exceptions génériques et peu descriptives quant à la configuration de l'ordinateur. Cette technique est toutefois difficile à mettre en place et complique le déverminage.

Interception et reprise de messages

Un intrus aux intentions hostiles peut intercepter un message et le renvoyer à plusieurs reprises à un même homologue, peut-être même sans le modifier au passage.

Les principales options pour éviter que cela ne cause des dégâts sont de chiffrer les messages et de les numéroter. Le chiffrement n'empêchera pas un même message d'être envoyé à plusieurs reprises, mais empêchera une attaque de dégénérer en transformation ou en espionnage.

Il existe toute une panoplie de variantes à cette approche : changer la date ou le numéro de série d'un paquet, échanger les numéros de deux paquets, enlever un paquet, l'émettre plusieurs fois, *etc*.

Toutes ces approches peuvent être combattues par un protocole dont la structure est robuste, mais concevoir un tel protocole est beaucoup plus difficile qu'il n'y paraît.

Tactiques célèbres

Il existe des tactiques d'assaut sur des services Web qu'on pourrait qualifier de classiques. En voici quelques-unes.

Injection SQL[194]

La tactique qu'on nomme injection SQL consiste à transformer une requête SQL transmise sur un lien réseau de manière à en changer subtilement le comportement.

Ces transformations peuvent être tout simples, comme modifier la requête émise par cet extrait de code Java[195] :

```
String sql = "SELECT * FROM USERS WHERE PASSWORD='"+pwd+"'";
ResultSet rs = conn.createStatement().executeQuery(sql);
```

pour obtenir quelque chose comme :

```
SELECT * FROM USERS WHERE PASSWORD='' OR ''='';
```

ce qui a la vilaine particularité de contenir une clause WHERE qui est toujours vraie. Notez qu'on peut en arriver à la seconde à partir de la première en utilisant un contenu construit volontairement de manière à s'intégrer à la requête qui sera émise.

Un autre type d'injection est de modifier ou d'ajouter des champs à des requêtes DELETE ou UPDATE … SET pour corrompre la BD ou créer des vecteurs d'attaque supplémentaires. Imaginez une requête SQL de la forme DELETE … WHERE auquel on aurait ajouté, dans la clause WHERE, une tautologie!

Le problème est d'actualité : un article[196] de juillet 2006 mettait de l'avant que les banques sont de plus en plus assaillies de cette manière, alors qu'un autre[197], datant de 2006 mais mis à jour à la fin de janvier 2007, souligne que l'injection SQL arrive présentement au deuxième rang des vulnérabilités les plus exploitées dans les systèmes répartis, juste après le *Cross-Site Scripting* (XSS) sur lequel nous reviendrons plus bas.

Une protection possible contre cette tactique est de désactiver le recours aux littéraux, pour éviter des clauses telles que WHERE … OR 1=1. La plupart des moteurs de BD offrent cette option. Il faut alors que le code générant les requêtes SQL utilise des *Prepared Statements* (quel est le terme français?) et des constantes symboliques plutôt que littérales.

---

[194] Un exemple frappant et qui tourne un peu les auteurs en dérision peut être vu à l'adresse http://worsethanfailure.com/Articles/Now_Hiring_SQL_Injectors.aspx

[195] Source : http://www.h2database.com/html/frame.html?advanced.html%23sql_injection&main

[196] http://www.net-security.org/secworld.php?id=4076

[197] http://portal.spidynamics.com/blogs/msutton/archive/2006/09/26/How-Prevalent-Are-SQL-Injection-Vulnerabilities_3F00_.aspx

Pour vous aider, privilégiez aussi les procédures stockées et les vues aux requêtes sur des tables. Une saine hygiène de gestion de votre BD réduit les risques sans toutefois les éliminer. Offrir un accès externe à une BD exigera de vous et de votre équipe de la discipline et de la rigueur. Essayez de penser comme un tiers hostile et d'être proactifs, donc de voir les problèmes avant qu'ils ne surviennent; vous n'aurez peut-être pas de deuxième chance.

Une autre protection possible serait de valider les types des expressions suppléées avant de construire la requête. Ceci peut permettre de détecter certaines transformations dangereuses (telles que ='3' avec le littéral 3 qui est entier, et qu'un tiers hostile aurait voulu transformer en 3' OR '3'='3 pour obtenir ='3' OR '3'='3').

Cela dit, SQL est un langage expressif et laisser transiter des requêtes SQL sur un lien réseau est une très, *très* mauvaise idée. Privilégiez la circulation de requêtes dans un format plus restrictif que SQL sur vos liens réseaux et gardez SQL pour les accès locaux.

Cette remarque s'applique à tout langage suffisamment descriptif et n'est pas limitée à SQL. Pour quelques exemples, voir :

```
http://www.site-reference.com/articles/Website-Development/Malicious-Code-Injection-It-s-Not-Just-for-SQL-Anymore.html
```

Si vous travaillez avec PHP, vous aimerez peut-être cet extrait de livre :

```
http://www.apress.com/ApressCorporate/supplement/1/437/1590595084-2957.pdf
```

Cross-Site Scripting *(XSS)*

La tactique nommée *Cross-Site Scripting*, qu'on abrège habituellement sous la forme XSS[198], consiste en la divulgation potentielle d'information à des tiers hostiles à travers des techniques reposant sur des scripts côté client. Certains[199] tiennent d'ailleurs un discours mettant de l'avant qu'un site exposant une vulnérabilité de type XSS n'est pas nécessairement un site hostile mais bien un site possédant une vulnérabilité qu'il est possible d'exploiter.

Pour décrire une exploitation de type XSS, mieux vaut un exemple, et l'un des mieux connus est le cas *XSS, Trust, Barney*[200] qui montre comment un individu un peu retors mais pas nécessairement malicieux a démontré comment, en injectant du code HTML ou JavaScript dans un livre de visiteurs (un *Guest Book*) il parvenait à faire afficher un dinosaure violet là où on se serait attendu à voir la signature et les remarques d'un visiteur anodin[201].

---

[198] Et non pas CSS puisque cette forme est maintenant associée aux *Cascading Style Sheets*. De très vieux textes sur XSS utiliseront peut-être la forme CSS, donc mieux vaut être conscient(e) de la nuance.

[199] Voir http://neosmart.net/blog/2006/what-xss-isnt/

[200] Voir http://www.webmonkey.com/webmonkey/00/18/index3a.html

[201] Une autre classique révélant le contenu d'un témoin dans une URL à partir d'une page malicieuse côté client est décrit sur http://www.informit.com/articles/article.asp?p=603037&rl=1

Comme le met de l'avant le site Wiki sur le sujet[202], le terme `XSS` est mal choisi mais les spécialistes de sécurité vivent avec, par habitude et par manque de temps pour raffiner le vocabulaire. On y définit trois catégories de vulnérabilités `XSS`, exemples réels d'incidents à l'appui. Ces catégories sont :

- le type `0`, qui est local, apparaît quand du code client (souvent JavaScript) construit une partie du contenu de la page dans laquelle il s'exécute à partir de sources externes. Il y a alors un risque que ce contenu contienne lui-même du code `HTML` et transforme à la fois le sens attribué au contenu de la page et les risques qui lui sont associés[203];
- le type `1`, type le plus commun, survient lors d'interactions entre du code client et un script côté serveur destiné à générer une nouvelle page Web. L'intervention de scripts malicieux côté client peut mener à la génération de pages Web par le script serveur contenant des données qui n'auraient pas dû y apparaître. Ces attaques sont souvent associées à une forme d'ingénierie sociale ou de hameçonnage (de *Phishing*) à partir d'une page Web falsifiée mais ressemblant étrangement à une page Web réelle (disons la page de connexion des services en ligne d'une banque);
- le type `2`, enfin, exploite des données entreposées côté serveur, ce qui peut être particulièrement malicieux si ces données sont utilisées pour générer des pages Web à grande échelle (pour l'interface d'un système de messagerie instantanée, par exemple).

Les exploitations de type `XSS` permettent aux malfaiteurs créatifs de s'exprimer. À titre d'exemple, une démonstration a été faite[204] de la possibilité de générer un comportement viral à partir de tactiques `XSS` : un fureteur y examine une page Web dans laquelle un fichier est créé côté serveur puis chargé et rafraîchi régulièrement par d'autres fureteurs examinant la même page, ce qui permet la propagation et la dissémination de données qui se reproduisent elles-mêmes d'un fureteur aux autres. Cette manière d'exploiter le modèle est bénigne mais ouvre la porte à des variantes qui le sont moins.

Pour d'autres détails sur les exploitations de type `XSS`, incluant quelques exemples opérationnels, voir `http://www.cgisecurity.com/articles/xss-faq.shtml`

[202] `http://en.wikipedia.org/wiki/Cross-site_scripting`

[203] Un exemple simple est `http://www.webappsec.org/projects/articles/071105.shtml` qui montre comment il serait possible de révéler, sans réel effort, le contenu des témoins d'un document.

[204] Voir `http://www.bindshell.net/papers/xssv/`

Sécurité de services Web et déploiement

L'accent mis sur la sécurité pour un service Web sera influencé par le type de déploiement qu'on en fera. En effet, on ne s'y prendra pas de la même manière pour sécuriser un service Web déployé sur un intranet, pour sécuriser un service Web déployé sur un extranet (espèce d'hybride semblable à un intranet mais accessible à des partenaires commerciaux) ou pour sécuriser un service Web exposé à Internet tout entier.

Étrangement, les options de sécurisation sont plus grandes sur un intranet, alors que les besoins sont plus grands si un service est exposé au monde entier à travers Internet.

L'intranet d'une entreprise est un lieu privilégié, où (en théorie) l'entreprise contrôle la plateforme de tous les homologues, la technologie des clients, de même que celles des serveurs et des services offerts. Ceci permet entre autres d'exploiter, si on en a envie, les services de sécurité des plateformes sous-jacentes, surtout si le parc informatique de l'entreprise est homogène[205].

De plus, on estime habituellement que les communications `C/S` sont sécurisées par le fait même que l'intranet est un lieu auquel l'accès est sécurisé *a priori*. Ce savoir peut permettre une économie d'échelle intéressante et peut aider à augmenter le débit des transactions, réduisant le besoin de chiffrer et de valider les messages.

Il est moins probable qu'une entreprise ait le contrôle sur les outils et les plateformes utilisées sur son extranet. Même s'il est possible que la confiance règne sur l'extranet et entre les différentes firmes qui y contribuent, les probabilités qu'une firme ou l'autre ait une forme de contrôle sur les plateformes et sur les produits qui y sont utilisés sont plus faibles, et il devient important de faire reposer la stratégie de sécurité sur des outils plus standards et plus répandus (p. ex. : le protocole `TLS`, autrefois nommé `SSL`, et `HTTPS`).

Sur Internet, il faut s'en tenir à des outils standards et disponibles sur un maximum de plateformes et, surtout si le service Web représente un risque de sécurité, présumer que tout client, tout message est susceptible d'être dangereux ou d'être à risque par nature ou parce que susceptible d'avoir été manipulé d'une manière ou l'autre par un intermédiaire hostile.

Il est à noter que plus un serveur est sollicité pour des considérations de sécurité (chiffrement en tour ou en partie des messages, validation de l'identité des homologues, examen des signatures numériques, *etc.*) et moins il sera *échelonnable* à une masse importante de clients. C'est une simple histoire de temps de calcul.

[205] Sinon, on peut tirer profit de l'hétérogénéité et utiliser les forces de chacune des plateformes à leur maximum.

Pour en savoir plus

Un *Whitepaper* très pertinent, rédigé en commun par IBM et *Microsoft* sur la sécurité dans les services Web, est disponible à l'adresse suivante :

```
http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/
```

Je vous en suggère fortement la lecture. Le sujet est des plus pertinents, et cet article traite de manière large l'essentiel du sujet.

Sécuriser un SCS, surtout s'il est ouvert sur le Web, n'est pas une mince tâche. En fait, c'est une variante technique d'un problème ouvert, peut-être même un problème indécidable de par la formulation trop vague qui l'accompagne.

Annexe 00—Code de la classe `AfficheurDOM`[206]

La classe `AfficheurDOM` dérive de `ConsommateurDOM` (voir la section ***Consommer un document XML avec DOM***) et implémente, dans sa méthode `traiter()`, un affichage en arbre de la structure du document reçu en paramètre. Le code suit; vous remarquerez qu'une très petite partie de ce code a vraiment trait au décodage du document, l'essentiel s'attardant à des considérations d'affichage.

```
import org.w3c.dom.Document;
//
// Éléments pour l'interface personne/ machine
//
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.JSplitPane;
import javax.swing.JEditorPane;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;
//
// Éléments pour l'arbre et son modèle
//
import javax.swing.tree.*;
import javax.swing.event.*;
import java.util.*;
//
// Notre petite classe
//
public class AfficheurDOM extends ConsommateurDOM {
   public static void main (String [] args) {
      if (args.length == 0) {
         System.err.println ("Usage: java AfficheurDOM fichierXML [fichierXML ...]");
      } else {
         for (String s : args) {
            AfficheurDOM ad = new AfficheurDOM ();
            ad.consommer (s);
         }
      }
   }
```

[206] Un peu lâche, je me suis **fortement** inspiré du code sur cette page, qui est correct sans être renversant : http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/dom/work/DomEcho02.java

```
   //
   // Le traitement du document à proprement dit
   //
   protected void traiter (Document doc) {
      Affichage aff = new Affichage (doc);
      aff.init ();
      aff.addWindowListener (
         new WindowAdapter () {
            public void windowClosing (WindowEvent we) {
               System.exit (0);
            }
         }
      );
      aff.pack();
      aff.setVisible (true);
   }
   //
   // Classe présentant un Document dans un JTree
   //
   private class Affichage extends JFrame {
      Document m_Document;
      public Affichage (Document doc) {
         super ("Affichage d'un document XML avec DOM");
         m_Document = doc;
      }
      public void init () {
         final int HAUTEUR = 460;
         final int LARGEUR_GAUCHE = 300;
         final int LARGEUR_DROITE = 340;
         final int LARGEUR_TOTALE = LARGEUR_GAUCHE + LARGEUR_DROITE;
         final int ÉPAISSEUR_BORDURE = 5;
         final int ESPACEMENT = ÉPAISSEUR_BORDURE * 2;
         //
         // Nous allons afficher le document dans un arbre
         // (c'est assez naturel comme représentation)
         //
         JTree Arborescence = new JTree(new AdaptateurDOMTreeModel(m_Document));
         //
         // Côté gauche...
         //
         JScrollPane vueEnArbre = new JScrollPane(Arborescence);
         vueEnArbre.setPreferredSize
            (new Dimension (LARGEUR_GAUCHE, LARGEUR_TOTALE));
         //
         // Côté droit...
         //
         JEditorPane panneauHTML = new JEditorPane("text/html","");
         panneauHTML.setEditable(false);
```

```
        JScrollPane vueHTML = new JScrollPane(panneauHTML);
        vueHTML.setPreferredSize
           (new Dimension(LARGEUR_DROITE, LARGEUR_TOTALE));
        //
        // Vue séparée en deux...
        //
        JSplitPane panneauSéparé =
           new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, vueEnArbre, vueHTML);
        panneauSéparé.setContinuousLayout( true );
        panneauSéparé.setDividerLocation( LARGEUR_GAUCHE );
        panneauSéparé.setPreferredSize
           (new Dimension(LARGEUR_TOTALE + ESPACEMENT, LARGEUR_TOTALE + ESPACEMENT));
        setLayout(new BorderLayout());
        add(panneauSéparé, BorderLayout.CENTER);
     }
  }

  //
  // Tableau contenant les noms des types de noeuds DOM. Les indices
  // correspondent aux valeurs des types dans org.w3c.dom.Node, plus bas
  //
  private static final String[] NOMS_TYPES_NOEUDS = {
     "none",
     "Element",
     "Attr",
     "Text",
     "CDATA",
     "EntityRef",
     "Entity",
     "ProcInstr",
     "Comment",
     "Document",
     "DocType",
     "DocFragment",
     "Notation"
  };

  //
  // Enrobe un noeud DOM en donnant accès au texte à afficher dans l'arbre.
  //
  public class AdaptateurNoeudDOM {
     private org.w3c.dom.Node m_NoeudDOM;
     public AdaptateurNoeudDOM (org.w3c.dom.Node NoeudDOM) {
        m_NoeudDOM = NoeudDOM;
     }
     //
     // Méthode pour convertir un AdaptateurNoeudDOM en String
     //
```

```
    public String toString() {
       String Résultat = NOMS_TYPES_NOEUDS[m_NoeudDOM.getNodeType()];
       String nom = m_NoeudDOM.getNodeName();
       if (nom.length () != 0 && !nom.startsWith("#")) {
          Résultat += ": " + nom;
       }
       if (m_NoeudDOM.getNodeValue() != null) {
          if (Résultat.startsWith("ProcInstr")) {
             Résultat += ", ";
          } else {
             Résultat += ": ";
          }
          //
          // Nettoyage au cas où le texte commencerait par des blancs ou un saut de ligne
          //
          String s = m_NoeudDOM.getNodeValue().trim();
          int ndxNouvelleLigne = s.indexOf("\n");
          if (ndxNouvelleLigne >= 0) {
             s = s.substring(0, ndxNouvelleLigne);
          }
          if (s.length () > 0) {
             Résultat += s;
          } else {
             Résultat = "(vide)";
          }
       }
       return Résultat;
    }
    //
    // Cherche l'index d'un enfant
    //
    public int index(AdaptateurNoeudDOM Enfant) {
       int NombreEnfants = compterEnfants();
       for (int i = 0; i < NombreEnfants; i++) {
          AdaptateurNoeudDOM anDOM = getEnfant(i);
          if (Enfant.m_NoeudDOM == anDOM.m_NoeudDOM) {
             return i;
          }
       }
       return -1; // hum...
    }
    public int compterEnfants() {
        return m_NoeudDOM.getChildNodes().getLength();
    }
    public AdaptateurNoeudDOM getEnfant (int ndx) {
       org.w3c.dom.Node noeudDOM =
            m_NoeudDOM.getChildNodes().item(ndx);
       return new AdaptateurNoeudDOM(noeudDOM);
```

```
        }
    }

    //
    // Adapte un document pour en faire un modèle de Jtree (du «boilerplate code»)
    //
    public class AdaptateurDOMTreeModel implements javax.swing.tree.TreeModel {
        private Vector <TreeModelListener> m_ListeObservateurs;
        private Document m_Document;
        public AdaptateurDOMTreeModel (Document doc) {
            m_ListeObservateurs= new Vector<TreeModelListener>();
            m_Document = doc;
        }
        //
        // Opérations de base d'un TreeModel
        //
        public Object getRoot() {
            return new AdaptateurNoeudDOM(m_Document);
        }
        public boolean isLeaf(Object aNode) {
            AdaptateurNoeudDOM node = (AdaptateurNoeudDOM) aNode;
            return node.compterEnfants() <= 0;
        }
        public int getChildCount(Object parent) {
            AdaptateurNoeudDOM noeud = (AdaptateurNoeudDOM) parent;
            return noeud.compterEnfants();
        }
        public Object getChild(Object parent, int index) {
            AdaptateurNoeudDOM noeud = (AdaptateurNoeudDOM) parent;
            return noeud.getEnfant(index);
        }
        public int getIndexOfChild(Object parent, Object enfant) {
            AdaptateurNoeudDOM noeud = (AdaptateurNoeudDOM) parent;
            return noeud.index((AdaptateurNoeudDOM) enfant);
        }
        public void valueForPathChanged(TreePath path, Object newValue) {
            // sans intérêt pour nous
        }
        //
        // On doit implémenter celles-ci mais on ne s'en servira pas
        //
        public void addTreeModelListener(TreeModelListener observateur) {
            if ( observateur != null && ! m_ListeObservateurs.contains( observateur ) ) {
                m_ListeObservateurs.addElement( observateur );
            }
        }
        public void removeTreeModelListener(TreeModelListener observateur) {
            if ( observateur != null ) {
```

```
            m_ListeObservateurs.removeElement( observateur );
         }
      }
      public void fireTreeNodesChanged( TreeModelEvent e ) {
         Enumeration observateurs = m_ListeObservateurs.elements();
         while ( observateurs.hasMoreElements() ) {
            TreeModelListener observateur =
               (TreeModelListener) observateurs.nextElement();
            observateur.treeNodesChanged( e );
         }
      }
      public void fireTreeNodesInserted( TreeModelEvent e ) {
        Enumeration observateurs = m_ListeObservateurs.elements();
        while ( observateurs.hasMoreElements() ) {
           TreeModelListener observateur =
              (TreeModelListener) observateurs.nextElement();
           observateur.treeNodesInserted( e );
        }
      }
      public void fireTreeNodesRemoved( TreeModelEvent e ) {
        Enumeration observateurs = m_ListeObservateurs.elements();
        while ( observateurs.hasMoreElements() ) {
          TreeModelListener observateur =
             (TreeModelListener) observateurs.nextElement();
          observateur.treeNodesRemoved( e );
        }
      }
      public void fireTreeStructureChanged( TreeModelEvent e ) {
        Enumeration observateurs = m_ListeObservateurs.elements();
        while ( observateurs.hasMoreElements() ) {
          TreeModelListener observateur =
             (TreeModelListener) observateurs.nextElement();
          observateur.treeStructureChanged( e );
        }
      }
   }
}
```

Annexe 01—Fichier `PetitService.wsdl`

Le fichier `PetitService.wsdl`, contenant la description `WSDL` générée pour le service Web `PetitService`, suit :

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="PetitService" targetNamespace="urn:TiService" xmlns:tns="urn:TiService"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types/>
  <message name="IEvaluateur_plusPetit">
    <part name="int_1" type="xsd:int"/>
    <part name="int_2" type="xsd:int"/>
    <part name="int_3" type="xsd:int"/></message>
  <message name="IEvaluateur_plusPetitResponse">
    <part name="result" type="xsd:int"/></message>
  <portType name="IEvaluateur">
    <operation name="plusPetit" parameterOrder="int_1 int_2 int_3">
      <input message="tns:IEvaluateur_plusPetit"/>
      <output message="tns:IEvaluateur_plusPetitResponse"/></operation></portType>
  <binding name="IEvaluateurBinding" type="tns:IEvaluateur">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="plusPetit">
      <soap:operation soapAction=""/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
namespace="urn:TiService"/></input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
namespace="urn:TiService"/></output></operation></binding>
  <service name="PetitService">
    <port name="IEvaluateurPort" binding="tns:IEvaluateurBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/></port></service></definitions>
```

Annexe 02—Fichier `mapping.xml`

Le fichier `mapping.xml` généré par `wscompile` respecte le standard `JSR 109` décrivant la correspondance entre les types `JAX-RPC` de la plateforme Java et ceux de `SOAP`, permettant de voir à quel point les deux sont proches (notez en particulier, la jonction entre espaces nommés `XML` et paquetages Java).

Chaque service exposé apparaît sous la forme d'un élément `XML` de type `service-interface-mapping`. On y voit la correspondance des noms de classes, d'interfaces, de noms de services et de ports `WSDL`.

Le fichier généré suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<java-wsdl-mapping version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://www.ibm.com/webservices/xsd/j2ee_jaxrpc_mapping_1_1.xsd">
<package-mapping>
<package-type>trucs</package-type>
<namespaceURI>urn:TiService</namespaceURI>
</package-mapping>
<package-mapping>
<package-type>trucs</package-type>
<namespaceURI>urn:TiService</namespaceURI>
</package-mapping>
<service-interface-mapping>
<service-interface>trucs.PetitService</service-interface>
<wsdl-service-name xmlns:serviceNS="urn:TiService">serviceNS:PetitService</wsdl-service-
name>
<port-mapping>
<port-name>IEvaluateurPort</port-name>
<java-port-name>IEvaluateurPort</java-port-name>
</port-mapping>
</service-interface-mapping>
<service-endpoint-interface-mapping>
<service-endpoint-interface>trucs.IEvaluateur</service-endpoint-interface>
<wsdl-port-type xmlns:portTypeNS="urn:TiService">portTypeNS:IEvaluateur</wsdl-port-type>
<wsdl-binding xmlns:bindingNS="urn:TiService">bindingNS:IEvaluateurBinding</wsdl-binding>
<service-endpoint-method-mapping>
<java-method-name>plusPetit</java-method-name>
<wsdl-operation>plusPetit</wsdl-operation>
<method-param-parts-mapping>
<param-position>0</param-position>
<param-type>int</param-type>
<wsdl-message-mapping>
<wsdl-message xmlns:wsdlMsgNS="urn:TiService">wsdlMsgNS:IEvaluateur_plusPetit</wsdl-message>
<wsdl-message-part-name>int_1</wsdl-message-part-name>
```

```
<parameter-mode>IN</parameter-mode>
</wsdl-message-mapping>
</method-param-parts-mapping>
<method-param-parts-mapping>
<param-position>1</param-position>
<param-type>int</param-type>
<wsdl-message-mapping>
<wsdl-message xmlns:wsdlMsgNS="urn:TiService">wsdlMsgNS:IEvaluateur_plusPetit</wsdl-message>
<wsdl-message-part-name>int_2</wsdl-message-part-name>
<parameter-mode>IN</parameter-mode>
</wsdl-message-mapping>
</method-param-parts-mapping>
<method-param-parts-mapping>
<param-position>2</param-position>
<param-type>int</param-type>
<wsdl-message-mapping>
<wsdl-message xmlns:wsdlMsgNS="urn:TiService">wsdlMsgNS:IEvaluateur_plusPetit</wsdl-message>
<wsdl-message-part-name>int_3</wsdl-message-part-name>
<parameter-mode>IN</parameter-mode>
</wsdl-message-mapping>
</method-param-parts-mapping>
<wsdl-return-value-mapping>
<method-return-value>int</method-return-value>
<wsdl-message xmlns:wsdlMsgNS="urn:TiService">wsdlMsgNS:IEvaluateur_plusPetitResponse</wsdl-
message>
<wsdl-message-part-name>result</wsdl-message-part-name>
</wsdl-return-value-mapping>
</service-endpoint-method-mapping>
</service-endpoint-interface-mapping>
</java-wsdl-mapping>
```