



E N S E I R B
M A T M E C A
B O R D E A U X



<?xml?>

TALENTED TOGETHER

Unissons nos Talents

Enseirb – Support de cours Web / XML

Objet du document

- Ce document a pour but de présenter les notions de « client / serveur » avec son application dans la navigation Web (HTML) en s'appuyant sur le langage XML et ses dérivés (XSD, XSL, XPath)
- Ce support a pour vocation d'introduire les notions nécessaires à la réalisation les TDs

Votre serviteur :

Sopra
group

Mathieu Lombard
Senior Consultant
Business Consulting – Division Atlantique

mathieu.lombard@sopraconsulting.com ■



Déroulement des cours et des TDs

- Il y aura 5 cours et 7 TDs
- Après chaque cours, un TD permettra d'appliquer ce qui a été appris
 - Chaque TD s'appuiera sur ce qui a été fait lors du précédent
 - si un élève loupe un TD, il doit le rattraper avant d'aborder le suivant
 - Le TDs peuvent se faire en binôme (c'est même recommandé)
- Le 6^{ème} et le 7^{ème} TD feront l'objet de l'évaluation
 - Le 6^{ème} sur le bon avancement du projet
 - Le 7^{ème} sur le développement d'une évolution en séance
 - Les mêmes binômes que lors de TDs peuvent être conservés
- La notation se répartira comme suit
 - 40% pour le 6^{ème} TD (via des tests automatiques)
 - 40% pour le 7^{ème} TD (via des tests automatiques)
 - 20% de pondération selon l'application et l'implication de l'élève ou du groupe (appréciation du professeur)

Sommaire du cours

- Le cours se déroulera sur 5 séances :

- [Sommaire – Cours 1 Le client et le serveur](#)
- [Sommaire – Cours 2 Le langage XML et l'API DOM](#)
- [Sommaire – Cours 3 La validation des documents XML](#)
- [Sommaire – Cours 4 L'utilisation de Xpath](#)
- [Sommaire – Cours 5 Les feuilles de style XSL](#)

Sommaire – Cours 1

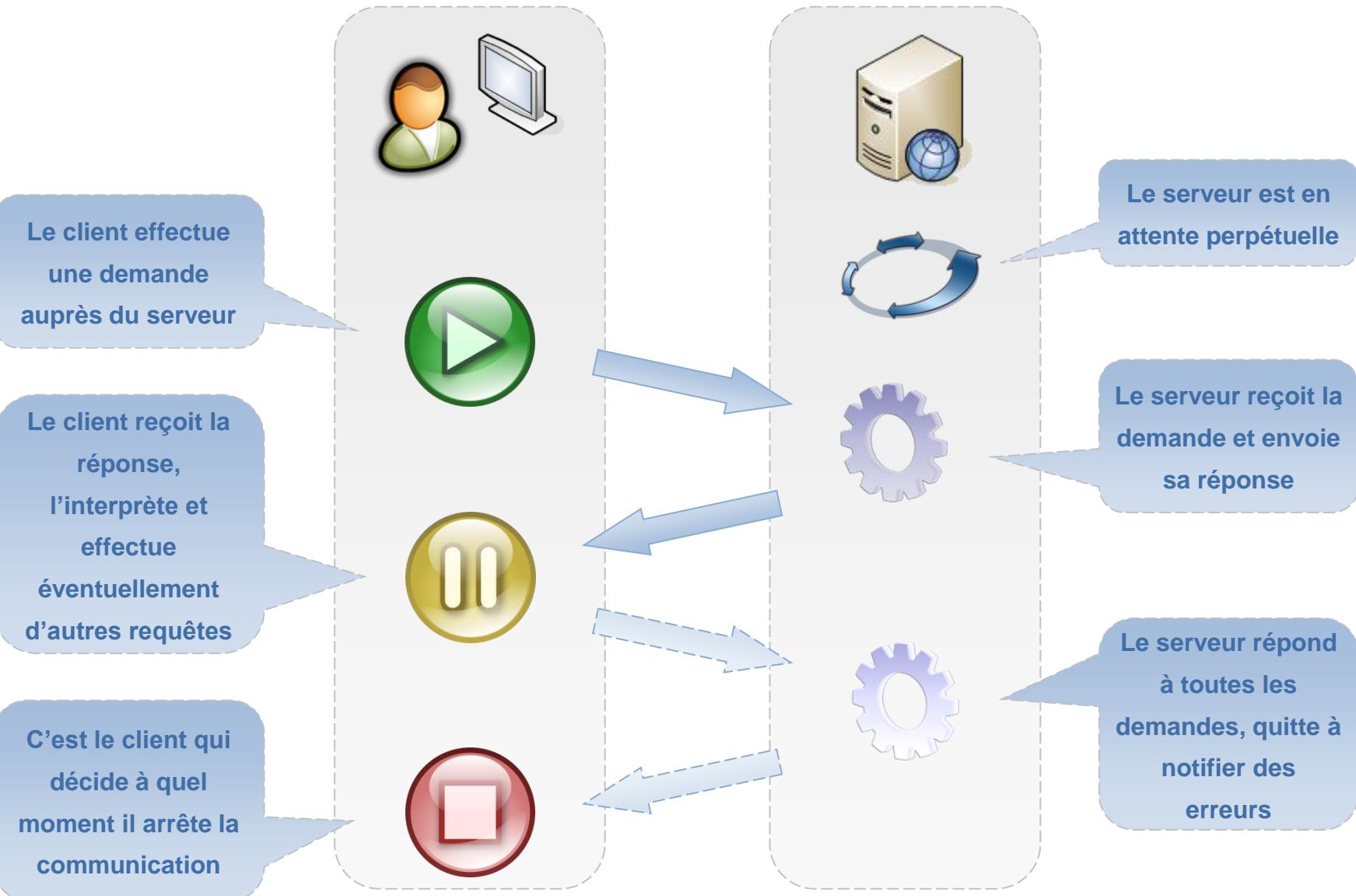
Le client et le serveur

- Client-serveur
 - Mode de fonctionnement
- Le protocole de communication HTTP
 - Historique
 - Utilisation dans le cadre d'un site Web
- Focus sur HTML
 - Extension avec AJAX
- Les Web Services
 - SOAP et REST

Le mode de fonctionnement « Client / Serveur »

- L'analogie classique est : consommateur / fournisseur
 - Le consommateur effectue des demandes auprès d'un fournisseur
 - Le fournisseur répond à la demande du consommateur
- Le serveur est en attente permanente d'une éventuelle requête d'un éventuel client
 - Le client initie la communication
 - Le serveur établit un « échange 2 à 2 » (c'est-à-dire sur un canal unique avec chaque client)
 - Le serveur répond à la demande du client et à ses éventuelles futures demandes
 - Le client interprète les réponses du serveur pour
 - Les afficher à l'écran
 - Ou effectuer tout autre traitement
- Ce mode de fonctionnement est générique et le client, comme le serveur, peuvent être des
 - Hommes
 - Programmes
 - Machines

Schéma de fonctionnement du mode « client / serveur »

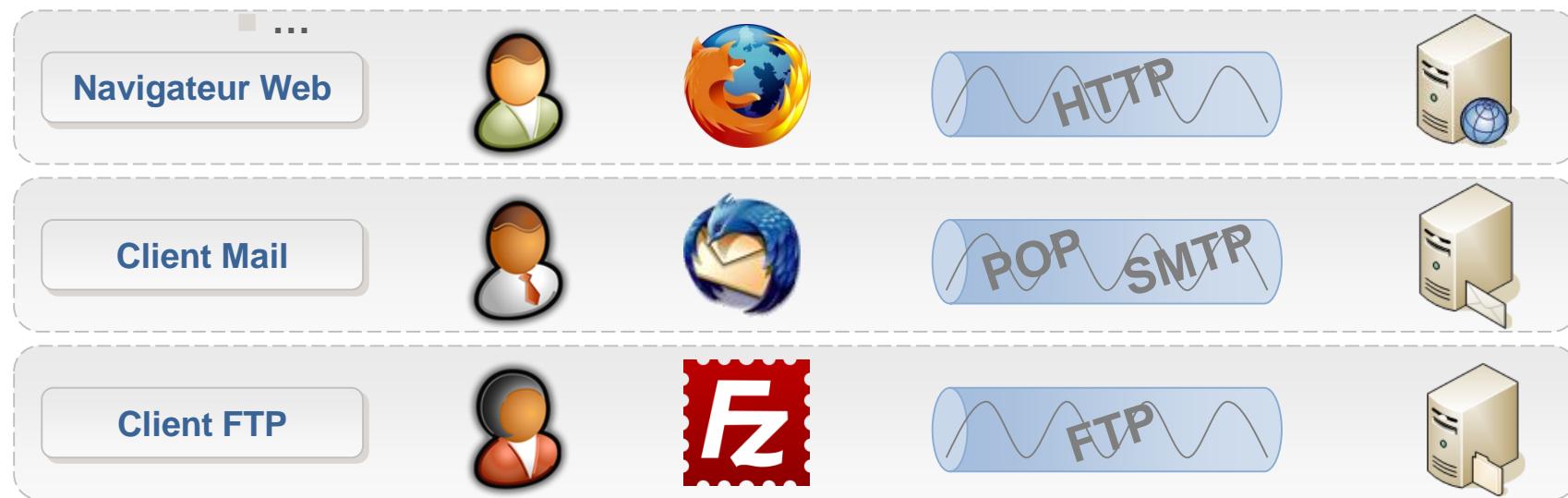


Le serveur rend service !

- **Le serveur n'a d'autre vocation que de servir des demandes**
- **La nature des services dépend totalement du contexte d'utilisation**
 - Renvoyer des données au format HTML pour être visualisées par un navigateur Web
 - Renvoyer un format particulier de données (Google Map)
 - Effectuer un traitement propre à une entreprise (authentification par exemple)
 - ...
- **Un service peut être**
 - **Synchrone**, la requête reste bloquée jusqu'à ce que la réponse soit reçue (demande d'une page Web par exemple)
 - **Asynchrone**, le traitement de la requête n'est pas garanti d'être immédiat (envoi d'un mail par exemple)

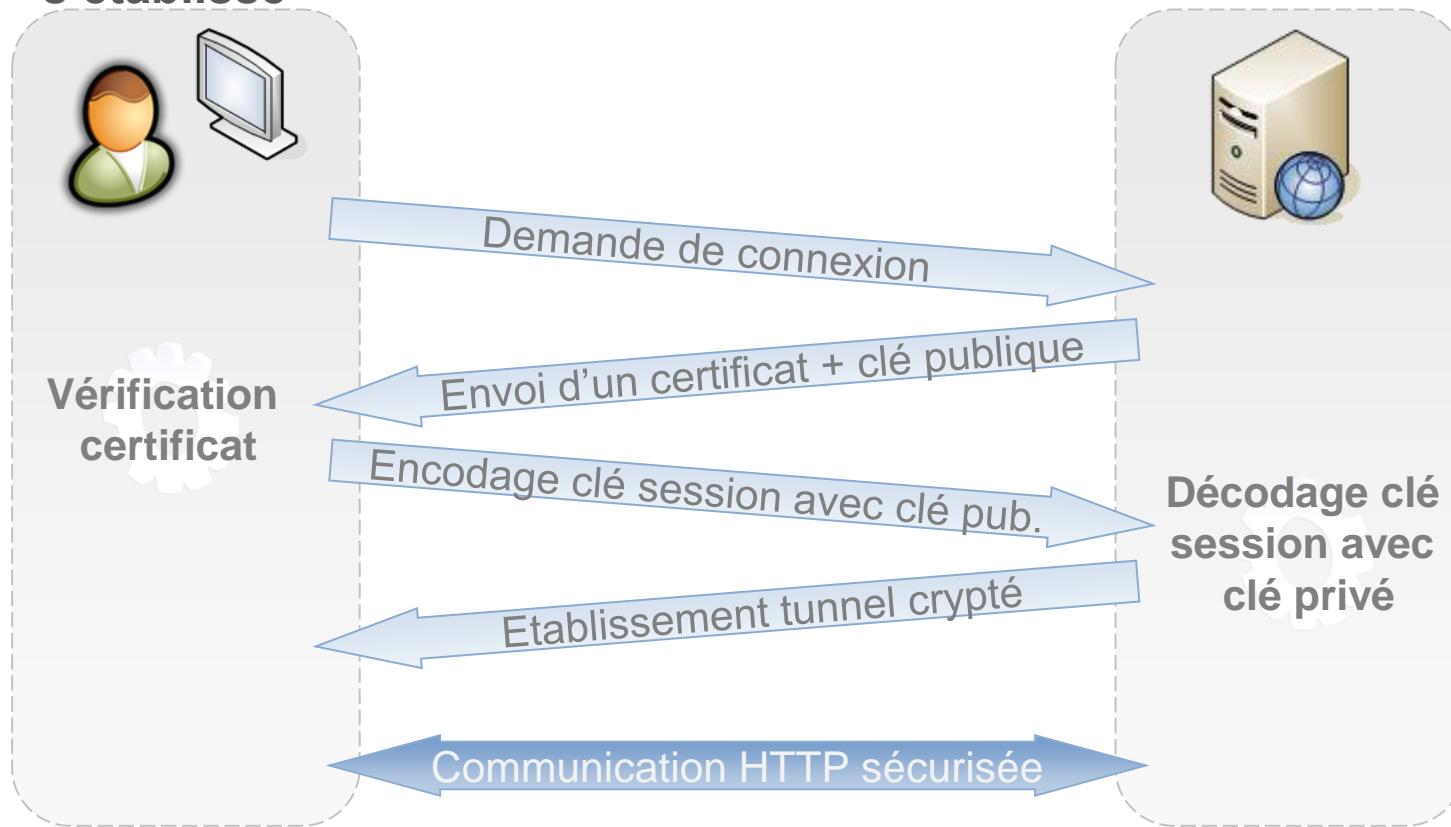
Pour communiquer, il faut parler la même langue

- Le client et le serveur doivent communiquer selon le même protocole
- Selon le besoin adressé, des protocoles dédiés ont été créés pour s'adapter au mieux à la demande, par exemple :
 - HTTP : Hyper Text Transfert Protocol
 - SMTP : Simple Mail Transfert Protocol
 - POP : Post Office Protocol
 - FTP : File Transfert Protocol
 - IRC : Internet Relay Chat Protocol



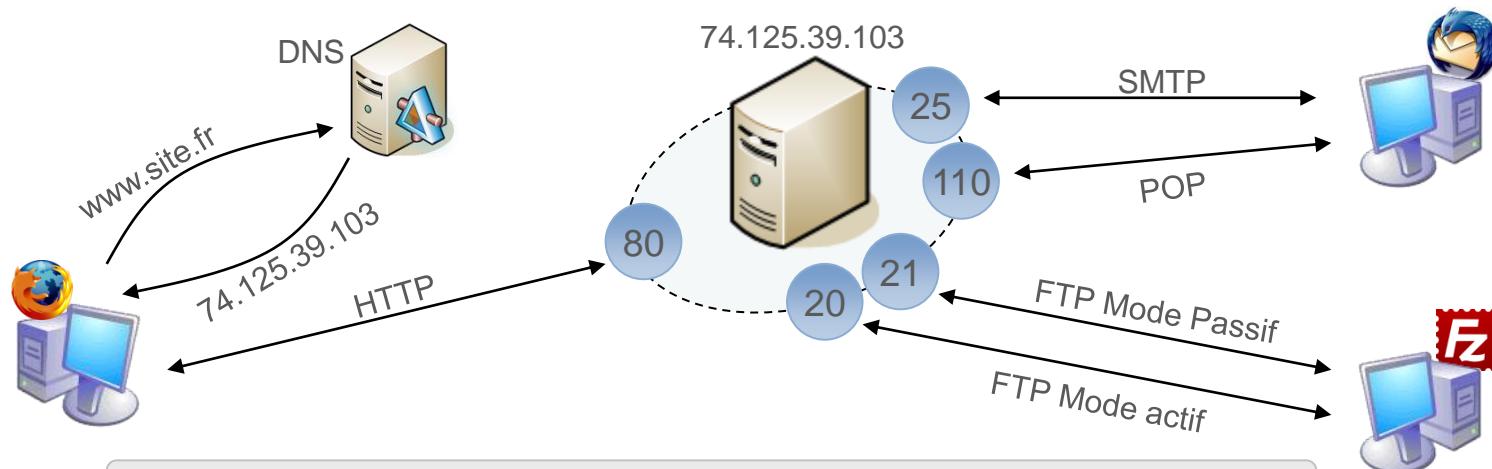
Exemple du protocole HTTPS

- Le protocole HTTPS se base sur HTTP en rajoutant une brique de Sécurité (se basant sur SSL)
- HTTPS est un bon exemple où le client et le serveur échangent préalablement avant que la communication à proprement dit ne s'établisse



Un serveur est accessible par le biais d'un nom et d'un port

- Que se passe-t-il quand je tape **www.google.fr** ?
 - Un serveur DNS (Domain Name System) **Résout** le nom du site pour en déduire son adresse IP
 - Le client (navigateur) établit une connexion vers l'adresse IP du serveur
- Un serveur pouvant rendre plusieurs **services** (HTTP, FTP, SMTP, ...), il faut que les clients puissent les différencier
 - On utilise alors la notion de **port**
 - Un serveur **écoute** sur certains ports de sa machine et interprète les connexions entrantes selon un protocole précis
 - Remarque : quand on accède à un site depuis son navigateur, le port, lorsqu'il n'est pas précisé, est 80

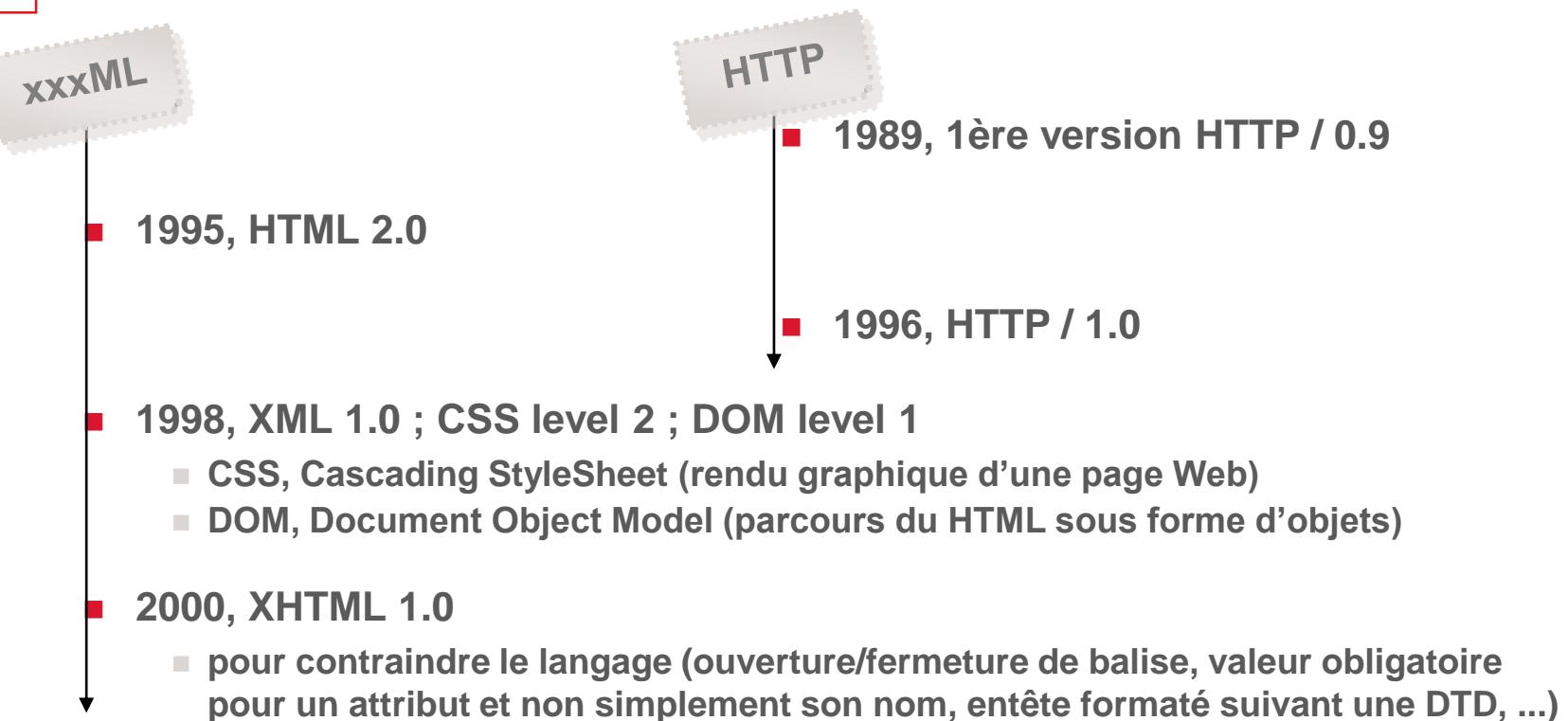


Chaque protocole a son port par défaut (mais peut être changé)

Sommaire

- Client-serveur
 - Mode de fonctionnement
- Le protocole de communication HTTP
 - Historique
 - Utilisation dans le cadre d'un site Web
- Focus sur HTML
 - Extension avec AJAX
- Les Web Services
 - SOAP et REST

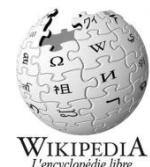
HTTP a 20 ans et son utilisation Web ne cesse de croître



De nombreux navigateurs :
Firefox 1.0 en 2004, IE en V8
en 2009, Google Chrome 0.2
en 2008, Safari, ...

En février 2009,
216.000.000
sites recensés

Source : http://fr.wikipedia.org/wiki/World_Wide_Web#Historique



Premier pas dans le protocole HTTP

- HTTP est un protocole d'échange de ressources (fichiers, flux de données) et il permet de transmettre des pages web mais s'applique aussi à webdav par exemple
- Le protocole HTTP supporte plusieurs commandes
 - GET, pour récupérer une ressource sans la modifier (par exemple, demander la page d'accueil d'un site)
 - POST, pour modifier une ressource (par exemple, ajouter un commentaire sur un blog)
 - d'autres moins utilisées (HEAD, OPTIONS, CONNECT, TRACE, PUT, DELETE)
- Exemple de demande de la page www.google.fr depuis Firefox

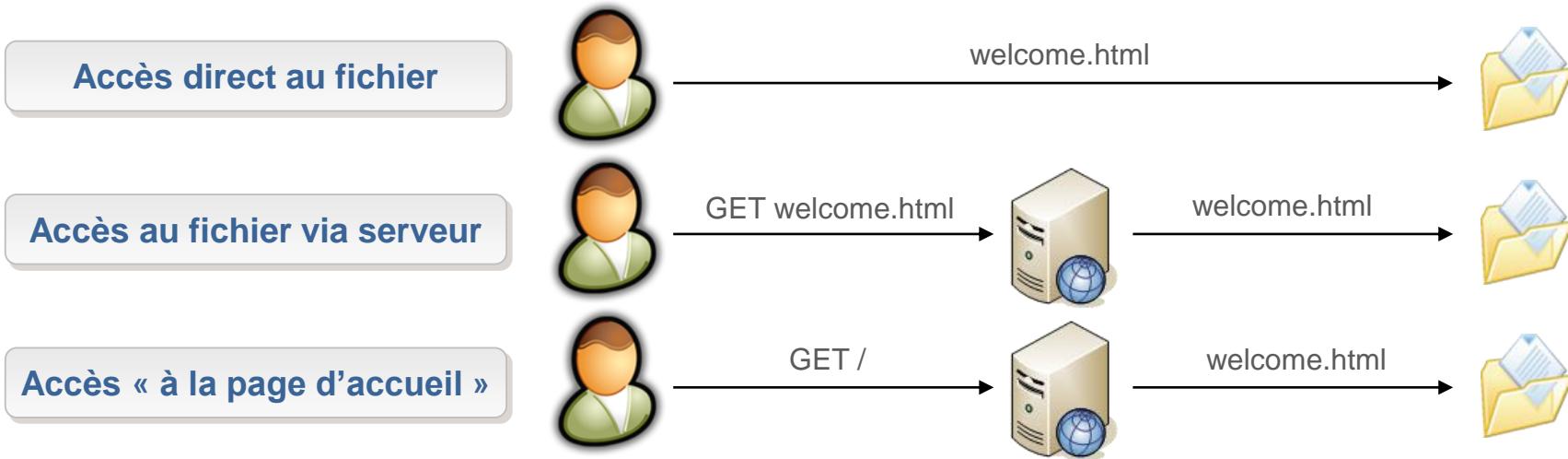
```
(Request-Line)      GET / HTTP/1.1
Host              www.google.fr
User-Agent        Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.1.2) Gecko/20090729 Firefox/3.5.2
Accept            text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language   fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding  gzip,deflate
Accept-Charset   ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive        300
Proxy-Connection keep-alive
Cookie           PREF=ID=ba83bdxfc838d9db9:U=ab1ce7e538628124:TM=1249920117
```

- On retrouve la commande GET qui demande la racine / de l'hôte www.google.fr

Pour plus d'infos sur les autres données : <http://www.w3.org/Protocols/rfc2616/rfc2616.html>

Le mode de fonctionnement client / serveur permet d'offrir une interface

- Le but premier d'un serveur est de répondre aux demandes « fonctionnelles » d'un client
 - Lorsqu'il rend un service, il dit « ce qu'il fait » (le **QUOI**) et non « comment il le fait » (le **COMMENT**)
- Exemple d'accès à la page d'accueil (`welcome.html`) d'un site Web :



- L'utilisateur demande la page d'accueil sans savoir que cela se traduit par le renvoi d'un fichier, cela est avantageux car :
 - Si le fichier en question doit changer de nom ou d'emplacement, c'est transparent pour l'utilisateur (ex : <http://www.mozilla-europe.org/fr/firefox>)
 - Si le contenu renvoyé par le serveur dépend d'un contexte (utilisateur, date, ...), le `welcome.html` peut être adapté puis renvoyé

Plusieurs technologies sont utilisables pour développer un serveur Web

- Un serveur se décompose en
 - Machine physique (processeur, mémoire vive, ...)
 - Serveur logiciel hébergeant les applications
 - Applications développées dans un langage interprétable par le serveur
- Comme souvent, le monde est séparé en 2 (majoritairement) concernant les serveurs d'application

Microsoft

- Serveur IIS
- Langages supportés
 - CGI
 - ASP
 - ASP.Net
 - PHP !

Java

- Nombreux serveurs (Tomcat, ...)
- Nombreuses technologies sous-jacentes (Servlet, JSP, ...)

- Concernant le monde des « serveurs web », on retrouve l'incontournable « Apache », fonctionnant majoritairement sous Unix et interprétant Perl, PHP, Python, Ruby, CGI, ...

Dans le monde Java, les 2 notions de base sont Servlet et JSP

- **Une Servlet est un point d'entrée du serveur**
 - Une Servlet donnée peut être vue comme l'un des **services** que rend le serveur
 - Une Servlet se focalise sur le **fond** plus que sur la forme (on écrit du HTML brut)
- **Une page JSP (Java Server Page) est une page HTML qui contient des parties rendues dynamiques**
 - On n'enrichit pas « à proprement dit » la DTD XHTML car ce n'est pas du XML
 - Une JSP se focalise sur la **forme** plus que sur le fond (on rend dynamique du HTML déjà existant)
- **Le bon compromis est l'utilisation des 2 techniques**
 - La Servlet sert de point d'entrée
 - Elle interprète les paramètres et effectue les traitements « métiers » (accès à une base de données pour récupérer des informations, calculs divers, ...)
 - Elle transmet les paramètres calculés à une JSP qui s'occupe de faire le rendu graphique

Il existe de nombreux frameworks permettant « d'augmenter » ces capacités de base

La gestion des servlet via le fichier web.xml

- Dans le cadre d'une application Web étant exécutée via un « conteneur de Servlet » (par exemple Tomcat), il faut lister les servlets (les points d'entrée) offerts par l'application

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>EnseirbWebXMLWebapp</display-name>
    <servlet>
        <description></description>
        <display-name>DefaultServlet</display-name>
        <servlet-name>DefaultServlet</servlet-name>
        <servlet-class>fr.enseirb.webxml.servlet.DefaultServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DefaultServlet</servlet-name>
        <url-pattern>/DefaultServlet</url-pattern>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    ...
</web-app>
```

Classe JAVA qui sera invoquée

URL = point d'entrée

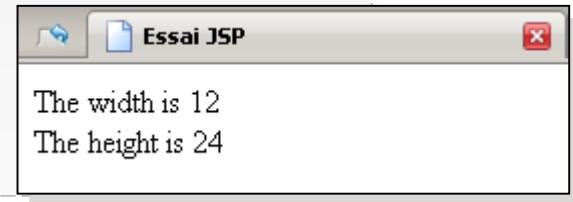
Exemple d'une JSP

- Pour fabriquer une page JSP,
 - On part d'une page HTML
 - On rend dynamique les parties qui doivent l'être

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<%
    String width = (String) session.getAttribute("width");
    String height = (String) session.getAttribute("height");
%>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>Essai JSP</title>
    </head>
    <body>
        The width is <%=width%><br>
        The height is <%=height%>
    </body>
</html>
```



- Dans l'exemple, les 2 valeurs (width et height) sont calculées et non écrites directement dans le HTML
- Le calcul de ces valeurs est effectué en Java (ou elles peuvent être passées en paramètre de l'URL) et renvoyé à la JSP en tant que paramètre

Un serveur renvoie « ce que l'on veut »

- Quand on effectue une requête auprès d'un serveur (en Java, on « attaque » une Servlet généralement), on attend une réponse pour
 - Afficher une page Web (HTML), c'est le cas des navigateurs
 - Effectuer un autre traitement, le format de données est alors à décider entre le client et le serveur ; on parle de **contrat de service** (qui permet de définir les paramètres en entrée / sortie avec leur type et cardinalité)
- On peut alors considérer qu'un serveur Web renvoyant du HTML offre le service suivant : « envoi d'informations pré-formatées pour être visualisées selon la DTD XHTML »
- Mais il existe bien d'autres manières d'accéder à un serveur
 - XMLHttpRequest et JSON
 - Web Services (REST et SOAP)
 - ... tout autre besoin spécifique

Sommaire

- Client-serveur
 - Mode de fonctionnement
- Le protocole de communication HTTP
 - Historique
 - Utilisation dans le cadre d'un site Web

- Focus sur HTML
 - Extension avec AJAX

- Les Web Services
 - SOAP et REST

Description de HTML

- HTML est le seul langage (avec ses compléments que sont JavaScript, CSS) interprété par les navigateurs Web, c'est dire son importance !
 - Néanmoins, avec l'adjonction de plugins, un navigateur peut afficher autre chose, c'est le cas de Flash par exemple
- HTML suit une DTD (cf. Cours N°2)
 - Depuis le XHTML, le langage est plus contraint qu'avant (ce qui apporte un meilleur respect des standards)
 - La dernière version est XHTML 1.0 (polémique sur la future version...)

```
<html>
    <head>
        <title>Essai JSP</title>
    </head>
    <body>
        The width is <em class="important">24</em><br>
        The height is <em class="important">12</em>
        
    </body>
</html>
```

- HTML est donc basé sur XML et partage donc les mêmes notions
 - Balises, avec une hiérarchie entre elles
 - Attribut, avec une valeur fonctionnelle différente selon l'attribut (cf. l'attribut « class » qui décrit un style)

Ajax permet d'étendre les mécanismes de chargement du HTML depuis le serveur

- **AJAX (Asynchronous Javascript And XML) est une méthode vouée à**
 - **Effectuer des appels client vers serveur alors que la page HTML est déjà chargée**
 - C'est-à-dire qu'un premier aller-retour a déjà eu lieu
 - Les suivants ne seront alors pas bloquants pour visualiser la page Web et ils viendront la modifier éventuellement ou simplement transmettre des données au serveur
 - **Via DOM, modifier le code HTML de la page déjà chargée**
- **On parle de communication asynchrone car le mécanisme utilisé a pour but d'effectuer des opérations**
 - Non strictement nécessaires (en théorie)
 - Non bloquantes
 - Lors le chargement de la page elle-même
 - Lors d'une requête pendant l'utilisation de la page
- **Avec l'arrivée du Web 2.0 (dont 1 des buts est d'améliorer l'expérience utilisateur des internautes), l'utilisation d'AJAX est devenu courante**
 - Plutôt que de devoir recharger la totalité des pages, on ne recharge qu'une partie
 - Certains contrôles peuvent s'effectuer à la volée plutôt que d'attendre la validation totale des données des formulaires

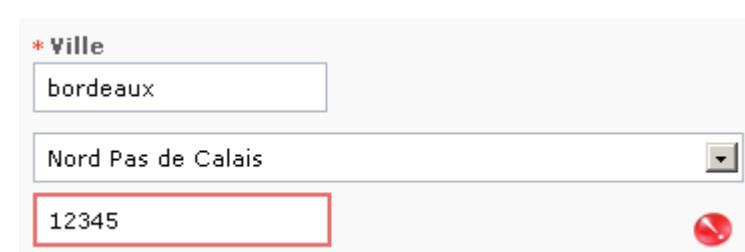


Exemples de l'apport d'AJAX dans les sites Web

- **www.airfrance.fr**, quand on saisit son identifiant et mot de passe, la page affiche des informations personnelles sans se recharger totalement



- **www.monster.fr**,
 - la validation de la forme de l'e-mail s'effectue en JS (pas AJAX) en ajoutant un message à la page
 - la validation du CP s'effectue en interrogeant le site via AJAX quand on valide le formulaire

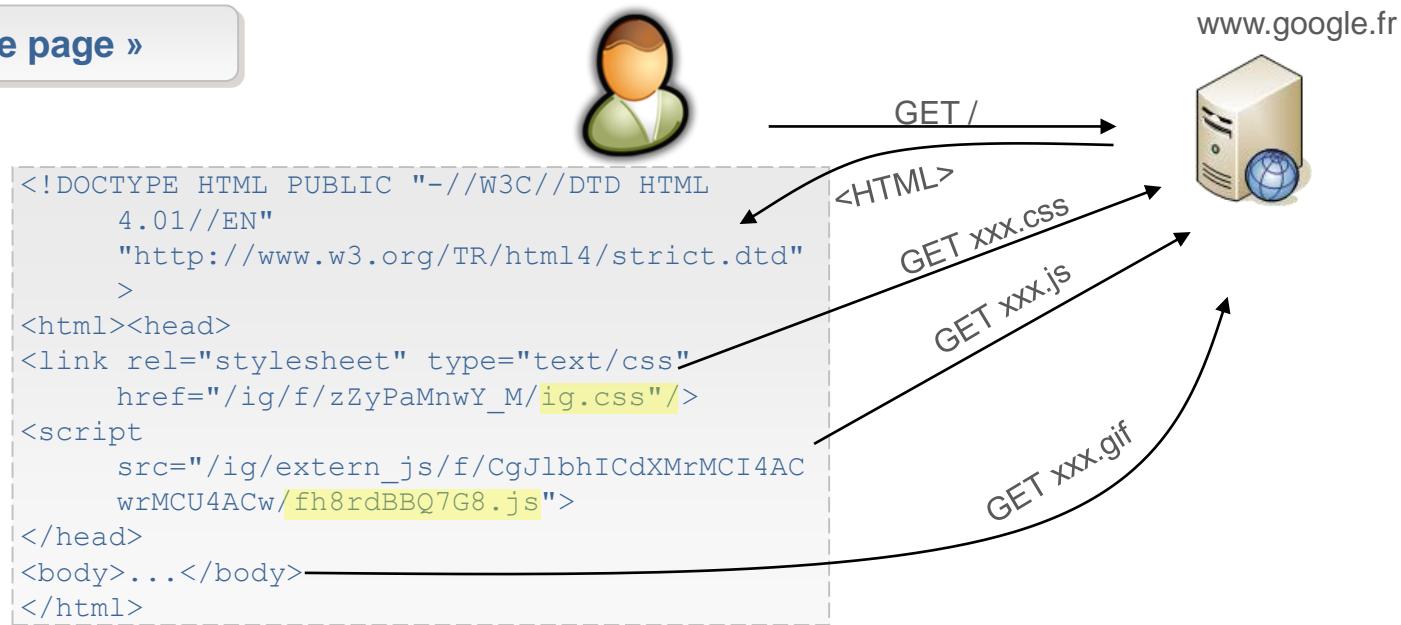


Monster pourrait encore améliorer les choses en validant le CP, toujours via AJAX, au moment de la saisie

Différence de fonctionnement entre une requête « de page » et une requête « AJAX »

Requête « de page »

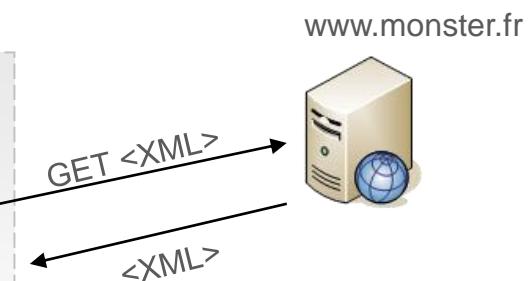
Le navigateur analyse le HTML au fur et à mesure et effectue les requêtes nécessaires



Requête « AJAX »



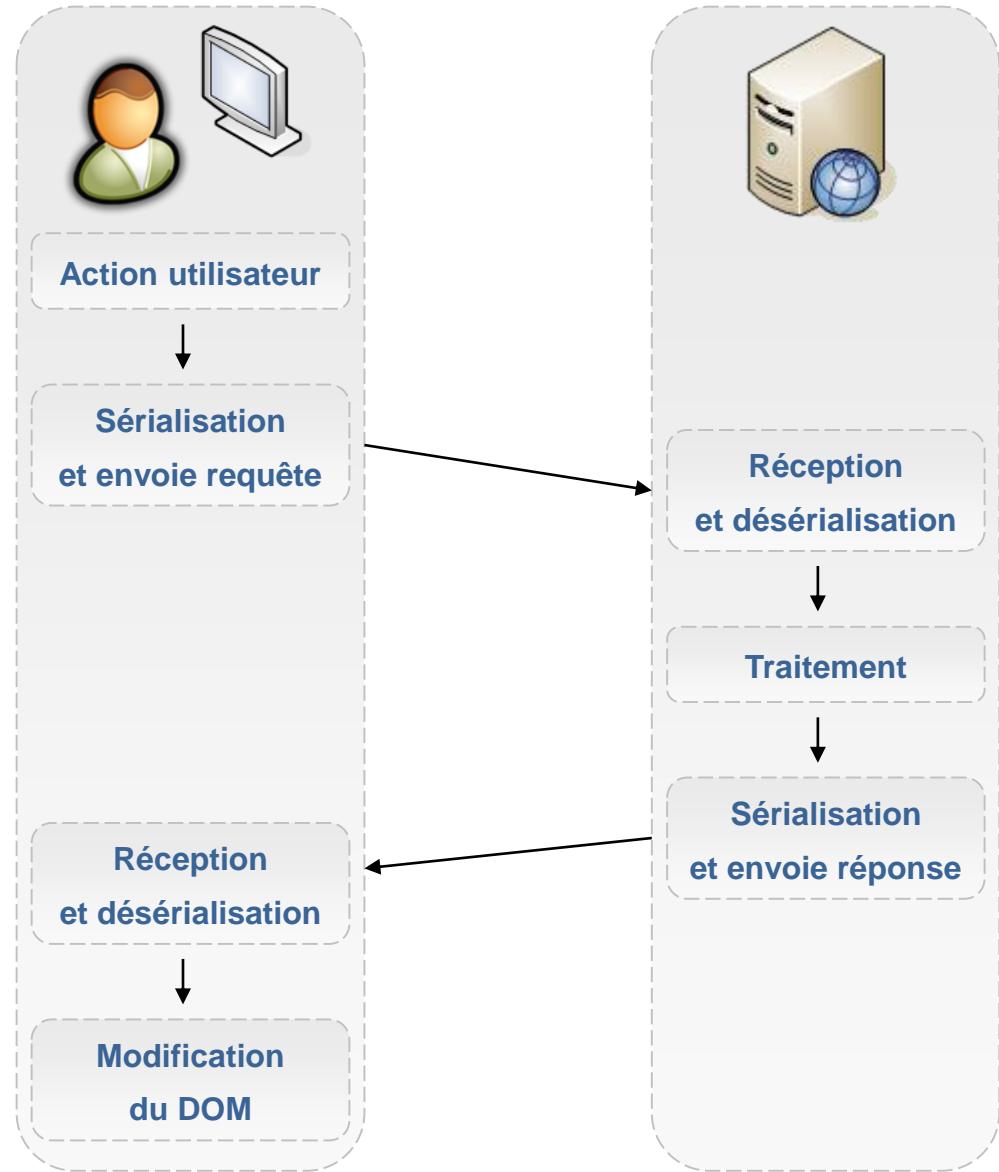
```
...  
<a href="#" id="BtnSaveUserInfo"  
    class="globalButton gbOrangeButton"  
    onclick="trackPageUpdate();return  
    ViewEditAjaxForm_BecomeMember_CreateAcc  
    ount.save()">S'inscrire<span></span></a  
...  
...
```



Les principes de fonctionnement d'AJAX

- AJAX permet d'effectuer des appels depuis le client vers le serveur afin

- De modifier l'écran côté client (message de validation, mise à jour d'un formulaire, ...) -> envoie d'une requête **GET** et modification du **DOM** (cf. Cours N°2)
- Stocker de l'information côté serveur (sauvegarde du brouillon sous GMail par exemple) -> envoie d'une requête **POST**



L'objet XMLHttpRequest a permis de démocratiser la communication client-serveur

- Le but de XMLHttpRequest est de pouvoir échanger des données entre le client et le serveur au format XML
- L'idée originale vient de ... Internet Explorer 5.0 (en 1998)
 - Puis il a été repris au fur et à mesure, entre autre par Mozilla 4 ans plus tard
 - Aujourd'hui encore, il persiste de nombreuses incompatibilités entre navigateurs (en particulier pour créer l'objet)
- L'API est décrite à : <http://www.w3.org/TR/XMLHttpRequest/>
- XMLHttpRequest sert de base aux frameworks AJAX
 - Et ceci, même si le protocole utilisé n'est pas du XML « pur » (cf. JSON)

```
function createXhrObject() {  
    if (window.XMLHttpRequest) {  
        return new XMLHttpRequest();  
    } else if (window.ActiveXObject) {  
        var names = ["Msxml2.XMLHTTP.6.0",  
                    "Msxml2.XMLHTTP.3.0",  
                    "Msxml2.XMLHTTP",  
                    "Microsoft.XMLHTTP"];  
        for(var i in names) {  
            try{ return new  
ActiveXObject(names[i]); }  
            catch(e) {}  
        }  
    }  
    return null; // non supporté  
}
```

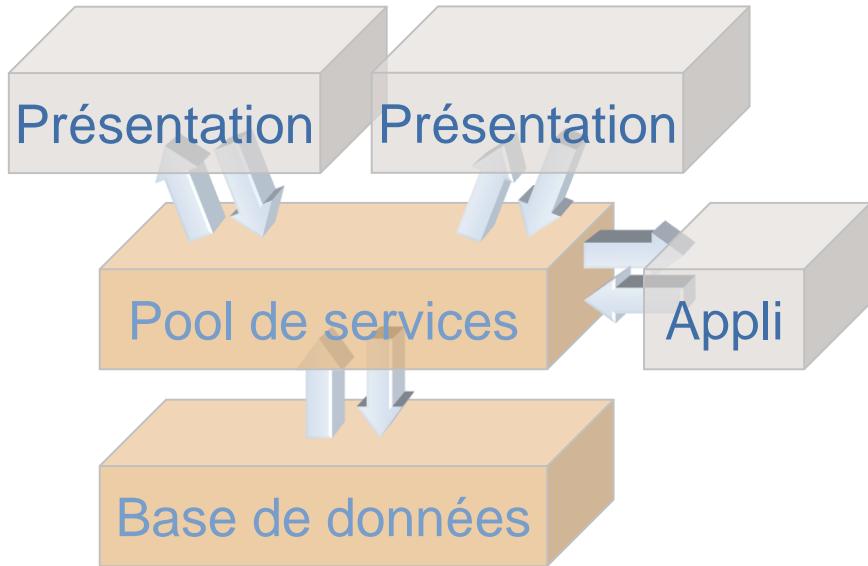
```
var request = createXhrObject();  
request.open('get', '/action', false);  
request.send();  
request.onreadystatechange = function () {  
    if (request.readyState == 4) {  
        if (request.status == 200) {  
            var xml = request.responseXML;  
        } else {  
            alert( "There was a problem with  
the URL." );  
        }  
        http_request = null;  
    }  
};
```

Sommaire

- Client-serveur
 - Mode de fonctionnement
- Le protocole de communication HTTP
 - Historique
 - Utilisation dans le cadre d'un site Web
- Focus sur HTML
 - Extension avec AJAX
- Les Web Services
 - SOAP et REST

Les Web services augmentent l'interopérabilité

- L'atout principal dans l'utilisation des WS est de pouvoir décorrérer le **quoi du comment**
 - On offre une **interface** qui « répond à un besoin »
 - On traite de façon opaque **l'implémentation** pour adresser la demande
- L'utilisation des WS a introduit des nouvelles manières de concevoir
 - En particulier, la **SOA** (Services Oriented Architecture) est apparue (et est très à la mode !)
 - Le but ultime est que toutes les couches d'une application se « **rendent services** » entre elles via un **protocole unique**



ATOUTS

- Les services (la logique « métier ») ne sont pas dupliqués
- On peut changer l'implémentation des couches (changer de techno, faire évoluer un framework) sans casser le système global
- Quelque soit le canal, on s'assure de pointer toujours vers le même code

Les différents types de Web services

- Un Web services est un service accessible sur le Web...
 - Attention tout de même, par abus de langage, « Web services » est souvent synonyme de SOAP
- Les 2 types de WS les plus répandus sont :
 - REST (Representational State Transfer)
 - SOAP (Simple Object Access Protocol)

Points communs

- Ils s'appuient tous les 2 sur le protocole HTTP

REST

- Simple d'utilisation (appel via une URL paramétrée) ; com. navigateur <-> serveur possible
- Souple à utiliser mais interface imprécise et plus limitée que SOAP (transfert « d'objets » complexe)
- *Le Web s'appuie finalement sur des WS REST*

SOAP

- L'utilisation de SOAP nécessite un client particulier ; com. serveur <-> serveur obligatoire
- La syntaxe doit suivre schéma XSD (certaines contraintes peuvent être spécifiées grammaticalement)
- Le WSDL (Web Services Description Language) permet une auto-déclaration via le contrat de service

Focus sur les Web services REST

- Un WS REST est un point d'accès offert par un serveur
 - L'accès se fait via une URL que l'on peut paramétrer
 - Le format de réponse est à décider entre le client et le serveur
 - Il n'est pas obligatoire d'utiliser un contrat de service

■ Exemple d'utilisation d'un service de flux RSS :

<http://feeds.feedburner.com/eclipselive?format=xml>
<http://feeds.feedburner.com/eclipse/fnews?format=xml>
<http://feeds.feedburner.com/eclipse/fnews?format=html>

Current Feed Content

Global Eclipse Training Series

Posted: Tue, 08 Sep 2009 09:30:00 EST

The Eclipse Foundation and Eclipse member companies series. The training is an excellent opportunity for Eclipse Rich Client Platform (RCP), Equinox & OSGie sessions, providing practical experience through scheduled in 23 different cities around the world.

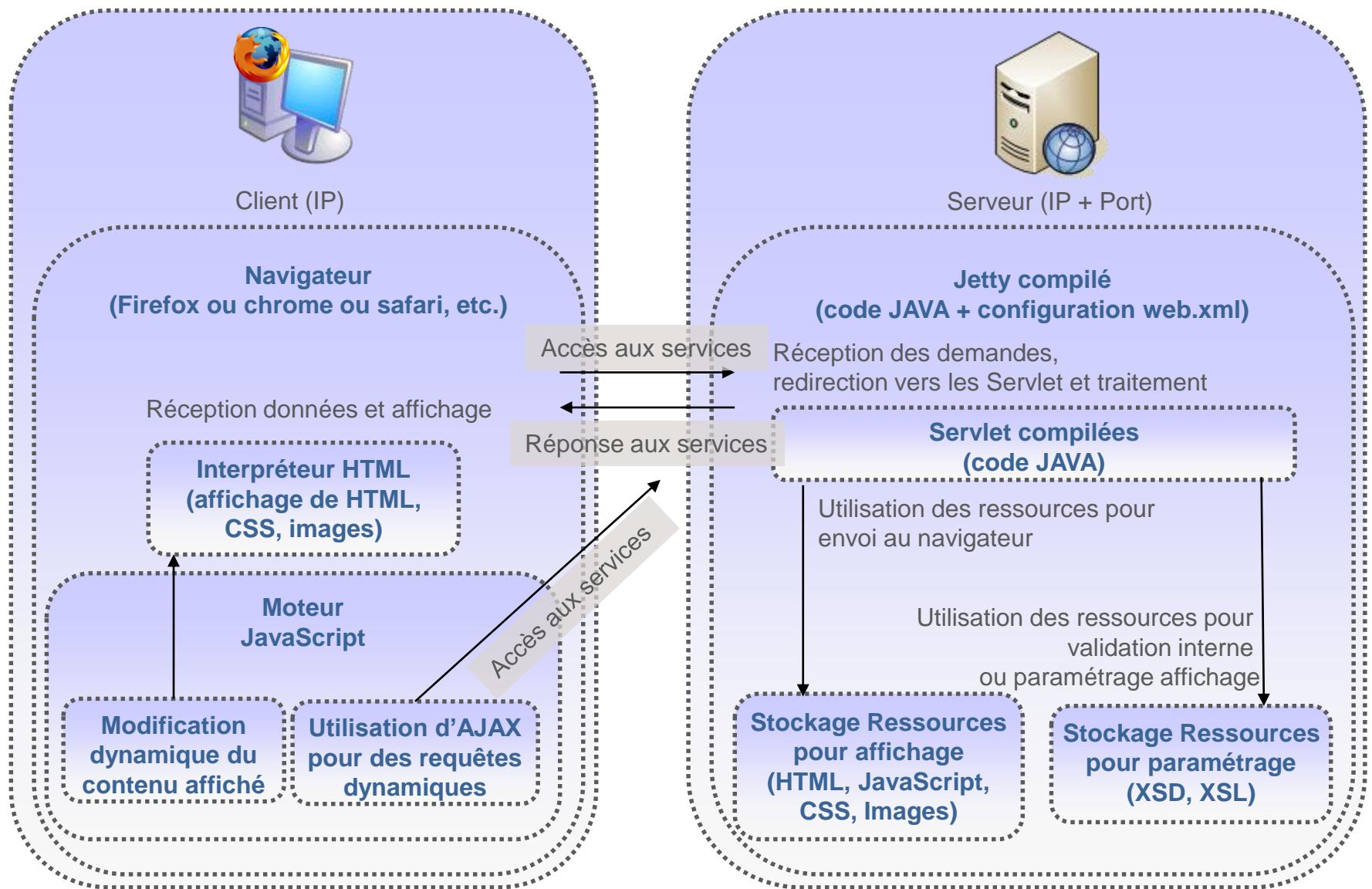
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" media="screen" href="/~d/stylesheets/rss.xsl"?>
<channel>
  <title>Eclipse News</title>
  <link>http://www.eclipse.org</link>
  <description>Eclipse News</description>
  <image>
    <url>http://www.eclipse.org/images/EclipseBannerPic.jpg</url>
    <title>eclipse.org</title>
    <link>http://www.eclipse.org</link>
  </image>
  <atom10:link xmlns:atom10="http://www.w3.org/2005/Atom">
    <title><![CDATA[Global Eclipse Training Series]]></title>
    <link>http://feedproxy.google.com/~r/eclipse/fnews</link>
    <description>&lt;p&gt;
      The Eclipse Foundation and Eclipse member companies series. The training is an excellent opportunity for Eclipse Rich Client Platform (RCP), Equinox & OSGie sessions, providing practical experience through scheduled in 23 different cities around the world.
    &lt;/p&gt;
  </atom10:link>
</channel>
```

Le HTML est le format choisi pour communiquer entre un navigateur et un serveur Web

Synthèse

- **Le mode de fonctionnement client-serveur permet de décorreler**
 - Le client qui bénéficie du service
 - Le serveur qui rend le service
- **Ce mode s'est développé énormément**
 - Avec la création de sites Web
 - Et poursuit sa progression avec les Web Services
- **Avec des demandes ergonomiques toujours plus exigeantes et l'introduction du Web 2.0, l'utilisation d'AJAX s'est démocratisée**
 - En partant de l'objet XMLHttpRequest
 - Et en aboutissant (pour le moment) à JSON
- **Les architectures des nouveaux Systèmes d'informations sont basées sur la SOA**
 - Pour augmenter l'évolutivité et la maintenabilité

Schéma d'architecture pour les TDs



Q&R

Questions

Réponses

Sommaire – Cours 2

Le langage XML et l'API DOM

■ XML

- Objectif
- Historique

■ Le langage XML

- Syntaxe
- Structure arborescente

■ Ecriture d'un document XML

- Le XML « bien formé »

■ Parcours d'un document XML

- Focus sur DOM
- Parenthèse sur SAX

L'objectif de XML : séparer le fond de la forme

■ HTML porte à la fois

- Le fond : la donnée brute que l'on veut transmettre
- La forme : le formatage / la mise en forme de la donnée

■ XML a pour objectif de transmettre uniquement la donnée « brute » et de laisser au client le soin de la mettre en forme

■ Sur cet exemple :

- La même information est mise en forme de 2 manières (taille de police, couleur, longueur du texte, liens)
- Il n'y a qu'une seule source de données ; www.usinenouvelle.com qui s'affiche dans 2 environnements (www.usinenouvelle.com et google.com)

usinenouvelle.com

Accueil > Technos et Innovations

HTC attaque Apple en justice pour la 3ème fois

Le 17 août 2011 par Barbara Leblanc

▶ Apple



© DR

Les plaintes n'en finissent pas à l'encontre du géant américain. Après son conflit avec Samsung, c'est au tour du fabricant taiwanais de smartphones de l'accuser de violation de brevets

HTC annonce le 16 août qu'il porte plainte contre Apple devant devant la Commission américaine du Commerce international (ITC) et devant un tribunal du Delaware. En cause ? Il accuse le fabricant américain d'avoir enfreint ses droits de propriété intellectuelle liés à trois brevets. Les produits visés sont les iPods, iPhone et les iPads.

L'objectif de ce dépôt de plainte est de "protéger la propriété intellectuelle de HTC, ses partenaires industriels et le plus important, ses clients", précise le groupe. Le taiwanais réclame le versement de dommages et l'interdiction pour l'américain d'importer aux Etats-Unis tout matériel violent les brevets.

Mais les deux géants n'en sont pas à leur premier conflit. HTC a déjà porté plainte contre Apple devant l'ITC à deux reprises.

[ARTICLES LIÉS](#)
Apple et Samsung en conflit aux Pays-Bas aussi
Apple fait interdire une tablette Samsung en Europe

[LIENS SPONSORISÉS](#)

HTC attaque Apple en justice pour la 3ème fois - L'Usine Nouvelle



Les plaintes n'en finissent pas à l'encontre du géant américain. Après son conflit avec Samsung, c'est au tou... [Afficher d'autres articles](#)

L'U... Paramètres

news.google.com

Exemple de différence d'objectif entre l'utilisation de XML et HTML

```
...
<item>
    <title><![CDATA[HTC attaque Apple en justice pour la 3ème fois]]></title>
    <link>http://www.usinenouvelle.com/article/htc-attaque-apple-en-justice-pour-la-3eme-fois.N157110?xtor=RSS-215</link>
    <guid>http://www.usinenouvelle.com/article/htc-attaque-apple-en-justice-pour-la-3eme-fois.N157110?xtor=RSS-215</guid>
    <description><![CDATA[Les plaintes n'en finissent pas à l'encontre du géant américain. Après son conflit avec Samsung, c'est au tour du fabricant taiwanais de smartphones de l'accuser de violation de brevets <a href="http://www.usinenouvelle.com/article/htc-attaque-apple-en-justice-pour-la-3eme-fois.N157110?xtor=RSS-215">Lire l'article</a> ]]></description>
    <enclosure url="http://www.usinenouvelle.com/mediatheque/2/1/2/000142212_3.jpg?xtor=RSS-215" length="9776" type="image/jpeg" />
    <pubDate>Wed, 17 Aug 2011 08:43:00 +0200</pubDate>
</item>
...
```

Source des données :
au format XML

Page HTML sur le site
news.google.com

```
...
<h1 class="articleTitre">HTC attaque Apple en justice pour la 3ème fois</h1>
<span class="regular11px color909090">Le 17 août 2011 par Barbara Leblanc
</span>
...
<p class="ArticleChapeau">Les plaintes n'en finissent pas à l'encontre du géant américain. Après son conflit avec Samsung, c'est au tour du fabricant taiwanais de smartphones de l'accuser de violation de brevets</p>
<p>HTC annonce le 16 août qu'il porte plainte contre Apple devant devant la Commission américaine du Commerce international (ITC) et devant un tribunal du Delaware. En cause ? Il accuse le fabricant américain d'avoir enfreint ses droits de propriété intellectuelle liés à trois brevets. Les produits visés sont les iPods, iPhone et les iPads.</p>
<p>L'objectif de ce dépôt de plainte est de <em>"protéger la propriété intellectuelle de HTC, ses partenaires industriels et le plus important, ses clients"</em>, précise le groupe. Le taiwanais réclame le versement de dommages et l'interdiction pour l'américain d'importer aux Etats-Unis tout matériel violent les brevets.</p>
<p>Mais les deux géants n'en sont pas à leur premier conflit. HTC a déjà porté plainte contre Apple devant l'ITC à deux reprises.</p>
</div>
...
```

Page HTML sur le site
usinenouvelle.com

```
...
<h2 class="title">
    <a target="_blank" class="usg-AFQjCNEhh3EA5-eK6EyvwWIXJhwiawnveA sig2-V1G3MKn91G9NhZW7N4h7oA did-def58cdfa2bb2326 article"
        href=
        "http://www.usinenouvelle.com/article/htc-attaque-apple-en-justice-pour-la-3eme-fois.N157110"
        id="MAA4AEgGUABgAWoCZnJ6AXQ" name="MAA4AEgGUABgAWoCZnJ6AXQ"><span class="titletext">HTC attaque Apple en justice pour la 3ème fois</span></a>
    </h2>
    <div class="sub-title">
        <span class="source source-pref sid-753372">L'Usine Nouvelle</span> -
        <span class="date">Il y a 1 heure</span>
    </div>
    <div class="body">
        <div class="snippet">
            Les plaintes n'en finissent pas à l'encontre du géant américain. Après son conflit avec Samsung, c'est au tour du fabricant taiwanais de smartphones de l'accuser de violation de brevets HTC annonce le 16 août qu'il porte plainte contre Apple devant <b>...</b>
        </div>
    </div>
    ...

```

Ce qu'apporte XML par rapport à HTML

- Permettre l'échange de données sans pré-formatage
- En particulier, ne transmettre que les données « **utiles** » (pas de bruit lié au formatage, HTML par exemple)
- En cas de changement de présentation (mise en forme), la donnée reste la même et l'échange n'est pas impacté

- Remarque : HTML se base sur XML (c'est un langage à balises)
 - Sa syntaxe comporte à la fois des éléments permettant d'afficher de la donnée et de la mettre en forme

Historique

- La premier langage dont le but était de séparer les données de leur présentation est SGML
 - Il a été initié par IBM en 1979,
 - Puis normalisé en ISO 8879 en 1996
- XML est apparu en 1998 sous le crédo :
 - « Son but est de permettre au SGML générique d'être transmis, reçu et traité sur le web de la même manière que l'est HTML aujourd'hui. »
 - La version 1.1 est sortie en 2004
 - Info sur : <http://www.w3.org/XML/>
- XML est un dialecte simplifié de SGML, plus strict, orienté web

Source : http://fr.wikipedia.org/wiki/Extensible_markup_language



Les objectifs de XML

- Utilisation **immédiate sur internet**
- Compatible avec **SGML**
 - Format texte
 - Sémantique et structure des données
 - Séparation entre les données et leur présentation
- Analyse syntaxique et programmation aisées
- Documents lisibles et **faciles à écrire pour un être humain**
- Recommandations formelles et concises

Les (nombreuses) utilisations de XML

- **Scalable Vector Graphics (SVG)**
 - Dessin vectoriel (inkscape, etc)
- **Mathematical Markup Language (MathML)**
 - Expressions mathématiques (maple, mathematica, firefox, etc...)
- **User interface markup language (XUL, XAML, etc)**
 - Description d'interfaces utilisateur (XUL) + comportement du programme (XAML), etc.
- **Really Simple Syndication (RSS 2.0)**
 - Publication de contenu (<http://www.rssboard.org/rss-specification>)
- **Au-delà de ces « standards »**
 - L'utilisation de XML est courante dans le monde informatique pour configurer des applications, échanger des données entre client-serveur, ...
 - Chaque développeur décline le XML pour coller à ses besoins

Sommaire – Cours 2

Le langage XML et l'API DOM

- XML
 - Objectif
 - Historique
- Le langage XML
 - Syntaxe
 - Structure arborescente
- Ecriture d'un document XML
 - Le XML « bien formé »
- Parcours d'un document XML
 - Focus sur DOM
 - Parenthèse sur SAX

Exemple de syntaxe, basé sur SVG

- Une ellipse blanche de centre (220, 100) et de rayons (190, 20) à l'intérieur d'une ellipse jaune de centre (240, 100) et de rayons (220, 30)

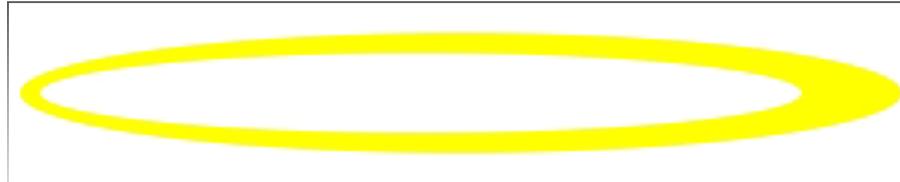


Traduction selon la
syntaxe SVG

```
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org /2000/svg">  
    <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:yellow"/>  
    <ellipse cx="220" cy="100" rx="190" ry="20" style="fill:white"/>  
</svg>
```



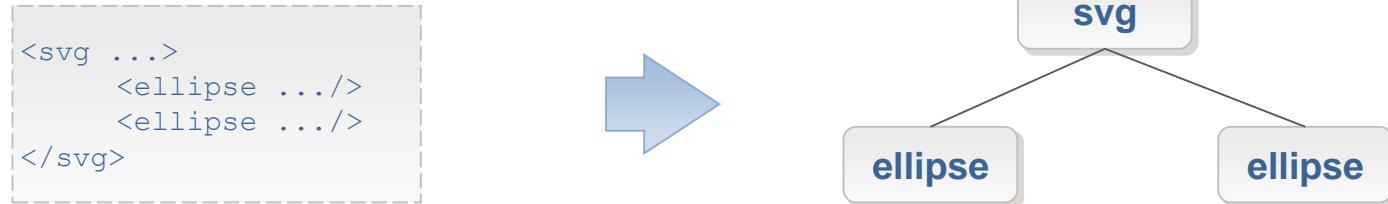
Résultat obtenu par
un interpréteur SVG



- Il est important de noter que :
 - Structurellement, une ellipse est toujours composée des coordonnées de son centre et de ses 2 rayons
 - Plusieurs ellipses différentes peuvent être déclinées à partir de cette structure

Un document XML est un arbre

- Un document XML a une structure arborescente avec 1 racine
- Chaque ramification ou feuille est un nœud
 - On appelle « Elément » un bloc de texte compris entre une balise ouvrante et une balise fermante de même nom ou vide
 - Un Elément est un nœud
- Ce sont les balises « < » et « > » qui permettent de découper le document XML en nœuds
- Par exemple :
 - Arbre dont le nœud racine est « svg » et qui a 2 Eléments feuilles « ellipse »



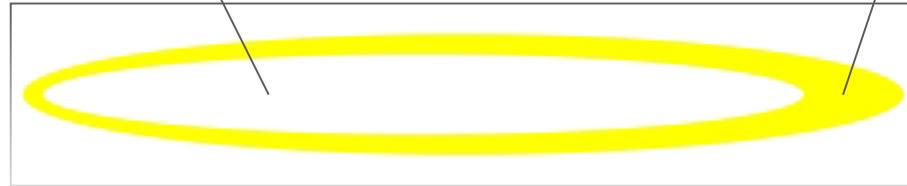
Un Elément comporte des attributs

- Pour caractériser un Elément, des attributs peuvent être utilisés
 - C'est un couple « nom » / « valeur » qui permet de préciser l'Elément qui le porte

```
...  
<ellipse cx="220" cy="100" rx="190" ry="20" style="fill:white"/>  
...
```

≠

```
...  
<ellipse cx="240" cy="100" rx="220" ry="30" style="fill:yellow"/>  
...
```



Un Elément est caractérisé par ses attributs et sa sous-arborescence

- Le canvas graphique est caractérisé par les 2 ellipses qui le composent
- Chaque ellipse est caractérisée par ses attributs

```
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org /2000/svg">
  <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:yellow"/>
  <ellipse cx="220" cy="100" rx="190" ry="20" style="fill:white"/>
</svg>
```

- Finalement, une caractérisation des ellipses pourrait se traduire comme suit :

```
<ellipse>
  <cx>240</cx>
  <cy>100</cy>
  <rx>220</rx>
  <ry>30</ry>
  <style>fill:yellow</style>
</ellipse>
```

- Le choix entre attributs ou sous-éléments dépend des cas :
 - Les sous-éléments sont plus évolutifs (un sous-arbre peut être créé)
 - Les attributs ne sont pas structurés mais sont moins verbeux

La section CDATA – pour du texte non analysé

- La section CDATA `<![CDATA ...]>` permet d'insérer du texte « brut »
 - Qui ne sera pas analysé lors du parsing du document XML
 - Qui peut contenir des caractères « interdits », par exemple « < » ou « > »
 - Qui peut éventuellement contenir du texte XML mais qui ne doit pas être considéré comme faisant partie de l'arbre du document XML principal

```
<article id="123">
    <title>Le Web, c'est super.</title>
    <descriptionHTML>
        <![ CDATA[<h1>Web et XML : le bon ménage</h1><p>Vous pouvez trouver toutes
              les infos <a href="http://uuu.enseirb.fr/~herbrete/IF205/">ici</a></p>]]>
    </descriptionHTML>
</article>
```

- Il faut bien noter la différence entre ces 2 expressions:
 - `<code><![CDATA[if (x < 30) x=x+1;]]></code>`
 - `<![CDATA[<code>if (x < 30) x=x+1; </code>]]>`
- Dans le 2^{ème} cas, la balise `<code>` n'appartient pas à l'arborescence du document XML

Ajout de commentaires dans le document XML

- **Comme dans un langage de programmation, l'utilisation de commentaires est recommandée**
 - Cela permet par exemple de justifier le choix de certaines valeurs d'attributs ou d'indiquer à quoi correspond tel nœud
 - Cela s'effectue comme en HTML : <!-- commentaire -->

```
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org /2000/svg">
    <!-- cette ellipse sera celle placée au fond de l'image -->
    <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:yellow"/>
    <!-- l'ellipse blanche est légèrement décalée par rapport à la jaune -->
    <ellipse cx="220" cy="100" rx="190" ry="20" style="fill:white"/>
</svg>
```

- **Attention, l'utilisation de commentaires n'est pas possible au sein d'une balise, à côté des attributs**
- **L'utilisation des commentaires permet « d'empêcher » la prise en compte d'une portion de XML (comme en langage de programmation) :**

```
<!-- l'ellipse jaune est temporairement désactivée
    <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:yellow"/>
-->
```

Sommaire – Cours 2

Le langage XML et l'API DOM

- XML
 - Objectif
 - Historique
- Le langage XML
 - Syntaxe
 - Structure arborescente
- Ecriture d'un document XML
 - Le XML « bien formé »
- Parcours d'un document XML
 - Focus sur DOM
 - Parenthèse sur SAX

XML : de la souplesse mais de la rigueur !

- XML est un langage extrêmement souple et les choix suivants peuvent être faits :
 - Le nom de balises
 - Le fait que tel élément doit avoir 0, 1 ou plus de fils
 - Les attributs d'un élément
 - ...
- Mais cette souplesse est permise car un cadre rigoureux encadre la « bonne formation » d'un document XML
 - Si un document XML est mal formé, il ne pourra pas être analysé par un interpréteur : les données seront illisibles !
- Un certains nombres de règles doivent ainsi être respectées
 - Elles sont énoncées ci-après

Utilisation des caractères de l'encodage spécifié

- L'encodage désigne l'ensemble des caractères interprétables et la façon dont ils sont codés (nombre de bits entre autre)
- Par défaut, pour le XML, c'est l'UTF-8 qui est utilisé
 - D'autres encodages peuvent être utilisés
 - <http://www.iana.org/assignments/character-sets>
- Pour spécifier un autre encodage, il faut utiliser le « Prologue »
 - Ce prologue n'est pas « obligatoire » mais fortement recommandé dans tous les cas
 - De plus, il permet de définir d'autres caractéristiques du document XML (cf. cours N° 3)

Code	Correspondant
US-ASCII	ASCII (7 bits)
UTF-8	Compressed Unicode (1-3 octets)
UTF-16	UCS ISO-10646 (2 octets)
ISO-8859-1	Latin-1, ASCII + accents
ISO-8859-6	ASCII + Arabic
GB2312	ASCII + Kanji
ISO-2022-CN	Caractères chinois

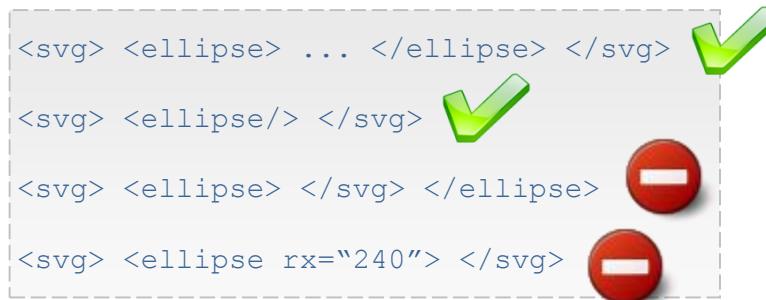
```
<?xml version="1.0" encoding="utf-8"?>
<svg>
    ...
</svg>
```

Règles de nommage des balises et des attributs

- Les caractères autorisés sont
 - Les lettres [a-zA-Z]
 - Les chiffres [0-9]
 - Les caractères : « _ », « - », « : » et « . »
- Le nom d'une doit commencer avec une lettre, « _ » ou « : »
 - Les noms qui commencent avec (x|X)(m|M)(l|L) sont réservés
 - Le symbole « : » a un but lié aux espaces de noms (cf. Cours N°3)
- Les noms sont sensibles à la casse
 - <svg> est différent de <Svg>
 - De même que <svg id="123"/> est différent de <svg ID="123"/>

Règles concernant les balises

- Toute balise ouverte doit être fermée et l'enchaînement doit suivre un arbre strict (pas d'enchâssement)



- Un document XML ne doit avoir qu'une seule racine

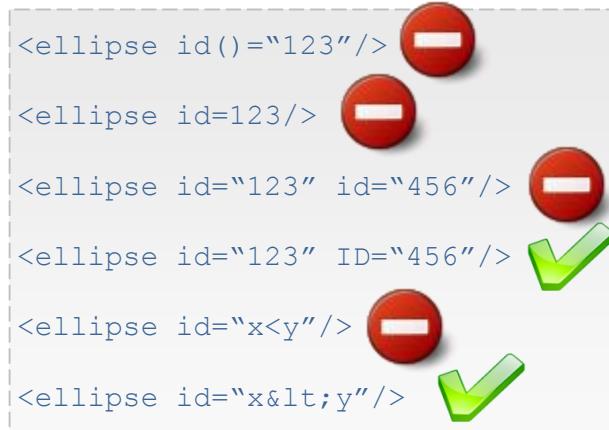


Règles concernant les attributs

- La valeur d'un attribut doit toujours être indiquée entre guillemets « " » ou « ' »
- Un attribut ne peut avoir qu'une occurrence dans une balise

- Tout caractère de la syntaxe doit être échappé
 - Sauf s'ils sont contenus dans une section CDATA

Caractère	Remplacé par
ampersand (&)	&
apostrophe (')	'
quotation ("")	"
greater-than (>)	>
less-than (<)	<



Règles concernant les commentaires et CDATA

- Les commentaires et les sections CDATA ne peuvent pas
 - Etre enchaînés
 - Etre à l'intérieur de balise
- Pour les commentaires, la chaîne « -- » est interdite

```
<ellipse id="123" <!-- id de l'ellipse jaune --> />   
<ellipse id="123"/> <!-- ellipse jaune <!-- celle qui est sous la blanche --> -->   
<ellipse id="123"/> <!-- id à décrémenter 2 fois (--) -->   
<!-- ellipse jaune --><ellipse ... /><!-- puis la blanche -->   
<![CDATA[Ne pas interpréter : <![CDATA[<h1>Title</h1>]] ; sinon problèmes !]] 
```

Sommaire – Cours 2

Le langage XML et l'API DOM

- XML
 - Objectif
 - Historique
- Le langage XML
 - Syntaxe
 - Structure arborescente
- Ecriture d'un document XML
 - Le XML « bien formé »
- Parcours d'un document XML
 - Focus sur DOM
 - Parenthèse sur SAX

L'arbre DOM se construit à partir du flux XML

- Ce sont les caractères « < » (délimitation des balises), « " » (valeur d'un attribut) qui permettent à un interpréteur de construire l'arbre XML à partir du flux
 - Le caractère « & » est aussi utile pour les Entités (non traité ici)
- Mais un flux XML contient aussi des caractères de mise en forme (retours à la ligne, espaces, ...)
 - Ils n'interviennent pas dans la hiérarchie des nœuds
 - Mais ils sont tout de même pris en compte dans l'arbre DOM

- Chaque nœud de l'arbre DOM est représenté par un triplet (type, nom, valeur) :

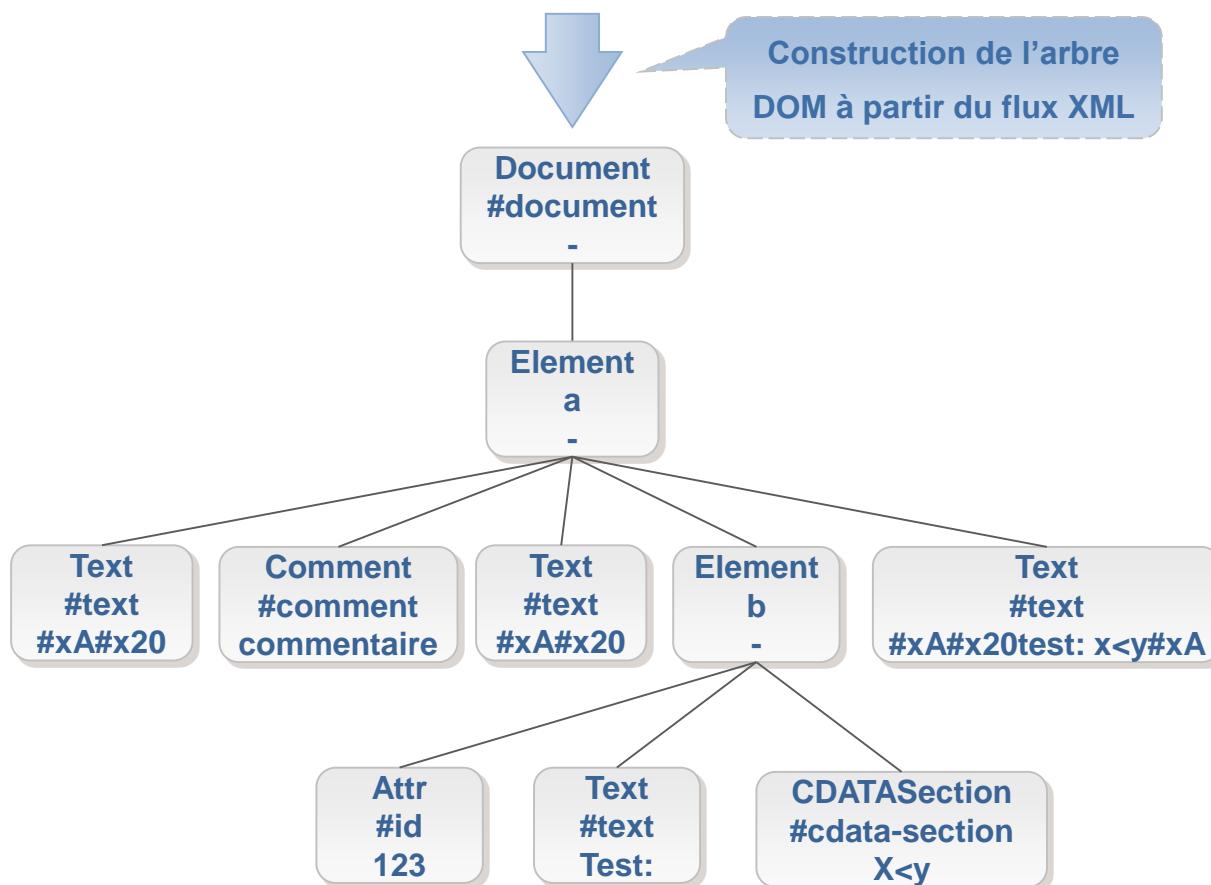
Type
Nom
Valeur

Type	Nom	Valeur
Document	#document	-
Element	nom de la balise	-
CDATASection	#cdata-section	texte de la section
Attr	nom de l'attribut	valeur de l'attribut
Text	#text	texte
Comment	#comment	texte du commentaire
ProcessingInstruction	cible	paramètres

- Remarque : l'arbre DOM ne représente pas le Prologue (car celui-ci a pour but d'indiquer comment interpréter le flux lui-même)

Exemple d'arbre DOM

```
<?xml version ="1.0"?>
<a>
    <!-- commentaire -->
    <b id="123">test: <! [CDATA[ x<y ] </b>
    test: x &lt; y
</a>
```



L'API DOM

- **DOM signifie : Document Object Model**
 - Cela permet de voir chaque nœud d'un document XML comme un objet ayant des propriétés (attributs) et des enfants (Element)
- **Cet « API » permet, à partir d'un flux XML, de créer un arbre complet le représentant**
 - En cas de « gros flux » XML, l'utilisation de SAX est recommandée (cf. ci-après)
- **L'API DOM permet**
 - Naviguer dans l'arbre (parcours des fils, récupération des valeurs des attributs, ...)
 - Modifier l'arbre (ajout d'éléments, modification de valeurs, ...)
- **(Ce cours ne se consacre pas à comprendre comment « fonctionne » DOM mais uniquement à comprendre comment « utiliser » l'API DOM)**

Les classes et méthodes les plus utiles

■ Utilisation de l'objet « Element » (spécialisation de l'interface Node) et des ses méthodes

■ Parcours / navigation

- `getElementById(x)`, retourne le nœud dont l'attribut id vaut x
- `getElementsByName(x)`, retourne une liste de nœuds dont le nom (de balise) est x
- `<node>.firstChild`, renvoie le premier enfant du nœud `<node>`
- `<node>.nextSibling`, renvoie le « frère » suivant du nœud `<node>`
- `<node>.getAttribute(x)`, retourne la valeur de l'attribut x du nœud `<node>`

■ Crédit / Modification

- `createElement(type, name)`, crée un nouveau nœud du type et du nom indiqués
- `<node>.appendChild(<child>)`, ajoute un nœud `<child>` au nœud `<node>` en tant que dernier fils
- `<node>.name = value`, affecte la valeur de l'attribut `<name>` à la valeur `<value>` sur le nœud `<node>`

Exemple d'utilisation en Dynamic HTML

```
<html>
<head>
    <title>This is dynamic HTML !</title>
    <script type="application/javascript">
        function red() {
            document.getElementById("titre").style.color = "red";
        }
    </script>
</head>
<body>
    <h1 id="titre">This is dynamic HTML !</h1 >
    <button type="button" onclick="red()">Passer en rouge</button>
</body>
</html>
```



This is dynamic HTML !

[Passer en rouge](#)

Parenthèse sur SAX (Simple Api for XML)

- Une 2^{ème} manière d'interpréter un flux XML est SAX
- C'est une programmation événementielle qui exécute un « handler » quand
 - Le document commence
 - Une balise s'ouvre
 - Il y a des espaces et/ou des caractères
 - ...
 - Une balise se ferme
 - Le document se ferme
- Ainsi, l'interpréteur lit le flux XML « au fil de l'eau » et ne traite que le strict minimum dont il a besoin
 - Cela permet d'économiser de la mémoire vive (par le stockage de l'arbre DOM)
 - Mais cela peut ajouter des difficultés si le XML contient plusieurs parties qui se font références entre elles

Q&R

Questions

Réponses

Sommaire – Cours 3

La validation des documents XML

- Pourquoi valider des documents XML ?
 - Le contrat d'interface
 - Le contrat de service (SOAP vs REST)
- DTD – Document Type Definition
- XSD – XML Schema Definition
- Relax NG – Regular LAnguage for XML Next Generation
- Utilisation de la validation XML
 - Mise en œuvre
 - Application

Pourquoi décrire des documents XML

- **Usuellement, les documents XML ont vocation à être échangés**
 - Un des buts de XML est de structurer les données de façon à les rendre plus facilement interprétables
 - Encore faut-il savoir comment lire ces données...
- **Afin de pouvoir échanger des données, il faut se mettre d'accord sur leur format**
 - Comme c'est le cas en programmation avec la signature d'une méthode
 - On définit son nom, ses paramètres en entrée et son retour
- **L'objectif des DTD, XSD et RelaxNG est donc le même : permettre de décrire les données qui transitent**
 - Cela permet au consommateur (le client) de savoir comment sont structurées les données et ainsi savoir comment parcourir le document XML afin de récupérer ce qui l'intéresse
- **La validation XML permet alors de s'assurer qu'un document XML est « valide » (respecte une description) afin de pouvoir le parcourir**

La validation XML, prérequis à tout doc XML ?

- **Un cheminement logique serait alors de toujours décrire un document XML et**
 - Publier le contrat d'interface
 - Inciter le client à l'utiliser pour valider le doc XML
- **Cela a du sens pour**
 - Des échanges de flux XML via le Web
 - Ainsi, les parties client et serveur sont synchronisées sur la structure des données qu'elles échangent
 - Mais cela est moins utile pour un fichier de configuration interne à une application par exemple
 - Car c'est souvent le même développeur qui écrit le fichier XML et le parseur associé (et qui les maintient en parallèle)
- **La rédaction d'un DTD, XSD ou RelaxNG a un coût (lors de la création mais surtout lors de la maintenance) et il ne faut pas généraliser son utilisation**
 - Tout en gardant à l'esprit que c'est **un moyen rigoureux d'exposer des données**

Le contrat de service d'un WebService SOAP

- L'un des atouts majeurs des WebServices SOAP est que
 - Les données sont structurées (via un XSD)
- Ainsi, quand un WS SOAP est exposé, le consommateur (client) connaît immédiatement la structuration du flux XML qu'il recevra
 - Via le [WSDL](#) : Web Service Description Langage
- Les détracteurs de SOAP (qui prônent REST) pensent que cela apporte surtout de la lourdeur
 - Le XSD SOAP est très verbeux
 - Néanmoins, des XSD pour REST voient le jour !
- Le bon compromis semble alors être de s'orienter vers REST (pour sa simplicité de mise en œuvre) et de définir un [contrat de service en XSD](#)
 - Le [WADL](#) : Web Application Description Langage

Sommaire – Cours 3

La validation des documents XML

- Pourquoi valider des documents XML ?
 - Le contrat d'interface
 - Le contrat de service (SOAP vs REST)
- DTD – Document Type Definition
- XSD – XML Schema Definition
- Relax NG – Regular LAnguage for XML Next Generation
- Utilisation de la validation XML
 - Mise en œuvre
 - Application

DTD - Vue d'ensemble

- Une DTD permet de
 - Définir les éléments
 - Leur structure
 - Leur attributs
- La DTD provient du monde du SGML et est utilisée dans celui du HTML
- La DTD est définie par le W3C : <http://www.w3.org/TR/xml/>
 - Et c'est la seule manière reconnue par le W3C pour valider un document XML
- Aujourd'hui, les DTD sont peu utilisées car
 - Elles sont complexes à rédiger et à lire
 - Elles ne permettent pas de décrire certaines contraintes qu'XSD ou RelaxNG permettent

DTD – Exemples

■ La DTD suivante décrit

- Un document XML de racine « a » ayant autant de fils « b » vides que souhaité

```
<!DOCTYPE a [  
  <!ELEMENT a (b)*>  
  <!ELEMENT b EMPTY>  
]>
```

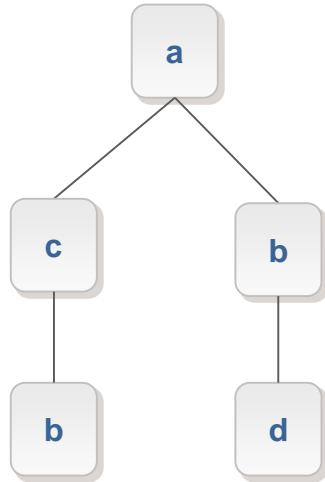
■ La DTD suivante décrit

- Un document XML de racine « a » pouvant avoir un attribut « x » et un fils « b » devant avoir un attribut « y » qui vaut 3 par défaut

```
<!DOCTYPE a [  
  <!ELEMENT a (b)*>  
  <!ATTLIST a x CDATA #IMPLIED>  
  <!ELEMENT b EMPTY>  
  <!ATTLIST b y CDATA "3">  
]>
```

DTD – Les insuffisances

- Une DTD ne se décrit pas en XML
 - Et impose une nouvelle syntaxe
- Il est impossible de typer les attributs
- Il est impossible de définir 2 éléments de même nom
 - Même s'ils sont situés à différentes profondeurs (et donc dans un contexte différent)



```
<!DOCTYPE a [  
<!ELEMENT a (c,b)>  
<!ELEMENT b (d)?>  
<!ELEMENT c (b)>  
<!ELEMENT d EMPTY>  
  <!ELEMENT b EMPTY>  
]>
```



On ne peut pas définir « b » plusieurs fois et donc, on ne peut pas le contraindre dans certains contextes et pas d'autres.
De plus, une DTD ne permet pas de définir un nœud en fonction d'un contexte

Sommaire – Cours 3

La validation des documents XML

- Pourquoi valider des documents XML ?
 - Le contrat d'interface
 - Le contrat de service (SOAP vs REST)
- DTD – Document Type Definition
- XSD – XML Schema Definition
- Relax NG – Regular LAnguage for XML Next Generation
- Utilisation de la validation XML
 - Mise en œuvre
 - Application

XSD – Vue d'ensemble

- Un XSD permet de
 - Définir les éléments
 - Définir les attributs
 - Déclarer (et utiliser) des types
 - Simples (string, integer, ...)
 - Complexes, description de sous-arbres
- XSD est un langage lui-même basé sur XML
 - Qui est lui-même défini par un schéma !
- XSD est défini par le w3c : <http://www.w3.org/XML/Schema>
 - Part 0, Primer : introduction/tutoriel (non normatif)
 - Part 1, Structures : cadre structurel du langage
 - Part 2, Datatypes : types de valeurs et définition de types
- Actuellement, XSD est la manière la plus répandue de décrire un document XML
 - Mais RelaxNG se démocratise
- (XSD sera le plus détaillé de ce cours)

XSD – xs:element

- L'objet « element » est le type de base permettant de décrire un « Element » au sens XML
- Un élément est dit simple s'il ne contient que du texte (qui peut être typé)
 - C'est-à-dire qu'il n'a pas d'attribut ni de fils
- Format : `<xs:element name="xxx" type="yyy"/>`
 - Une valeur par défaut peut être spécifiée et une valeur fixée peut être indiquée
- Exemple

Schéma XSD (partiel)

```
<xs:element name="login" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="isAdmin" type="xs:boolean"
    default="false"/>
```

Exemple XML

<login>bob</login>
<age>25</age>
<isAdmin>true</isAdmin>



<login>bob</login>
<age>vieux</age>
<isAdmin>vrai</isAdmin>



XSD – xs:attribute

- L'objet « attribute » permet de décrire un attribut d'une balise XML
 - xs:attribute permet donc d'enrichir xs:element en lui ajoutant la possibilité d'avoir des attributs
- Comme les éléments, un attribut est typé et peut avoir des valeurs par défaut ou fixées
- Format : <xs:attribute name="xxx" type="yyy"/>
- Exemple

Schéma XSD (partiel)

```
<xs:attribute name="login" type="xs:string"/>
<xs:attribute name="age" type="xs:integer"/>
<xs:attribute name="isAdmin" type="xs:boolean" default="false"/>
```

Exemple XML

```
<??? login="bob" age="25" isAdmin="true"/>
```



```
<??? login="bob" age="vieux" isAdmin="vrai"/>
```



XSD – Le typage

- L'ensemble des types « simples » est référencé :
<http://www.w3.org/TR/xmlschema-0/#CreatDt>
 - Plus de 40 types y sont normalisés
 - Les plus utilisés sont les suivants

Nom du type	Description
string	Chaîne de caractères
integer	..., -1, 0, 1, ...
nonNegativeInteger	0, 1, 2, ...
byte	-128, ..., -1, 0, 1, ..., 127
decimal	-1.23, 0, 34.67, 10001.01
float	-INF, 0, 12.78E-2, 13, NaN, ...
date	2007-12-10
language	en-US, fr, ...

- L'atout majeur de XSD est que de nouveaux types peuvent être déclarés (en se basant sur les types de base)
 - Par exemple : Client (nom, prénom, adresse(numéro, rue, CP, ville))

XSD – xs:restriction

- En plus de typer les éléments et les attributs, des restrictions peuvent être ajoutées, elles permettent par exemple
 - De réduire les valeurs admises à une plage de valeurs
 - D'obliger le choix parmi une énumération
 - Limiter la longueur d'une chaîne de caractères
 - ...
- Format : <xs:restriction base="type"> différents pattern selon le cas </xs:restriction>
- Exemple

```
<xs:element name="studentid">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="150"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="title">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Mr"/>
      <xs:enumeration value="Mme"/>
      <xs:enumeration value="Mlle"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XSD – Les éléments complexes

- A partir de **xs:element** et **xs:attribute** décrit ci-avant, XSD peut être étendu pour décrire un large panel de documents XML
 - En particulier, l'utilisation de **xs:complexType** et **xs:sequence** permet de décrire l'arborescence d'un document XML
- Il faut alors bien distinguer
 - La déclaration des types complexes
 - Qui permet de définir un type d'objet qui a un sous-arbre (sous-éléments et/ou attributs)
 - Leur utilisation

■ Exemple

Schéma XSD (partiel)

```
<xs:complexType name="user">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="login"/>
</xs:complexType>

<xs:element name="teacher" type="user"/>
<xs:element name="student" type="user"/>
```

Exemple XML

```
<teacher login="fherbre">
  <firstname>Frédéric</firstname>
  <lastname>Herbreteau</lastname>
</teacher>
<student login="max">
  <firstname>Max</firstname>
  <lastname>La Menace</lastname>
</student>
```

XSD – Exemples (1/2)

■ Le XSD suivant décrit

- Un document XML de racine « a » ayant autant de fils « b » que souhaité

```
<?xml version ="1.0" encoding ="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="a">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="b" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Il est à noter l'utilisation du XMLSchema qui décrit la structure du XSD lui-même

XSD – Exemples (2/2)

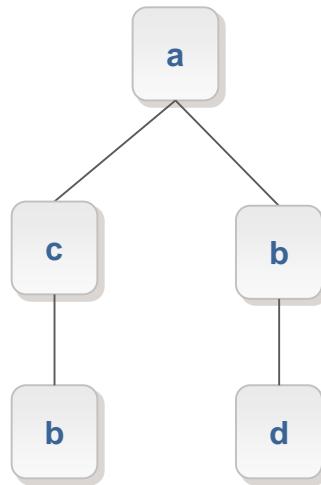
■ Le XSD suivant décrit

- Un document XML de racine « a » pouvant avoir un attribut « x » (de type chaîne de caractères), et un sous-élément « b » devant avoir un attribut « y » (de type entier) valant 3 par défaut

```
<?xml version ="1.0" encoding ="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="a">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="b" minOccurs = "0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:attribute name="y" type="xs:integer" default="3"/ >
                    </xs:complexType>
                </xs:element >
            </xs:sequence >
            <xs:attribute name="x" type="xs:string"/>
        </xs:complexType >
    </xs:element >
</xs:schema >
```

XSD – Plus de possibilités que DTD

- En XSD (et contrairement à une DTD), il est possible de décrire l'arbre suivant (c'est-à-dire de le distinguer de tous les autres arbres) :

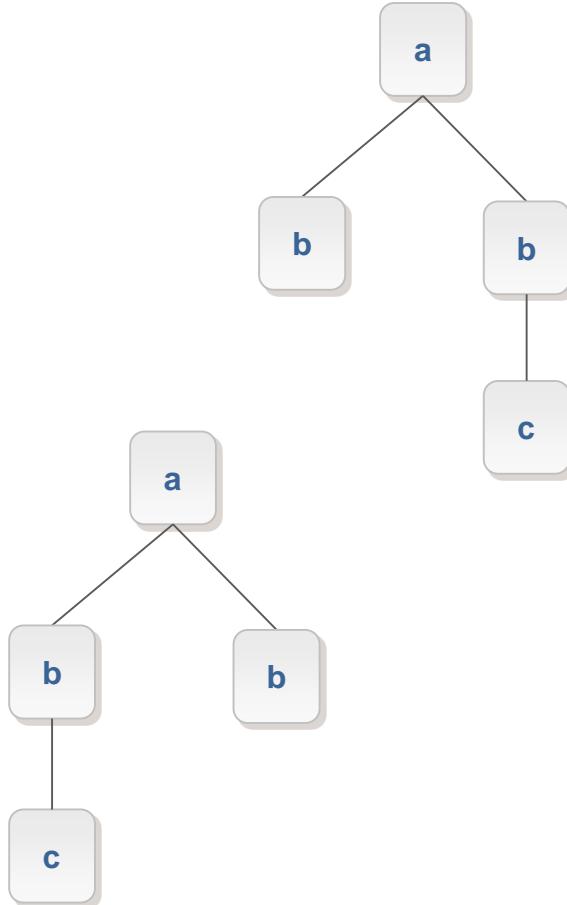


```
<?xml version ="1.0" encoding ="utf-8"?>
<xs:schema xmlns:xs="...">
  <xs:element name="a">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="c">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="b"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element >
        <xs:element name="b">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="d"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Il est à noter que c'est le contexte (la profondeur dans l'arbre) qui permet de distinguer les 2 définitions de l'élément « b »

XSD – Mais encore des impossibilités

- Il reste impossible de décrire (et distinguer) les arbres suivants :



```
<?xml version ="1.0" encoding ="utf-8"?>
<xs:schema xmlns:xs="...">
  <xs:element name="a">
    <xs:complexType>
      <xs:all minOccurs ="1">
        <xs:element name="b"/>
        <xs:element name="b">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="c"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Ce schéma est valide, d'un point de vue « syntaxe XSD » mais
n'est pas interprétable par les validateurs XSD

Sommaire – Cours 3

La validation des documents XML

- Pourquoi valider des documents XML ?
 - Le contrat d'interface
 - Le contrat de service (SOAP vs REST)
- DTD – Document Type Definition
- XSD – XML Schema Definition
- Relax NG – Regular LAnguage for XML Next Generation
- Utilisation de la validation XML
 - Mise en œuvre
 - Application

RelaxNG - Vue d'ensemble

- RelaxNG est un langage permettant de décrire les contraintes régissant la structure d'un document XML
 - Il ne se focalise pas sur le typage des attributs mais conserve ce qui est fait dans le cas de XSD (<http://www.w3.org/TR/xmlschema-2>)
- RelaxNG est un dialecte XML (est décrit en XML)
 - Mais il existe aussi une forme non-XML qui est moins verbeux
 - Son format se veut restreint (peu de constructions) et sa spécification concise
- Relax NG, spécification OASIS est décrit : <http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>)
 - Ce n'est pas un langage supporté par le W3C
- Aujourd'hui, RelaxNG n'est pas encore démocratisé
 - Car sa plus-value reste faible dans la majorité des cas pratiques
 - XSD est suffisant pour couvrir les cas usuels

RelaxNG – Exemples (1/2)

- Le RelaxNG suivant décrit

- Un document XML de racine « a » ayant autant de fils « b » que souhaité

```
<?xml version="1.0" encoding="utf-8"?>
<rg:grammar xmlns:rg="http://relaxng.org/ns/structure /1.0">
  <rg:start>
    <rg:element name="a">
      <rg:zeroOrMore>
        <rg:element name="b">
          <rg:empty/>
        </rg:element>
      </rg:zeroOrMore>
    </rg:element>
  </rg:start>
</rg:grammar>
```

RelaxNG – Exemples (2/2)

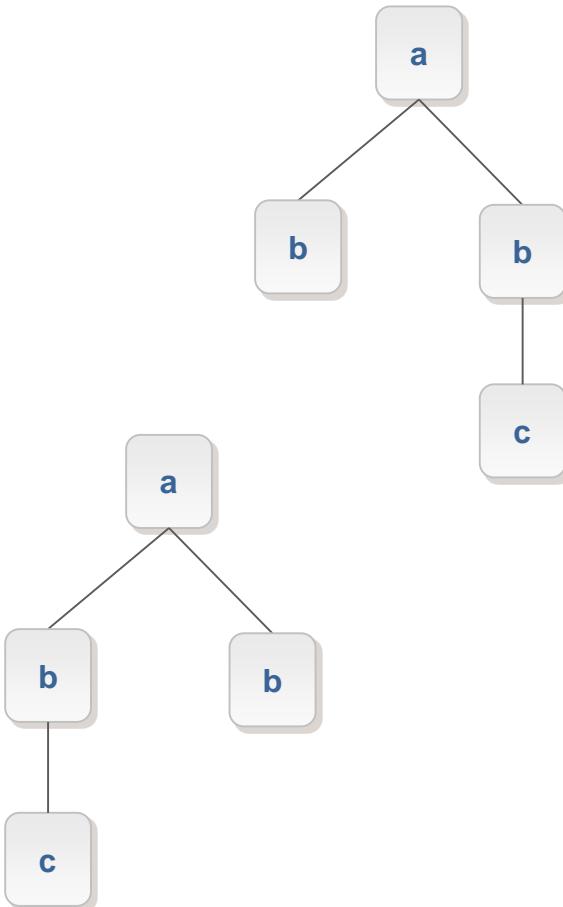
■ Le RelaxNG suivant décrit

- Un document XML de racine « a » pouvant avoir un attribut « x » (de type chaîne de caractères), et un sous-élément « b devant doit avoir un attribut « y » (de type entier) valant 3 par défaut

```
<?xml version="1.0" encoding="utf-8"?>
<rg:grammar xmlns:rg="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary ="http://www.w3.org/2001/XMLSchema-datatypes">
  <rg:start>
    <rg:element name="a">
      <rg:zeroOrMore>
        <rg:element name="b">
          <rg:empty/>
          <rg:attribute name="y" a:defaultValue ="3">
            <rg:data type=" integer"/>
          </rg:attribute>
        </rg:element>
      </rg:zeroOrMore>
      <rg:optional>
        <rg:attribute name="x">
          <rg:data type=" string"/>
        </rg:attribute>
      </rg:optional>
    </rg:element>
  </rg:start>
</rg:grammar>
```

RelaxNG – Plus expressif que XSD

- Contrairement à XSD, RelaxNG permet de décrire (et distinguer) les arbres suivants :



```
...
<rg:element name="a">
  <rg:interleave>
    <rg:element name="b">
      <rg:empty/>
    </rg:element>
    <rg:element name="b">
      <rg:element name="c">
        <rg:empty/>
      </rg:element>
    </rg:element>
  </rg:interleave>
</rg:element>
...

```



Il est à noter que ce genre de cas arrive rarement (et que c'est peut-être mieux car trop complexe à utiliser)

Sommaire – Cours 3

La validation des documents XML

- Pourquoi valider des documents XML ?
 - Le contrat d'interface
 - Le contrat de service (SOAP vs REST)
- DTD – Document Type Definition
- XSD – XML Schema Definition
- Relax NG – Regular LAnguage for XML Next Generation
- Utilisation de la validation XML
 - Mise en œuvre
 - Application

XSD pour spécifier des interfaces

- XSD permet de définir la forme d'un document XML
 - Ce document peut être un fichier
 - Mais peut aussi être un flux
- Ainsi, XSD permet de valider des interfaces de communications dans une communication client / serveur
- Le moyen le plus répandu pour communiquer entre 2 applications sur le Web est le WebServices ; 2 types existent
 - SOAP, contraint via un WSDL (Web Service Definition Langage) qui définit les services offerts, les paramètres en entrée et en sortie
 - REST, qui ne constraint pas les interfaces
 - Mais le format WADL (Web Application Definition Langage) se démocratise pour permettre de définir précisément les interfaces
- Ainsi, les services REST permettent de bénéficier à la fois de la souplesse des protocoles HTTP standards en formalisant les échanges



Q&R

Questions

Réponses

Sommaire – Cours 4

L'utilisation de XPath

■ A quoi sert XPath

- Introduction
- Comparaison vis-à-vis de DOM et SAX

■ Le langage XPath

- L'axe, le filtre et le prédictat
- Les instructions de parcours et leurs raccourcis

■ Application de XPath dans une architecture client-serveur

- Mise en œuvre
- Application

Pourquoi utiliser XPath

- **XPath est un moyen de requête d'un document XML qui se veut concis et lisible**
 - Et dont le but est de simplifier l'accès aux données qui se trouvent réparties dans l'arbre XML
- **Que ce soit par DOM ou SAX, l'accès aux données est fastidieux**
 - Avec DOM, il faut descendre « manuellement » dans l'arbre jusqu'à obtenir la valeur désirée
 - Avec SAX, il faut stocker le contexte dans lequel on se trouve pour être certain d'obtenir la valeur désirée
- **XPath apporte une vue globale de l'arbre (comme peut le faire DOM) en rendant moins verbeux l'accès à la donnée**
 - Les moteurs XPath récents bénéficient de performances proches de SAX
- **L'objectif de XPath est donc de masquer la complexité inhérente à parcourir un document XML tout en rendant efficace ce parcours**

XPath vis-à-vis de DOM et SAX

Accès à la ville de
l'étudiant « bob »

```
<?xml version ="1.0"?>
<school>
  <students>
    <student name="bob">
      <address>
        <city>Bordeaux</city>
      </address>
    </student>
    ...
  </students>
  <teachers>
    ...
  </teachers>
</school>
```

DOM

- Chargement du document XML en intégralité
- Sur la racine du document : getElement('Students') puis itération sur les Element « Student »
- Quand le nom correspond, on continue le parcours à partir du nœud : getElement('address') puis getElement('city').getText()

SAX

- Crédation des Handler
- A chaque « startElement », on stocke l'endroit où on est dans l'arbre
- Quand on arrive sur le nœud « student » avec comme nom « bob », on le flag
- Quand on arrive sur le nœud « city » et que le flag est placé, on récupère la valeur

XPath

- Instanciation du « parser »
- Requête XPath : /school/students/student[@name='bob']/address/city
- Pour avoir la valeur « Bordeaux », il faut rajouter .../city/**text()**

XPath et l'externalisation du parcours

- Dans la comparaison avec DOM et SAX, on constate que
 - D'une part, XPath réclame moins de code (à condition d'utiliser les bonnes librairies)
 - D'autre part, XPath permet de modifier le parcours d'un arbre sans (forcément) à avoir à recompiler du code (Java ou autre)
- C'est un atout majeur : on ne code pas du XPath, on le configure
 - On ne « programme » pas des boucles de parcours mais on paramètre une « chaîne de caractères »
 - La complexité liée à l'algorithme de parcours est reléguée à l'outil d'interprétation de XPath, qui est fait une fois pour toute

Cette approche de « configuration » plutôt que
« codage » est largement répandue dans les
« bonnes pratiques » de programmation actuelles

Historique et références

- La spécification de XPath 2.0 date de 2007
 - C'est le W3C qui l'édite : <http://www.w3.org/TR/xpath20>
- Le langage XSLT se base sur Xpath
- Une évolution de XSL, XQuery, se base aussi sur Xpath

La simplicité de XPath en a fait
un standard du monde XML

Sommaire – Cours 4

L'utilisation de XPath

- A quoi sert XPath
 - Introduction
 - Comparaison vis-à-vis de DOM et SAX
- Le langage XPath
 - L'axe, le filtre et le prédicat
 - Les instructions de parcours et leurs raccourcis
- Application de XPath dans une architecture client-serveur
 - Mise en œuvre
 - Application

Le modèle arborescent XDM

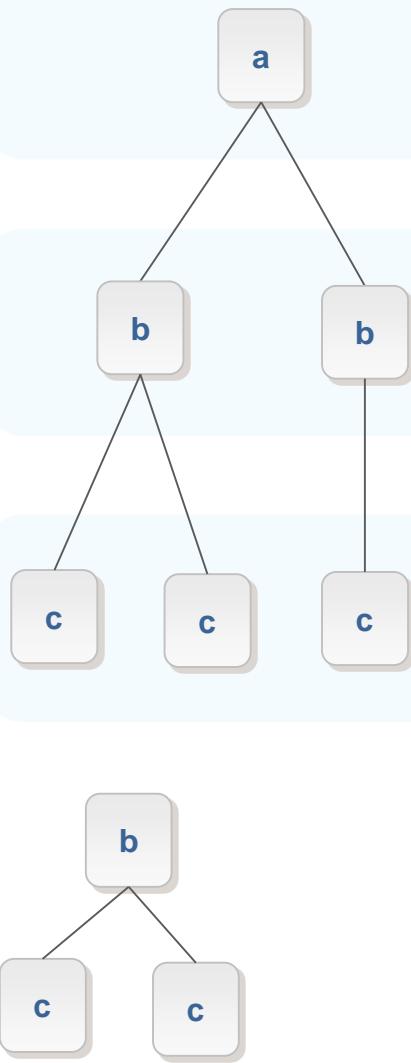
- XDM signifie : XPath Data Model
 - À comparer à DOM : Document Object Model
- Il est semblable au modèle DOM sauf (notamment) :
 - Tout nœud Text qui ne contient que des espaces est éliminé de l'arbre (sauf si un ancêtre le protège)
 - => cela simplifie l'arbre au final
- Chaque élément a une string value strval :
 - Pour les noeuds Text, Attr, Namespace, ProcessingInstruction et Comment, il s'agit du champ valeur
 - Pour les nœuds Document et Element, il s'agit de la concaténation des strval de leurs sous-nœuds
 - => chaque nœud « vaut » quelque chose et cela est très utile (cf. la suite)
- Description complète : <http://www.w3.org/TR/xpath-datamodel/>

Le chemin XPath

- Un chemin XPath est une succession d'étapes :
 - [/]étape1/étape2/.../étapeN
- Un chemin est absolu s'il débute par « / », sinon il est relatif
- Une étape s'évalue dans un contexte (ensemble de nœuds+...) et produit le contexte suivant
- Le contexte initial d'un chemin absolu est le nœud racine.
 - Pour un chemin relatif, c'est un nœud courant
- L'opérateur « | » réalise l'union de chemins

On retrouve ici (entre « / » et « | ») des classiques
de la programmation => XPath se veut intuitif

XPath par l'exemple (écriture simplifiée)



- /a => renvoie l'élément racine de nom « a »
- /a/b => renvoie le ou les éléments de nom « b », fils du nœud racine de nom « a »
- /a/b/c => renvoie le ou les éléments de nom « c », fils d'un nœud de nom « b », lui-même fils du nœud racine de nom « a »
- Si le nœud courant est le « b » de gauche ci-dessus, alors le XPath « c » renverra les 2 nœuds fils
 - Comme le contexte de départ est différent, on se trouve ici dans un cas où « /a/b/c » renvoie plus d'éléments que « c avec « b » comme nœud courant »

Les étapes d'un chemin XPath

- Une étape d'un chemin XPath se décompose comme suit :
axe::filtre[prédicat]...[prédicat]
 - L'axe définit la direction du parcours (nœud fils, nœuds frères, ...)
 - Le filtre identifie les nœuds retenus (par leur nom, leur type)
 - Les prédictats qui ne conservent que les nœuds qui les satisfont (position, valeur d'attribut, ...)
- Le contexte d'évaluation passe successivement au travers de l'axe, puis du filtre et enfin, des prédictats

La théorie paraît ici complexe mais

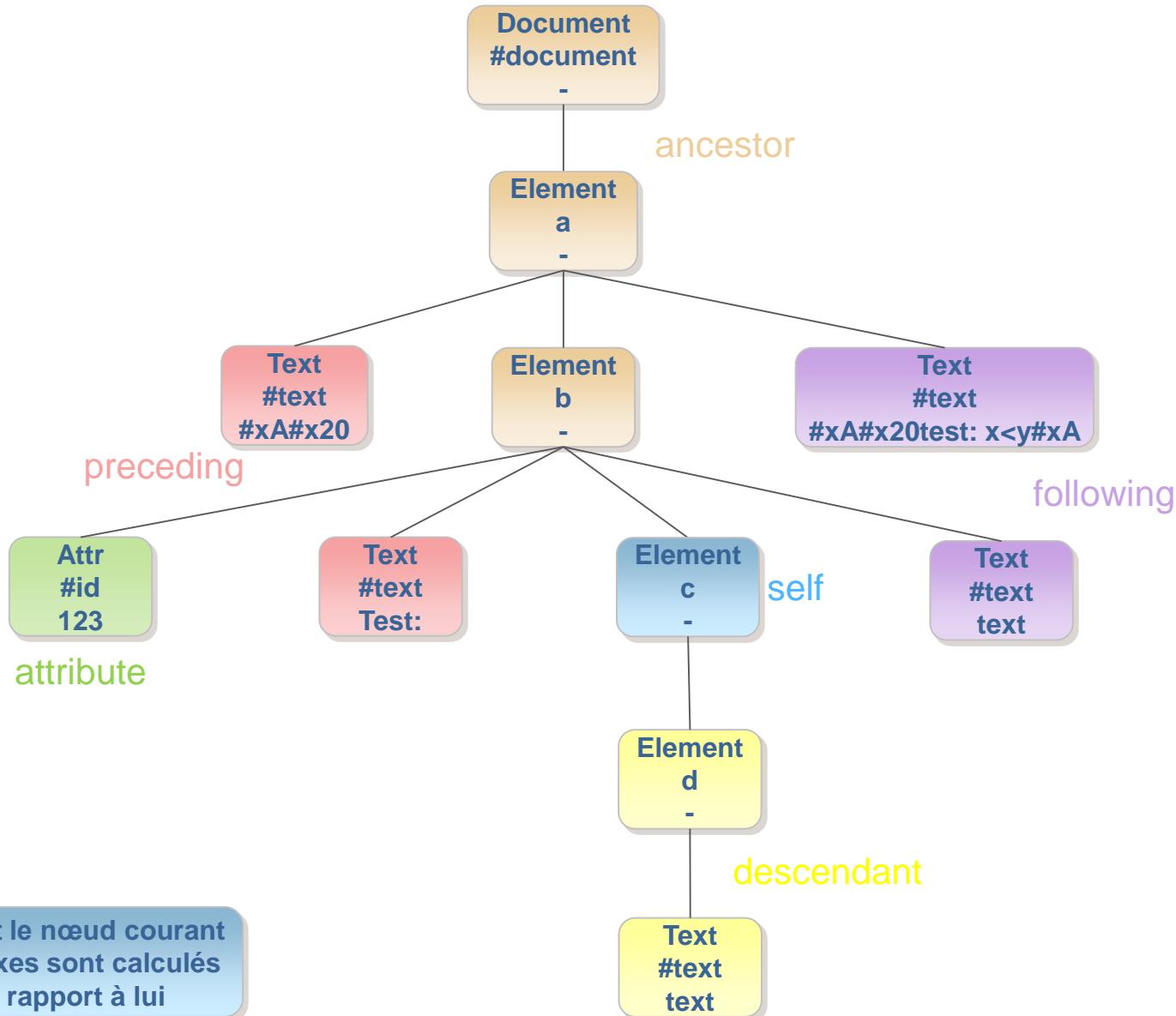
- des simplifications existent
- Ce sont souvent les mêmes chemins XPath « type » qui reviennent

Les différents axes de XPath

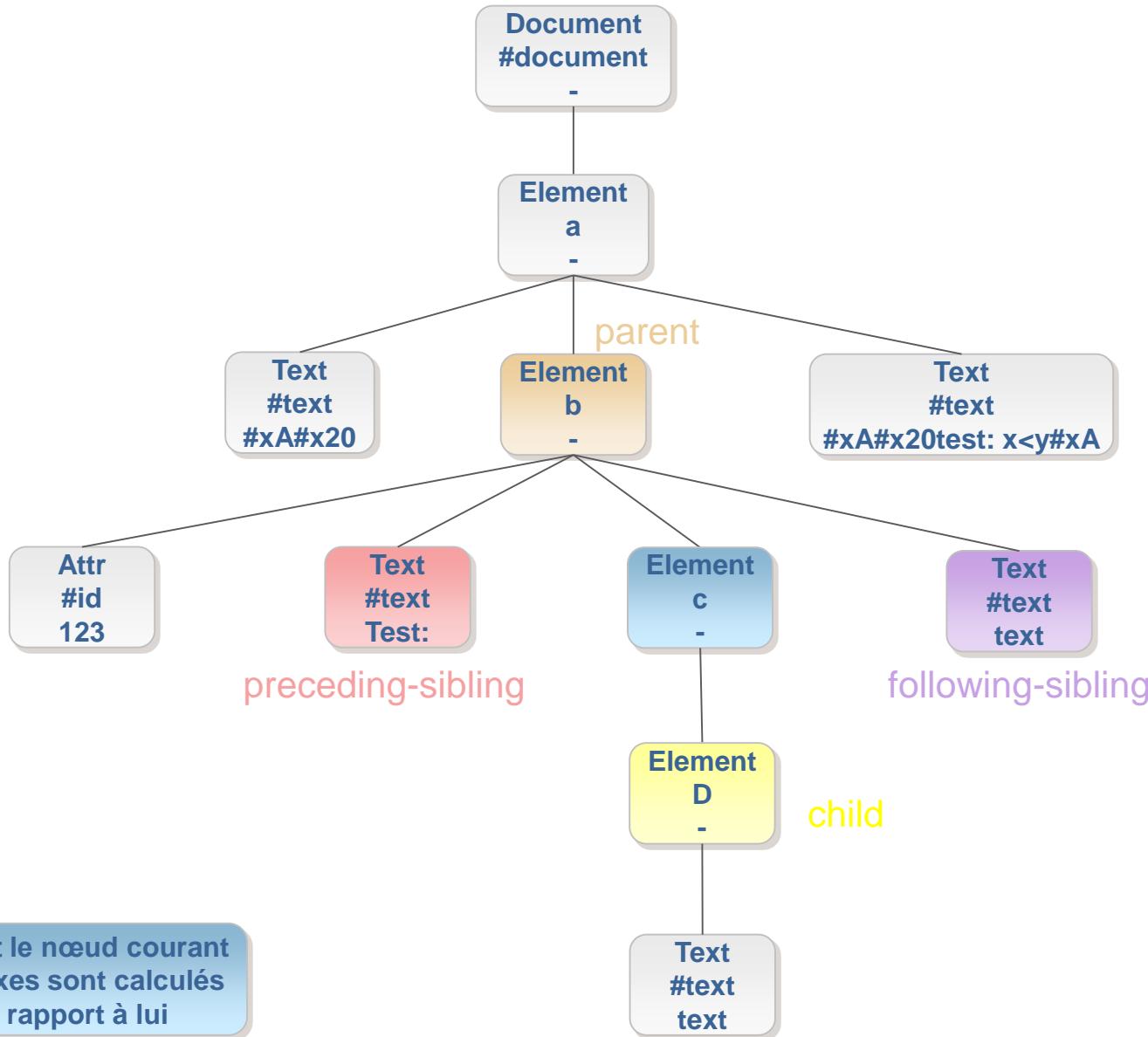
Axe	Signification
ancestor	Nœuds ancêtres
ancestor-or-self	Nœuds ancêtres et nœud courant
attribute	Nœuds « Attr » fils
child	Nœuds fils
descendant	Nœuds descendants
descendant-or-self	Nœuds descendants et nœud courant
following	Nœuds suivants, parcours préfixe
following-sibling	Nœuds frères suivants
namespace	Nœuds « Namespace » fils
parent	Nœud parent
preceding	Nœuds précédents, parcours préfixe
preceding-sibling	Nœuds frères précédents
self	Nœud courant

Ces axes s'entendent « par rapport au nœud courant »

Les axes de XPath – Exemples (1/2)



Les axes de XPath – Exemples (2/2)



Le filtre dans XPath

- Le filtre permet de sélectionner des nœuds selon leur nom ou leur type :

Filtre	Signification
QName (ex : a, . . .)	Nœud élément de nom correspondant
*	Noeuds ayant le type principal (cf. ci-après)
text()	Noeuds de type Text
comment()	Noeuds de type Comment
processing-instruction()	Noeuds de type ProcessingInstruction
node()	Noeuds quelconques

- Le type principal est :
 - Attr pour l'axe attribute,
 - Namespace pour l'axe namespace,
 - et Element pour les autres axes

Le prédictat dans XPath

- Un prédictat est une combinaison booléenne de comparaisons d'objets (ensemble de nœuds, nombre, booléen ou chaînes de caractères)
- Les objets comparés sont obtenus via des expressions ou des chemins Xpath
 - Ainsi, une expression XPath peut à la fois définir le chemin vers tous les nœuds ciblés avec comme critères un autre chemin XPath qui filtre sur les mêmes nœuds ciblés ou non
- Exemples (écriture simplifiée) :
 - `/a/b/c[@<nom_attr_c> = 3]` => filtre les nœuds de nom « c » dont l'attribut vaut 3
 - `/a/b/c[../@<nom_attr_b> = 3]` => filtre les nœuds de nom « c » dont le père (ici de nom « b ») a un attribut valant 3

Exemples de chemin XPath complets

- **descendant::text()** sélectionne tous les nœuds Text dans le sous arbre du nœud courant
- **child::*[attribute::x="3"]** sélectionne les nœuds de type Element, fils du nœud courant, et ayant un attribut x qui a pour valeur 3
- **child::a/attribute::x** sélectionne les nœuds attributs de nom x sous un nœud élément de nom a, fils du nœud courant
- **child::*[self::a or attribute::x="3"]** sélectionne les nœuds Element, enfants du nœud courant, de nom a ou ayant un attribut x de valeur 3

Abréviations – la théorie

- Pour les éléments du langage XPath les plus utilisés, des « raccourcis d'écriture » permettent de simplifier l'écriture (et la lecture) des chemins Xpath
- L'axe `child::` peut être omis
- L'axe `attribute::` est abrégé en `@`
- L'expression `/descendant-or-self::node()` s'abrége en `//`
- L'expression `self::node()` est abrégée en « `.` »
- L'expression `self::parent()` est abrégée en « `..` »

On retrouve ici des notions de
parcours d'arborescence de fichiers

Abréviations – les exemples

Expression	Abréviation	Signification
child::a/child::b	a/b	Nœuds de nom « b », fils des nœuds de nom « a », eux-mêmes fils du nœud courant
child::node()[attribute::x="3"]	node()[@x="3"]	Nœuds fils du nœud courant et dont l'attribut « x » vaut 3
/descendant-or-self::node()/child::b	//b	Les nœuds descendants du nœud courant et de nom « b »
a[self::node()/b/@x="3"]	a[./b[@x="3"]] voire même a[b[@x="3"]]	Les nœuds de nom « a » dont au moins 1 des nœuds fils de nom « b » a un attribut « x » valant 3
a/b[self::parent()]/@x="3"]	a/b[../@x="3"]	Les nœuds de nom « b » dont le nœud parent de nom « a » a un attribut valant 3

Sommaire – Cours 4

L'utilisation de XPath

- A quoi sert XPath
 - Introduction
 - Comparaison vis-à-vis de DOM et SAX
- Le langage XPath
 - L'axe, le filtre et le prédicat
 - Les instructions de parcours et leurs raccourcis
- Application de Xpath dans une architecture client-serveur
 - Mise en œuvre
 - Application

XPath pour filtrer, côté client comme côté serveur

- **XPath permet de récupérer des valeurs (ou des listes de nœuds) et cela peut servir**
 - Côté serveur, pour filtrer un document XML avant de l'envoyer
 - Côté client, pour afficher certains valeurs d'un document XML reçu intégralement mais dont seulement une partie est utile
 - Dans l'utilisation d'AJAX, le séquencement classique est alors
 - Demande de l'utilisateur pour obtenir une information
 - Requête auprès du serveur
 - Réception du flux XML
 - Utilisation de XPath pour récupérer une valeur
 - Affichage de l'information demandée
- **XPath étant un langage pratique, il sert aussi de base à d'autres langages**
 - Dans le cas de XSL, la récupération de valeur ou l'itération sur un groupe de nœud s'effectue via XPath



Q&R

Questions

Réponses

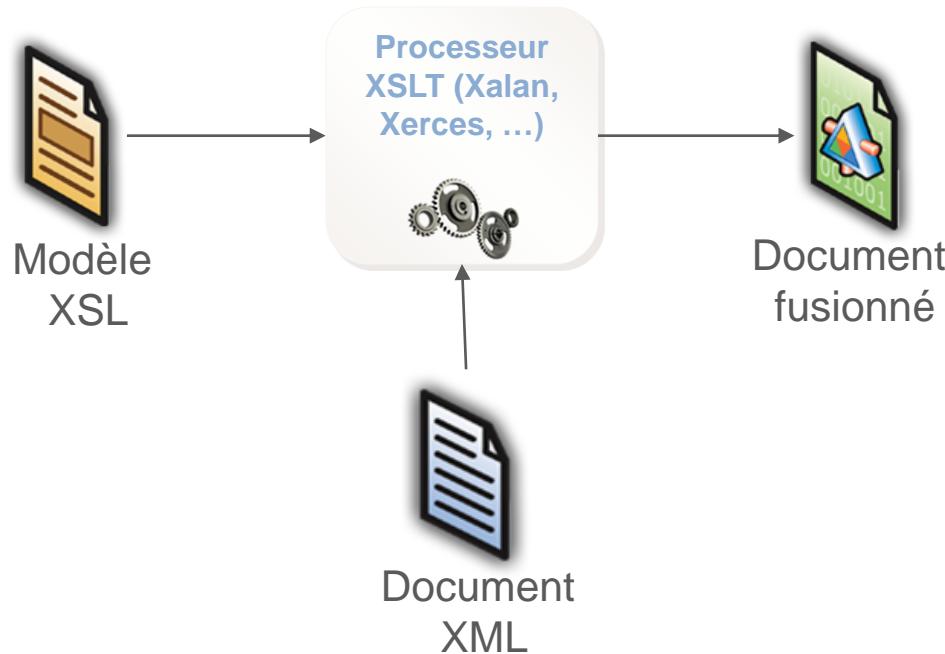
Sommaire – Cours 5

Les feuilles de style XSL

- **La transformation XSL**
 - Son objectif
 - Ses composants
 - Son principe de fonctionnement
- **Le langage XSL**
 - Les instructions
 - Les normes à connaître
- **Application de XSL dans une architecture client-serveur**
 - Mise en œuvre
 - Application

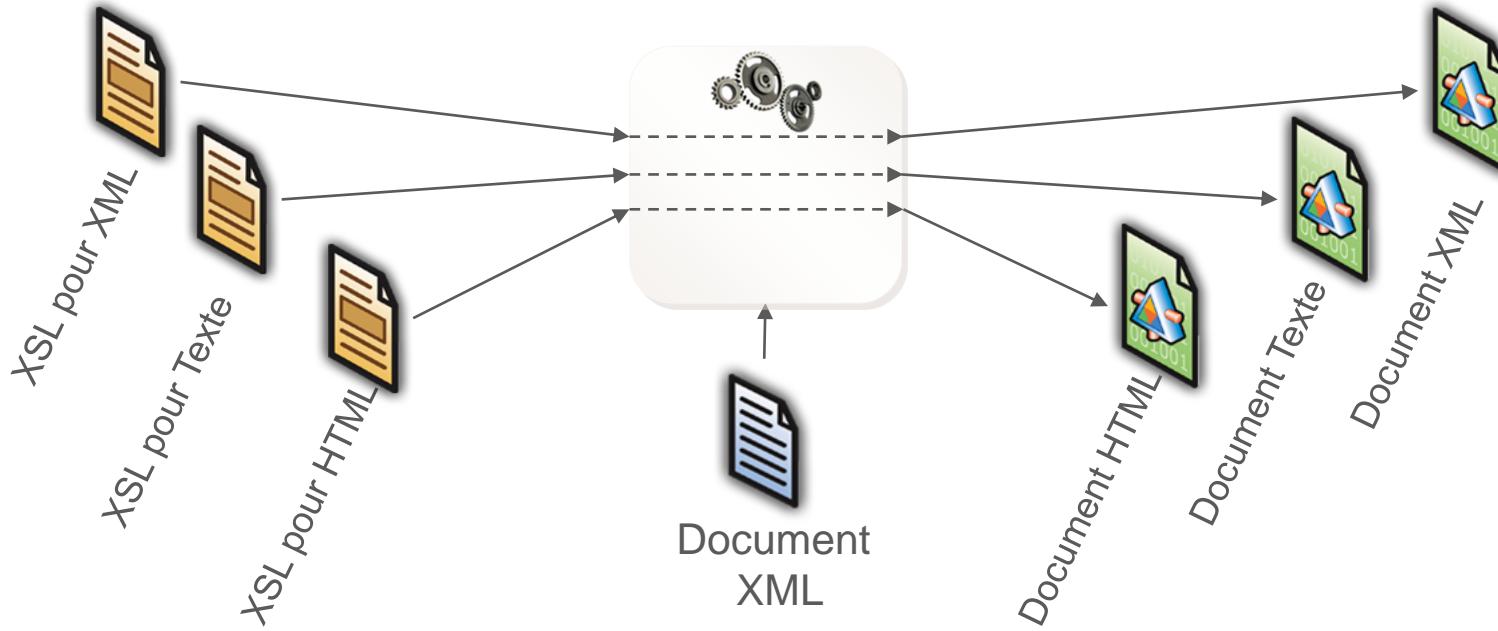
Introduction à XSL

- L'objectif de **XSL** (*eXtensible Stylesheet Language*) est de fournir un langage permettant de transformer un fichier XML source dans un format textuel (formaté ou non, XML ou non) différent



Objectif de XSL

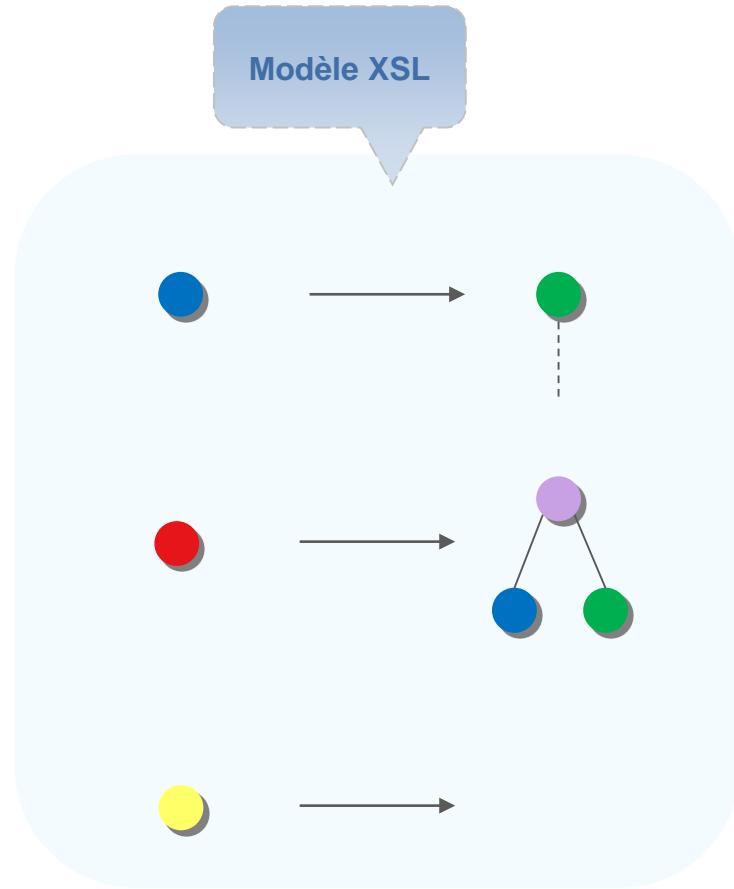
- XSL permet de formater des données XML afin de les présenter selon un format cible pré-défini
 - Plus précisément, c'est la présentation (in hérente à chaque modèle XSL) qui est « paramétrée » par les données XML
 - (c'est juste une question de point de vue !)
- L'avantage est ainsi qu'à partir d'un même jeu de données, leur affichage peut être adapté
 - Le modèle de données ne change pas, c'est uniquement la présentation qui en est faite



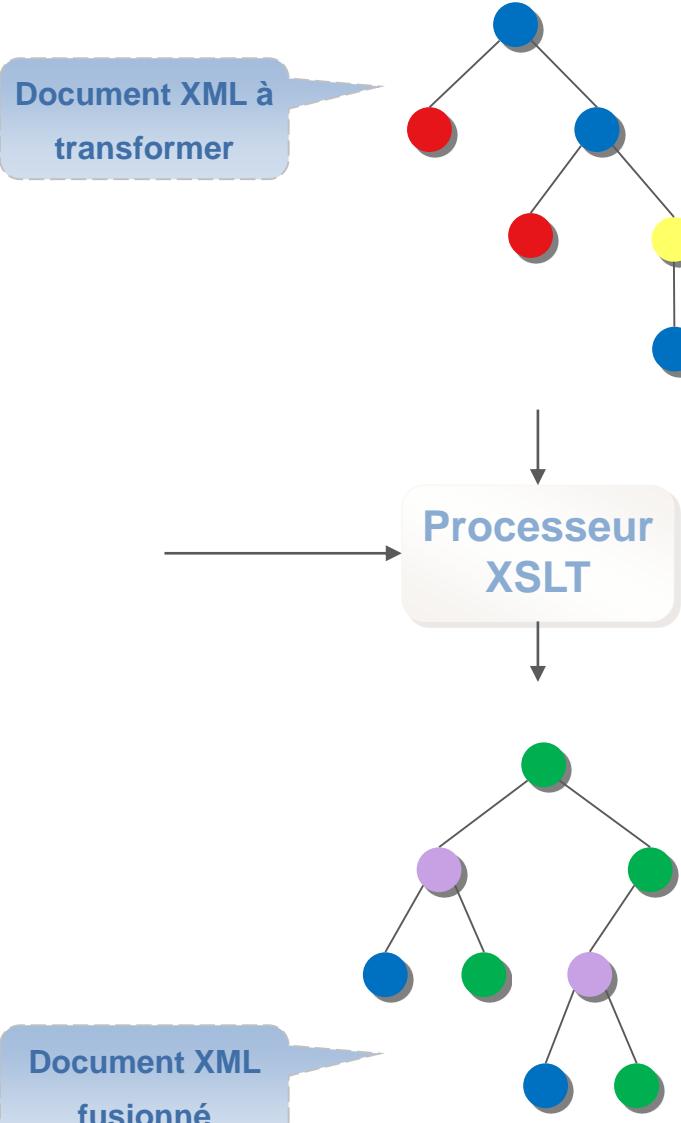
Composants de XSL

- XQuery 1.0 and XPath 2.0 Data Model pour la représentation arborescente des documents XML
 - W3C 2007, <http://www.w3.org/TR/xpath-datamodel/>
- XSLT 2.0 pour la transformation des arbres XDM
 - W3C 2007, <http://www.w3.org/TR/xslt20/>
- XSL-FO (FO pour Formatting Object) est un langage servant d'intermédiaire entre les données XML brutes et un document final de présentation
 - XSL-FO permet ainsi rendre indépendant le rendu final (via HTML, PDF, ...) de la transformation des données
 - Par exemple, on ne dit pas « bold » qui est une routine HTML, mais plutôt « emphasis » qui indique qu'on veut mettre en avant le texte (sans présumer de la façon dont il sera mis en avant)
 - XSL-FO a aussi pour vocation de pouvoir générer des fichiers PDF (ce qui n'est pas possible uniquement avec XSL car ce n'est pas un format textuel)
- XSL-FO 1.1 pour la présentation
 - W3C 2006, <http://www.w3.org/TR/xsl/>

Exemple d'une transformation XSL (1/2)



On transforme ici un document XML en un autre document XML, mais le format de sortie peut être différent



Exemple d'une transformation XSL (2/2)

```
<?xml version="1.0"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="blue">
        <green>
            <xsl:apply-templates/>
        </green>
    </xsl:template>
    <xsl:template match="red">
        <purple>
            <blue/>
            <green/>
        </purple>
    </xsl:template>
    <xsl:template match="yellow"/>
</xsl:stylesheet>
```

```
<?xml version ="1.0"?>
<blue>
    <red/>
    <blue>
        <red/>
        <yellow>
            <green/>
        </yellow>
    </blue>
</blue>
```

Processeur
XSLT

```
<?xml version ="1.0"?>
<green>
    <purple>
        <blue/>
        <green/>
    </purple>
    <green>
        <purple>
            <blue/>
            <green/>
        </purple>
    </green>
</green>
```

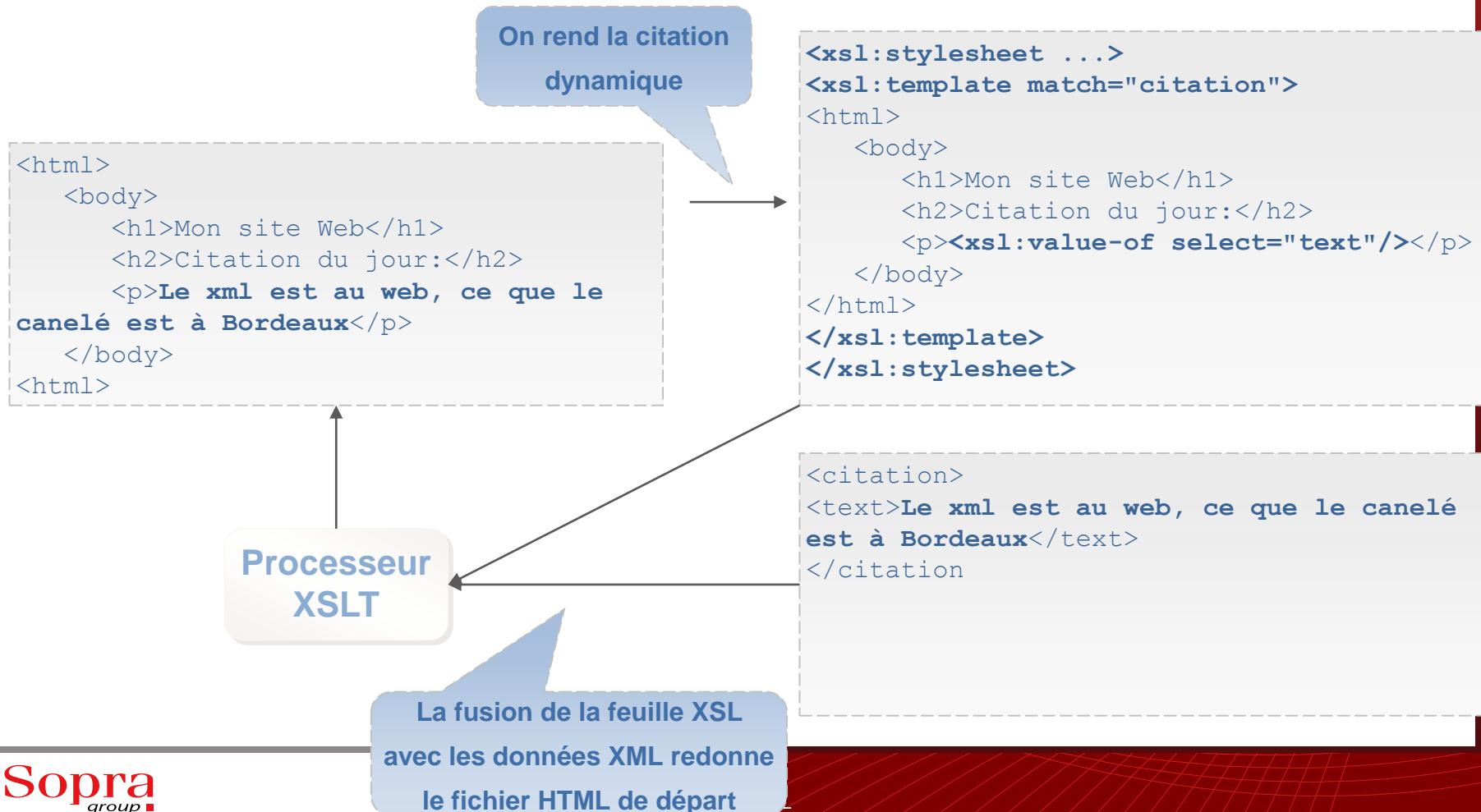
Sommaire – Cours 5

Les feuilles de style XSL

- La transformation XSL
 - Son objectif
 - Ses composants
 - Son principe de fonctionnement
- Le langage XSL
 - Les instructions
 - Les normes à connaître
- Application de XSL dans une architecture client-serveur
 - Mise en œuvre
 - Application

La mécanique d'un modèle XSL

- On peut considérer une feuille XSL comme suit
 - C'est le format du document final que l'on veut obtenir
 - Auquel on substitue des « routines » XSL pour placer des données dynamiquement



La feuille de style et le modèle XSL

■ Un modèle (template) XSL est composé

- D'un **motif** auquel appliquer le modèle
- D'un **contenu**, lui-même composé
 - D'**éléments statiques** (qui seront retranscrits de façon brutes)
 - D'**éléments dynamiques** qui correspondent aux instructions XSL (repérées par des balises **<xsl: ...>**)

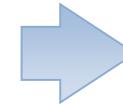
■ Exemples :

```
<xsl:template match="citation">
  <h1>Formatage HTML</h1>
  <p><xsl:value-of select="text" /></p>
</xsl:template>
```



```
<h1>Formatage HTML</h1>
<p>blah</p>
```

```
<xsl:template match="citation">
  <title>Formatage XML</title>
  <text><xsl:value-of select="text" /></text>
</xsl:template>
```



```
<title>Formatage XML</title>
<text>blah</text>
```

```
<xsl:template match="citation">
# Formatage TEXT
<xsl:value-of select="text" />
</xsl:template>
```



```
# Formatage TEXT
blah
```

Le motif XSL

- Un motif XSL (pattern) est une union de chemins XPath n'utilisant que les axes child et attribute, ainsi que les abréviations // et /
 - C'est ce motif qui détermine si le template s'applique ou non au document XML (et le nœud courant) en entrée
- Attention, le motif « a » s'interprète self::a et non pas child::a mais a/b s'interprète tel quel (child:: implicite)
 - Cf. <http://www.w3.org/TR/xslt20/#dt-pattern>
- Exemples :
 - a reconnaît tout nœud Element de nom a
 - * reconnaît tout nœud (de type principal)
 - a|b reconnaît tout nœud Element de nom a ou de nom b
 - a//b/text() reconnaît tout nœud Text ayant un père de nom b qui possède un ancêtre de nom a

Les instructions XSL – xsl:apply-templates

- L'instruction **xsl:apply-templates** permet d'appliquer récursivement les modèles sur les nœuds sélectionnés
 - Si aucun ensemble de nœuds n'est explicitement sélectionné, tous les nœuds fils sont alors utilisés par défaut
- Format : **<xsl:apply-templates match="critères des nœuds"> ... </xsl:apply-templates>**

Structure XML

```
<school>
  <teachers>
    <teacher>
      <address/>
    </teacher>
  </teachers>
  <students>
    <student>
      <address/>
    </student>
  </students>
</school>
```

Feuille XSL

```
<xsl:template match="/">
  <h1>Les adresses des profs sont:</h1>
  <xsl:apply-templates select="//teacher/address"/>

  <h1>Les adresses des étudiants sont:</h1>
  <xsl:apply-templates select="//student/address"/>
</xsl:template>

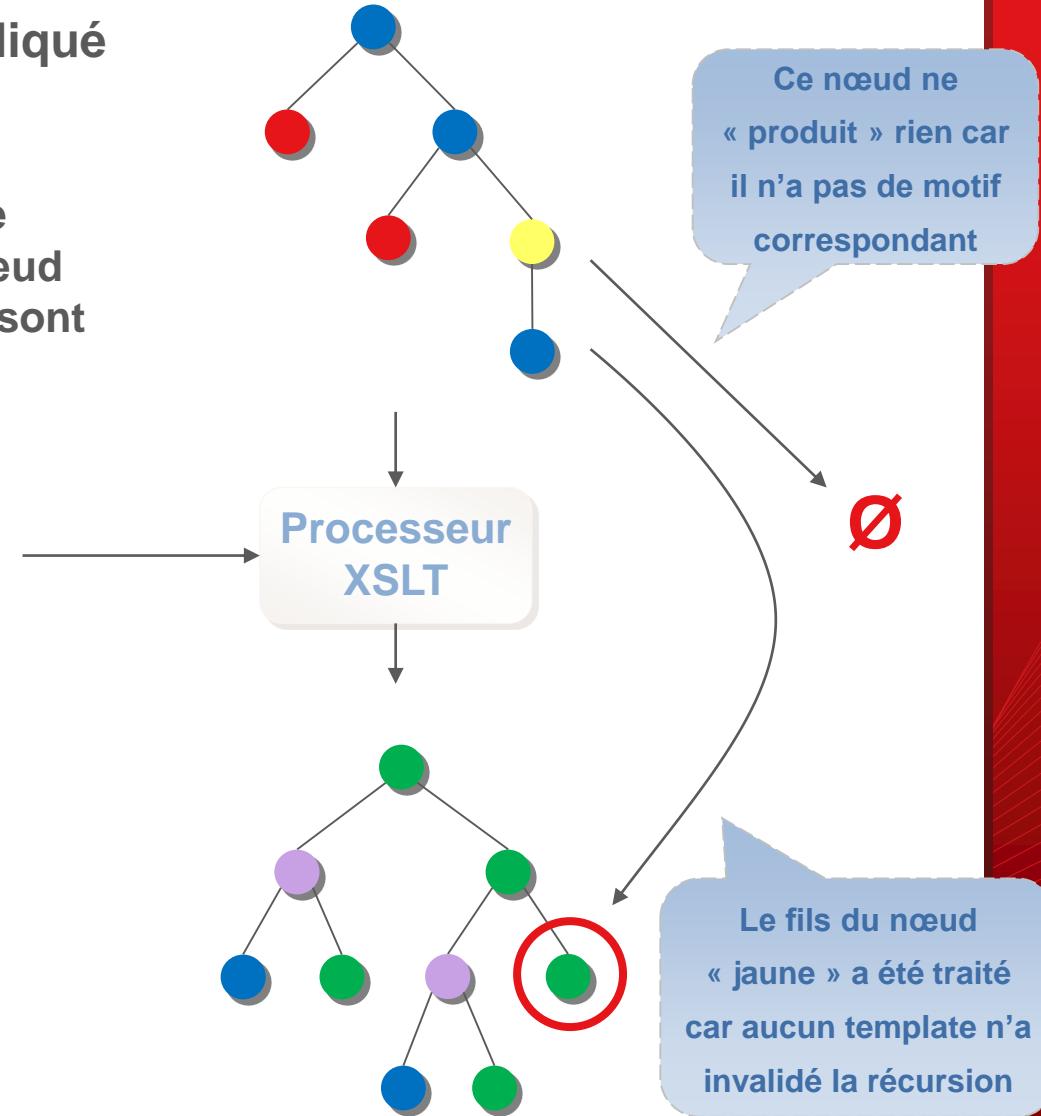
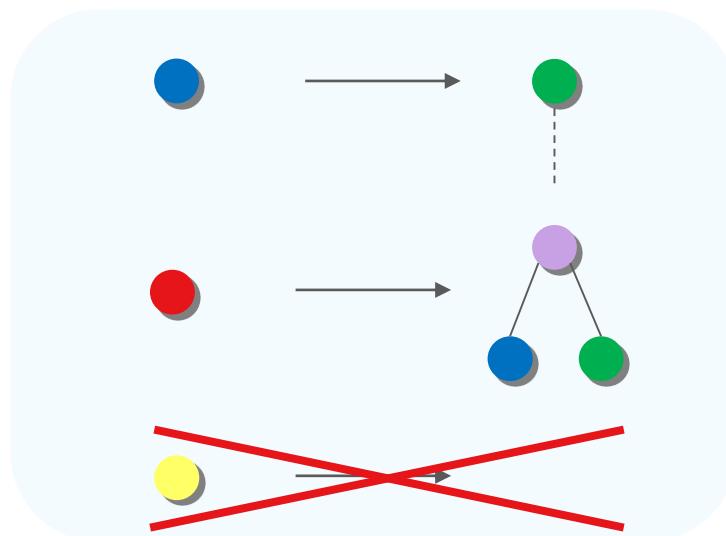
<xsl:template match="address">
  <!-- formatage de l'adresse -->
</xsl:template>
```

L'instruction **xsl:apply-templates** permet de réutiliser des modèles XSL quand des mêmes motifs apparaissent plusieurs fois dans l'arbre XML de départ

xsl:apply-templates – modèles par défaut (1/2)

- Par défaut, un modèle est appliqué récursivement

- Et ceci, même si un nœud intermédiaire ne trouve pas de modèle correspondant : ce nœud est alors ignoré, mais ses fils sont pris en compte



xsl:apply-templates – modèles par défaut (2/2)

- Les instructions suivantes sont « insérées » par défaut dans toutes feuilles XSL
- Récursion « infinie » (cf. slide précédent)

```
<xsl:template match ="* | /">
  <xsl:apply-templates/>
</xsl:template>
```

- Recopie des nœuds Text et Attr

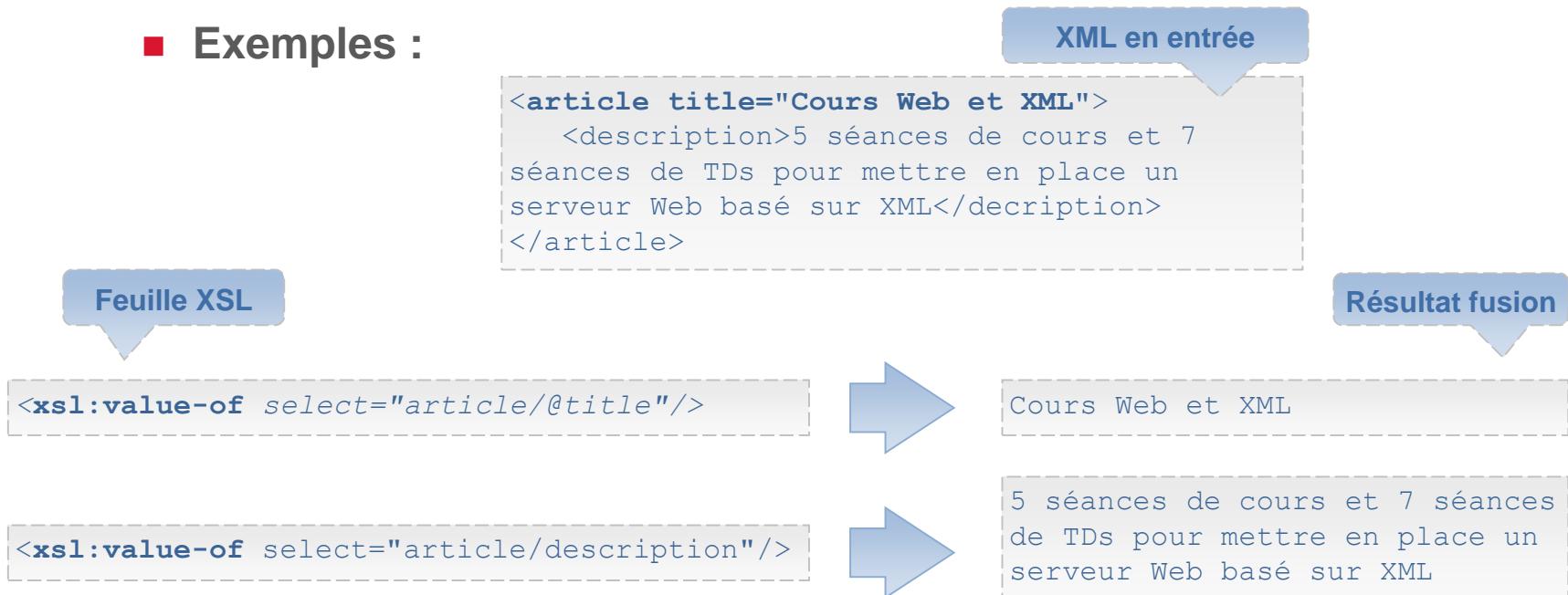
```
<xsl:template match="text () | @* ">
  <xsl:value-of select =".."/>
</xsl:template>
```

- Les nœuds Comment et ProcessingInstruction sont ignorés

```
<xsl:template match="processing -instruction () | comment () "/>
```

Les instructions XSL – xsl:value-of

- Extraction de la valeur (string-value) d'un nœud ou d'un attribut référencé dans l'instruction par un chemin XPath
- La valeur extraite est substituée à `<xsl:value-of.../>` dans le document final
- Format : `<xsl:value-of select="choix du nœud"/>`
- Exemples :



Les instructions XSL – xsl:for-each

■ Itération sur les nœuds sélectionnés

- L'utilisation de l'instruction **for-each** correspond au « style itératif » complémentaire du « style récursif » que porte l'instruction **xsl:apply-templates**

■ Format : **<xsl:for-each select="critères des nœuds"> ... </xsl:for-each>**

■ Exemples

Feuille XSL

```
<xsl:template match="/lessons">
  <xsl:for-each select="lesson[@number > 3]">
    <xsl:value-of select="@title"/>
  </xsl:for-each>
</xsl:template>
```

XML en entrée

```
<lessons>
  <lesson title="Seveur Web" number=1/>
  <lesson title="Les base de XML" number=2/>
  <lesson title="Les schémas XSD" number=3/>
  <lesson title="Les chemins XPath" number=4/>
  <lesson title="Les feuilles XSL" number=5/>
</lessons>
```



Résultat fusion

```
Les chemins Xpath
Les feuilles XSL
```

Les instructions XSL – xsl:if et xsl:choose

■ Selon la valeur de certains éléments, on peut vouloir changer le comportement de la feuille de style

- Attention, l'instruction « else » n'existe pas
- Pour avoir une suite de « if / else if », il vaut mieux utiliser xsl:choose

■ Format

- `<xsl:if test="critère des nœuds"> ... </xsl:if>`
- `<xsl:choose><xsl:when test="critère des nœuds"> ... <xsl:when> ... <xsl:otherwise> ... </xsl:otherwise></xsl:choose>`

■ Exemples

```
<xsl:if test="@author='bob'>
    C'est encore bob l'auteur !
</xsl:if>
```

```
<xsl:choose>
    <xsl:when test="@rights='superadmin'">
        Vous disposez des droits de super administrateur
    </xsl:when>
    <xsl:when test="@rights='admin'">
        Vous disposez des droits d'administrateur
    </xsl:when>
    <xsl:otherwise>
        Vous disposez des droits utilisateurs
    </xsl:otherwise>
</xsl:choose>
```

Les instructions XSL – xsl:variable

- Il peut parfois être utile de stocker un résultat intermédiaire qui impactera le comportement de la feuille de style
 - Comme cela se fait via des langages de programmation « classique »
- Une fois une valeur affectée à une variable, elle ne peut plus être changée
- Format : **<xsl:variable name="nom de la variable"> valeur de la variable </xsl:variable>**
 - La valeur peut aussi être donnée via l'attribut « select »
- Exemple

```
<xsl:variable name="url"/>
  http://www.enseirb-matmeca.fr/~<xsl:value-of select="@login"/>
</xsl:variable>
...
Vous pouvez accéder à la page via : <xsl:value-of select="$url"/>
```

Les instructions XSL – xsl:param

- Une feuille de style XSL peut être paramétrée afin de changer son comportement
 - Cela permet de garder du code commun et de ne changer que ce qui est nécessaire
- Attention, l'accès à un paramètre ou à une variable se fait de la même manière : le nommage est donc important !
- Format : `<xsl:param name="nom de la variable"/>`
 - Une valeur par défaut peut être spécifiée avec l'attribut « select »
- Exemple

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="user"/>
  <xsl:template match="/">
    Bonjour <xsl:value-of select="$user"/>
    <xsl:if test="$user=' '">
      Veuillez vous identifier.
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

En fait, un moteur XSLT prend 3 éléments en entrée

- Le document XML
- La feuille de style XSL
- ET les paramètres

- **xsl:element**

- **Exemple pour les TDs**

```
<xsl:element name="a">

<xsl:attribute name="href">../task/list?ownerNameFilter=<xsl:value-of
select='@name' /></xsl:attribute>

<xsl:attribute name="target">_self</xsl:attribute>

<xsl:value-of select='@name' />

</xsl:element>
```

Permet d'obtenir la forme <a href="../task/list?ownerNameFilter=matLo"
target="_self">matLo

Sommaire – Cours 5

Les feuilles de style XSL

- La transformation XSL
 - Son objectif
 - Ses composants
 - Son principe de fonctionnement
- Le langage XSL
 - Les instructions
 - Les normes à connaître
- Application de XSL dans une architecture client-serveur
 - Mise en œuvre
 - Application

XSLT du côté serveur de préférence

- XSLT permet de formater des données
 - Du côté serveur, l'accès aux données est garantie et le formatage peut être décidé en fonction de l'appelant (qui décide du format qu'il désire : XML brut, HTML, ...)
 - Du côté client, l'accès aux données s'effectue via un appel au serveur et l'utilisation de XSL impliquerait que la feuille soit aussi récupérée côté serveur (elle ne peut pas être stockée côté client)
 - Non seulement cela est lourd
 - Mais en plus ça ne semble pas utile car à partir du moment où un appel serveur est fait, le formatage peut lui être demandé
 - (Même si cela reste possible en JavaScript)



Q&R

Questions

Réponses