Services Web
Cours 2

Isis TRUCK
Université Paris 8

Références

- http://www-igm.univ-mlv.fr/~dr/XPOSE2004/woollams/index.html
- https://benoitpiette.com/labo/introduction-aux-web-services.html
- http://www.christian-faure.net/2012/10/08/services-web-ou-api/
- http://www.gchagnon.fr/cours/xml/servicesweb.html
- https://openclassrooms.com/courses/les-requetes-http
- https://doc.ubuntu-fr.org/wget
- https://fr.wikipedia.org/
- https://www.w3schools.com/xml/xml_dtd_intro.asp
- https://www.irif.fr/~carton/Enseignement/XML/Cours/Namespace/index.html
- Et beaucoup d'autres…

Plan

- Cours 1 : Préliminaires : couches réseau
  - Modèle théorique : OSI
  - Modèle pratique : TCP/IP
  - HTTP
- Cours 2 : Services Web ou API ?
  - Service Web (Génèse, …, XML, DTD, XSD…)
  - API, API publique
  - Contrat
- Cours 3 : SOAP ?
- CORBA
- REST

Services Web : Génèse

- Au départ : **les applications réparties**
- Exemple : *banque de dépôt*
  - offre possibilité de consulter son compte client par différents moyens : Mobile ou Internet
  - Il faut une architecture adaptative et ouverte avec une seule application de traitement mais plusieurs interfaces *front-end* (application AppStore, GooglePlay… ou web)
  - Donc application B2C, mais également B2B car la banque peut proposer des produits financiers à destination de ses partenaires
  - On parle également **d'application agrégée** lorsque le produit proposé est une agrégation de plusieurs autres produits.
- Autre exemple : la *réservation d'un voyage* est en fait une application agrégée car une combinaison d'une réservation de billets de transport, d'hôtels, de voitures de location, etc.
- Agrégation d'informations récupérées auprès de plusieurs fournisseurs (ce sont bien des Services Web) : compagnies aériennes, hôtels, loueurs de véhicules…

Services Web : Génèse

- Une application Web communicante est un assemblage de services Web qui peuvent être **internes ou externes** et fournis par divers partenaires et fournisseurs.
- Mais quelle différence entre Service Web et appli distante traditionnelle ?
  - Archi réparties dépendent et évoluent en fonction des :
    - Protocoles d'échanges et d'accès (Corba, RMI, .NET, ActiveX… *cf*. ***Programmation orientée composants – qu'est-ce que c'est ?***)
    - Langages d'implémentation (Java, C++, …)
    - Interfaces d'interaction et de présentation
  - Mais les clients doivent, eux aussi, évoluer pour pouvoir recevoir de telles architectures.

Services Web : Génèse

- Le Web, quant à lui, est plus sommaire :
  - Débit plutôt faible
  - Caractéristiques du protocole (HTTP) bien plus basiques que Corba, RMI, etc.
  - Interactions moins riches (html, CSS, etc.)

- Côté Web, les configurations clientes sont légères (un navigateur) et le protocole HTTP est installé (compris) partout.
- Donc HTTP s'impose…

- D'où la volonté de combiner les caractéristiques des architectures distribuées et les contraintes du Web (client léger et mise en œuvre de HTTP) => **Apparition des Services Web**

Service Web : qu’est-ce que c’est ?

- Service Web ?
  - Communication machine à machine sur le Web
  - déporter le traitement de données d'un poste client, vers un poste serveur sur lequel "tourne" l'application
  - Trois raisons pourraient inciter à opter pour un tel traitement déporté :
    - la machine distante peut être en possession des données, celle du client, non ;
    - la machine distante peut disposer d'une puissance de calcul supérieure (attention, cela ne suffit pas : il faut également tenir compte de la rapidité du débit entre les deux machines)
    - la machine distante dispose de logiciels plus adaptés au traitement des données.

Service Web : principes de base

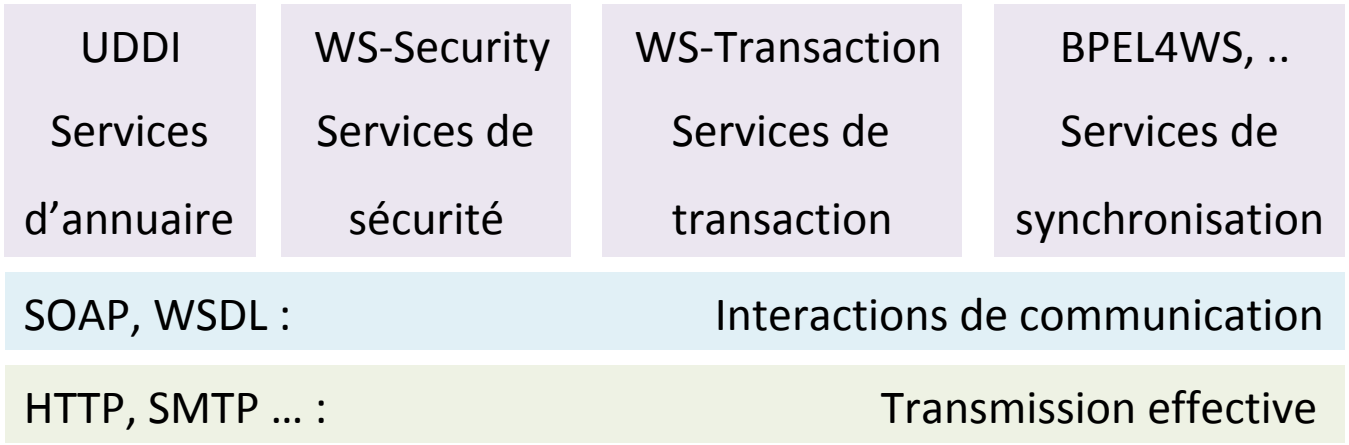
- HTTP ne sait transporter que du texte (HTML…)
- Les échanges (requêtes et réponses) sont donc au format texte
- Le format texte pour représenter les informations est **XML**
- Les messages sont au format **XML**.

Services Web : résumé

- Service Web : kesako ?
  - mettre des ressources à disposition (gratuites ou non) sur Internet, via un protocole d'échanges standardisé, pour des programmes écrits dans des langages quelconques.
  - Il faut :
    - un encodage (toujours XML) ;
    - un transport (souvent HTTP) ;
    - une organisation des requêtes et des réponses.
  - La procédure de fonctionnement d'un service Web est la suivante :
    - le service Web définit un format pour les requêtes et les réponses ;
    - un ordinateur demandeur effectue une requête ;
    - le service Web effectue une action, et renvoie la réponse à l'ordinateur demandeur.
  - Un service Web peut par exemple :
    - récupérer un cours de bourse
    - faire une demande automatiquement mise à jour d'un prix ;
    - accéder à un calendrier universel faisant les conversions entre calendriers internationaux et connaissant, pour chaque pays, les dates des jours fériés ;
    - traduire un passage de texte
    - valider un code postal international...
  - Pour pouvoir utiliser un service Web, plusieurs étapes sont nécessaires :
    - il faut savoir le trouver...
    - ... puis connaître la méthode pour y accéder...
    - ... enfin savoir l'utiliser correctement.

Services Web

- Quelques protocoles (cf. plus loin)
  - SOAP : utilise XML pour représenter les données
  - WSDL : utilise XML, XML Schéma



Services Web : XML

- XML ?
  - *eXtensible Markup Language*
  - Langage de balisage (comme HTML*, mais **générique**)
  - On crée ses **propres balises** et **ses propres règles**. On invente son dictionnaire et sa grammaire. Ex. docx, format XML (***Office Open XML ?***)
  - C'est une recommandation du W3C
  - Syntaxe stricte, facile à mettre en œuvre
  - Facile à lire
  - Interopérabilité
  - Destiné à être interprété par un programme pour :
    - Afficher quelque chose dans une page Web
    - Commander une machine (ex. bras robotisé, ***regarder par ex. http://air.imag.fr/images/6/6a/RenduprojetRobot.pdf***)
    - Stocker des informations pour mieux les échanger (ex. docx)
    - Etc.

Services Web : XML

- XML
  - est né suite aux limites avérées du HTML (impossibilité de créer ses propres balises, langage trop restrictif… d'où la naissance plus tard du XHTML)
  - But de XML : étendre les fonctionnalités de HTML
  - XML est un héritier (une simplification) de SGML (*Standard Generalized Markup Language*)
  - XML permet la **vérification de la structure** d'un document via une **grammaire type** définie dans une **DTD** (*Document Type Declaration*)
  - XML est un **langage de description et d'échange de données structurées**

Services Web : XML

- XML : 5 principes
  - Lisible pour la machine comme pour l'humain
  - Définition sans ambiguïté du contenu d'un document
  - Définition sans ambiguïté de la structure d'un document
  - Séparation entre documents et relations entre documents
  - Séparation entre structure du document et présentation du document

Services Web : XML

- XML : exemple

Quand décide-t-on de mettre une balise ou bien un attribut ?

```
<bibliothèque>
    <livre année="2002">
        <titre>Prélude à fondation</titre>
        <auteur>
            <nom>Asimov</nom>
            <prénom>Isaac</prénom>
        </auteur>
        <éditeur>Pocket</éditeur>
        <prix>6</prix>
    </livre>
    <livre année="1995">
        <titre>La marche des millénaires</titre>
        <auteur>
            <nom>Asimov</nom>
            <prénom>Isaac</prénom>
        </auteur>
        <auteur>
            <nom>White</nom>
            <prénom>Frank</prénom>
        </auteur>
        <éditeur>Flammarion</éditeur>
        <prix>8.45</prix>
    </livre>
    <livre année="2005">
        <titre>Avant Dune</titre>
        <auteur>
            <nom>Anderson</nom>
            <prénom>Kevin J.</prénom>
        </auteur>
        <auteur>
            <nom>Herbert</nom>
            <prénom>Brian</prénom>
        </auteur>
        <éditeur>Pocket</éditeur>
        <prix>12.3</prix>
    </livre>
</bibliothèque>
```

Source: Sylvie CALABRETTO, Tiphaine ACCARY. XML, Gestion de contenu Web

Services Web : XML

- document XML *bien formé* : respecte les règles de la grammaire
- règles d'un document bien formé
  - toute balise ouverte doit être fermée, ex: <livre> </livre>
  - les balises sont correctement imbriquées. Ex : Entrelacement mal formé   ….
  - les valeurs d'attributs sont entre guillemets
- les balises autofermantes correspondent à des documents vides et sont notées: Ex : `<img source="image.gif" />`
- les caractères < & sont notés < &
- un document commence par une déclaration XML
  `<?xml version="1.0" encoding ="UTF-8" standalone="yes" ?>`
- Cet encodage autorise l'utilisation des accents dans les noms des éléments (et des attributs), mais, dans la pratique, on les évitera pour des raisons de compatibilité

Services Web : XML

- XML : ***namespaces***
  - Les documents XML contiennent donc des balises qui codifient des grammaires et un certain vocabulaire selon une certaine spécialité (la géographie – le géocodage, par exemple – la biologie, etc.)
  - XML permet une **interopérabilité** qui suppose que plusieurs DTD peuvent se retrouver dans un même document XML (contenant une agrégation des documents XML d'origine).
  - Ceci suppose qu'il n'y a pas d'ambiguïté dans le nommage des différents éléments de chaque document.
  - Les domaines de noms XML (*namespaces*) sont une recommandation du W3C pour résoudre le problème du conflit de noms
  - **Principe** : on *préfixe* chaque nom d'élément d'un *nom de domaine unique* auquel il fait référence.

Services Web : XML

- XML : ***namespaces***
  - Exemple de déclaration d'un élément
    - sans préfixe : `head`
    - avec préfixe : `html:head`
  - Un espace de noms est déclaré via l'attribut xmlns
    - Soit en déclarant l'espace de noms dans l'élément
    - Soit en associant un préfixe pour une utilisation plus fine.

Services Web : namespaces

- Le **mélange de plusieurs vocabulaires** est illustré par l'exemple suivant
  - Pour insérer des métadonnées dans des documents, il faut disposer d'éléments pour les présenter.
  - Il existe déjà un **standard**, Dublin Core, pour organiser ces métadonnées.
  - `dc` comprend une quinzaine d'éléments dont `title`, `creator`, `subject` et `date`
  - Il est préférable d'utiliser ce vocabulaire, plutôt que d'introduire un nouveau vocabulaire.
  - Voici le document principal d'un livre au format DocBook
  - Les métadonnées sont contenues dans un élément `metadata`.
  - Celui-ci contient plusieurs éléments du Dublin Core (préfixe dc)
  - L'élément `include` de `XInclude` fait partie d'un autre espace de noms marqué par le préfixe `xi`

Services Web : namespaces

```
<?xml version="1.0" encoding="iso-8859-1"?>
<book version="5.0" xml:lang="fr"
      xmlns="http://docbook.org/ns/docbook"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:xi="http://www.w3.org/2001/XInclude">

 <!-- Titre DocBook du document -->
 <title>Langages formels, calculabilité et complexité</title>

 <!-- Métadonnées -->
 <metadata>
   <dc:title>Langages formels, calculabilité et complexité</dc:title>
   <dc:creator>Olivier Carton</dc:creator>
   <dc:date>2008-10-01</dc:date>
   <dc:identifier>urn:isbn:978-2-7117-2077-4</dc:identifier>
 </metadata>

 <!-- Import des chapitres avec XInclude -->
 <xi:include href="introduction.xml"  parse="xml" />
 <xi:include href="chapter1.xml"      parse="xml" />
 <xi:include href="chapter2.xml"      parse="xml" />
 <xi:include href="chapter3.xml"      parse="xml" />
 <xi:include href="chapter4.xml"      parse="xml" />
 <index />
</book>
```

Services Web : namespaces

- Analyse du document précédent :
  - Les espaces de noms évitent les conflits de noms entre différents vocabulaires.
  - Le dialecte DocBook dispose d'un élément title de même nom que l'élément `title` du Dublin Core.
  - Ces deux éléments ne sont pas confondus car l'élément `title` du Dublin Core a le préfixe `dc`.

Services Web : DTD

- DTD : *Document Type Declaration*
  - Définit la **structure** et les **éléments** et **attributs autorisés** d'un document XML
  - Permet de **valider la conformité** d'un document XML par rapport à sa définition
  - Exemple 1 : document XML :

```
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend</body>
</note>
```

  - DTD associée :

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Services Web : DTD

- Exemple 2 :

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes" ?>

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S8</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
    <modele>iPhone 8</modele>
  </telephone>

  <telephone>
    <marque>Nokia</marque>
    <modele>2</modele>
  </telephone>
</boutique>
```

Services Web : DTD

- Exemple 2 : DTD associée

```
<!ELEMENT boutique (telephone*)>
<!ELEMENT telephone (marque, modele)>
<!ELEMENT marque (#PCDATA)>
<!ELEMENT modele (#PCDATA)>
```

  - ***Que signifie l'étoile après « téléphone » ?***

  - En s'aidant des pages de www.w3schools.com (https://www.w3schools.com/xml/xml_dtd_intro.asp, notamment les DTD Elements, Attributes, etc.), ***écrire la DTD du fichier XML écrit dans l'exercice 1 du TP3***

Services Web : DTD

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<repertoire>
    <!-- Jean AYMAR -->
    <personne sexe="masculin">
        <nom>AYMAR</nom>
        <prenom>Jean</prenom>
        <adresse>
            <numero>12</numero>
            <voie type="rue">rue des roses</voie>
            <codePostal>75015</codePostal>
            <ville>PARIS</ville>
            <pays>FRANCE</pays>
        </adresse>
        <telephones>
            <telephone type="fixe">01 02 03 04 05</telephone>
            <telephone type="portable">06 07 08 09 10</telephone>
        </telephones>
        <emails>
            <email type="personnel">jean.aymar@gmail.com</email>
            <email type="professionnel">jean.aymar@societe.com</email>
        </emails>
    </personne>
```

Services Web : DTD

```
<!- Emilie JOLIE -->
    <personne sexe="feminin">
        <nom>JOLIE</nom>
        <prenom>Emilie</prenom>
        <adresse>
            <numero>43</numero>
            <voie type="boulevard">boulevard Haussmann</voie>
            <codePostal>17000</codePostal>
            <ville>LA ROCHELLE</ville>
            <pays>FRANCE</pays>
        </adresse>
        <telephones>
            <telephone type="bureau">04 05 06 07 08</telephone>
        </telephones>
        <emails>
            <email type="professionnel">contact@emilie-jolie.fr</email>
        </emails>
    </personne>
</repertoire>
```

Services Web : DTD

- 2 façons de déclarer une DTD, donc 2 types de DTD : interne ou externe
  - *Interne* : elle est située au sein du document XML, dans le DOCTYPE. Exemple :

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes" ?>

<!DOCTYPE boutique [
  <!ELEMENT boutique (telephone*)>
  <!ELEMENT telephone (marque, modele)>
  <!ELEMENT marque (#PCDATA)>
  <!ELEMENT modele (#PCDATA)>
]>

<boutique>
  <telephone>
    <marque>Samsung</marque>
    ...
</boutique>
```

Services Web : DTD

- *Externe* : la DTD est stockée dans un fichier à part, dont l'extension est `.dtd` Très intéressant pour la modularité notamment, càd si la DTD est commune à plusieurs fichiers XML
  - **DTD externe PUBLIC** : utilisée lorsque la DTD est une norme.
    - Syntaxe : `<!DOCTYPE racine PUBLIC "identifiant" "url">`
    - Exemple dans les documents XHTML 1.0 :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//
EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

  - **DTD externe SYSTEM** : utilisée dans le cas contraire
    - Syntaxe : `<!DOCTYPE racine SYSTEM "uri">`
    - Exemple :

```
<!DOCTYPE boutique SYSTEM "doc1.dtd">
```

Services Web : DTD

- **DTD externe SYSTEM** : Exemple complet

```
<?xml version = "1.0" encoding="UTF-8" standalone="no" ?>

<!DOCTYPE boutique SYSTEM "doc1.dtd">

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>
        ...
</boutique>
```

Services Web

- DTD intéressantes mais **insuffisantes** et ont des **faiblesses** :
  - Sont assez pauvres : **ne supportent pas les espaces de noms** donc impossible d'importer des schémas externes
  - et elles s'écrivent **dans un format qui n'est pas XML** : il faut donc apprendre une **nouvelle syntaxe** et on **ne peut pas utiliser les outils** existants comme DOM ou SAX pour *parser* les DTD, comme on *parse* les documents HTML et XML

- *=> Schémas XML*
  - XML Schema est une alternative XML aux DTD
  - En fait, XML Schéma apparaît comme le successeur des DTD car il est par nature extensible et s'appuie sur XML

Services Web

- Ainsi XML Schema décrit (en XML) la structure d'un document XML c'est-à-dire :
  - les éléments qui composent un document,
  - les attributs,
  - la hiérarchie entre les éléments,
  - l'ordre des sous éléments,
  - le nombre de sous éléments,
  - les types des éléments et attributs,
  - les valeurs par défaut, le format ou la restriction des valeurs d'un élément ou d'un attribut.
- On parle ainsi de **XML Schema Definition** (XSD).

XML Schémas

Eléments de base : (s'aider du site w3schools: https://www.w3schools.com/xml/schema_intro.asp)

Les types simples les plus courants sont de types :

`xs:string` *(NB:* `xs` *est un préfixe de nommage. L'usage veut qu'on utilise* `xs` *ou* `xsd`*)*

`xs:decimal`

`xs:integer`

`xs:boolean`

`xs:date`

`xs:time`

exemple :

```
<xs:element name="Services Web" type="xs:string"/>
```

déclaration d'une valeur par défaut : `<xs:element name="code_postal" type="xs:string" default="93526"/>`

déclaration d'une valeur figée : `<xs:element name="universite" type="xs:string" fixed="Paris8"/>`

Les éléments de type simple **ne peuvent pas** contenir de sous-éléments.
Seuls les éléments **complexes** le peuvent.

XML Schémas

Eléments complexes :
la définition d'un élément complexe peut se faire directement au niveau de l'élément lui-même ou par référence au nom du type complexe (ce qui permet à plusieurs éléments de partager le même type complexe).

La définition se fait alors par l'utilisation du tag `xs:complexType`.

Un type complexe peut enrichir un autre type complexe on non (tag `<xs:extension base="type_de_base">`).

Un type complexe peut aussi en restreindre un autre (exemple `<xs:restriction base="xs:integer">`).

Il est possible de mélanger du texte libre avec des tags (exemple : bonjour<prenom>simon</prenom>) : `<xs:complexType mixed="true">`.

XML Schémas

- Une suite d'éléments complexes doit être déclarée dans un `<xs:sequence>`, `<xs:choice>` ou `<xs:all>` :

  - `<xs:all>` spécifie que les éléments fils peuvent apparaître une fois ou pas du tout, et dans n'importe quel ordre. Ce connecteur n'a pas d'équivalent dans une DTD. Seul le `?` permet de gérer l'occurrence (0 ou 1 fois) dans une DTD.
  - `<xs:sequence>` spécifie que les éléments fils peuvent seulement apparaître dans l'ordre mentionné. Ce connecteur a la même signification qu'une succession de déclaration d'éléments séparés par des virgules, dans une DTD.
  - `<xs:choice>` spécifie qu'un seul des éléments fils peut apparaître. Ce connecteur a la même signification que le | dans une DTD.

XML Schémas

Attributs :
Seuls les éléments complexes peuvent avoir des attributs. La déclaration des attributs par défaut ou fixe est identique aux éléments :

- rendre un attribut obligatoire : <xs:attribute name="email" type="xs:string" use="required"/>
- rendre un attribut facultatif : <xs:attribute name="url" type="xs:string" use="optional"/>

Restrictions sur les attributs ou éléments (voir exemples slides suivants) :

- Plage de valeurs : <xs:minInclusive value="minimum" /> et <xs:maxInclusive value="maximum"/>,
- Liste de valeurs : <xs:enumeration value="une_valeur" />,
- Conformité à un motif : <xs:pattern value="[A-Z][A-Z][A-Z]" /> ou <xs:pattern value="([az])*" />,

XML Schémas

- Traitement des espaces : <xs:whiteSpace value="preserve"/> (les espaces sont laissés tels quels). Autres valeurs : replace (remplacer les LF,CR, TAB... par des espaces) ou collapse (remplacer les CR,LF... mais aussi supprimer les espaces avant/ après et concaténer les successions d'espace en un seul),

- sur la longueur : <xs:length value="8"/> ou <xs:minLength value="5"/> et <xs:maxLength value="8"/>,

- il existe aussi des restrictions sur les décimales (fractionDigits et totalDigits)

XML Schémas

Exemples de restrictions (encore appelées *facets*)

- Sur des plages de valeurs numériques :

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML Schémas

- Sur des ensembles de valeurs :

```
<xs:element name="voiture">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Solution 2 (permet de **réutiliser** la restriction) :

```
<xs:element name="voiture" type="typeVoiture"/>

<xs:simpleType name="typeVoiture">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

XML Schémas

- Sur des motifs (expressions régulières) :
  Ex : on veut un code postal français

```
<xs:element name="CPFrancais">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9]{5}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Voir les regex de XML Schema ici :
https://www.regular-expressions.info/xml.html

XML Schémas

- Sur la longueur des chaînes de caractères :

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- NB : il y a un **ordre** dans la définition XSD : d'abord les éléments `<xs:element>` (ou `<xs:sequence>` ou `<xs:all>`…) puis les attributs `<xs:attribute>`

XML Schémas

- On peut également **créer un type complexe à partir d'un type simple**.
  - Par exemple, pour définir un élément contenant une valeur simple et un attribut, comme dans `<quantite unit="g">250</quantite>`
  - On va donc dériver un type complexe à partir d'un type simple :

```
<xs:complexType name="quantite">
  <xs:simpleContent>
    <xs:extension base="xs:positiveInteger">
      <xs:attribute name="unit" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

XML Schémas

- XML Schemas indispensables comme outil d'interopérabilité pour des applications réparties :
  - entre des applications web
  - dans des approches objets répartis comme SOAP ou WSDL
  - entre des bases de données hétérogènes

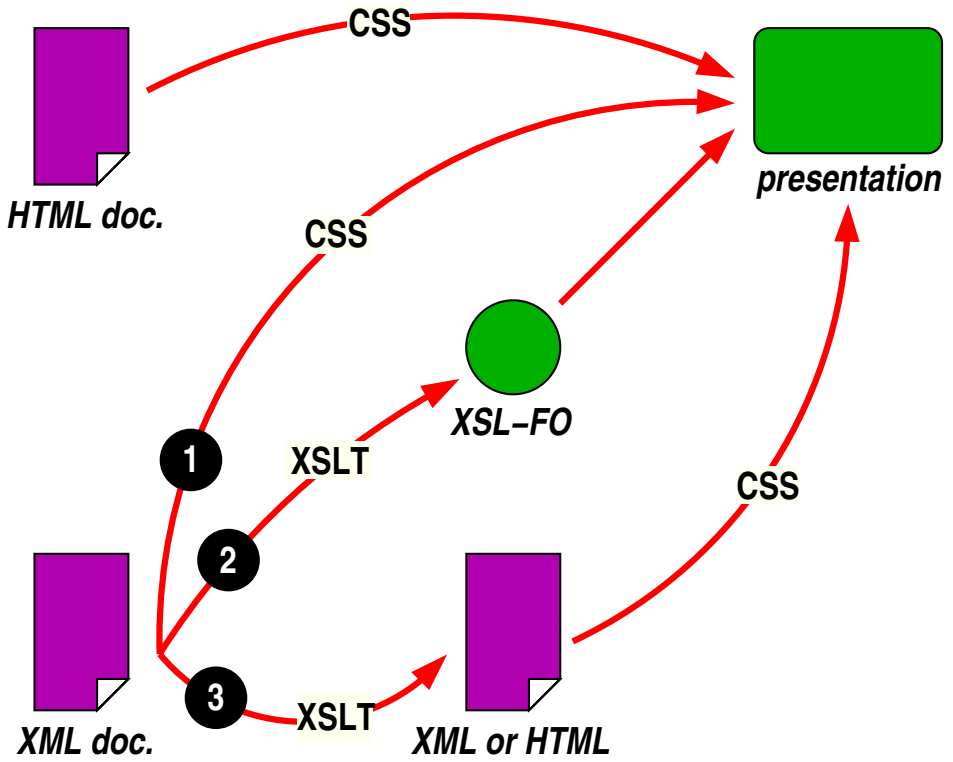
- Pour les normes, spécifications et recommandations du W3C, voir
  https://www.w3.org/XML/Schema

XSL ?

- XSL : ***eXtensible Stylesheet Language***
- langage évolué pour la définition de feuilles de style
- XSL n'est pas un simple langage de formatage de documents XML, comme CSS.
- C'est beaucoup plus que cela : il permet de retraiter un document XML, de **réarranger** sa structure.
- => XSL permet de transformer un document XML en un autre document, souvent XML, mais pas forcément, par exemple en HTML, TeX, RTF, PostScript, etc.

XSL

- 1 : le doc XML ne demande pas de transformation. Seul le CSS est utilisé.
- 2 : nécessité d'utiliser un sous-langage de XSL (FO : formatting objects)
- **3 : on génère via XSLT (voir plus loin) un XML ou HTML et on l'affiche dans un navigateur grâce à CSS**



Source : *https://www.w3.org/Style/CSS-vs-XSL.en.html*

XSL

- La feuille de style XSL est enregistrée dans un fichier externe et son nom comporte l'extension ".xsl"
- Dans le document XML, on indique ceci :

```
<?xml-stylesheet type="text/xml" href="URL"?>
```

- Structure d'une feuille de style XSL
  - XSL étant une application XML, une feuille de style XSL est … un document XML !
  - La feuille de style contient donc une déclaration XML et tous ses éléments sont placés dans l'élément racine. D'autre part, les éléments XSL sont préfixés par xsl: (XSL utilise les domaines de noms).
  - L'élément racine contient principalement des **modèles** (templates) pour l'affichage du document XML.

XSL

- chaque **modèle** contient des informations sur l'affichage d'une branche des éléments du document
- S'il n'y a qu'un seul modèle, alors il s'applique sur la racine du document XML
- Une feuille de style XSL comporte une déclaration XML, un élément racine `xsl:stylesheet` qui englobe tous les autres éléments et précise que les éléments préfixés par `xsl:` appartiennent au domaine de nom xsl.
- Le modèle est appliqué à la **branche spécifiée par l'attribut `match`** de l'élément `template`.
- En CSS, la valeur de l'attribut `match` correspondrait au **sélecteur de la règle**.

XSL

- Souvent, la branche indiquée est la racine du document XML (ne pas confondre avec **l'élément racine** qui, lui, est un **enfant de la racine du document**)

- La transformation d'un document XML par une feuille de style XSL s'effectue donc par un **modèle traitant un nœud donné**

- Chaque modèle est divisé en deux parties : un nœud cible indiqué par l'attribut match et une action sur le nœud :

```
<xsl:template match="noeud_cible">
      action (par exemple : <html>, etc.)
</xsl:template>
```

XSL

- Un modèle contient deux types d'éléments :
  - des éléments XML bien formés pour représenter les éléments html ;
  - des éléments XSL
    - Par ex., `xsl:value-of` ou `xsl:for-each`
    - qui permettent d'accéder au contenu des éléments du document XML
    - l'attribut `select` indique le nom de l'élément XML (à partir de l'élément courant) auquel on veut accéder.

- On peut classer les éléments avec `xsl:sort`
- On peut filtrer les éléments avec une syntaxe crochets (et @ s'il s'agit d'un attribut)
  - Par ex. :

```
<xsl:for-each
        select="catalog/cd[type='reggae']">
```

XSL / XSLT

- Pour que la transformation se fasse, il faut également préciser quelle sortie on veut obtenir :
  - Par exemple, une sortie vers un fichier HTML :

```
<xsl:output method="html" version="4.0" encoding="UTF-8" indent="yes"/>
```

- La feuille de style XSL est enregistrée dans un fichier externe et son nom comporte l'extension ".xsl"
- Dans le document XML, on indique ceci :

```
<?xml-stylesheet type="text/xml" href="URL"?>
```

- La transformation se fait en utilisant un fichier xsl et un fichier xml. On parle alors de XSLT (XSL transformations) et on utilise (par exemple, sous Linux) : **`xsltproc`**
- **L'interopérabilité est donc parfaitement atteinte puisque l'on peut automatiser les traitements de transformation des fichiers (XML) sous divers formats**

Services Web ou API ?

- Service Web vs. API
  - D'un point de vue générique, un service web est une API (une API qui a la particularité d'utiliser le protocole HTTP) ; et une API web serait alors le mécanisme que connaissent les développeurs (librairies, modules, classes, etc.) mais appliqué au protocole HTTP.
  - Mais comment une interface (API) peut elle être en même temps un protocole (HTTP) ?
  - Une interface protocolaire est un protocole de transport qui devient un protocole de transfert
  - **Une API web peut donc être comprise comme une "interface protocolaire"**. Le génie du web est d'avoir permis une **architecture qui supporte cette approche qui fait du protocole de transfert une interface**.

Services Web ou API ?

- Service Web vs. API
  - C'est donc le **protocole de transfert qui prime dans une API web**.
  - On peut aller plus loin : les protocoles de transports deviennent des protocoles de transfert *quand ils acquièrent un statut d'interface*.
  - C'est cette sublimation d'un protocole de transport (TCP/IP) en une interface (uniforme) qui donne des protocoles de transfert.

Services Web ou API ?

- API publique
  - Ceux qui vendaient des architectures de services vendent à présent des architectures d'API.
  - Mais il y a maintenant quelque chose de différent en ce sens que c'était généralement les mêmes personnes qui mettaient en place les services web et qui les utilisaient. C'était la même équipe qui étudiait les besoins de communication entre 2 ou n applications et mettaient en place les services web appropriés. Avec l'approche par les API, la démarche est différente en ce sens que ce n'est pas forcément la même équipe de développeurs qui produit les services web et les utilise ; ce fait devient une évidence quand on ne parle plus d'API privées et internes mais d'API publiques.
  - Ex Google APIs

Services Web ou API ?

- Contrat
  - Tout transfert est un contrat qui lie deux parties.
  - Une API sous-entend un contrat, tout comme un service Web.

  - On peut voir les contrats comme un regroupement d'assertions sur les propriétés d'un programme.
  - Chacune d'elle constitue alors une **expression booléenne à satisfaire** pour son exécution

Services Web ou API ?

- Résumé
  - D'un point de vue général, un service web est une API (qui a la particularité d'utiliser le protocole HTTP) ; et une API web serait alors le mécanisme de librairie (module, classe) bien connu des développeurs, mais appliqué au protocole HTTP.
  - Mais HTTP, en tant que protocole de transfert, est également qualifié d' « Interface » (cf. Roy Fielding, avec REST – plus loin)
  - Ce qui est troublant, c'est donc que ce qui est explicitement un protocole (HTTP) soit en même temps qualifié d'interface. En effet, dans le modèle OSI, le terme d'interface était réservé à la communication entre des protocoles de niveaux différents sans qu'il n'y ait de protocole de transport utilisé.
  - Si les interfaces n'étaient pas des protocoles de communications, c'était parce que l'on considérait que le passage d'un niveau de protocole à un autre se faisait au sein de la même machine.
  - Il y a aussi une question de *mode* (la mode « API » a dépassé la mode « Services Web »… pour l'instant…)

Services Web ou API ?

- Services Web ou API ?
  - Qui utilise les Services Web et qui les développe ?
    - En principe, les mêmes équipes
  - Qui utilise les API et qui les développe ?
    - ce n'est pas forcément la même équipe de développeurs
    - on ne parle plus d'API privées et internes mais **d'API publiques**
  - D'où le focus mis sur les bonnes pratiques : simplicité, bonne documentation…