Infrastructures du Commerce Électronique (NFE102)

2 – Echanges de données sur le Web

Auteurs: Raphaël Fournier-S'niehotta, Philippe Rigaux et Nicolas Travers
(rigauxp,fournier,traversn @ cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Table of Contents

Table of Contents

XML en bref

XML est un standard défini par le World-Wide-Web Consortium (W3C).

- Les documents XML peuvent être sérialisés dans un encodage normalisé (typically iso-8859-1, ou utf-8), et transmis sur le réseau sans perte d'information.
- XML est un format générique qui peut être spécialisé en "dialectes" conçus pour des domaines spécifiques (e.g., XHTML).
- Le W3C normalise des standards associés: DOM (le "modèle" XML), XSchema (typage), XPath (chemins), XSLT (restructuration), XQuery (interrogation), et beaucoup d'autres.

Remarque

1. XML est une version simplifiée de SGML, utilisé depuis longtemps pour les documents techniques.
2. HTML, jusqu'à la version 4.0, est aussi une variante de SGML. XHTML est un dialecte XML.

Forme sérialisée, forme arborescente

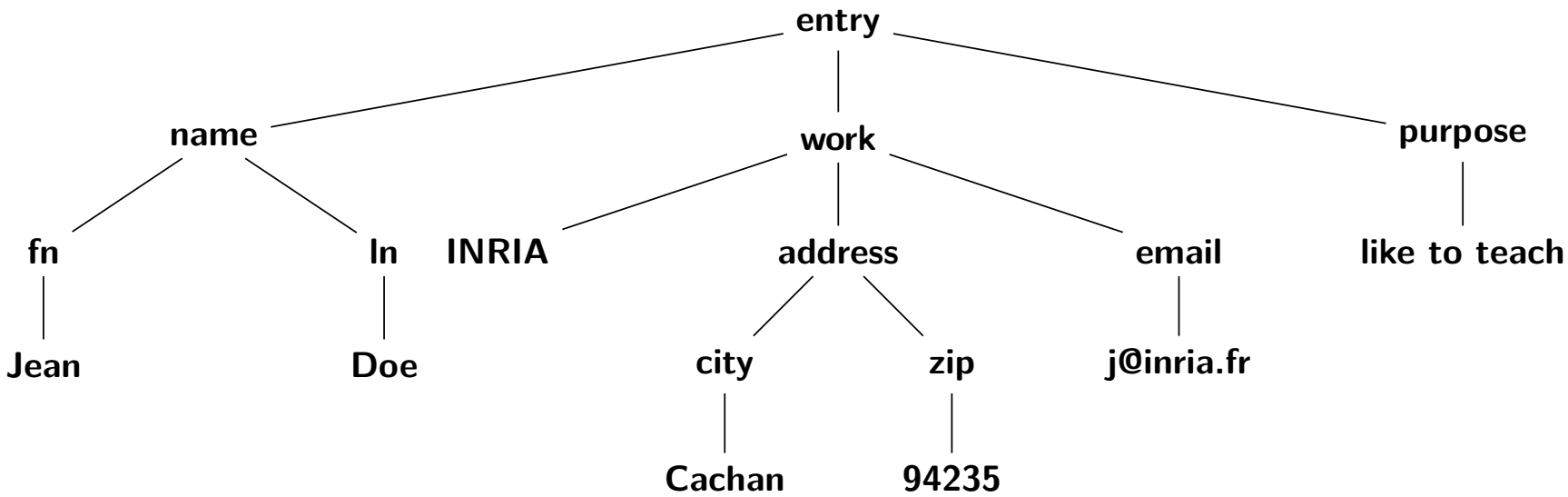
L'application :

1. Reçoit un document sous forme sérialisée,
2. L'analyse (parsing) pour le manipuler sous forme arborescente
3. Puis le sérialise pour le sauvegarder.

- La forme sérialisée est un codage séquentiel de l'arbre qui obéit à une syntaxe particulière.
- La forme arborescente est définie par un modèle, le DOM.

Les documents XML sont des arbres

Les applications manipulent les documents XML comme des arbres ordonnés.



Représentation *sérialisée* d'un document XML

La forme sérialisée est la plus connue pour XML :

```
<entry><name><fn>Jean</fn><ln>Doe</ln></name><work>INRIA<adress><city>
Cachan</city><zip>94235</zip></adress><email>j@inria.fr</email>
</work><purpose>like to teach</purpose></entry>
```

La structure hiérarchique est partiellement rendue par l'indentation:

```
<entry>
   <name>
      <fn>Jean</fn>
      <ln>Doe</ln> </name>
   <work>
      INRIA
      <address>
         <city>Cachan</city>
         <zip>94235</zip> </address>
      <email>j@inria.fr</email> </work>
   <purpose>like to teach</purpose>
</entry>
```

XML describes structured content

Un texte sans aucune stucture est difficile à interpréter pour une application.

```
The book "Fundations of Databases", written by Serge Abiteboul,
Rick Hull and Victor Vianu, published in 1995 by Addison-Wesley
```

XML permet d'introduire de la structure.

```
<bibliography>
  <book>
    <title> Foundations of Databases </title>
    <author> Abiteboul </author>
    <author> Hull </author>
    <author> Vianu </author>
    <publisher> Addison Wesley </publisher>
    <year> 1995 </year> </book>
  <book>...</book>
</bibliography>
```

On peut accéder aux auteurs, réorganiser le contenu, etc.

Table of Contents

Un document XML I

Un document XML est composé de :

1. Un prologue
   1. Déclaration de document XML `<?xml version = "1.x" [encoding = "norme" ]? >`
   2. Suivie éventuellement (sans contrainte d'ordre)
      - D'une déclaration ou (exclusif) une référence à une DTD
        `<!DOCTYPE element racine [schema racine ] >` ou
        `<!DOCTYPE element racine SYSTEM "uri schema racine" >`
      - De commentaires `<!-- texte -->`
      - D'instructions de traitement (par ex. appel à XSLT)
        `<?instruction attribut1 = "valeur" ... attribut n ="valeur" ? >`
2. Un corps :
   - Une imbrication d'éléments et de textes
   - Des commentaires

Quelques exemples de base

Remarque: les retours à la ligne n'ont aucune signification.

```
<document/>

<document> Hello World! </document>

<document>
  <salutation> Hello World! </salutation>
</document>

<?xml version="1.0" encoding="utf-8" ?>
<document>
  <salutation color="blue"> Hello World! </salutation>
</document>
```

Le dernier exemple est le seul complètement correct : il comprend le prologue.

Forme hiérarchique des documents : le modèle DOM

Un document XML est interprété comme un arbre.

Cette interprétation est fixée par un modèle, le DOM (Document Object Model) normalisé par le W3C.

Dans le DOM, chaque nœud de l'arbre a un type, et une description. Par exemple :

1. le nom du nœud,
2. la valeur du nœud.
3. le contenu d'un nœud.

Remarque

La description exacte dépend du type du nœud.

Principaux types : Element et Text

Les nœuds de type **Element** correspondent au balisage dans la forme sérialisée.

1. ils définissent la structure du document.
2. ils ont un nom mais pas de valeur.
3. ils ont un contenu : le sous-arbre dont ils sont racine.

Les nœuds de type **Text** correspondent au contenu du document.

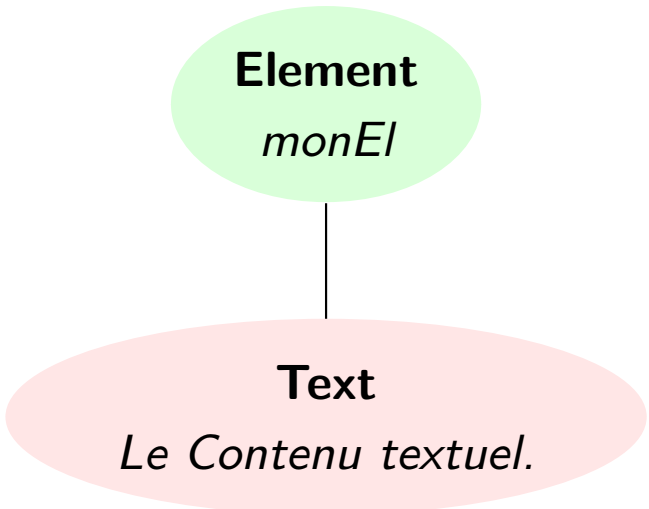
1. ils constituent les feuilles de l'arborescence (un nœud **Text** n'a pas de fils) et n'ont donc pas de contenu.
2. ils ont une valeur mais pas de nom

Correspondance forme sérialisée - forme DOM

Voici un élément contenant du texte.

```
<monEl>
  Le contenu textuel.
</monEl>
```

Ce document représenté en DOM: chaque nœud a un **type**, soit **Document** soit **Text**.



Imbrication des éléments

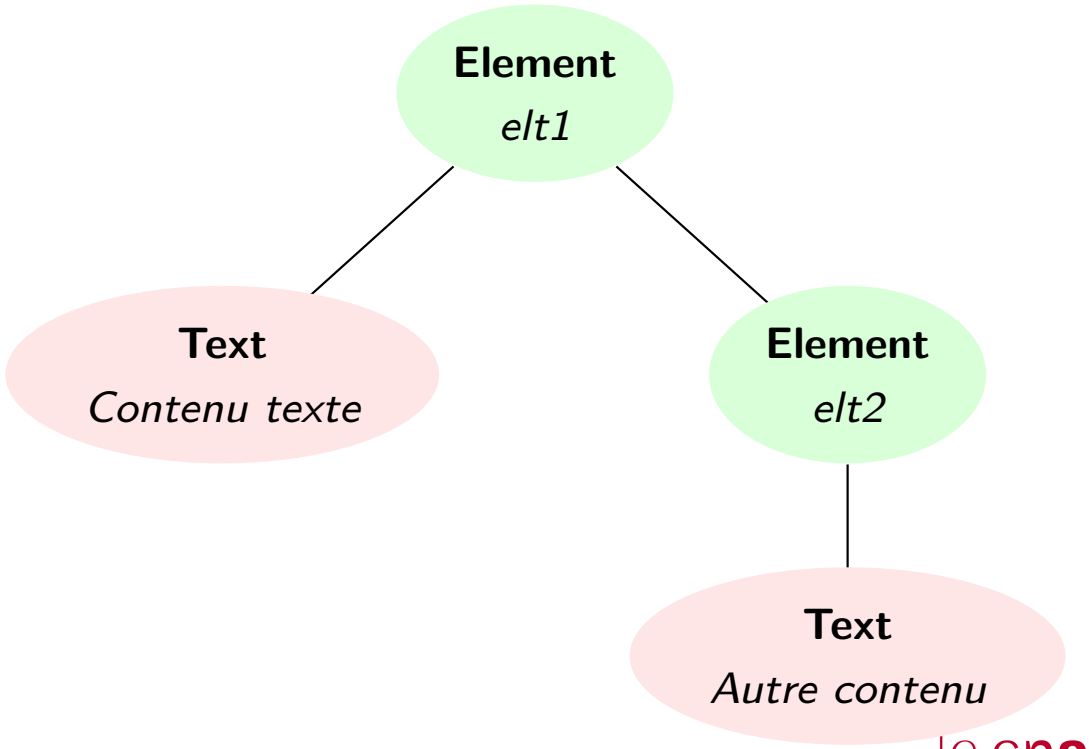
Le contenu d'un élément est

1. tout ce qui est compris entre la balise ouvrante et la balise fermante (dans la forme sérialisée).
2. le sous-arbre dont le nœud est la racine (dans la forme DOM).

Ce contenu comprend d'autres élément, des nœuds **Text**, et autres gadgets de moindre importance (commentaires, etc.)

Exemple d'éléments imbriqués

```
<elt1>
  Contenu texte
  <elt2>
    Autre contenu
  </elt2>
</elt1>
```



Quelques remarques en vrac (anecdotique)

- Les sous-éléments sont ordonnés.
- Pas de caractères spéciaux (mais "- _ . /" autorisés) dans les noms des éléments, pas d'espaces.
- Un élément peut être vide: <nom_elt></nom_elt>
  Notation abrégée : <nom_elt/> (Exemples d'éléments vides (XHTML): <br/> ou)
- (Contrairement à HTML,) XML est *case sensitive* donc <genie> $\neq$ <Genie>
- Les caractères spéciaux non autorisés peuvent être remplacés par une référence:
  & doit être remplacé par sa référence & (version numérique : &#28;)
  < devient < (ou <)
  > devient > (ou >)
  " devient " (ou ')
  ' devient ' (ou ")
- Il n'y a qu'un seul élément racine
- Toute balise ouverte doit être fermée et sont contenu bien formé (ouvert/fermé)
- Pour vérifier la validation du document :
  http://www.w3schools.com/xml/xml_validator.asp

Un peu plus loin : les attributs

Les attributs sont des paires clé/valeur attachées à un élément.

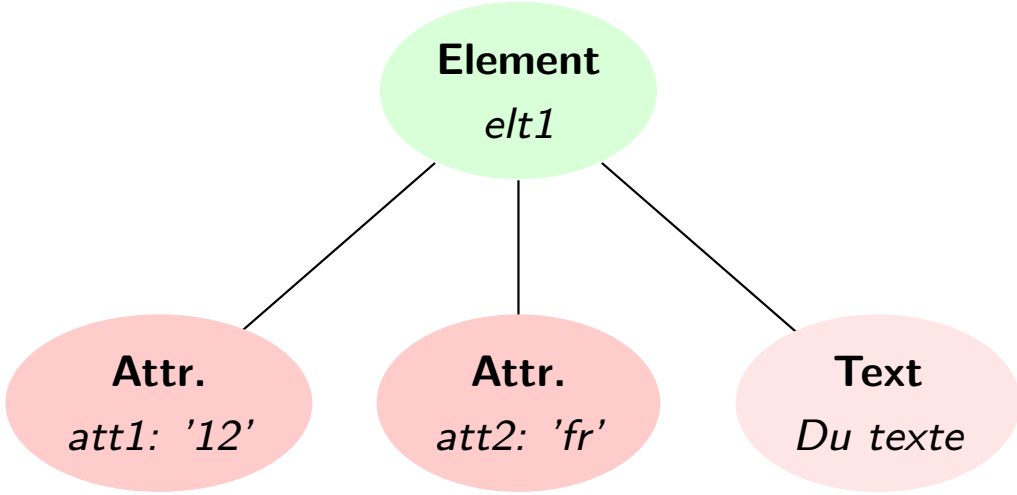
1. on les place dans la balise ouvrante (forme sérialisée) ;
2. ce sont des noeud-fils spéciaux du nœud Element dans la forme DOM.

Un attribut a une valeur : un chaÃőne de caractères.

Un élément avec deux attributs.

```
<elt1 att1='12' att2='fr'>
  Du texte
</elt1>
```

Les attributs ne sont par ordonnés, et on ne peut pas avoir deux attributs avec le même nom dans un élément.



Autre exemple I

```
<les_genies>
   <genie date_naiss="1912-06-23" date_deces="1954-06-07">
      Alan Turing
   </genie>
   <genie date_naiss="1906-04-28" date_deces="1978-01-14">
      Kurt Godel
   </genie>
</les_genies>
```

- `date_naiss` est un attribut du premier élément `<genie>`
- `1912-06-23` est sa valeur

On aurait très bien pu utiliser des noeuds **Text** plutôt que des attributs.

Remarque

Les attributs sont (en principe) utilisés pour des "méta-données" qualifiant le contenu (auteur, date de création). Peu important.

Pour compléter l'essentiel : la racine du document

Un document XML sous forme sérialisée commence toujours par un prologue :

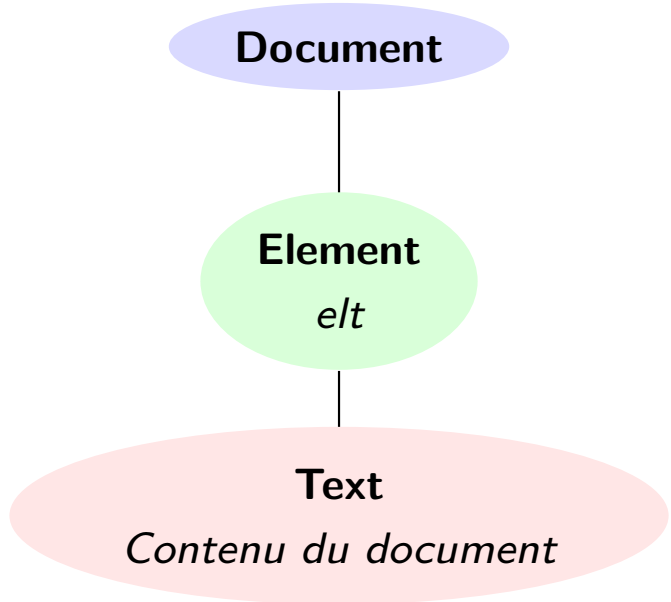
```
<?xml version="1.0"  encoding="utf-8" ?>
```

et le contenu d'un document est toujours contenu dans un unique élément, appelé l'élément racine.

Un document avec son prologue et l'élément racine.

```
<?xml version="1.0" encoding="utf-8" ?>
<elt>
  Contenu du document.
</elt>
```

Dans la représentation DOM, le prologue est représenté par un nœud **Document** appelé "*root node*".



Remarque

Le prologue peut contenir d'autres informations (DTD, etc.)

Table of Contents

Les entités et les références

Les entités sont des symboles qui désignent des fragments. Utile pour réutiliser du contenu, sans le répéter.

Une entité est déclarée (dans le prologue), puis référencée.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE a [
  <!ENTITY monNom "Charles Martel">
  <!ENTITY maSignature SYSTEM "signature.xml">
]>

<a>
  Mon nom est &monNom;.

  &maSignature;
</a>
```

Entités prédéfinies

Tout un ensemble de symboles qui sont interprétées comme du marquage / balisage si on les utilise directement.

Si on les veut littéralement, il faut utiliser des références d'entités.

Déclaration	**Référence**	**Symbole.**
`<!ENTITY lt "&#60;">`	`&lt;`	<
`<!ENTITY gt "&#62;">`	`&gt;`	>
`<!ENTITY amp "&#38;">`	`&amp;`	&
`<!ENTITY apos "&#39;">`	`&apos;`	'
`<!ENTITY quot "&#34;">`	`&quot;`	"

Sections litérales

Un analyseur (parseur) XML cherche à analyser tout le contenu pour détecter du marquage structurel.

Problème: et si on ne veut pas que le contenu soit analysé ?

```
<?xml version='1.0'?>
<program>
if ((i < 5) && (j > 6))
   printf("error");
</program>
```

Solution: soit on utilise des entités (lourd), soit on insère la texte à protéger dans une section littérale.

```
<?xml version='1.0'?>
<program>
<![CDATA[if ((i < 5) && (j > 6))
           printf("error");
]]>
</program>
```

Table of Contents

La validation I

Valider :

1. Associer un schéma permettant d'ajouter des contraintes sur les données :
   - noms des éléments,
   - sous-éléments possibles d'un élément, ordre d'apparition et nombre d'occurrences,
   - attributs d'un élément,
   - type : chaîne, type énuméré, clef, clef étrangère,
   - occurence : obligatoire ou non,
   - valeur : par défaut ou fixée,
   - etc.

   Pour les documents XML, les schémas sont exprimés sous la forme d'une grammaire.
2. Vérifier que le document est conforme à la grammaire.

Objectifs

Assurer une meilleure qualité des données transmises ou reçues[1].
Améliorer l'interopérabilité des applications.

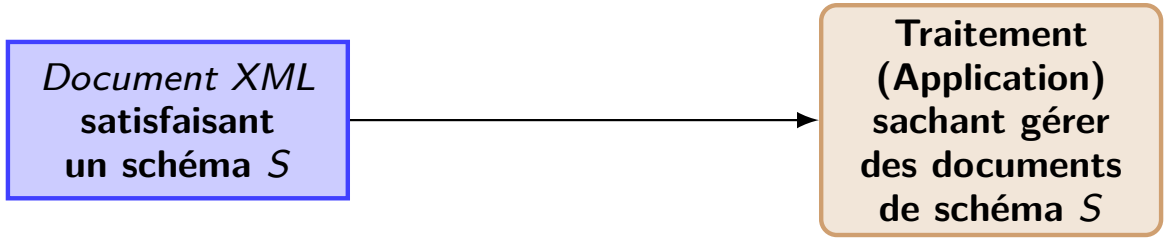
La validation II

Document valide

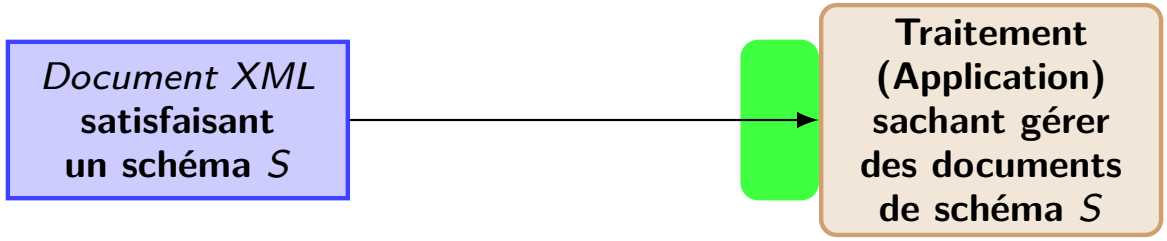
Un document est *valide* s'il respecte le schéma qui lui est associé.

[1] Évidemment, associer un schéma n'est généralement pas suffisant pour assurer une bonne qualité des données mais le schéma y participe

Valider un document XML I



Valider un document XML II



Une DTD en deux mots

Sous forme d'une grammaire :

- → simple à écrire et à comprendre,
- → rapide à écrire,
- → peu expressive.

```
<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (AUTHOR+,HEADLINE, BYLINE, LEAD, BODY, NOTES*)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT AUTHOR(#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>
```

Listing 1: Exemple simple d'une DTD

Définition d'un élément I

- Élément vide <!ELEMENT br EMPTY>
  ⇒
 ou
</br>
- Feuille <!ELEMENT auteur (#PCDATA)>
  ⇒<auteur>Victor Hugo</auteur>
- Sans contrainte sur les sous-éléments <!ELEMENT auteur ANY>
  ⇒ <auteur><prenom>Victor</prenom><nom>Hugo</nom></auteur>

Définition d'un élément II

Avec contrainte sur les sous-éléments

```
<!ELEMENT auteur (nom)>
<!ELEMENT auteur (nom?)>
<!ELEMENT auteur (livre*)>
<!ELEMENT auteur (nom, prenom)>      ← séquence
<!ELEMENT cours (magistral | projet)>      ← choix
<!ELEMENT autheur (nom, prenom?)>
<!ELEMENT livre (isbn,sommaire?)>
<!ELEMENT livre (isbn,titre,auteur+,critiques*)>
<!ELEMENT cours ((date,salle)+,(url|fichier+)+)>
```

cardinalités :

(vide)	: exactement 1 occurrence
*	: 0 à ∞ occurrences
+	: 1 à ∞ occurrences
?	: 0 ou 1 occurrence

Contraintes sur la DTD :
Il doit exister au plus une déclaration `<!ELEMENT ..>` par élément.
Tous les éléments doivent être définis.

Simples DTD avec cardinalités

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

Définition d'attributs

```
<!ATTLIST book
    ISBN CDATA #REQUIRED
    lang (fr|en) #IMPLIED
    subject CDATA #IMPLIED>
```

- Nom de l'élément
- Nom de l'attribut associé
- Type de valeur (CDATA, enumeration, ID, IDREF, IDREFS)
- Attribut obligatoire (REQUIRED par défaut) / optionnel (IMPLIED) / (FIXED)
- Valeur par défaut

Déclaration interne d'une DTD

la DTD est définie dans le document lui-même (élément racine).

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

<!DOCTYPE note [
        <!ELEMENT note (to,from,heading,body)>
        <!ELEMENT to (#PCDATA)>
        <!ELEMENT from (#PCDATA)>
        <!ELEMENT heading (#PCDATA)>
        <!ELEMENT body (#PCDATA)>
 ]>

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

Déclaration externe d'une DTD

la DTD est définie dans un autre document, appelé dans le document contenant les données.

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

La ligne 2 indique au processeur 1) qu'une DTD est associée et 2) où la trouver.

Table of Contents

XPath

XPath est un langage de requêtes

- non XML,
- permettant l'accès à des parties d'une donnée XML via l'expression de chemin menant à un ensemble de nœuds (sous-arbres) d'un document XML.

XPath est utilisé dans :

XQuery	XPath fait partie du langage d'interrogation XQuery,
XSLT	pour la sélection de la partie des données à transformer,
XML Schema	pour les expressions de contraintes d'unicité,
XPointer	pour l'identification de fragments,
XLink	pour ancrer les liens hypertextes,
Services Web	chercher dans une ressource,
...	

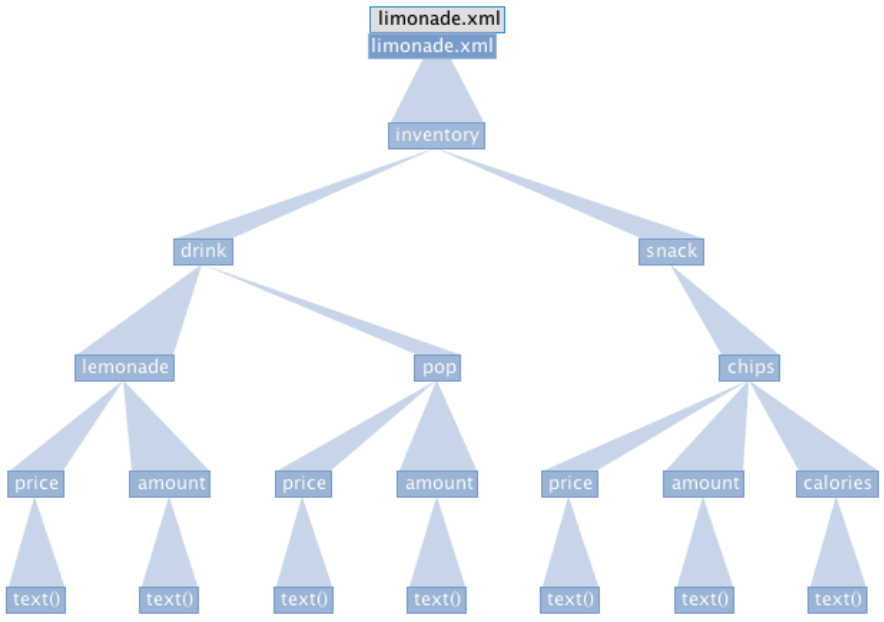
Une intuition

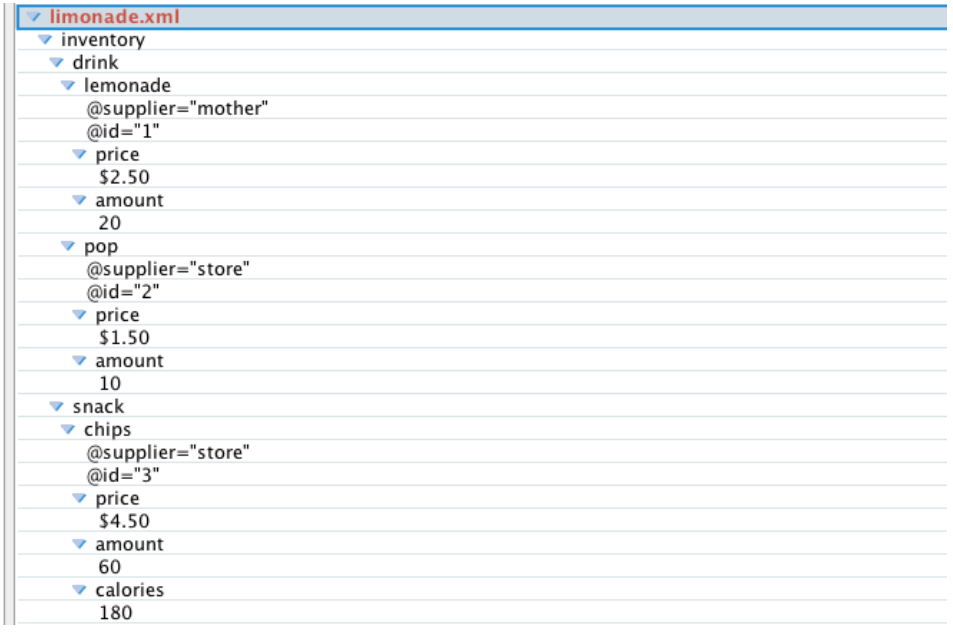
Permet de “pointer” (localiser) des sous-arbres d’un document XML.

Intuition

Par navigation : en navigant dans l’arbre de ”pas en pas” à partir de la racine, on va chercher à atteindre certaines de ses parties.

Quelques exemples simples de chemins





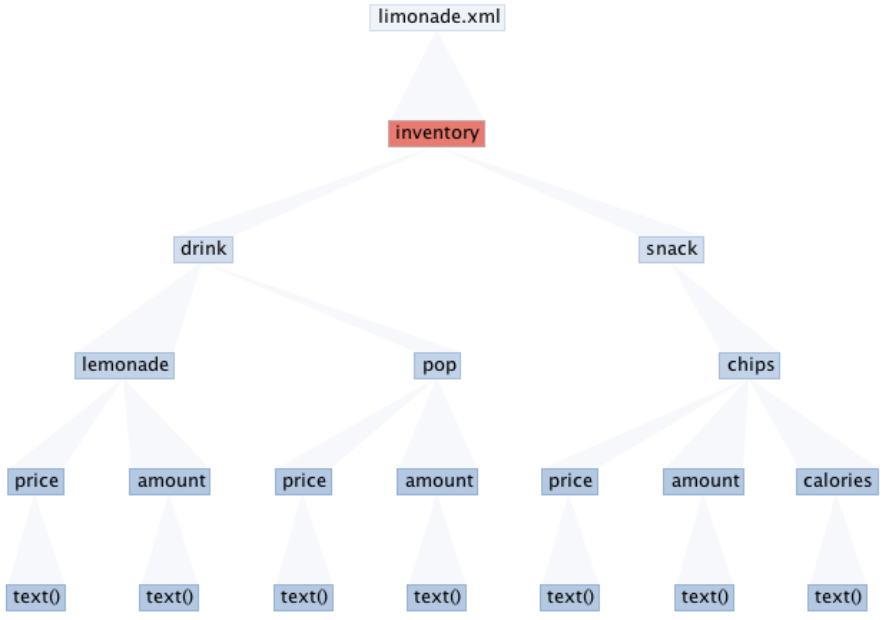
Évaluer les requêtes suivantes, sur le document XML se trouvant ci-dessus (vue BaseX[2])

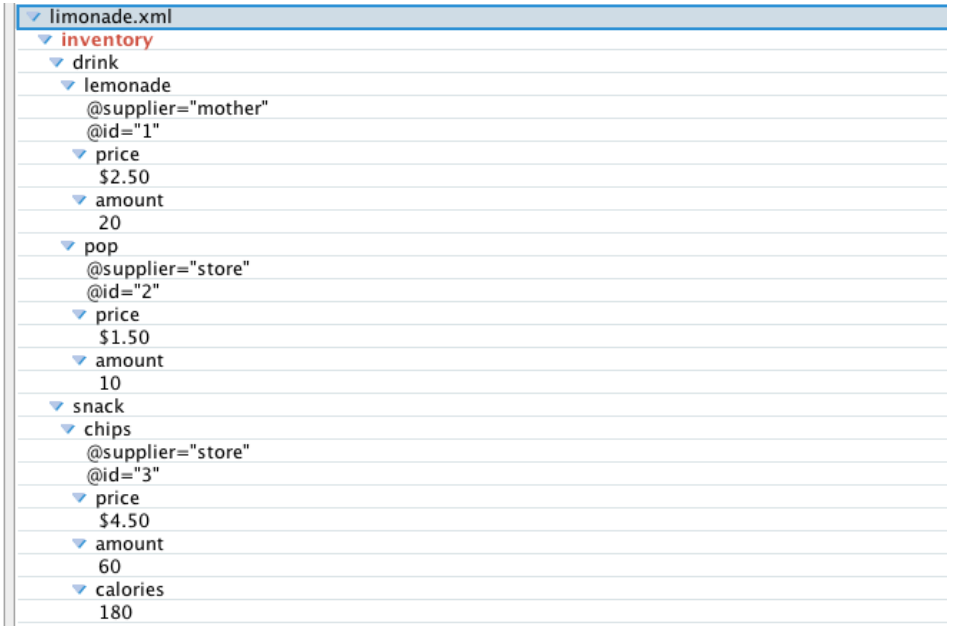
1) `/inventory`
2) `/inventory/drink`
3) `/inventory/drink/*`
4) `/inventory/drink/lemonade`
5) `/inventory/drink/*/price`
6) `/inventory/drink//@supplier`

[2] http://www.basex.org

le cnam

Quelques exemples simples de chemins



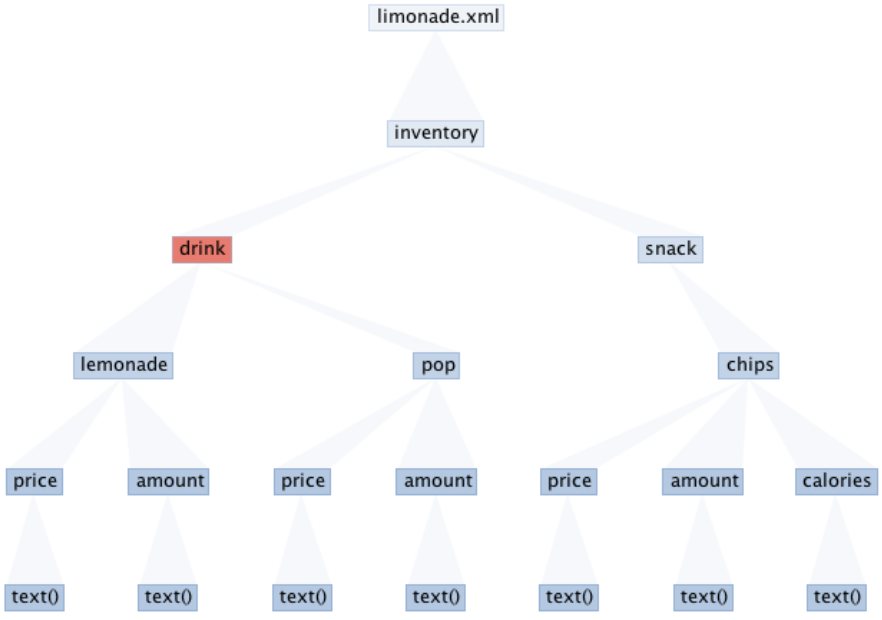


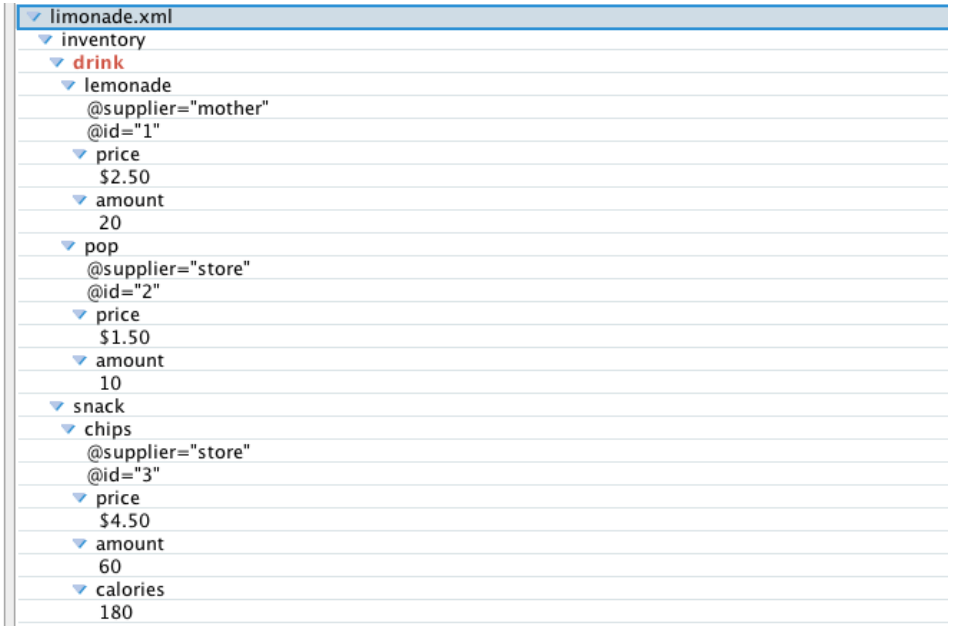
Évaluer les requêtes suivantes, sur le document XML se trouvant ci-dessus (vue BaseX[2])

1) `/inventory`
2) `/inventory/drink`
3) `/inventory/drink/*`
4) `/inventory/drink/lemonade`
5) `/inventory/drink/*/price`
6) `/inventory/drink//@supplier`

[2] http://www.basex.org

Quelques exemples simples de chemins





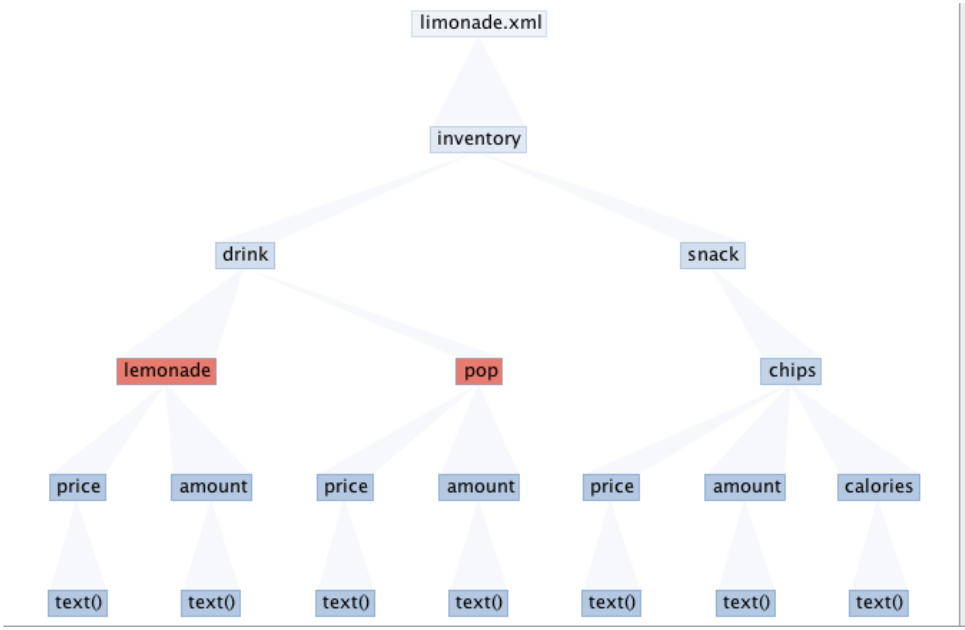
Évaluer les requêtes suivantes, sur le document XML se trouvant ci-dessus (vue BaseX[2])

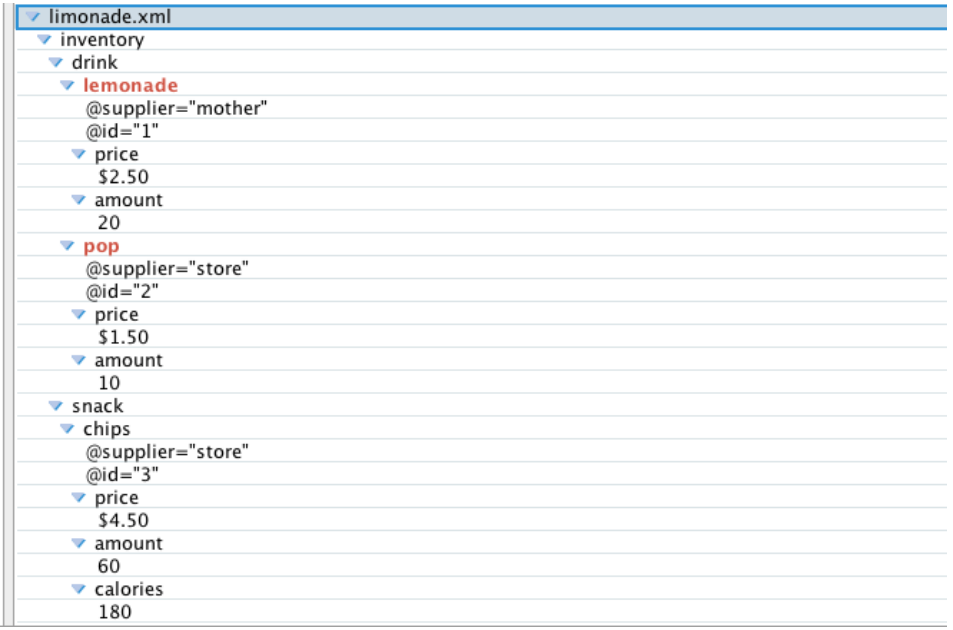
1) `/inventory`
2) `/inventory/drink`
3) `/inventory/drink/*`
4) `/inventory/drink/lemonade`
5) `/inventory/drink/*/price`
6) `/inventory/drink//@supplier`

[2] http://www.basex.org

le cnam

Quelques exemples simples de chemins



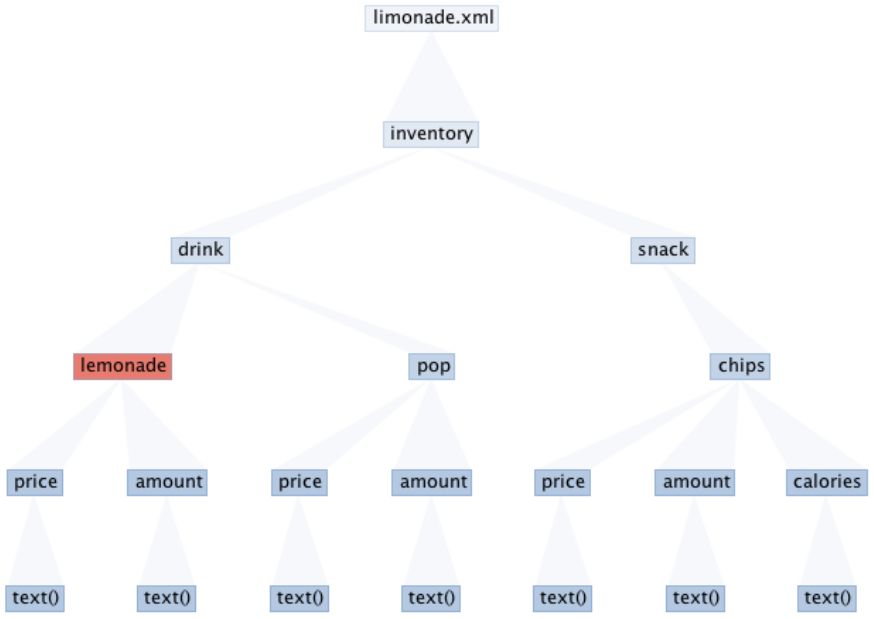


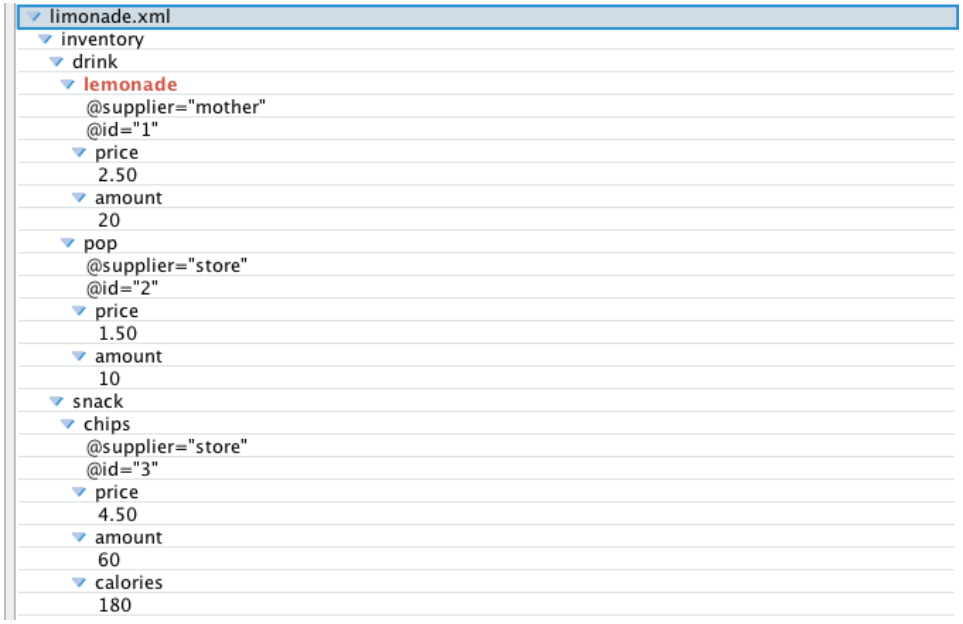
Évaluer les requêtes suivantes, sur le document XML se trouvant ci-dessus (vue BaseX[2])

1) `/inventory`
2) `/inventory/drink`
3) `/inventory/drink/*`
4) `/inventory/drink/lemonade`
5) `/inventory/drink/*/price`
6) `/inventory/drink//@supplier`

[2] http://www.basex.org

Quelques exemples simples de chemins



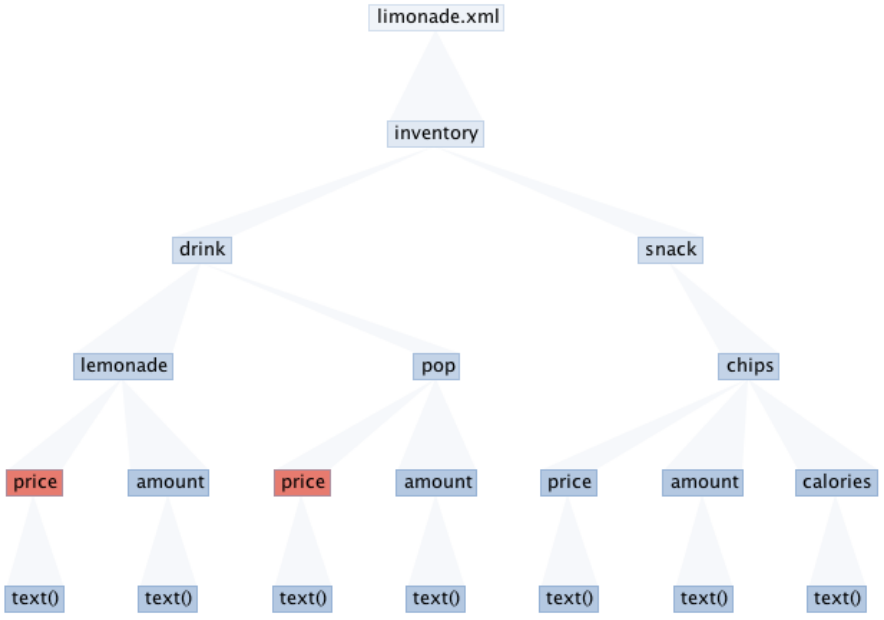


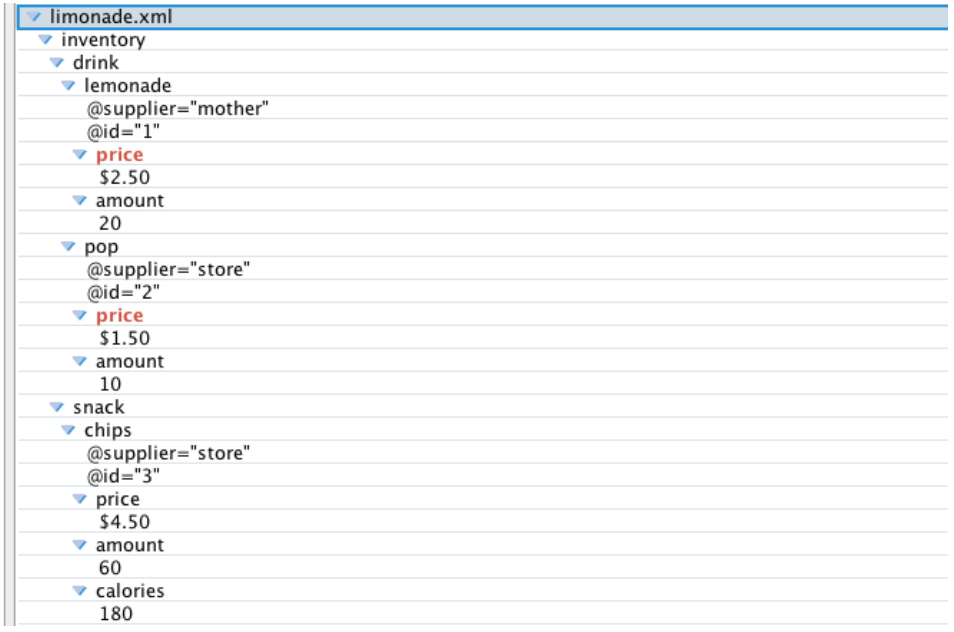
Évaluer les requêtes suivantes, sur le document XML se trouvant ci-dessus (vue BaseX[2])

1) `/inventory`
2) `/inventory/drink`
3) `/inventory/drink/*`
4) `/inventory/drink/lemonade`
5) `/inventory/drink/*/price`
6) `/inventory/drink//@supplier`

[2] http://www.basex.org

Quelques exemples simples de chemins





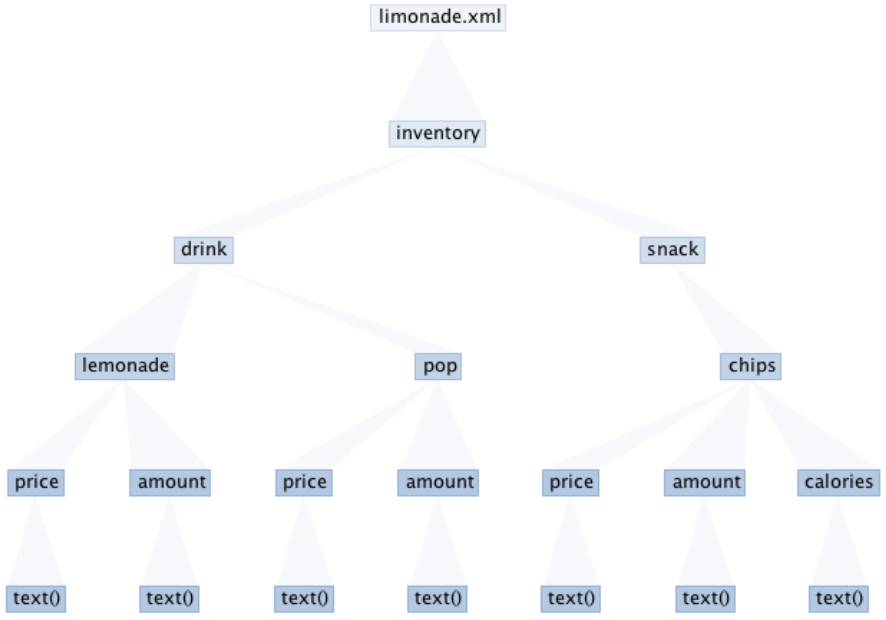
Évaluer les requêtes suivantes, sur le document XML se trouvant ci-dessus (vue BaseX[2])

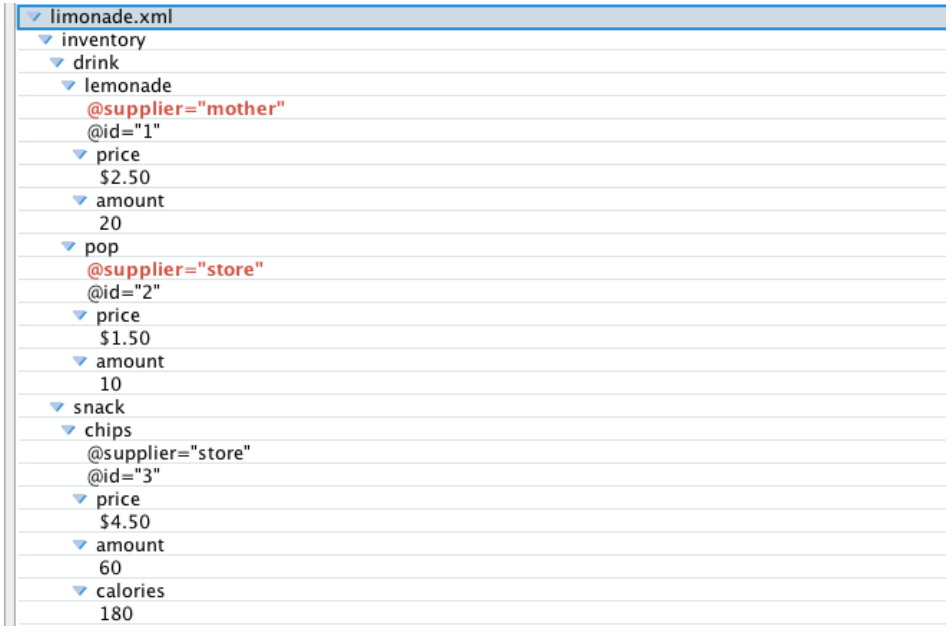
1) `/inventory`
2) `/inventory/drink`
3) `/inventory/drink/*`
4) `/inventory/drink/lemonade`
5) `/inventory/drink/*/price`
6) `/inventory/drink//@supplier`

[2] http://www.basex.org

le cnam

Quelques exemples simples de chemins





Évaluer les requêtes suivantes, sur le document XML se trouvant ci-dessus (vue BaseX[2])

1) `/inventory`
2) `/inventory/drink`
3) `/inventory/drink/*`
4) `/inventory/drink/lemonade`
5) `/inventory/drink/*/price`
6) `/inventory/drink//@supplier`

[2] http://www.basex.org

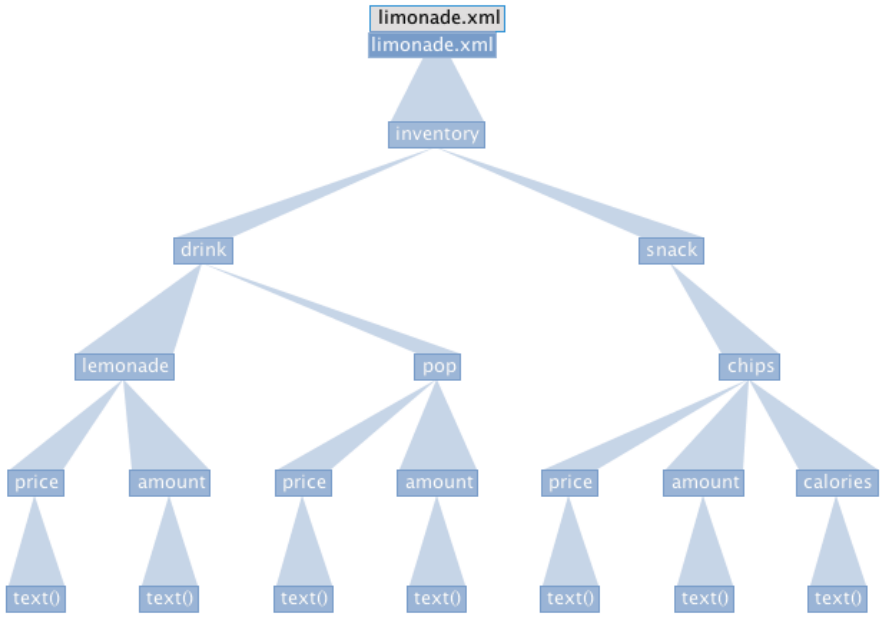
Chemin de localisation : intuition

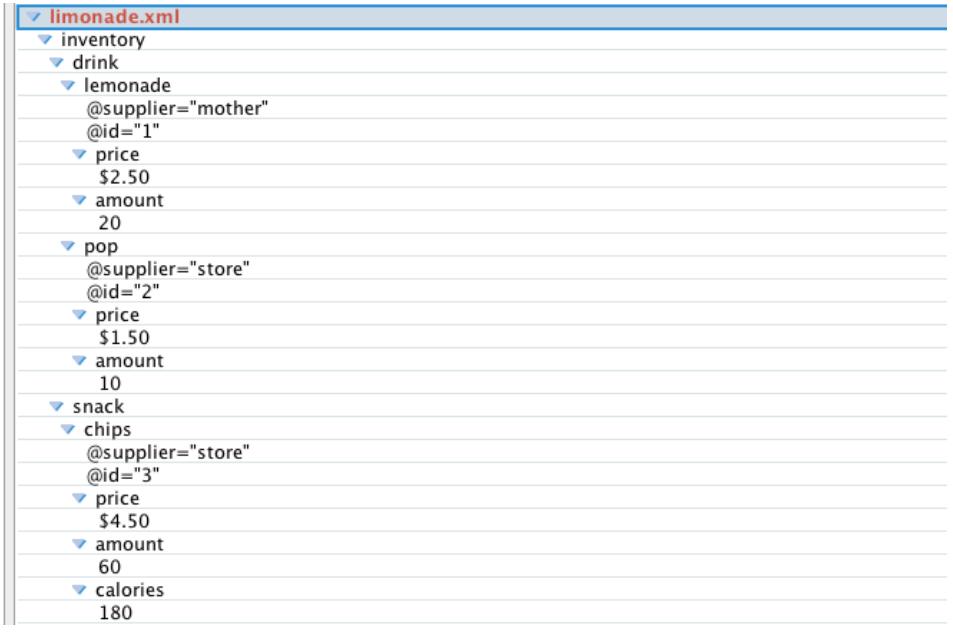
Intuition

Un chemin de localisation XPath est une suite de mouvements, chaque mouvement étant éventuellement suivi de tests sur les nœuds atteints.

Chaque *pas* "ramène" un ensemble de nœuds (de sous-arbres associés) à partir desquels est évalué le pas suivant.

Quelques exemples simples de chemins avec tests

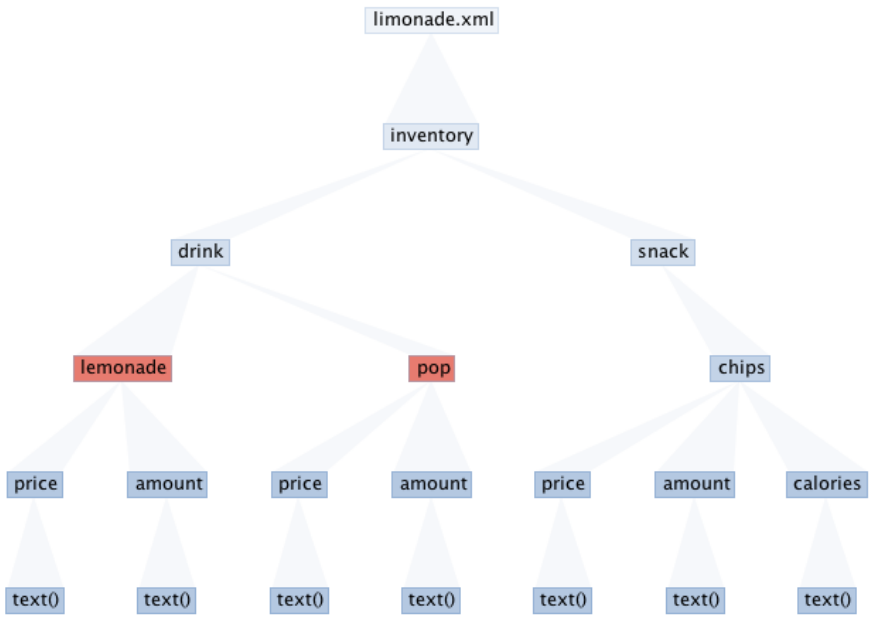


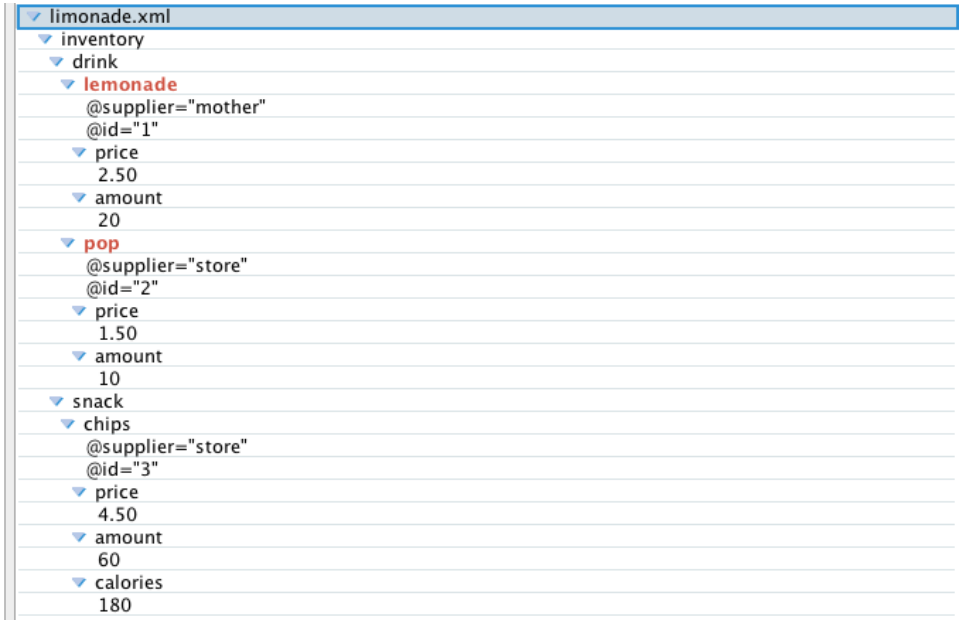


Évaluer (sur papier) les requêtes suivantes, sur le fichier limonade.xml se trouvant ci-dessus.

1) `/inventory/drink/*[price<3]`
2) `/inventory/drink//*[price<3][amount>10]`
3) `/inventory//*[@supplier="store"]/price`

Quelques exemples simples de chemins avec tests

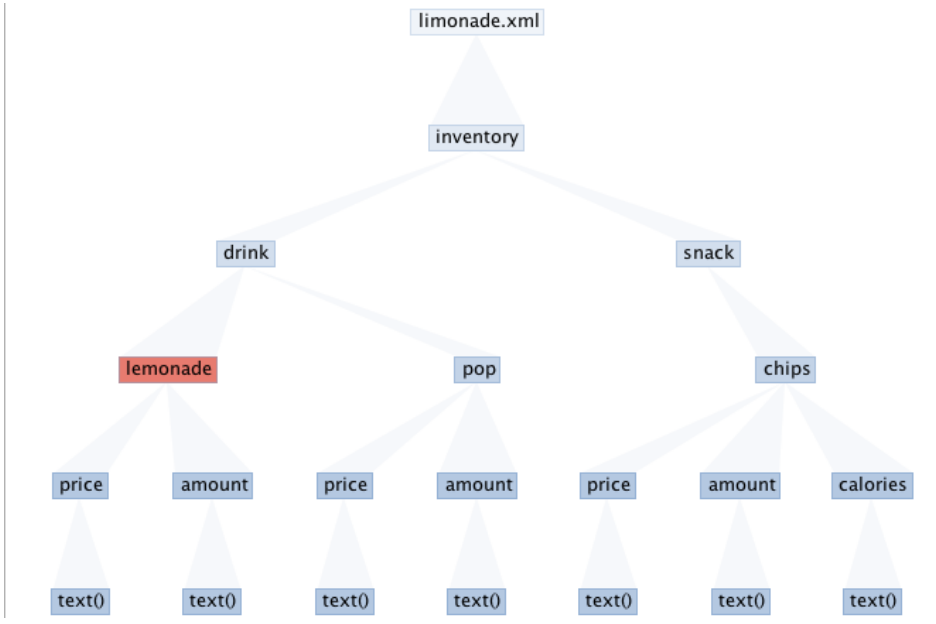


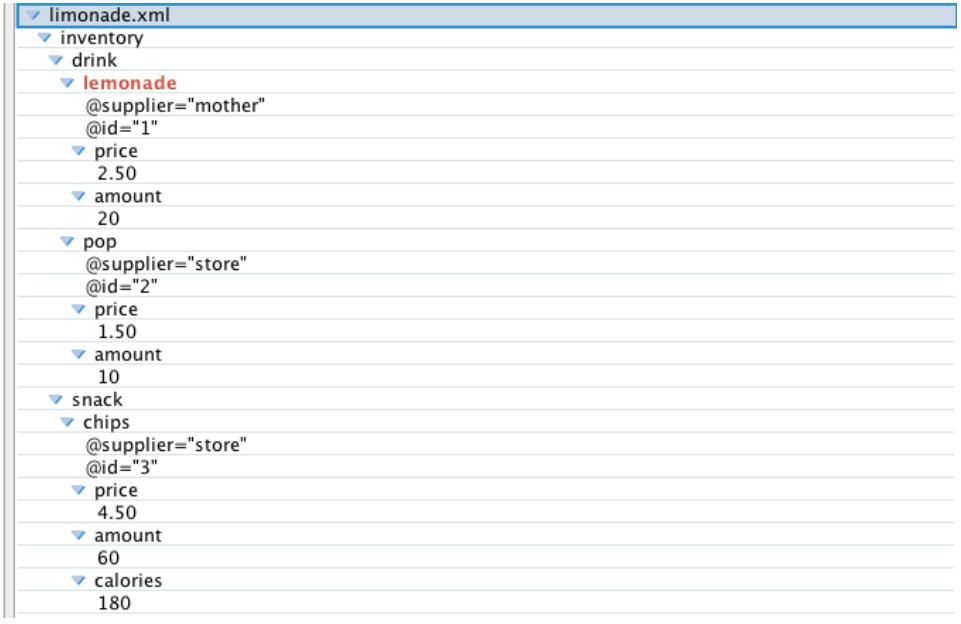


Évaluer (sur papier) les requêtes suivantes, sur le fichier limonade.xml se trouvant ci-dessus.

1) `/inventory/drink/*[price<3]`
2) `/inventory/drink//*[price<3][amount>10]`
3) `/inventory//*[@supplier="store"]/price`

Quelques exemples simples de chemins avec tests

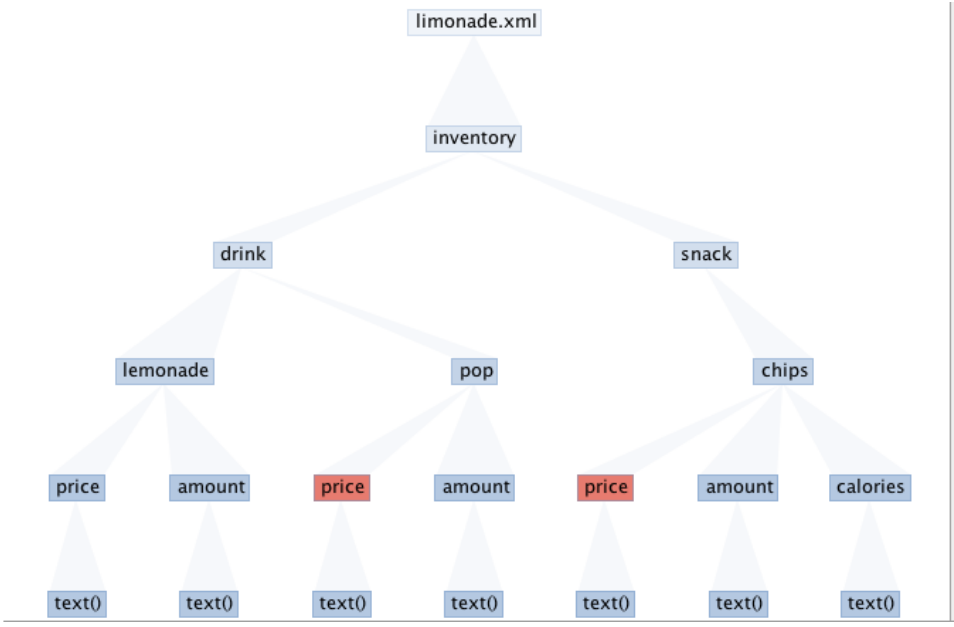


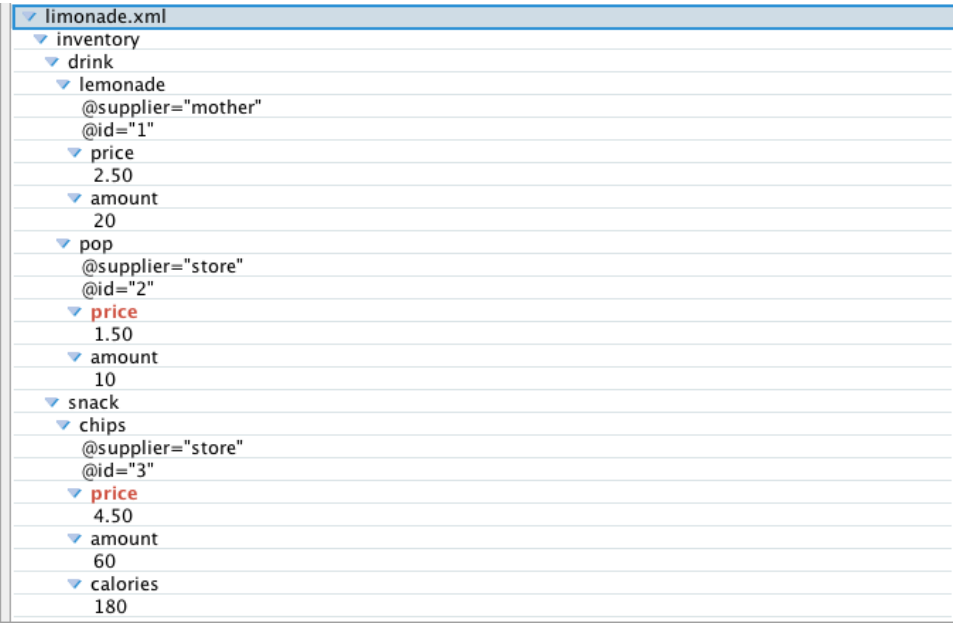


Évaluer (sur papier) les requêtes suivantes, sur le fichier limonade.xml se trouvant ci-dessus.

1) `/inventory/drink/*[price<3]`
2) `/inventory/drink//*[price<3][amount>10]`
3) `/inventory//*[@supplier="store"]/price`

Quelques exemples simples de chemins avec tests





Évaluer (sur papier) les requêtes suivantes, sur le fichier limonade.xml se trouvant ci-dessus.

1) `/inventory/drink/*[price<3]`
2) `/inventory/drink//*[price<3][amount>10]`
3) `/inventory//*[@supplier="store"]/price`

Types de mouvements

- Fils (child::), chemin par défaut
- Père (parent:: ou ../)
- Descendant (descendant:: ou //)
- Attribut (attribute:: ou @)
- Ancêtre (ancestor::)
- Précédents ou Suivants (following:: / preceding::)
- Frères avants ou après (following-sibling:: / preceding-sibling::)

XPath simplifié

Les deux chemins suivants sont équivalents :

- `/inventory//*[@supplier="store"]/price`
- `/child::inventory/descendant::*/child::*[attribute::supplier="store"]/child::price`

Table of Contents

JavaScript Object Notation

- Initialement créé pour la sérialisation et l'échange d'objets JavaScript;
- Langage pour l'échange de données semi-structurées (et éventuellement structurées);
- Format texte indépendant du language de programmation utilisé pour le manipuler.

Utilisation première : échange de données dans un environnement Web (par exemple applications Ajax)
Extension : sérialisation et stockage de données.

Les bases de JSON I

Paire ordonnée *clef-valeur* (*key-value*)

```
"title": "The Social network"
```

Types atomiques de données : chaînes de caractères (entourées par les classiques guillemets anglais (droits)), nombres (entiers, flottants) et valeurs booléennes (`true` ou `false`).

```
"year": 2010
```

Les bases de JSON II

Un *objet* est un ensemble de paires clef-valeur.

Au sein d'un ensemble de paires, une clef apparait au plus une fois (NB : les types de valeurs peuvent être distincts).

```
{"last_name": "Fincher", "first_name": "David"}
```

Un objet peut être utilisé comme valeur (dite *complexe*) dans une paire clef-valeur.

```
"director": {
  "last_name": "Fincher",
  "first_name": "David",
  "birth_date": 1962
}
```

Les bases de JSON III

Un *tableau* (*array*) est une liste de valeurs (dont le type n'est pas forcément le même).

```
"actors": ["Eisenberg", "Mara", "Garfield", "Timberlake"]
```

Les bases de JSON IV

Un *document* est un objet. Il peut être défini par des objets et tableaux imbriqués autant de fois que nécessaire.

```
{
  "title": "The Social network",
  "summary": "On a fall night in 2003, Harvard undergrad and computer\n
        programming genius Mark Zuckerberg sits down at his computer\n
        and heatedly begins working on a new idea. (...)",
  "year": 2010,
  "director": {"last_name": "Fincher",
                  "first_name": "David"},
  "actors": [
       {"first_name": "Jesse", "last_name": "Eisenberg"},
       {"first_name": "Rooney", "last_name": "Mara"}
    ]
}
```

Qu'est-ce qu'un objet JSON ?

object :
- {}
- { *members* }

members :
- *pair*
- *pair* , *members*

pair :
- *string* : *value*

array :
- []
- [*elements*]

elements :
- *value*
- *value* , *elements*

value :
- *string*
- *number*
- *object*
- *array*
- true
- false
- null

Représenter des données dans le format JSON ou dans le format XML I

Voici un document JSON :

```
{
        "title": "The Social network",
        "year": 2010,
        "genre": "drama",
        "summary": "On a fall night in 2003, Harvard undergrad and computer
                    programming genius Mark Zuckerberg sits down at his computer(...)",
        "country": "USA",
        "director": {
                      "last_name": "Fincher",
                      "first_name": "David",
                      "birth_date": 1962
                     },
        "actors": [
           { "first_name": "Jesse", "last_name": "Eisenberg", "birth_date": 1983, "role": "Mark Zuckerberg" },
           { "first_name": "Rooney", "last_name": "Mara", "birth_date": 1985, "role": "Erica Albright" },
           { "first_name": "Andrew", "last_name": "Garfield", "birth_date": 1983, "role": " Eduardo Saverin" },
           { "first_name": "Justin", "last_name": "Timberlake", "birth_date": 1981, "role": "Sean Parker" }
        ]
}
```

Listing 2: Document JSON

Représentez les données de ce document dans le format XML.

Représenter des données dans le format JSON ou dans le format XML II

Voici un document XML :

```
<?xml version="1.0" >
<biblio subject="XML" xmlns="http://www.biblioAppliNFE204">
  <book ISBN="9782212090819" lang="fr" subject="applications">
    <author> <firstname>Jean−Christophe</firstname> <lastname>Bernadac</lastname> </author>
    <author>
      <firstname>FranÃğois</firstname> <lastname>Knab</lastname>
    </author>
    <title>Construire une application XML</title>
    <publisher> <name>Eyrolles</name> <place>Paris</place> </publisher>
    <datepub>1999</datepub>
  </book>
  <book ISBN="9782212090529" lang="fr" subject="general">
    <author> <firstname>Alain</firstname> <lastname>Michard</lastname> </author>
    <title>XML, Langage et Applications</title>
    <publisher>
      <name>Eyrolles</name> <place>Paris</place>
    </publisher>
    <datepub>1998</datepub>
  </book>
  <book ISBN="9782840825685" lang="fr" subject="applications">
    <author> <firstname>William J.</firstname> <lastname>Pardi</lastname> </author>
    <title>XML en Action</title>
    <publisher> <name>Microsoft Press</name> <place>Paris</place> </publisher>
    <datepub>1999</datepub>
  </book>
</biblio>
```

Listing 3: Extrait d'un simple fichier XML

Représentez les données de ce document dans le format JSON. Que constatez-vous ?

Représenter des données dans le format JSON ou dans le format XML III

Voici une table contenant des données régulières.

name	firstname	email	office
Rigaux	Philippe	philippe.rigaux@cnam.fr	37.1.40
Travers	Nicolas	nicolas.travers@cnam.fr	37.1.41
Fournier	Raphael	fournier@cnam.fr	37.1.40

Représentez les données contenues dans cette table au format JSON. Que constatez-vous ?

JSON et références

```
{
  person: {name: "alan", phone: 3127786, email: "agg@abc.com"},
  person: &314
                {name: {first: "Sara", last: "Smith-Green"},
                phone: 2136877,
                email: "sara@math.xyz.edu",
                spouse: &443},
  person: &443
                { name: "Fred Green",
                  phone: 7786312,
                  Height: 183,
                  spouse: &314 }
}
```

JSON vs. XML

- JSON plus léger et intuitif que XML,
- Facile à parser pour n'importe quel langage de programmation,
- JSON n'a pas (encore) de langage de spécification de schéma standardisé,
- JSON n'a pas (encore) de langage de requête associé.

Table of Contents

Qu'est-ce qu'un Service Web

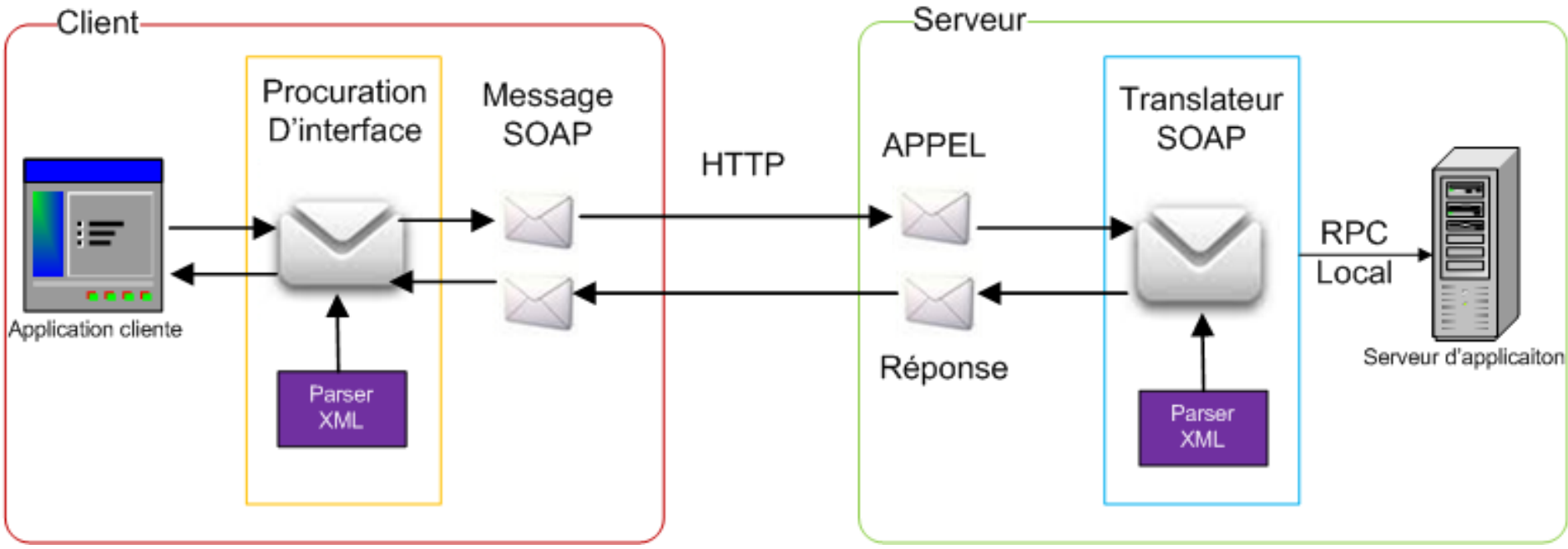
- Application modulaire
- Basée sur Internet
- Exécute une tâche précise avec un format spécifique (Basé sur XML)

$\Rightarrow$ Communication réseau : Application / Application

Technologies associées

- SOAP "*Simple Object Access Protocol*"
  - Définit la structure des messages XML utilisés par les applications pour dialoguer entre elles ;
- WSDL "*Web Service Description Language*"
  - Format de description des méthodes et des paramètres des services Web
- UDDI "*Universal Description, Discovery and Integration*"
  - Annuaire de services
  - Ensemble de WSDL associés à une sémantique
- HTTP pour transporter les requêtes et leurs réponses, et HTTPS pour la sécurité

Appel de Service Web



Message SOAP

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <soap:Header>
        <!-- optionnel -->
    </soap:Header>
    <soap:Body>
        <!-- requis -->
        <soap:Fault>
            ...
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

- Envelope : Document racine, contient l'ensemble des informations du message
- Header : Contient les informations d'entête
  - Élements : User (destinataire), Session (identifiant de session), Lang (langue d'échange)
- Body : Corps du message, contient les informations propre à l'échange de données
  - Spécifique à l'application
- Fault : Permet de reporter des erreurs durant l'échange de données

Exemple de Body

- Requête :

```
<soap:Body>
    <m:GetCourse xmlns:m="http://www.exemple.com/course">
        <m:CodeUE>NFE102</m:CodeUE>
    </m:GetCourse>
</soap:Body>
```

- Réponse :

```
<soap:Body>
    <m:GetCourseResponse xmlns:m="http://www.exemple.com/course">
        <m:title>Infrastructures du Commerce Electronique</m:title>
    </m:GetCourseResponse>
</soap:Body>
```

WSDL et UDDI

S2-WSDL-UDDI.png

Exemple de WSDL

Défini l'ensemble des informations transmises :

```
 <message name="GetCourse">
     <part name="CodeUE" type="xs:string">
</message>

<message name="GetCourseResponse">
     <part name="title" type="xs:string">
</message>
```

Implémentation de services Web

Principaux framework de mise en œuvre de Services Web :

- HP Web services (e-speak)
- IBM WebServices Toolkit (SOAP, UDDI et ebXML)
- Microsoft .NET (SOAP, UDDI, BizTalk)
- Oracle 9i iAS Web Services (SOAP, UDDI et J2EE)
- Sun Open Net Environment (SOAP, UDDI, ebXML)
- Systinet, WASP Toolkit
- The Mind Electric, Glue plateform
- BowStreet, Cape Clear, SilverStream...

Table of Contents

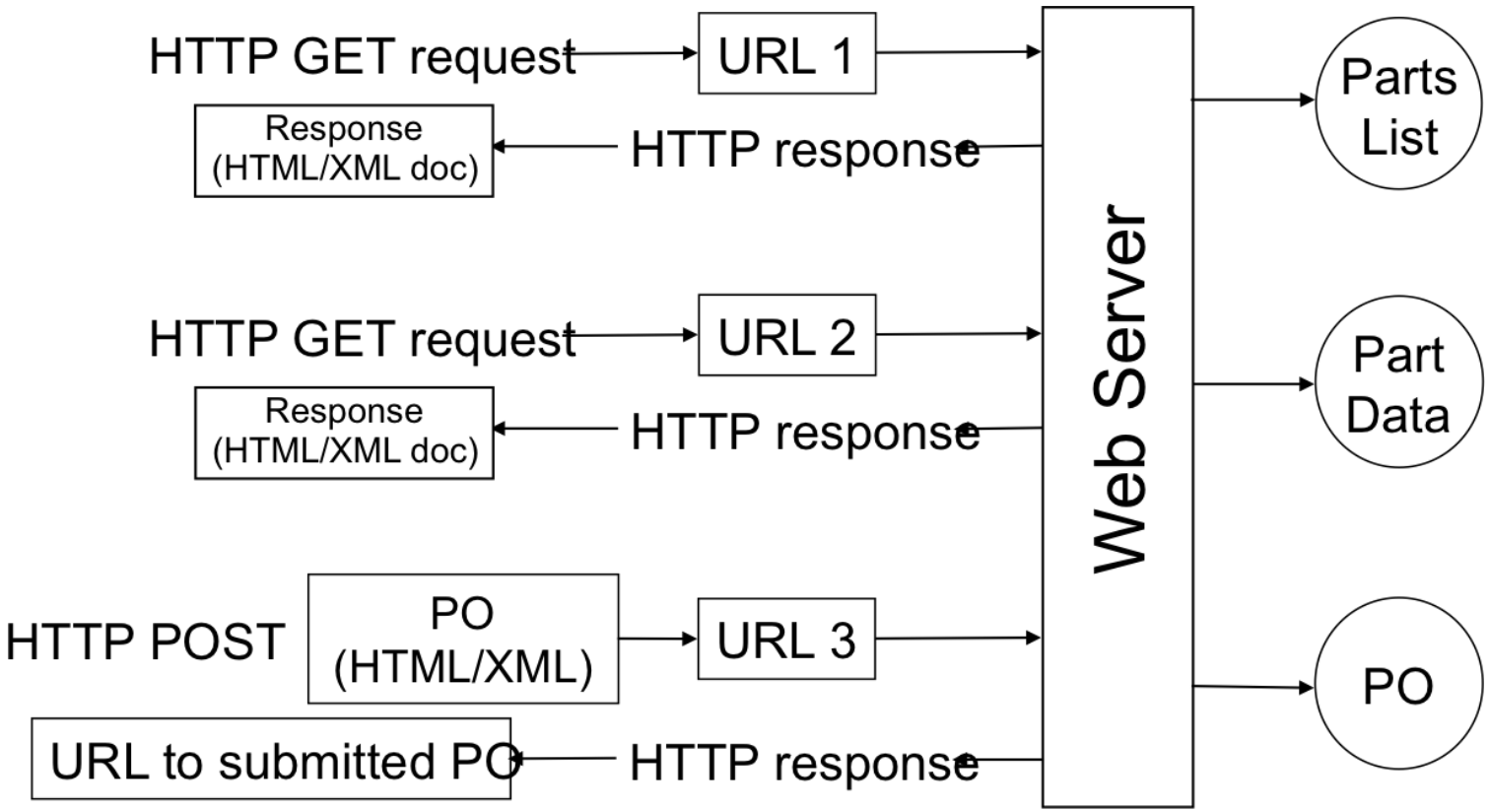
REST - Representational State Transfer

- REST a ÃľtÃľ crÃľÃľ par *Roy Fielding* (2000) pour dÃľfinir des "`design pattern`" sur des rÃľseaux
- `Fonctionnement` :
  - Le client rÃľfÃľrence une ressource Web Ãă l'aide d'une URL
  - Surcouche du protocole HTTP
  - Une reprÃľsentation de la ressource est retournÃľe
  - Le client se trouve dans un nouvel Ãľtat grÃ¢ce Ãă la ressource (ici, `Boeing747.html`)
  - Le nouvel Ãľtat peut conduire Ãă la demande d'une nouvelle ressource

REST - N'est pas un standard

- REST n'est pas spÃľcifiÃľ par le W3C
- Un gestionnaire REST ne sera pas vendu par une grosse compagnie
- Il est conseillÃľ d'utilisÃľ REST avec :
  - HTTP
  - URL
  - XML/JSon/HTML/GIF/JPEG/etc. (ReprÃľsentation)
  - text/xml, text/json, text/html, image/gif, image/jpeg, etc. (Types MIME)

RequÃłtes et DÃľpÃťts



RÃľcupÃľrer une liste d'ÃľlÃľments

- Une URL liÃľes Ãă une "collection" permet de fournir une liste de ressources :
- `curl -XGET http://www.parts-depot.com/parts`

Remarque

La maniÃÍre dont le service produit cette liste est transparente pour l'utilisateur (couplage faible)

Liste de retour

```
curl -XGET http://www.parts-depot.com/parts
```

⇒ Ressource retournÃľe :

```
<?xml version="1.0"?>
<Parts>
      <Part id="00345" href="http://www.parts-depot.com/parts/00345"/>
      <Part id="00346" href="http://www.parts-depot.com/parts/00346"/>
      <Part id="00347" href="http://www.parts-depot.com/parts/00347"/>
      <Part id="00348" href="http://www.parts-depot.com/parts/00348"/>
</Parts>
```

Remarque

Dans ce rÃľsultat, un lien est fourni Ãă chaque “part” pour avoir des informations supplÃľmentaires sur la ressource.
Le service dÃľtermine la maniÃÍre dont les donnÃľes sont utilisÃľes et interrogÃľes.

Principes de l'API REST

- Pour chaque service une ressource doit Ãłtre fournie
- Chaque URL identifie une ressource

Retrouver une ressource particuliÃÍre

curl -XGET http://www.parts-depot.com/parts/00345
⇒ Ressource retournÃľe :

```
<?xml version="1.0"?>\\
<Part>\\
      \hspace*{.5cm}<Part-ID>00345</Part-ID>\\
      \hspace*{.5cm}<Name>Widget-A</Name>\\
      \hspace*{.5cm}<Description>This part is used within the frap assembly</Description>
      \hspace*{.5cm}<Specification href="http://www.parts-depot.com/parts/00345/specifica
      \hspace*{.5cm}<UnitCost currency="USD">0.10</UnitCost>\\
      \hspace*{.5cm}<Quantity>10</Quantity>\\
</Part>
```

Informations supplÃľmentaires

- Il est possible de faire passer des requÃłtes complexes :
  - MÃľthode POST : document requÃłte
    `curl -XPOST http://www.parts-depot.com/parts/ -d '//UnitCost[currency='USD']'`
- Plusieurs services disponibles sur un mÃłme serveur : collections
  - ImplÃľmentation d'un service par ressource
- Le type de retour n'est pas fixe
  - `curl http://www.parts-depot.com/parts/?ouput=JSON`
- L'accÃs peut Ãłtre restreint
  - ClÃľ d'API (peut Ãłtre payant)
  - `curl http://www.parts-depot.com/parts/?APIKEY=f254fc91e7ca9fdc41f648658734fc27`