Le web sémantique

RDF/XML, API DOM : *XML contre-attaque*

RDF : Resource Description Framework

- http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
- W3C en 1997, version 1.1 en 2014
- Associer des méta-informations à des ressources :
  - Page web
  - Item sur une page web
  - Document multimédia
  - …
- On peut créer ses propres annotations (extension du principe des microformats)
- On peut matérialiser les liens entre les ressources
- Framework très généraliste (trop?)

RDF et la notion de triplet

- Toutes les ressources peuvent être décrites formellement sous forme de triplets
- **sujet** *verbe* complément
- **sujet** *prédicat/propriété* objet
- **Totor le castor** *s'intéresse à* Web sémantique
- **:totor** *foaf:interest* w3c:semWeb
- L'ensemble des triplets forme un graphe
- Toutes les ressources peuvent être sujet, prédicat ou objet

Syntaxes RDF

- Plusieurs syntaxes !
  - **RDF / XML**
  - Notation 3
  - Turtle (sous-ensemble de Notation 3)
  - N-triples (sous-ensemble de turtle)
  - JSON-LD
  - Etc.

RDF/XML

- Historiquement la première syntaxe (1997)
- Aussi vieux qu'XML
- Plus vieux que l'expression « web sémantique » (2001)
- Dernière recommandation : 1.1 (2014)

Pourquoi ?

- Avantages d'une syntaxe XML :
  - Tous les outils qui peuvent parser du XML peuvent parser du RDF/XML
  - Tous les langages ont des bibliothèques pour parser des documents XML
  - Ce n'est pas le cas des autres syntaxes
  - Ont peut utiliser d'autres outils faits pour manipuler du XML : XSLT, validation de schéma, etc.

Pourquoi pas ?

- XML est verbeux
- Les documents de gros volume peuvent être difficiles à parser
- XML n’est pas facile à comprendre pour un humain par rapport à Turtle ou N3

Rappel : les triplets

- Graphe RDF : ensemble de triplets
- Sujet prédicat objet
- Sujet verbe complément
- Ressource propriété valeur

RDF/XML

```
<rdf:RDF xmlns:rdf='...'>
  <rdf:Description rdf:about='sujet'>
    <predicat rdf:resource='objet'>
    <predicat>objet littéral</prédicat>
  </rdf:Description>
</rdf:RDF>
```

Turtle (rappel)

- Surensemble de N-triples
- Possibilité de créer des préfixes (même idée que les namespaces XML)
- Virgules et point-virgules pour éviter les répétitions

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <https://schema.org/> .
@prefix ex: <http://www.example.org/> .

ex:totor rdf:type schema:Person ;
         schema:name "Totor le Castor" .

schema:name rdf:type rdf:property .
```

RDF/XML

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:ex="http://www.example.org"
   xmlns:schema="https://schema.org/">
   <rdf:Description rdf:about="ex:totor">
      <rdf:type rdf:resource="schema:Person"/>
      <schema:name>Totor le Castor</schema:name>
   </rdf:Description>
   <rdf:Description rdf:about="schema:name">
      <rdf:type rdf:resource="rdf:property"/>
   </rdf:Description>
</rdf:RDF>
```

Comparaison

- 2 phrases (2 sujets dont on parle, 2 fois le « point »)
- 2 éléments `rdf:Description`
- Pour une description, à chaque prédicat, un élément XML du même nom
- Objet : `ressource` ou objet littéral

Tag de langue

```
ex:totor schema:name "Totor le Castor" ,
                     "Totor the Castor"@en .
```

```
<rdf:Description rdf:about="ex:totor">
  <schema:name>
    Totor le Castor
  </schema:name>
  <schema:name xml:lang="en">
    Totor the Castor
  </schema:name>
</rdf:Description>
```

Tag de langue

```
<rdf:Description rdf:about="ex:totor" xml:lang="fr">
  <rdf:type rdf:resource="schema:Person"/>
  <schema:jobTitle>
    Constructeur de barrages
  </schema:jobTitle>
  <schema:name>
    Totor le Castor
  </schema:name>
  <schema:name xml:lang="en">
    Totor the Castor
  </schema:name>
</rdf:Description>
```

Tag de langue : conversion

```
ex:totor a schema:Person ;
         schema:jobTitle "Constructeur de barrages"@fr ;
         schema:name "Totor le Castor"@fr ,
                     "Totor the Castor"@en .
```

rdf:type implicite

```
ex:totor a schema:Person .

<rdf:Description rdf:about="ex:totor">
   <rdf:type rdf:resource="schema:Person"/>
</rdf:Description>

<schema:Person rdf:about="ex:totor"/>
```

rdf:type implicite

```
ex:totor a schema:Person ;
         schema:name "Totor le Castor".
```

```
<rdf:Description rdf:about="ex:totor">
   <rdf:type rdf:resource="schema:Person"/>
   <schema:name>Totor le Castor</schema:name>
</rdf :Description>
```

```
<schema:Person rdf:about="ex:totor">
   <schema:name>Totor le Castor</schema:name>
</schema:Person>
```

Noeuds anonymes

```
[ ] # quelque chose
```

```
<rdf:Description/> < !-- Pas de about : nœud anonyme -->
```

Noeuds anonymes

```
[ a schema:Book ] # quelque chose qui est un livre
```

```
<rdf:Description> < !-- Pas de about : nœud anonyme -->
   <rdf:type rdf:resource="schema:Book"/>
</rdf:Description>
```

```
<schema:Book/> < !-- raccourci de Description + type -->
```

Noeuds anonymes : descriptions imbriquées

```
ex:elmore schema:owns [ a schema:Book ] .
```

```
<rdf:RDF>
   <rdf:Description rdf:about="ex:elmore">
      <schema:owns>
         <rdf:Description>
            <rdf:type rdf:resource="schema:Book"/>
         </rdf:Description>
      </schema:owns>
   </rdf:Description>
</rdf:RDF>
```

Noeuds anonymes : descriptions imbriquées

```
ex:elmore schema:owns [ a schema:Book ] .
```

```
<rdf:RDF>
   <rdf:Description rdf:about="ex:elmore">
      <schema:owns>
         <schema:Book/>
      </schema:owns>
   </rdf:Description>
</rdf:RDF>
```

Noeuds anonymes : références

```
ex:elmore schema:owns [ a schema:Book ] .
```

```
<http://example.org/elmore> <https://schema.org/owns> _:book1 .
_:book1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <https://schema.org/Book> .
```

```
<rdf:RDF>
   <rdf:Description rdf:about="ex:elmore">
      <schema:owns rdf:nodeID="book1"/>
   </rdf:Description>
   <rdf:Description rdf:nodeID="book1"> < !-- Pas de about : nœud anonyme -->
      <rdf:type rdf:resource="schema:Book"/>
   </rdf:Description>
</rdf:RDF>
```

Noeuds anonymes

```
ex:elmore schema:owns [ a schema:Book ] .
```

```
<rdf:RDF>
   <rdf:Description rdf:about="ex:elmore">
      <schema:owns rdf:nodeID="book1"/>
   </rdf:Description>
   <schema:Book rdf:nodeID="book1"/>
</rdf:RDF>
```

Noeuds anonymes : descriptions imbriquées

```
ex:elmore schema:owns
    [
        a schema:Book ;
        schema:hasCharacter [ schema:name "Jack Foley" ]
    ] .
```

Noeuds anonymes : descriptions imbriquées

```
<rdf:RDF>
   <rdf:Description rdf:about="ex:elmore">
      <schema:owns>
         <schema:Book>
            <schema:hasCharacter>
               <rdf:Description>
                  <schema:name>"Jack Foley"</schema:name>
               </rdf:Description>
            </schema:hasCharacter>
         </schema:Book>
      </schema:owns>
   </rdf:Description>
</rdf:RDF>
```

RDF/XML : typage des littéraux

```
ex:totor ex:age "23" .
```

```
<http://example.org/totor> <http://example.org/age> "23" .
```

```
<rdf:Description about="ex:totor">
    <ex:age>23</ex:age>
</rdf:Description>
```

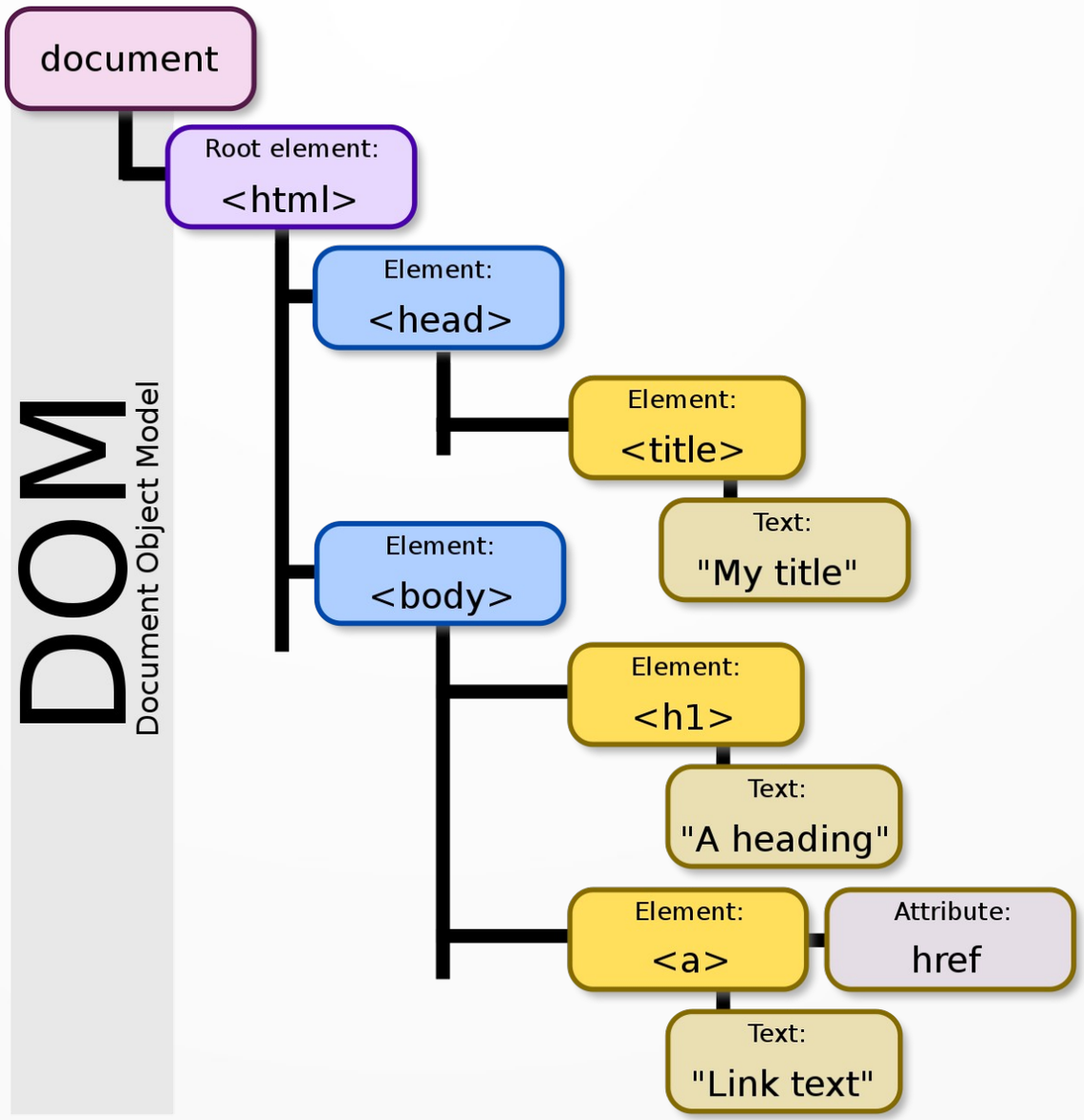
RDF/XML : typage des littéraux

```
ex:totor ex:age 23 .
```

```
<http://example.org/totor> <http://example.org/age>
"23"^^<http://www.w3.org/2001/XMLSchema#integer> .
```

```
<rdf:Description about="ex:totor">
  <ex:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
     23
  </ex:age>
</rdf:Description>
```

XML : API DOM



(source image : wikipédia)

XML : API DOM

- API pour manipuler les documents XML dans un programme
- Recommandation W3C
- Première version : 1997
- API générique, pour tous les langages
- Représentation du document comme un arbre stocké en mémoire

XML : API DOM

- Avantages :
  - Tous les langages implémentent API, souvent dans la bibliothèque standard
  - Facile à utiliser
- Inconvénients :
  - Pas adapté aux documents très volumineux
  - Pas adapté aux documents créés à la volée

XML : API SAX

- API SAX : Simple API for XML
- Plus bas niveau
- Permet de manipuler de très gros documents et des documents créés à la volée
- API de type événementiel
- On parse le document à la volée
- Pas une recommandation W3C
- Plus bas niveau que DOM

XML : API DOM

- Ici, les grands principes uniquement
- Chaque langage apporte des fonctionnalités supplémentaires
- Regardez la doc de votre langage préféré

XML : 7 types de nœuds

- Racine
  - L'ensemble du document
  - Se situe au-dessus de l'élément racine, parce qu'il peut y avoir des infos supplémentaires au-dessus

```
<?xml version='1.0'?>
<!-- ceci est un document RDF -->
<rdf:RDF> … </rdf:RDF>
```

XML : 7 types de nœuds

- Élément
  - Tout ce qu'il y a entre une balise ouvrante et une balise fermante
  - Peut contenir :
    - Attributs
    - Sous-éléments
    - Contenus textuels
    - Commentaires
  - L'ordre des nœuds fils est significatif (sauf pour les attributs)

XML : 7 types de nœuds

- Attribut
  - Paire clé/valeur
  - Forcément une feuille de l’arbre

XML : 7 types de nœuds

- Noeud textuel
  - Chaîne de caractères
  - Forcément une feuille de l'arbre
  - Zone de texte contiguë sans élément à l'intérieur

```
<p>
  Vous pouvez <a href='contact.html'>me contacter</a> si vous le souhaitez !
</p>
```

  - Un élément « p » qui contient 3 fils : du texte (« Vous pouvez »), un sous-élément, du texte (« si vous le souhaitez ! »)
  - Le sous-élément est un élément « a » qui contient 2 fils : un attribut (« href » → « contact.html ») et du texte (« me contacter »)

XML : 7 types de nœuds

- Commentaire
  - Font partie de la structure du document
  - Contiennent parfois des méta-informations pour les parseurs (à éviter)
  - Forcément une feuille

```
<!-- commentaire -->
```

XML : 7 types de nœuds

- Instruction de traitement
  - Contiennent des instructions pour les parseurs (à éviter)
  - Forcément une feuille

```
<?nom-instruction attribut='valeur'?>
```

XML : 7 types de nœuds

- Espace de nom
  - Un nœud est associé à un namespace
  - Attention, un namespace n'est pas un préfixe, mais bien l'URI complet

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   …
</rdf:RDF>
```

DOM : quelques fonctions

- Element.getElementById(string) → Element
- Element.getElementsByTagName(string) → NodeList
- Element.childNodes → NodeList
- NodeList.length → int
- NodeList[int] → Node
- Node.nodeType → int
- Node.nodeName → string
- Node.nodeValue → string

DOM : quelques fonctions

- Node.parentNode → Node
- Node.firstChild → Node
- Node.lastChild → Node
- Node.nextSibling → Node
- Node.previousSibling → Node

```
doc.getElementsByTagName('rdf:type').parentNode
// tous les nœuds ayant un fils rdf:type
```

DOM : quelques fonctions

- Node.removeChild(Node)
- Node.appendChild(Node)
- Node.replaceChild(Node, Node) // new, old
- Node.insertBefore(Node)

- Document.createElement(string) → Node
- Document.createTextNode(string) → TextNode
- Document.createComment(string) → CommentNode

DOM : quelques fonctions

- Node.attributes → AttributeList
- Node.getAttribute(string) → string
- Node.getAttributeNode(string) → Node
- Node.setAttribute(string, string)
- Node.removeAttribute(string)
- Node.removeAttributeNode(Node)

DOM : exemple en JS

```
for (let i = 0 ; i < doc.childNodes.length ; i++) {
   const child = doc.childNodes[i] ;
   if (child.nodeType == 1) { // Element
      const id = child.getAttributeNode('id');
      if (id !== null) {
         console.log('id=' + id) ;
      } else {
         child.setAttribute('id', 'my-id-' + i);
      }
   }
}
```

Exemple RDF

- Remplacer les nœuds rdf:Description ayant un fils rdf:type par la valeur correspondante

```
let types = doc.getElementsByTagName("rdf:type");
for (let i = 0; i < types.length; i++) {
   const name = types[i].getAttribute('rdf:resource') ;
   let n = doc.createElement(name) ;
   const parent = types[i].parentNode ;
   for (let j = 0 ; j < parent.childNodes.length ; j++) {
      const child = parent.childNodes[j];
      if (child !== types[i]) n.appendChild(child);
   }
   parent.parent.replaceChild(n, parent);
}
```