Institut national des sciences appliquées de Rouen

INSA de Rouen

Document Web Sémantique

Projet 2.4 - NoDEfr-2

Auteurs:
Killian POULIQUEN
Achraf TALBI
Ghalil BALGA

Enseignant:
Nicolas DELESTRE

May 17, 2021

Contents

Chapter 1

Présentation du sujet

1.1 Sujet 2 : NoDEfr-2, une norme pour la description des offres de formation

Un groupe d'experts de l'AFNOR travaillent depuis deux ans sur la spécification d'un modèle conceptuel pour la description des offres de formation (NoDEfr-2). Le méta modèle utilisé pour modéliser le NoDEfr-2 est celui proposé par l'ISO dans le cadre du MLR. Le paradigme de ce méta-modèle (ensembliste reposant sur la description de classes et d'éléments de données) est proche de ceux proposés par le W3C pour le web des données (RDFS) et le web sémantique (OWL2). Pour travailler les experts utilisent un tableur (document semi-formel): un onglet représente les classes, et les autres onglets (un par classe) leurs propriétés (éléments de données). L'objectif de ce projet est de formaliser automatiquement ce travail pour générer automatiquement des schémas RDFS et OWL.

Liens :

- Site NoDEfr
- Article MLR
- GitHub projet

1.2 Sujet 2.4 : NoDEfr-2 XML vers OWL

4 équipes ont été constituées pour réaliser le projet NoDEfr-2 :

- Equipe 2.1 chargée de formaliser une grammaire XSD correspondant aux données du tableur
- Equipe 2.2 chargée de générer à partir du tableur un fichier XML respectant la grammaire XSD du groupe 2.1
- Equipe 2.3 chargée de générer un schéma RDFS au format turtle à partir du fichier XML de l'équipe 2.2
- Equipe 2.4 chargée de générer un schéma OWL2 au format turtle à partir du fichier XML de l'équipe 2.2

Nous sommes l'équipe 2.4.

1.3 Utilisation

1.3.1 Répertoire code

Ce répertoire contient 3 fichiers :

- *xml2ttl.xsl* : feuille XSLT qui transforme le xml en OWL au format turtle
- *resources.xml* : version XML du noDEfr2 produite par le groupe projet 2.2
- *noDEfr2_OWL.ttl* : output au format turtle après application de la feuill XSLT sur le fichier XML

1.3.2 Execution

L'application de la feuille XSLT se fait via un processeur XSLT tel que Saxon.

Executez la commande suivante dans le répertoire courant pour lancer le processeur :

```
1  saxonb-xslt -o noDEfr2_OWL.ttl resources.xml xml2ttl.xsl
```

1.3.3 Validation

La validation de la syntaxe du fichier turtle peut se faire sur le site du W3C. Il suffit de copier coller le fichier *noDEfr2_OWL.ttl* et de lancer la validation.

Chapter 2

Formalisation du problème

Le but du projet est d'automatiser la formalisation de données sous forme de schéma RDFS et OWL2. Il ne s'agit donc pas d'un problème de fond mais de forme. C'est à dire que nous ne devons pas construire le modèle NoDEfr2 (car celui ci est déjà représenté dans un tableur) mais le formaliser. Dans notre cas, il s'agit de passer un fichier XML en fichier turtle représentant un schéma OWL2.

Nous pouvons identifier les problèmes généraux suivants :

- manipulation de plusieurs paradigmes
  - paradigme des langages à balise (fichier XML, feuille XSLT)
  - paradigme MLR Metadata for Learning Resources
  - paradigme OWL2
- différentiation méta-modèle et modèle

Pour réaliser le travail demandé, nous avons décomposé le problème comme ceci :

- Comprendre le tableur.
- Lister les colonnes du tableur que nous pouvons représenter en OWL2.
- Passer du paradigme MLR vers le paradigme OWL.
  - par exemple *co-domaine* devient *range*
- Présenter aux équipes 2.1 et 2.2 les éléments que nous avons besoins de retrouver dans le fichier XML.
- Construire la feuille XSLT à partir d'exemples XML.
- Corriger la feuille XSLT pour pouvoir l'appliquer au vrai fichier XML de l'équipe 2.2.
- Corriger la feuille XSLT pour que le fichier turtle soit syntaxiquement validé par le site du W3C

Chapter 3

Présentation des solutions

3.1 Convertisseurs en ligne

Il esxiste plusieurs convertisseurs en ligne qui proposent une conversion de fichier XML vers des fichiers turtle (TTL).
Avantages :

- solution clé en main
- fichier TTL de sortie valide

Inconvénients :

- le fichier XML d'entrée doit respecter la syntaxe XML/RDF
- solution très peu paramétrable
- largement disponible pour une conversion vers RDF mais moins disponible pour une conversion vers OWL2

Nous avons écarté cette solution car ce n'est pas une solution paramétrable et le fichier XML d'entrée n'est pas au format XML/RDF.
Exemple de syntaxe XML/RDF :

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

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:si="https://www.w3schools.com/rdf/">

<rdf:Description rdf:about="https://www.w3schools.com">
<si:title>W3Schools</si:title>
<si:author>Jan Egil Refsnes</si:author>
</rdf:Description>

</rdf:RDF>
```

3.2 Programme Python ou autre langage

Une solution envisageable pour résoudre notre problème est de produire un programme Python qui va charger le fichier XML puis écrire un fichier TTL.
Avantages :

- Les membres du groupe ont un niveau de compétence élevé en Python.
- Il existe un module Python *xml.etree.ElementTree* disponible pour manipuler des arbres XML.

Inconvénients :

- Le langage Python n'est pas spécialisé dans le traitement de données XML.
- Ce type de solution n'est pas abordé dans le cours de Document Web.

Cette solution est tout d'abord plus réaliste que la solution précédente (conversion en ligne). C'est une solution envisageable, mais ce n'est pas la solution optimale.

3.3 Feuille XSLT

XSLT est un langage à balise développé spécialement pour transformer un document XML dans un autre format.
Avantages :

- Langage spécialisé pour traiter des documents XML.
- Compilation simple grace à un processeur XSLT tel que *Saxon*.
- Débuggage simple grace aux messages d'erreurs produite pas les processeurs XSLT.

Concrètement, cette solution ne présente pas d'inconvénients. Le langage est développé spécialement pour transformer des documents XML. De plus, XSLT et Saxon sont des outils largement utilisés : nous sommes certains de trouver des solutions pour les problèmes que nous allons rencontrer. Pour finir, si nous nous retrouvons dans une impasse, il est très peu probable que ce soit à cause du fait qu'il manque des fonctionnalités de traitement de document XML dans le langage XSLT.

Ces éléments font de cette solution la plus fiable. C'est la solution que nous avons implémentée.

Chapter 4

Implémentation

4.1 Description du document XML en entrée

La philosophie adoptée pour représenter NoDEfr2 en format XML est : *"une classe est une suite de relations"*. De plus, on peut noter que chaque colonne du tableur a une balise qui lui correspond dans le document XML.
Extrait du fichier XML produit par l'équipe 2.2 :

```
<classe identifiant="RC0000">
<nom>Entite du monde de la formation</nom>
<definition>classe abstraite qui regroupe les proprietes communes a toutes les
  situations ou a l'ensemble de situations visant un ou plusieurs
objectifs de developpement personnel ou professionnel
</definition>
<sousClasseDe/>
<note>Super classe de toutes les classes. Elle regroupe toutes les proprietes des
  classes presentes dans ce modele  </note>
<relation identifiant="DES000001">
<nom>libelle</nom>
<definition>nommage humainement comprehensible de l'element</definition>
<indicateurLinguistique>oui</indicateurLinguistique>
<coDomaine>Litteral</coDomaine>
<regleDeContenu>MLR String</regleDeContenu>
<raffine/>
<exemple>Pour une instance de la classe RC0020 Offre de formation, le libelle pourrait
   etre " BTS services informatiques aux organisations"</exemple>
<note/>
<cardinaliteMinimale>1</cardinaliteMinimale>
<cardinaliteMaximale>1</cardinaliteMaximale>
<raison/>
</relation>
</classe>
```

Dans cette exemple, nous avons un extrait de la classe *RC0000*. Nous avons d'abord des informations sur la classe puis la liste des relations qui ont pour domaine la classe *RC0000* (dans l'extrait, nous n'avons gardé que la relation *DES000001*).
Pour généré un schéma OWL, les éléments et attributs qui nous interessent le plus sont les suivants :

- *classe* ainsi que les informations basiques (*nom, definition ...*)
  - *identifiant*
  - *sousClasseDe*
- *relation*

- *coDomaine*
- *cardinaliteMinimale*
- *cardinaliteMaximale*

4.2 Description de la sortie souhaitée

Dans un premier temps, nous avons étudié le tableur pour lister les éléments à représenter. Dans le paradigme OWL, ces éléments sont représentés sous la forme de classes, de prédicats et de restrictions sur ces prédicats.
Nous nous sommes d'abord intéressés au tableur car c'est ce dernier qui contient l'information d'origine. Ce sont les informations contenues dans le tableur que nous devons représenter au mieux. La représentation des données du tableur en document XML n'est qu'une étape. Après avoir étudier le tableur, nous avons communiqué aux équipes 2.1 et 2.2 les attentes que nous avions pour le document XML.

Table 4.1: Le tableau suivant rassemble la traduction des éléments et attributs XML en OWL :

XML	OWL
classe	rdfs:Class
nom	rdfs:label
definition	rdfs:comment
sousClasseDe	rdfs:subClassOf owl:intersectionOf
relation	owl:DataProperty owl:ObjectProperty
coDomaien	rdfs:range
cardinalitéMaximale	owl:Restriction owl:minCardinality
cardinalitéMinimale	owl:Restriction owl:maxCardinality

Les extraits de code suivants sont des exemples de classes et de relations que nous souhaitons générer en TTL :

```
noDEfr2:RC0015 rdf:type owl:Class ;
rdfs:label "Certificat, badge, diplome" ;
rdfs:comment "un element de cette classe est la reconnaissance des competences, des
  connaissances, des savoir-faire d'une personne"  ;
rdfs:subClassOf noDEfr2:RC0000 .
```

```
noDEfr2:DES015001 rdf:type  owl:DataProperty ;
rdfs:label "description" ;
rdfs:comment "Description du diplome, certificat ou badge" ;
rdfs:domain noDEfr2:RC0015 ;
rdfs:range  "MLR String" .
_:DES015001Restriction rdf:type owl:Restriction ;
owl:onProperty noDEfr2:DES015001 ;
owl:minCardinality "0" ;
owl:maxCardinality "1" .
```

4.3 Détails techniques

La section précédente donne une idée du résultat que nous attendons. Nous allons maintenant préciser quelques détails techniques de la feuille XSLT.
La feuille XSLT utilise 3 principaux templates :

- classe
- sousClasseDe
- relation

4.3.1 Templates

Le choix des templates *relation* et *classe* est trivial. Un troisième template *sousClasseDe* à été créer car il y a beaucoup de modifications à faire pour passer le contenu du document XML vers un schéma OWL lorsqu'il s'agit de l'héritage des classes. En OWL, les classes sont des ensembles, il est donc possible d'avoir plusieurs classes mères pour une seule classe fille. Pour représenter cela en OWL, on créer une classe anonyme avec le prédicat *owl:intersectionOf* puis on précise que la classe initiale pour laquelle on gère l'héritage possède l'attribut *rdfs:subClassOf classAnonyme*.

Les extraits de code suivants sont respectivement une partie XML et une partie TTL qui illustrent le paragraphe précédent.

```
<classe identifiant="RC0020">
<nom>Offre de formation</nom>
<definition>un element de cette classe est une formation decrite au niveau national,
  cette formation pouvant donner lieu a une attestation
ou a preparer a un diplome, une qualification, un certificat
</definition>
<sousClasseDe>RC0050, RC0030</sousClasseDe>
</classe>
```

```
noDEfr2:RC0020 rdf:type owl:Class ;
rdfs:label "Offre de formation" ;
rdfs:comment "un element de cette classe est une formation decrite au niveau national,
   cette formation pouvant donner lieu a une attestation ou a preparer a un diplome,
  une qualification, un certificat"  .
_:anonymeRC0020 rdf:type owl:Class ;
rdfs:intersectionOf (noDEfr2:RC0050 noDEfr2:RC0030) .
noDEfr2:RC0020 rdfs:subClassOf _:anonymeRC0020 .
```

4.3.2 Restrictions

Nous avons utilisé les Restrictions pour préciser la cardinalité de certaines propriétés. Pour cela, nous utilisons *owl:Restriction*.

Les extraits de code suivants sont respectivements une partie du document XML et une partie du document TTL.

```
<relation identifiant="DES040005">
<nom>description</nom>
<definition>complement d'information sur le montant et/ou le type</definition>
<indicateurLinguistique>oui</indicateurLinguistique>
```

```
<coDomaine>Litteral</coDomaine>
<regleDeContenu>MLR String</regleDeContenu>
<raffine/>
<exemple/>
<note/>
<cardinaliteMinimale>0</cardinaliteMinimale>
<cardinaliteMaximale>1</cardinaliteMaximale>
<raison/>
</relation>
```

```
noDEfr2:DES040005 rdf:type  owl:DataProperty ;
rdfs:label "description" ;
rdfs:comment "complement d'information sur le montant et/ou le type" ;
rdfs:domain noDEfr2:RC0040 ;
rdfs:range  "MLR String" .
_:DES040005Restriction rdf:type owl:Restriction ;
owl:onProperty noDEfr2:DES040005 ;
owl:minCardinality "0" ;
owl:maxCardinality "1" .
```

Chapter 5

Conclusion

5.1 Difficultés rencontrées

Cette section liste les difficultés rencontrées lors du développement du projet.

La mise en page TTL, c'est à dire le bon placement des espaces, des sauts de lignes, des vigules, points-vigules et points. Plusieurs pistes de solutions sont possibles pour résoudre ce problème :

- Modifier le document XML.
- Choisir entre une bonne lisibilité du TTL et une bonne lisibilité de la feuille XSLT.

Pour résoudre cette difficulté, nous avons choisit de priorisé la lisibilité du fichier TTL au détriment de la feuille XSLT. Dans les faits, la feuille XSLT fonctionne et nous sommes satisfait de la lisibilité du fichier TTL. Cependant, la feuille XSLT est très difficile à déchiffrer.

Lors des séances de TP du cours de Document Web, nous avons eu l'occasion d'utiliser un processeur XSLT pour transformer un document xml en fichier HTML. Nous n'avons pas utiliser XSLT pour transformer un document XML vers un fichier TTL. Cette difficulté est liée à la précédente.

Les caractère spéciaux comme les guillemets, les chevrons ou encore les simples côtes nous ont posé problème au début du projet. Nous avon surmonté cette difficulté en choisissant astucieusement nos caractères spéciaux ainsi que les caractères d'échappement.

La dernière difficulté notable est que XSLT est un langage à balise orienté fonctionnel. Il n'est pas simple de créer par exemple une variable que l'on souhaite incrémenter régulièrement. Ce genre de variable peut être utile pour les classes anonymes (identifiant unique). Pour résoudre ce problème, nous avons utilisé l'identifiant des classes ou des propriétés pour lesquelles nous utilisons une classe anonyme.

5.2 Améliorations

Une piste d'amélioration pour notre projet est la simplification et la clarification de la feuille XSLT.

5.3 Conclusion générale

Pour résumer le projet, nous avons produit une feuille XSLT pour transformer un document XML en fichier TTL. Ce fichier TTL représente un schéma OWL2 que nous avons construit à partir du document XML. Le document XML lui même se base sur un tableur qui représente le modèle NoDEfr2, un modèle

décrivant les offres de formations. Le modèle NoDEfr2 est décrit par le méta-modèle MLR (Metadata for Learning Ressources).