



Ce projet a été réalisé dans le cadre du cours de d'Ontologies et Web sémantique du Master 2 Pro. GI. par **Johan GIRARD**, **Pierre ODIN** et **Abdourahamane TOURÉ**.

Rapport Projet Ontologies et Web sémantique

"Ontologie des musées et application TrouverUnMusée.fr"

1 Partie 1 : L'ontologie et les données

1.1 Jeu de données

Le fichier de données sur lequel nous nous sommes basé pour créer notre ontologie et notre application contient la **liste des Musées de France en 2012**. Ces données sont disponibles sur le site `data.gouv.fr` (à l'adresse suivante : <https://www.data.gouv.fr/fr/datasets/liste-et-localisation-des-musees-de-france/>). Ce fichier contient une liste d'environ 1200 musées en indiquant pour chacun d'entre eux différentes informations comme son nom, sa localisation, ses horaires d'ouverture, etc...

1.2 L'ontologie (protege)

L'entité principale de notre ontologie est un Musée. Elle est équivalente à l'entité `Museum` de l'ontologie `schema` (<http://schema.org/>). Les entités suivantes permettent de caractériser un Musée :

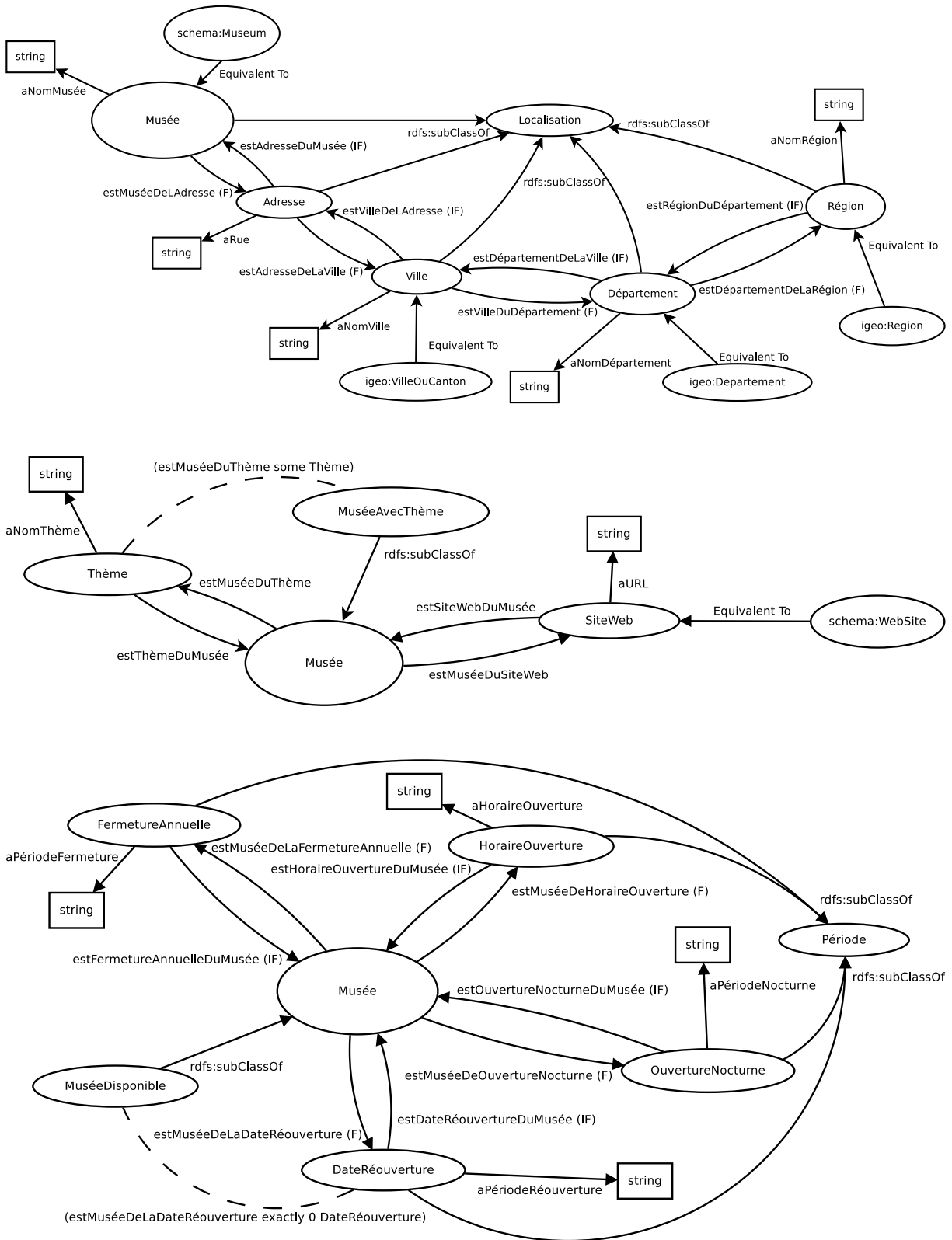
- Adresse qui est liée aux entités `Ville`, `Département` et `Région`. Ces entités sont équivalentes à des entités de l'ontologie `igeo` (<http://rdf.insee.fr/def/geo#>) et sont des sous-classe de l'entité `Localisation`.
- `Thème`
- `SiteWeb` (équivalente à l'entité `WebSite` de l'ontologie `schema`)
- `HorairesOuverture`, `OuvertureNocturne`, `FermetureAnnuelle`, `DateRéouverture` (un musée ayant une date de ré-ouverture est un musée actuellement en fermeture prolongée). Ces entités sont des sous-classe de l'entité `Période`.

L'entité `MuséeAvecThème` est une sous-classe de `Musée` qui regroupe les musées ayant un thème défini. L'entité `MuséeDisponible` est une autre sous-classe de `Musée` qui regroupe les musées qui ne sont pas actuellement en fermeture prolongée.

Toutes les propriétés sont `Asymmetric` et `Irreflexive`. Certaines propriétés sont `Fonctional` et d'autres sont `Inverse Fonctional` (respectivement notées (F) et (IF) dans le schéma de l'ontologie ci-après).

✦ Nous avons également complété l'ontologie avec une propriété nommée `estVilleDeLaRégion` qui utilisait l'option `SubPropertyOf` (Chain) mais nous avons dû la supprimer car elle bloquait l'utilisation du raisonneur.

Le schéma en trois parties de l'ontologie des musées réalisée est le suivant :



1.3 Importation des données

Nous avons utilisé notre propre programme Java pour peupler l'ontologie. La première étape a consisté à ajouter des individus avec un minimum de propriétés à l'ontologie. La seconde étape a consisté à utiliser le raisonneur de protege pour compléter les propriétés des individus et généré un fichier `musee.owl` contenant l'ontologie peuplée au format Turtle.

L'ensemble des classes Java utilisées sont fournies à titre indicatif. Elle se trouvent dans le package `genowl`. De plus, le fichier de données utilisé n'étant pas uniforme, nous avons dû effectuer des modifications "à la main" sur ce fichier. Enfin, des modifications ont également été effectuées sur le fichier `musee.owl` à la fin du processus.

1.3.1 Génération via un programme Java

Nous avons commencé par générer des identifiants pour les valeurs des attributs. Par exemple, pour l'attribut *Département*, le BAS-RHIN correspond à D_32. Nous avons ensuite associé chaque musée à un identifiant en rajoutant par exemple une colonne ID_DEP dans le fichier de données. Cette étape a été répétée pour plusieurs attributs (région, ville, horaire d'ouverture, etc...). L'étape suivante a consisté à générer le code Turtle des individus en indiquant les propriétés nécessaires (par exemple : individu D_32 de type Département avec la propriété `estDépartementDeRégion` R_10 et `aNomDépartement` "BAS-RHIN"). Enfin, nous avons généré les individus de la classe Musée en ajoutant les propriétés faisant le lien avec les individus précédemment créés.

Un traitement particulier a été fait pour l'entité Thème puisque le thème de chaque musée est extrait, dans la mesure du possible, du nom du musée.

1.3.2 Inférences dans protege

Les différentes parties de code Turtle générées avec notre programme Java ont ensuite été regroupées dans le fichier `musee.owl` contenant la structure de l'ontologie (définition des classes et des propriétés). Nous avons alors utilisé le raisonneur Hermit pour inférer les propriétés manquantes : `domain`, `range`, les propriétés inverses (`inverseOf`), etc...

Nous avons corrigé les caractéristiques des propriétés générées par le raisonneur. En effet, le raisonneur avait ajouté pour une raison inconnue la caractéristique *Transitive* à certaines propriétés alors qu'aucune des propriétés de l'ontologie n'est transitive.

Pour finir, nous avons ajouté au fichier `musee.owl` des correspondances entre nos entités et du vocabulaire existant (trouvé sur <http://lov.okfn.org/dataset/lov/>).

2 Partie 2 : Intégration de l'ontologie dans une application Java

2.1 Type de l'application

Nous avons développé une application de type web-app. Pour ceci, nous nous sommes reposés sur une architecture MVC et l'utilisation de Tomcat. De plus, afin d'effectuer nos requêtes SPARQL nous utilisons le framework Apache Jena. Pour simplifier le développement, nous avons travaillé sur un projet Eclipse, aussi si vous souhaitez exécuter l'application la démarche à suivre est la suivante :

- Importer le projet dans votre Eclipse.
- Ajouter un serveur d'exécution Tomcat à votre projet (Tomcat 8).
- Exécuter le projet, le serveur se lance et vous aurez alors accès au projet à la page `localhost:8080/WebOWL/`.

Si vous exécutez le projet sur une machine connectée au réseau de l'UFR ou utilisant un proxy, il est possible que les requêtes SPARQL vers DBpedia ne fonctionnent pas. Pour pallier à ce problème, une solution possible est de paramétrer la variable `proxy_on` de la classe `jena.RequêteDBpedia`.

2.2 Fonctionnalités






2.2.1 Recherche

Notre application est un moteur de recherche de musées. Elle permet de rechercher des musées suivants différents critères. Les critères de recherches sont utilisables en concurrence (union des résultats filtrés) et sont les suivants :

- recherche par **Nom** : vous pouvez rechercher les musées dont le nom contient le mot spécifié. La requête SPARQL traitant cette recherche utilise un FILTER sur les résultats.
- recherche par **Régions** et/ou par **Départements** et/ou par **Villes** : vous pouvez rechercher des musées suivant plusieurs localisations (plusieurs régions, départements ou villes). La requête SPARQL traitant cette recherche utilise des UNION. De plus, un système d'auto-complétion a été mis en place pour améliorer l'expérience utilisateur.
- recherche par **Thème** : vous pouvez rechercher les musées dont le thème contient le mot spécifié. La requête SPARQL traitant cette recherche utilise un FILTER sur les résultats.

A titre d'information, la création de ces requêtes SPARQL se trouve dans la classe `jena.RequêteMusée`. Enfin, la capture d'image suivante détaille la fonctionnalité de recherche de notre outil.

Note : si vous renseignez plusieurs champs de localisation (region, département et ville), le résultat sera l'ensemble des musées respectants au moins un des critères.

	Nom :	<input type="text" value="Folklorique"/>
	Région :	<div><div><input type="button" value="+"/></div><div><input type="text" value="Rhône-Alpes"/></div><div><input type="text" value="Pays de la Loire"/></div><div><input type="button" value="X"/></div></div>
	Département :	<div><div><input type="button" value="+"/></div><div><input type="text" value="GUADELOUPE"/></div><div><input type="text" value="LANDES"/></div><div><input type="button" value="X"/></div></div>
	Ville :	<div><div><input type="button" value="+"/></div><div><input type="text" value="PERPIGNAN"/></div><div><input type="text" value="SAMER"/></div><div><input type="text" value="PRIVAS"/></div><div><input type="button" value="X"/></div></div>
	Thème :	<input type="text" value="Traditions"/>

2.2.2 Résultats de recherche

Une fois la recherche demandée, une ou plusieurs requêtes SPARQL sont générées et exécutées. Un traitement d'union (en Java) est ensuite mis en place pour fournir un résultat cohérent. En outre, les résultats de la recherche sont formulés sous forme d'un tableau contenant le nom du musée, sa région (lien vers fiche Région DBPedia), son département et sa ville. De plus, chacune des colonnes est ordonnable par valeur ascendante ou descendante. Pour réaliser ce tri, nous rajoutons suivant la demande des ORDER BY sur les requêtes SPARQL.

A titre d'information, la création de ces requêtes SPARQL se trouve dans la classe `jena.RequêteMusée`. Enfin, la capture d'image suivante détaille la fonctionnalité d'affichage d'une recherche de notre outil.


Nom du musée 🏛️	Région 🗺️	Département 🗺️	Ville 🗺️
ARCHEA - Musée Intercommunal d'Histoire et d'Archéologie	Île-de-France	VAL D'OISE	LOUVRES
Agropolis-Muséum	Languedoc-Roussillon	HERAULT	MONTPELLIER
Association de l'Horlogerie Aliermontaise	Haute-Normandie	SEINE-MARITIME	SAINT-NICOLAS-D'ALIERMONT
Atelier - Musée du Chapeau	Rhône-Alpes	LOIRE	CHAZELLES-SUR-LYON
Atelier - Musée du Papier	Poitou-Charentes	CHARENTE	ANGOULEME
Atelier Musée Louis Levygue	Centre	LOIR ET CHER	NAVEIL
Atelier-Musée (Ecomusée de la Grande Landes)	Aquitaine	LANDES	LUXEY
Bibliothèque-musée	Pays de la Loire	SARTHE	SAINT-CALAIS
CAPC Musée d'Art Contemporain de Bordeaux	Aquitaine	GIRONDE	BORDEAUX


2.2.3 Fiche Région (DBpedia)


Nous avons mis en place un lien entre notre ontologie et les données de DBpedia en proposant une Fiche Région. Cette fiche propose plusieurs informations sur la région : un résumé, la préfecture, la superficie, la population et le site web de la région. Chacune de ces informations sont extraites de BDpedia.


Pour faire le lien entre notre ontologie et DBpedia, nous utilisons le nom de la région de notre ontologie (aNomRégion) qui correspond à un `rdfs:label` de la région correspondante dans DBpedia (qui appartient à la classe `AdministrativeRegion`). Cette façon de faire le lien nous a obligé à faire une première requête SPARQL qui récupère le nom et une seconde qui interroge DBpedia car nous avons un problème de type de string (@fr est différent de xds:string). Il aurait été plus judicieux de faire le lien directement entre les individus dans notre ontologie.

La classe `requêteDBpedia.java` du package `jena` est utilisé pour cette fonctionnalité. On peut également noter que la requête interrogeant DBpedia utilise un filtre pour récupérer un résumé en français (`FILTER(lang(?res) = "fr")`).

 **Nom : Haute-Normandie**

 **Description :** La Haute-Normandie, en référence à la présence de la capitale historique régionale, est une région de France créée en 1956 qui regroupe deux départements : la Seine-Maritime et l'Eure. Elle correspond à la partie orientale de l'ancienne province de Normandie.

 **Site web :** <http://www.region-haute-normandie.com>

 **Préfecture :** Rouen

2.2.4 Fiche Musée

TODO

3 Répartition du travail

Nous avons travaillé tous ensemble sur le projet sans faire une répartition stricte des différentes parties. Cependant, le travail de Johan GIRARD s'est un peu plus porté sur la partie requêtes SPARQL et aux traitements des résultats, celui de Pierre ODIN sur la création de l'ontologie dans protege et la génération d'individus et enfin, Abdourahamane TOURÉ s'est plus intéressé à la mise en place de l'application et aux fonctionnalités DBpedia et Google Map.