Université de Nice - Sophia Antipolis

Polytech Nice Sophia

Sciences Informatiques 5ème année

Web sémantique

PROJET FINAL

Florian Juroszek, Mathieu Paillart

Enseignante responsable : Catherine Faron-Zucker

Table des matières

1 Introduction

1.1 OWL

Web Ontonlogy Language (OWL) est un langage de représentation des connaissances qui se base sur le modèle RDF. Les ontologies sont un moyen formel de décrire le sens d'un champ d'informations. Il peut être considéré comme une extension de RDF et RDFS [1] et a été conçue pour décrire plus expressément les classes (par exemple `equivalentClass` ou `unionOf`) et les types de propriétés (`equivalentProperty` et `inverseOf` entre autres). De plus, l'inférence de type est largement facilité comparé à RDFS.

1.2 SHACL

Shapes Constraint Language (SHACL) est une recommandation du W3C (depuis 2017) pour valider un graphe RDF selon diverses conditions. Ces dernières peuvent être des conditions sur le nombre de valeur d'une propriété ou des plages numériques attendues par exemple.

1.3 SKOS

Simple Knowledge Organization System (SKOS) est également une recommandation du W3C (depuis 2009) qui vise à représenter la hiérarchie de concepts. Pour cela on écrit du RDF dont le méta-modèle est du OWL.

1.4 Domaine d'application

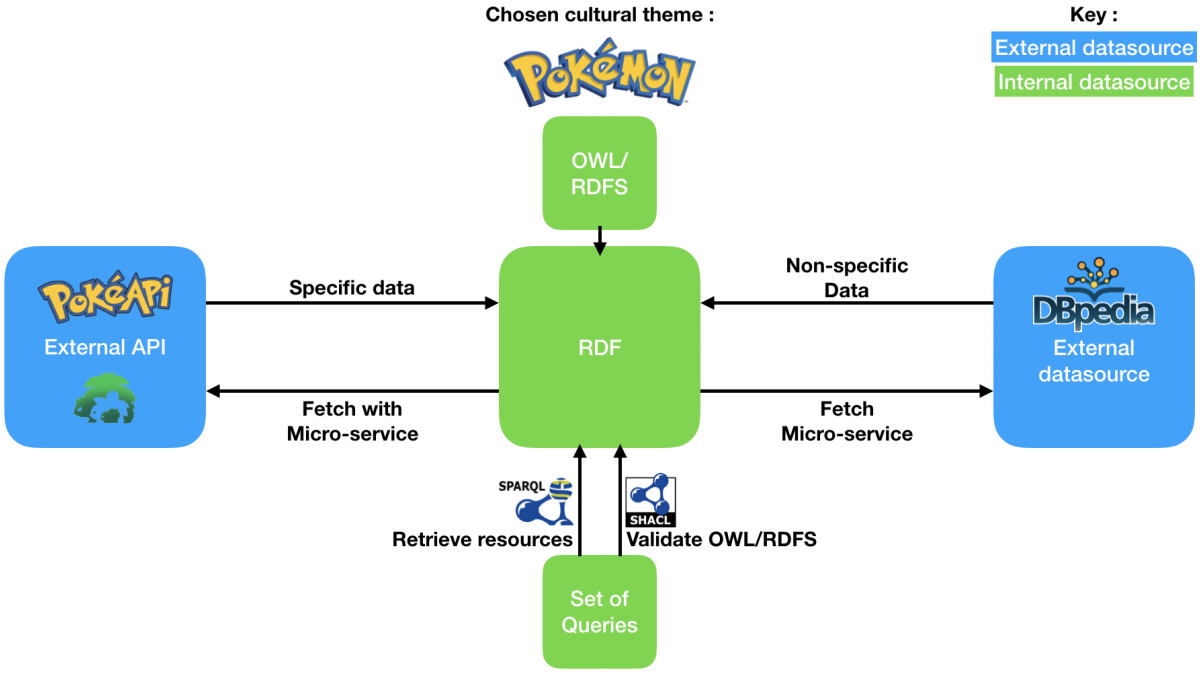
Nous avons choisit **Pokémon** comme domaine d'application culturel. Ce dernier est riche et varié, il permet en effet d'appliquer de nombreux types de propriétés avec les systèmes d'évolutions et de types ; des cardinalités avec un nombre limité de monstres pour les dresseurs.

1. http ://www.standard-du-web.com/web$_o$*ntology*$_l$*anguage.php*

2 Travail effectué

2.1 Architecture générale

Nous pouvons représenter les différentes interactions de notre système avec le schéma suivant :



Nous observons donc que le RDF a une place centrale car nous comptons peupler ce dernier depuis deux **API externes "PokéAPI" et "Pokémon TCG API"** grâce à des micro-services (cf 2.5). Ensuite nous croisons ces informations plus précises avec celles présentes sur DBpedia sur les films et séries Pokémon par exemple. Grâce au vocabulaire OWL/RDFS et les hiérarchies de concepts SKOS, nous pourrons ensuite requêter nos ressources et vérifier les différentes contraintes établies.

2.2 OWL ontology

Nous avons utilisé plusieurs mécanismes que met à disposition OWL comme :

— `owl:TransitiveProperty` pour définir les relations qui se propagent d'une ressource à son voisin. Nous l'utilisons pour les évolutions de Pokémon. Une Pokémon A qui a pour évolution un Pokémon B a par transitivité comme évolution le Pokémon C.

— `owl:IrreflexiveProperty` pour définir les relations qui ne peuvent pas relier une ressource à elle-même. Nous l'utilisons par exemple pour les évolutions car une Pokémon ne peut pas être sa propre évolution.

— `owl:inverseOf` pour définir deux relations qui se complètent dans le sens opposé. Nous l'utilisons pour les relations entre un Pokémon et son dresseur. Si un dresseur possède un Pokémon alors le Pokémon a inversement pour dresseur cette ressource.

— `owl:propertyDisjointWith` pour définir les relations qui ne peuvent exister entre un même sujet et un même objet. Nous l'utilisons pour l'état d'un Pokémon, c'est-à-dire,

qu'il ne peut pas être à la fois capturé et sauvage.

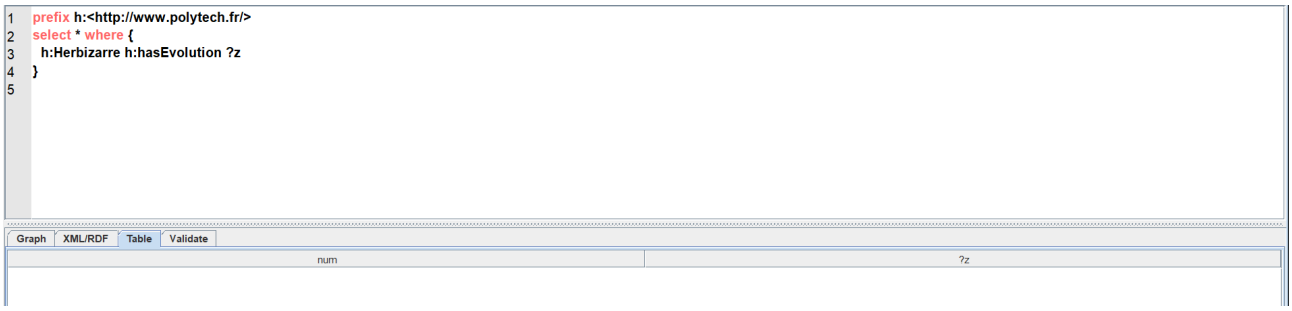
2.3 SPARQL

Nous avons écrit 8 requêtes SPARQL (dont 3 pour vérifier les contraintes avec SHACL) qui permettent de tester notre ontologie en récupérant des ressources ou en validant notre domaine. Vous pouvez les retrouver dans le dossier `sparql` de l'archive.

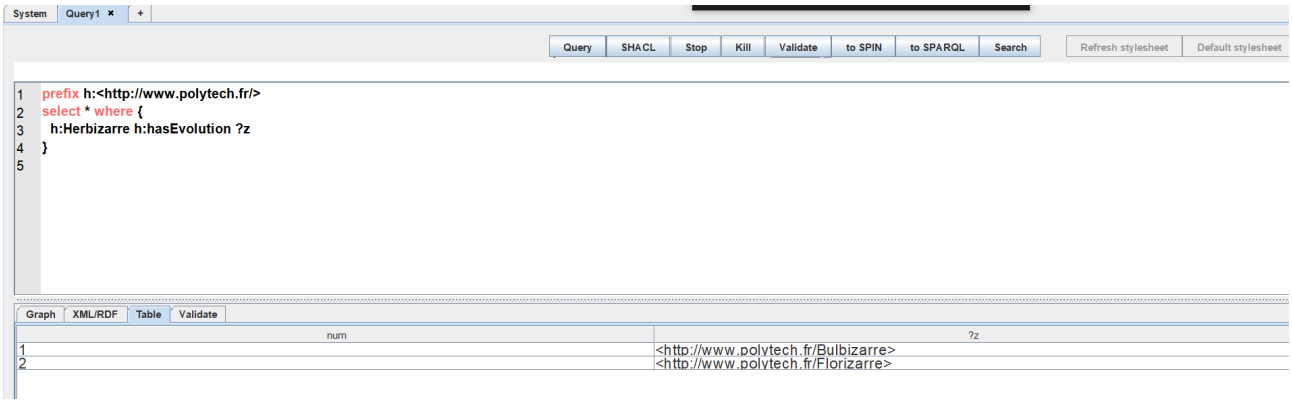
Pour effectuer les requêtes, nous avons utilisé le logiciel Corese [2] via lequel nous pouvons charger le vocabulaire OWL/RDFS et les données RDF.

2.4 Exemples

La première requêtes permet de tester les évolutions. Ici, nous recherchons les évolutions du Pokémon `Herbizarre` (`hasEvolution` relate les évolutions précédentes et futures d'une ressource). Nous constatons qu'il n'y a pas de transitivité et que nous obtenons 0 ressource.



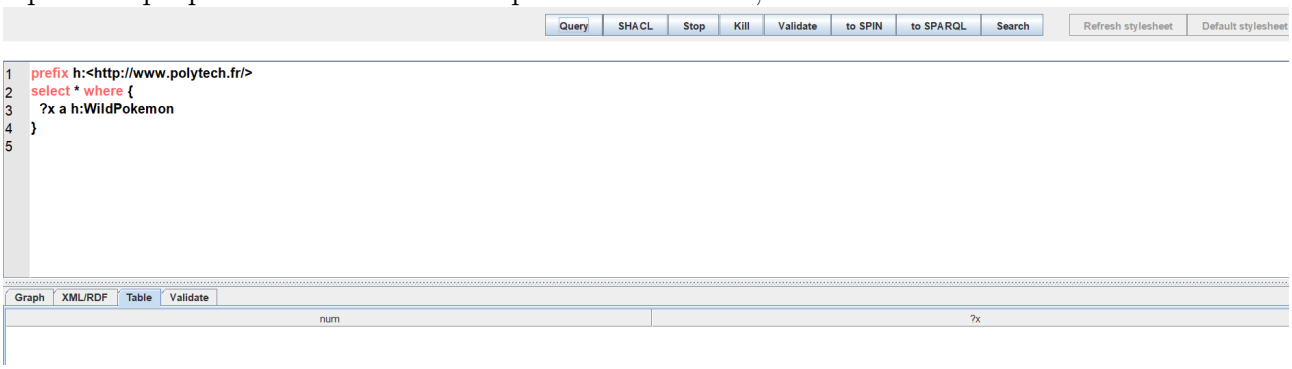
Puis en activant le rdfs/owl sur Corese, la transitivité nous permet d'obtenir les deux ressources de Pokémon qui représentent les évolutions d'`Herbizarre`.



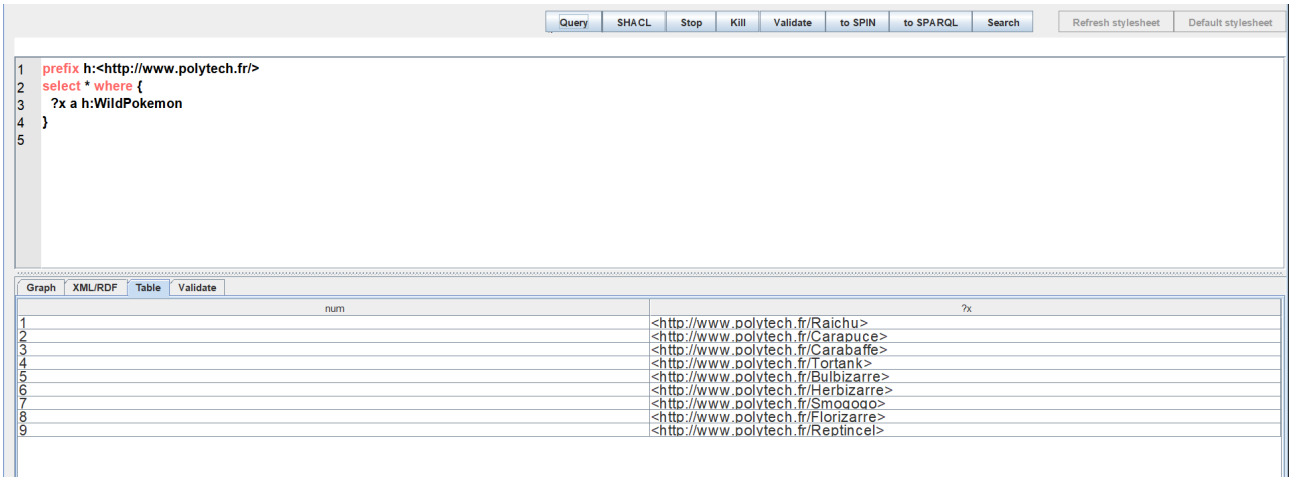
Pour le deuxième exemple, nous recherchons le nombre de type (parmis les Pokémon de notre RDF) qui sont battus par les types de la ressource `Dracaufeu`. Cette requête permet de voir que les forces sont bien les inverses des faiblesses des Pokémon.

2. https ://project.inria.fr/corese/corese-kgram/

Maintenant intéressons-nous aux contraintes avec SHACL. Avec cette requête, nous voulons voir que si la propriété `isWild` est vraie pour un Pokémon, alors c'est un `WildPokemon`.



Nous observons bien lorsque le rdfs/owl est activé les bons résultats qui sont tous les Pokémon sauvages que nous avons déclaré.



2.5 Micro-Services

Nous avons déjà commencé à travailler la partie micro-services, en effet nous avons connecté les deux API entre elles, en faisant par la même occasion une requête à dbpedia, afin d'obtenir la description d'un pokemon.

```
prefix schema: <http://schema.org/>.
prefix pok: <http://pokemon-ontology.org/>.

CONSTRUCT {
    ?pokemon pok:Pokemon
        schema:name ?name;
        pok:baseExperience ?baseExperience;
        pok:nationalPokedexNumber ?nationalPokedexNumber;
        foaf:isPrimaryTopicOf ?isPrimaryTopicOf.
}
WHERE{
    SERVICE <http://localhost/sparql-ms/pokeapi/findPokemon/
    ?keyword=charizard>
    {   ?PokeExp pok:baseExperience ?baseExperience .}

    SERVICE <http://localhost/sparql-ms/pokemontcg/findCards/
    ?keyword=charizard>
    { ?pokedexNumber pok:nationalPokedexNumber ?nationalPokedexNumber.
      ?pokemon schema:name ?name.
    }

    BIND(IRI(CONCAT("http://dbpedia.org/resource/", ?name))
    as ?name_url)

    service <http://dbpedia.org/sparql/> {
        select * where {
            ?name_url foaf:isPrimaryTopicOf ?isPrimaryTopicOf
        }
    }
}
```

Listing 1 – Requête de notre micro-Service

À l'heure actuelle, nous récupérons pour un Pokémon, son numéro national dans le Pokédex (il s'agit d'un identifiant), ainsi que son expérience de base et la description de Wikipedia.

3 Travail à venir

Pour la suite du projet, il nous reste principalement à d'une part faire le lien de notre RDF avec les ressources récupérées par les micro-services. D'autre part, continuer à complexifier notre vocabulaire notamment avec SKOS. Nous avons déjà prévu des requêtes SHACL sur les cardinalités représentant le nombre de Pokémon que peut posséder un dresseur ou encore des valeurs pour les statistiques des Pokémon comprises entre X et Y valeurs.