



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	2
1.1	OWL	2
1.2	SHACL	2
1.3	SKOS	2
1.4	Domaine d'application	2
2	Travail effectué	3
2.1	Architecture générale	3
2.2	OWL ontology	3
2.3	Alignement de l'ontologie	4
2.4	SPARQL	4
2.5	Exemples	4
2.6	Micro-Services	8
2.7	SHACL	11
2.8	SKOS	12
3	Conclusion	12
4	Annexes	13
4.1	Requêtes SPARQL	13
4.2	Validations SHACL	16

1 Introduction

1.1 OWL

Web Ontology 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 RDFS¹ 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 facilitée 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

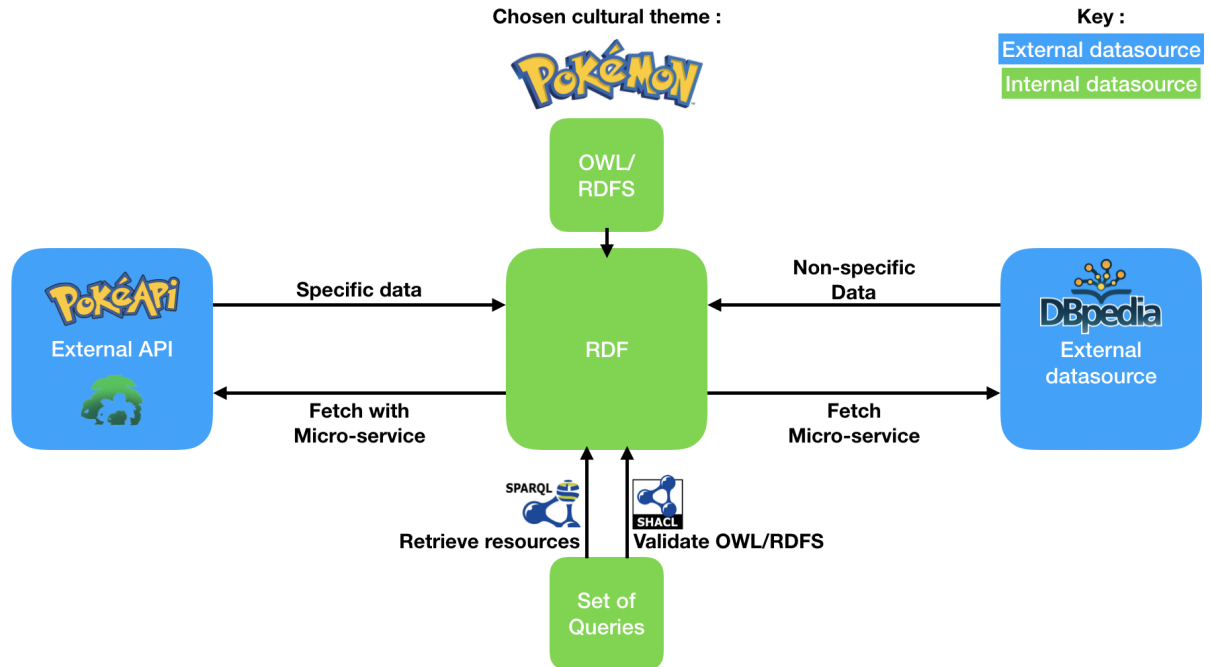
Nous avons choisi **Pokémon** comme domaine d'application culturel. Ce dernier est riche et varié, il permet en effet d'appliquer de nombreux types de propriétés avec les systèmes d'évolutions et de types ; des cardinalités avec un nombre limité de monstres pour les dresseurs.

1. http://www.standard-du-web.com/web_ontology_language.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. 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 pas 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 Alignement de l'ontologie

Nous aurions pu aligner notre ontologie concernant les dresseurs avec `human.rdfs` que nous avons vu en travaux dirigé mais nous avons préféré trouvé un autre référentiel plus pertinent. Nous avons décidé d'utiliser **RELATIONSHIP**, un vocabulaire permettant de décrire les relations entre les personnes. Ainsi nous avons pu définir une sous propriété de `hasMet` pour décrire qu'une dresseur a rencontré une autre lors d'un épisode de la série par exemple. De même pour la propriété `enemyOf` que nous avons adapté à notre vocabulaire avec notre `hasEnemy`.

2.4 SPARQL

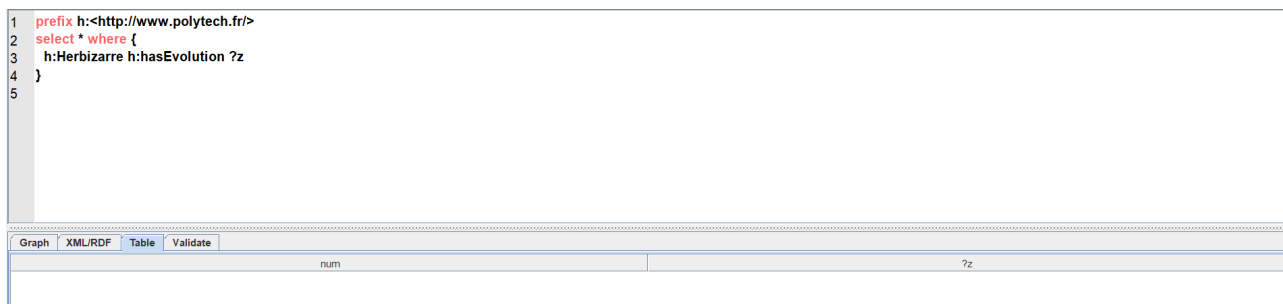
Nous avons écrit 8 requêtes SPARQL (dont 3 pour vérifier les contraintes avec SHACL) qui permettent de tester notre ontologie en récupérant des ressources ou en validant notre domaine. Vous pouvez les retrouver dans le dossier `sparql` de l'archive.

Pour effectuer les requêtes, nous avons utilisé le logiciel Corese² via lequel nous pouvons charger le vocabulaire OWL/RDFS et les données RDF.

2.5 Exemples

Veuillez retrouver en annexes d'autres requêtes que nous avons préparé pour vous démontrer notre travail.

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**.

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

System

Query1

+

Query

SHACL

Stop

Kill

Validate

to SPIN

to SPARQL

Search

Refresh stylesheet

Default stylesheet

1

prefix h:<http://www.polytech.fr/>

2

select * where {

3

h:Herbizarre h:hasEvolution ?z

4

}

5

Graph

XML/RDF

Table

Validate

num

?

z

1

<http://www.polytech.fr/Bulbizarre>

2

<http://www.polytech.fr/Florizarre>

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.

Query	SHACL	Stop	Kill	Validate	to SPIN	to SPARQL	Search	Refresh stylesheet	Default stylesheet
1	prefix	h:	<http://www.polytech.fr/>						
2	SELECT	?pokemon	(count(distinct ?strength) as	?strengthNumber)					
3	WHERE	{							
4		?pokemon	h:name	"Dracaufeu"					
5		?pokemon	h:hasType	?type					
6		?type	h:hasStrength	?strength	}				
7									

Graph	XML/RDF	Table	Validate
1	num	?pokemon	?strengthNumber
		<http://www.polytech.fr/Dracaufeu>	3

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**.

Query	SHACL	Stop	Kill	Validate	to SPIN	to SPARQL	Search	Refresh stylesheet	Default stylesheet
1	prefix	h:	<http://www.polytech.fr/>						
2	select	*	where	{					
3		?x	a	h:WildPokemon					
4	}								
5									

Graph	XML/RDF	Table	Validate
	num		?x

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

Query SHACL Stop Kill Validate to SPIN to SPARQL Search Refresh stylesheet Default stylesheet		
1	prefix h:<http://www.polytech.fr/>	
2	select * where {	
3	?x a h:WildPokemon	
4	}	
5		

Graph XML RDF Table Validate	num	?x
1		<http://www.polytech.fr/Raichu>
2		<http://www.polytech.fr/Carapuce>
3		<http://www.polytech.fr/Carabaffe>
4		<http://www.polytech.fr/Tortank>
5		<http://www.polytech.fr/Bulbizarre>
6		<http://www.polytech.fr/Herbizarre>
7		<http://www.polytech.fr/Smoogoo>
8		<http://www.polytech.fr/Florizarre>
9		<http://www.polytech.fr/Reptincel>

Une **requête plus complexe** permet d'appliquer les règles des combats Pokémon via les différentes capacités et types des Pokémon. En effet, les types des attaques peuvent être plus forts que les types de défense, nous utilisons donc des coefficients 0.5 ou 2 selon cela. La requête permet d'indiquer quelle est la puissance de l'attaque la plus forte (la plus efficace donc) à utiliser pour un Pokémon lorsqu'il en affronte un autre.

```

prefix h:<http://www.polytech.fr/>.
select distinct ?pokemon1 ?maxpower1 where {
  ?pokemon1 h:name "dragonite"
  ?pokemon1 h:hasType ?type1
  ?pokemon1 h:hasMove ?move1
  ?move1 h:moveType ?movetype1

  ?pokemon2 h:name "charizard"
  ?pokemon2 h:hasType ?type2
  ?pokemon2 h:hasMove ?move2
  ?move2 h:moveType ?movetype2

  {select (max(?power1) as maxpower1)
  where { ?movetype1 h:isStronger ?movetype1stronger
          ?movetype1 h:isWeaker ?movetype1weaker
          ?move1 h:power ?powermove1
          ?type2 h:isStronger ?test2
          BIND(IF(?movetype1stronger= ?type2,2*?powermove1,
                  IF(?movetype1weaker = ?type2, 0.5*?powermove1,
                    ?powermove1)) as ?power1)
        }
  }}

```

Listing 1 – Requête sparql appliquant les règles de pokemon

```
prefix h:<http://www.polytech.fr/>
select distinct ?pokemon1 ?maxpower1 where {
  ?pokemon1 h:name "dragonite"
  ?pokemon1 h:hasType ?type1
  ?pokemon1 h:hasMove ?move1
  ?move1 h:moveType ?movetype1

  ?pokemon2 h:name "charizard"
  ?pokemon2 h:hasType ?type2
  ?pokemon2 h:hasMove ?move2
  ?move2 h:moveType ?movetype2
```

aph	XML/RDF	Table	Validate	
		num		
			<http://www.polytech.fr/Dragonite>	300

2.6 Micro-Services

Concernant la partie micro-services, nous nous sommes connecté à deux API **PokeAPI** et **PokemonTCG**. Cela nous a permis de récupérer respectivement des informations générales sur les Pokémon (noms, identifiant dans le système de Pokemon) et des informations sur les cartes à jouer (points de vie, attaques par exemple). En parallèle de cela nous obtenons la description d'un Pokemon en effectuant une requête à DBPedia.

Pour expliquer plus dans le détail :

La requête à PokeAPI permet en plus de compléter notre RDF en renseignant tous les types de chaque Pokemon, toutes ces capacités ainsi que ces compétences.

Ce qui signifie qu'avec un RDF de 150 lignes, nous pouvons utiliser un **INSERT** afin d'insérer les différentes données que nous récupérerons dans notre RDF et cela permet de récupérer plus de 600 lignes de RDF avec des informations automatiquement agrégées pour chaque Pokémon.

La requête SPARQL suivante nous a permis de réaliser cela :

```
prefix pok: <http://www.polytech.fr/>.
INSERT {
    ?pokemon a pok:Pokemon;
    pok:nationalPokedexNumber ?nationalPokedexNumber;
    pok:baseExperience ?baseExperience;
    pok:weight ?weight;
    pok:height ?height;
    pok:hp ?hp;
    pok:hasAbility ?abilityURI;
    pok:hasMove ?moveURI;
    pok:hasType ?typeURIPokemon;
    pok:imageLink ?imageUrl;
    pok:wikipediaLink ?isPrimaryTopicOf.

    ?abilityURI a pok:Ability;
    pok:name ?abilityname.

    ?moveURI a pok:Move;
    pok:name ?movenamename;
    pok:pp ?pp;
    pok:accuracy ?accuracy;
    pok:power ?power;
    pok:typename ?typeURI;
    pok:effect ?effect.

    ?typeURI a pok:Type;
    pok:name ?typename.
}
WHERE{
    ?pokemon a pok:Pokemon
```

```

        ?pokemon pok:name ?name

bind (uri(concat("http://localhost/sparql-ms/pokemontcg/
findCards/?keyword=", ?name)) as ?pokemontcg)

        SERVICE ?pokemontcg
        {
            ?pokemontcgapi pok:nationalPokedexNumber ?nationalPokedexNumber;
            pok:hp ?hp;pok:imageUrl ?imageUrl.
        }

        bind (uri(concat("http://localhost/sparql-ms/pokeapi/
findPokemon/?keyword=", ?name)) as ?pokeapi_findpokemon)
        SERVICE ?pokeapi_findpokemon
{ SELECT * WHERE {
            ?pokemonpokeapi pok:baseExperience ?baseExperience;
            pok:weight ?weight;
            pok:height ?height;
            pok:abilityname ?abilityname;
            pok:movename ?movename;
            pok:type ?typenamePokemon.
        } LIMIT 4
}
BIND(IRI(CONCAT("http://www.polytech.fr/",concat(UCASE(substr(
?typenamePokemon, 1,1)),substr(?typenamePokemon,2)))) as ?typeURIPokemon)

bind(uri(concat("http://localhost/sparql-ms/pokeapi/
findMove/?keyword=",?movename)) as ?pokeapi_findmove)
SERVICE ?pokeapi_findmove
        { ?movepokeapi pok:pp ?pp; pok:power ?power;
            pok:accuracy ?accuracy; pok:effect ?effect;
            pok:typename ?typename.
        }

BIND(IRI(CONCAT("http://www.polytech.fr/",
concat(UCASE(substr(?typename, 1,1)),substr(?typename,2)))) as ?typeURI)
BIND(IRI(CONCAT("http://www.polytech.fr/",
concat(UCASE(substr(?abilityname, 1,1)),substr(?abilityname,2))))
as ?abilityURI)
BIND(IRI(CONCAT("http://www.polytech.fr/",
concat(UCASE(substr(?movename, 1,1)),substr(?movename,2)))) as ?moveURI)
BIND(IRI(CONCAT("http://dbpedia.org/resource/",
concat(UCASE(substr(?name, 1,1)),substr(?name,2)))) as ?name_url)

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

```

```

        ?name_url foaf:isPrimaryTopicOf ?isPrimaryTopicOf
    }
}
}

```

Listing 2 – Requête de notre micro-Service

La requête est assez complexe, mais nous voulons insérer les choses suivantes dans notre RDF :

- Des Pokémon (ils sont déjà présents, nous voulons juste rajouter certaines valeurs)
- Des *ability*, les capacités des Pokémon
- Des *move*, les attaques des Pokémon
- Des types, les types des Pokémon ou des attaques

Pour réaliser cela, dans un premier temps nous devons charger notre RDF minimal contenant les Pokémon ainsi que leur nom en minuscule afin de pouvoir effectuer les différentes requêtes sur les API.

Comme nous pouvons le voir dans le **WHERE**, pour chaque Pokémon, nous récupérons son nom.

Nous utilisons ensuite un **BIND** URI afin de pouvoir réaliser une requête pour chaque Pokémon puisque la variable **?name** va être différente pour chaque Pokémon.

Dans un premier temps nous réalisons la requête à PokémonTCG et récupérons l'identifiant national du Pokédex de notre Pokémon, ses HP, ainsi qu'une image de la carte Pokémon associée.

Ensuite nous réalisons deux requêtes à PokeAPI. Une première pour compléter le Pokémon avec des statistiques qui lui sont propres tels que son poids, sa taille, ses capacités, ses attaques et son type.

Nous avons délibérément choisi de mettre un **LIMIT 4** à la fin de cette query, car un Pokémon peut tout au long de son existence apprendre plusieurs attaques (plus de 100) sauf que dans les règles du jeu Pokémon, il ne peut apprendre que seulement 4 attaques (d'où le **LIMIT 4**).

Ensuite, nous réalisons une nouvelle requête à PokeAPI, cette fois pour récupérer des informations sur les attaques des Pokémon puisque précédemment nous avons récupéré le nom des attaques, nous pouvons donc réaliser cette requête.

Nous disposions déjà du nom de l'attaque, maintenant nous récupérons les PP, la puissance de l'attaque, la précision associée, l'effet ainsi que le type de l'attaque.

Les **BIND** IRI utilisés à la fin nous permettent d'utiliser notre base (namespace) avec le nom de chaque concept que nous souhaitons rajouter dans notre RDF, que j'ai cité précédemment. Cela nous permet donc de construire les différentes ressources dans le RDF (Par exemple, si nous récupérons l'attaque qui s'appelle "hydrocanon", nous allons créer une ressource avec l'URI suivant dans notre RDF : `<http://www.polytech.fr/Hydrocanon> a pok:Move`).

Ce phénomène est assez explicite de par le fait que nous avons indiqué dans le `INSERT` que les ressources que nous construisons s'appellent :

- `?abilityURI`
- `?moveURI`
- `?typeURI`

La dernière action que cette requête réalise est une query à DBpedia, afin d'obtenir le lien Wikipedia de chaque Pokémon. Ce qui fait office de description du Pokémon.

Dans le ZIP que nous vous avons rendu, il y a un dossier `micro-services` dans lequel vous pourrez trouver notre requête : `final_request.text` ainsi que dans le dossier `services` les deux micro-services que nous avons réalisé (Pokemontcg et Pokeapi).

2.7 SHACL

Nous avons validé notre RDF en utilisant quelques règles de l'univers Pokémon. Par exemple, nous vérifions qu'un Pokémon ait entre 1 et 3 types disjoints. On s'assure également que les valeurs des habilités, points de vie, puissance des attaques soient dans les bons intervalles. Par exemple, les points de pouvoir ne peuvent être compris qu'entre 0 et 40.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://www.polytech.fr/pokemon#> .

<TypeValidation> a sh:NodeShape ;
  sh:targetClass <Pokemon> ;
  sh:property [
    sh:path <hasType> ;
    sh:qualifiedMinCount 1 ;
    sh:qualifiedMaxCount 3 ;
    sh:qualifiedValueShapesDisjoint true;
  ] .

<PPValidation> a sh:NodeShape ;
  sh:targetClass <Pokemon> ;
  sh:property [
    sh:path <pp> ;
    sh:minCount 0 ;
    sh:maxCount 40 ;
  ] .

...
```

Listing 3 – Extrait de nos vérifications SHACL

2.8 SKOS

Nous faisons le lien entre le SKOS et le RDF en réutilisant les mêmes URI. En effet, après avoir discuté de cela avec Monsieur Corby, cela semblait la meilleure solution, l'utilisation d'une propriété n'ayant pas vraiment de sens selon nous.

Le thème (**ConceptScheme**) Pokémon peut être utilisé comme un concept global. On peut reprendre des concepts plus généraux comme **Monstre** ou **Animaux** avec la notion de `skos:broader` ; des concepts plus spécifiques `skos:narrower`. Nous avons utilisé aussi les différents types de labels, préférés et alternatifs ainsi qu'un `skos:hiddenLabel` pour la faute d'orthographe sur l'accent de Pokémon par exemple. Enfin, nous avons utilisé des exemples pour les concepts et les liens entre ces derniers avec `skos:related`.

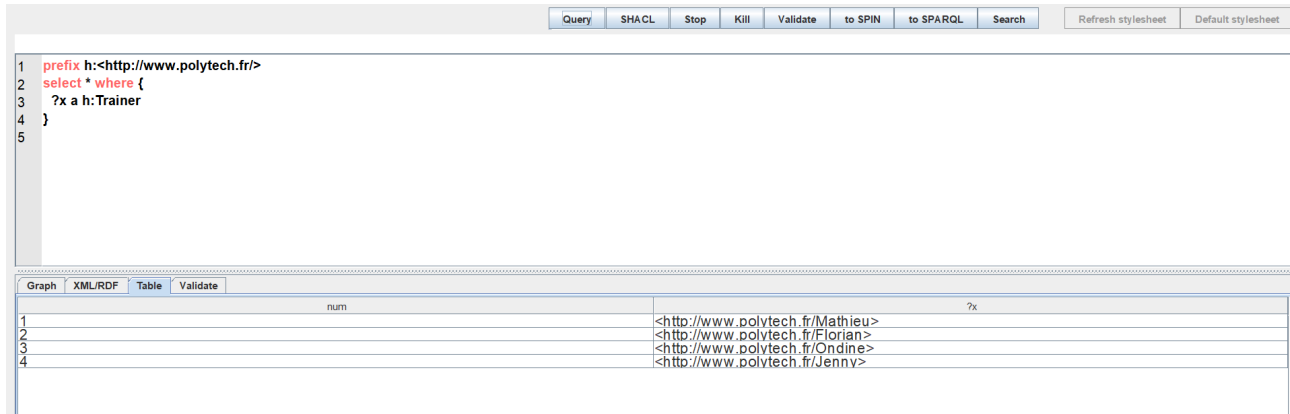
3 Conclusion

Ce projet a permis de mettre en application les différents langages de représentation et requêtes d'information dans les domaines du web de données et du web sémantique. Nous avons pu constater qu'un simple vocabulaire RDFS ne peut pas suffire à décrire de manière complète et efficace des ressources. De plus, l'utilisation de micro-services pour récupérer des données et les mettre en lien avec notre vocabulaire nous a permis de construire un RDF beaucoup plus complet, plus rapidement. Aussi, nous avons pu avoir une première expérience avec la notion de hiérarchie de concept qui permet de prendre du recul sur notre modélisation. Enfin, nous avons utilisé des contraintes vérifiables par des requêtes SHACL afin de s'assurer, avec les requêtes SPARQL également, de l'efficacité de notre vocabulaire.

4 Annexes

4.1 Requêtes SPARQL

- "Si une ressource possède un Pokémon, alors c'est un *traineur*"



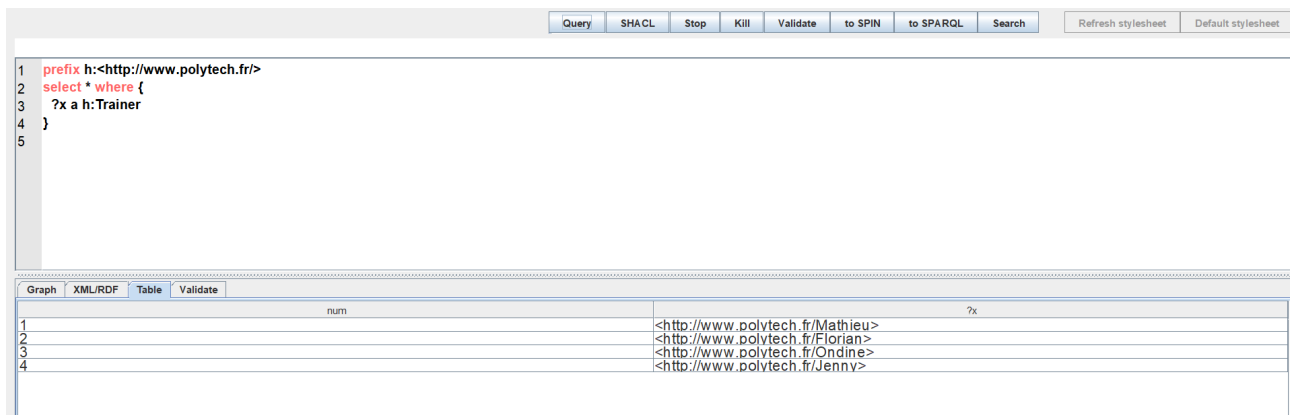
The screenshot shows a SPARQL query interface. At the top, there is a toolbar with buttons: Query, SHACL, Stop, Kill, Validate, to SPIN, to SPARQL, Search, Refresh stylesheet, and Default stylesheet. Below the toolbar is a text area containing the following SPARQL query:

```
1 prefix h:<http://www.polytech.fr/>
2 select * where {
3   ?x a h:Trainer
4 }
5
```

Below the query area, there is a table with two columns: 'num' and '?x'. The table contains four rows of results:

num	?x
1	<http://www.polytech.fr/Mathieu>
2	<http://www.polytech.fr/Florian>
3	<http://www.polytech.fr/Online>
4	<http://www.polytech.fr/Jenny>

FIGURE 1 – Sans OWL



The screenshot shows a SPARQL query interface, identical to Figure 1. It displays the same SPARQL query and the same table of results.

num	?x
1	<http://www.polytech.fr/Mathieu>
2	<http://www.polytech.fr/Florian>
3	<http://www.polytech.fr/Online>
4	<http://www.polytech.fr/Jenny>

FIGURE 2 – Avec OWL

- "Si une ressource a un dresseur (avec `hasTraineur`) alors c'est un Pokémon capturé"

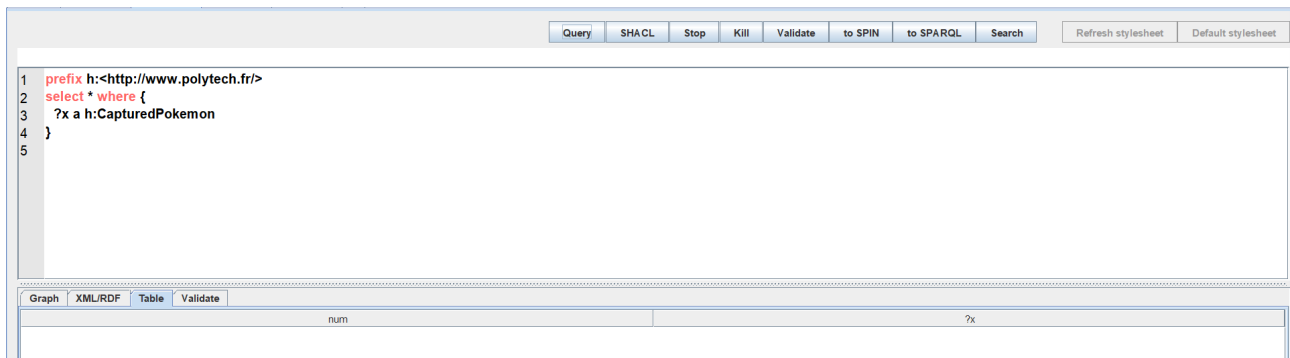


FIGURE 3 – Sans OWL

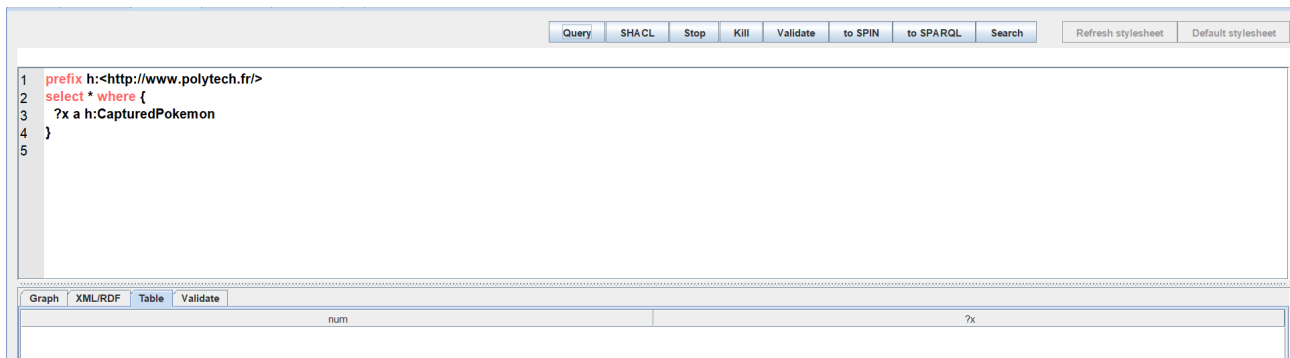


FIGURE 4 – Avec OWL

- "Si une ressource est sauvage (avec `isWild` vaut vrai) alors c'est un Pokémon sauvage"

Query	SHACL	Stop	Kill	Validate	to SPIN	to SPARQL	Search	Refresh stylesheet	Default stylesheet
-------	-------	------	------	----------	---------	-----------	--------	--------------------	--------------------

```

1 prefix h:<http://www.polytech.fr/>
2 select * where {
3   ?x a h:WildPokemon
4 }
5

```

Graph	XML/RDF	Table	Validate
num	?x		

FIGURE 5 – Sans OWL

Query	SHACL	Stop	Kill	Validate	to SPIN	to SPARQL	Search	Refresh stylesheet	Default stylesheet
-------	-------	------	------	----------	---------	-----------	--------	--------------------	--------------------

```

1 prefix h:<http://www.polytech.fr/>
2 select * where {
3   ?x a h:WildPokemon
4 }
5

```

Graph	XML/RDF	Table	Validate
num	?x		
1	<http://www.polytech.fr/Raichu>		
2	<http://www.polytech.fr/Carapuce>		
3	<http://www.polytech.fr/Carabaffe>		
4	<http://www.polytech.fr/Tortank>		
5	<http://www.polytech.fr/Bulbizarre>		
6	<http://www.polytech.fr/Herbizarre>		
7	<http://www.polytech.fr/Smoouoo>		
8	<http://www.polytech.fr/Florizarre>		
9	<http://www.polytech.fr/Reptincel>		

FIGURE 6 – Avec OWL

4.2 Validations SHACL

- "Vérifie que les Pokémon aient minimum 1 type et maximum 3."

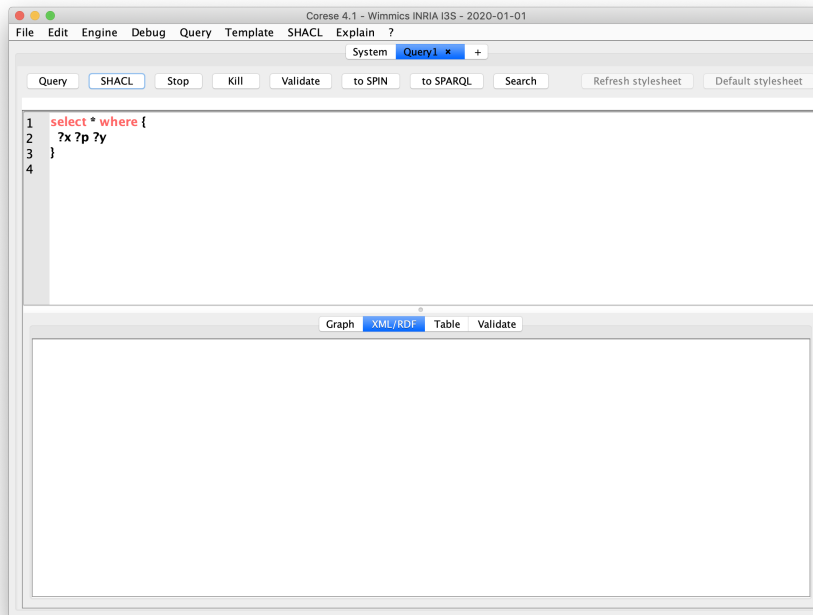


FIGURE 7 – Avec un RDF valide

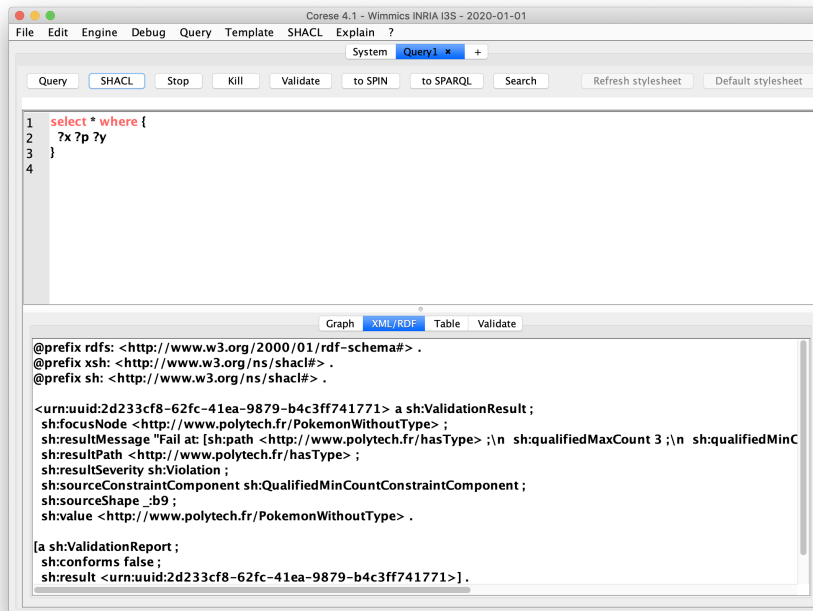


FIGURE 8 – En ajoutant une ressource de la classe Pokémon sans propriété `hasType`