Technical Report: Semantic Web Project on Pokémon Knowledge Graph Construction

NKHILI Nabil
ZAHIR Mohamed

Date : January 24, 2025

1. Introduction

This report documents the development of a Semantic Web project aimed at creating a Knowledge Graph (KG) from structured and unstructured data sources, specifically focusing on the domain of Pokémon. The goal of this project was to leverage RDF and SPARQL technologies to organize and interconnect data, enabling semantic queries and insights into the relationships between entities.

2. Project Overview

2.1 Objectives

- Extract structured data from Bulbapedia and other relevant sources.
- Transform the extracted data into RDF triples.
- Develop an ontology to standardize the representation of data in the Pokémon domain.
- Implement a SPARQL endpoint for querying the Knowledge Graph.

2.2 Technology Stack

The following technologies were employed:

Core Technologies:

- RDFLib: For creating and processing RDF triples.
- Apache Jena Fuseki: To serve as the SPARQL endpoint and triplestore.
- Turtle (.ttl): Format for storing RDF data.

Data Extraction Tools:

- MediaWiki API: For querying structured infobox data from Bulbapedia.
- Python libraries (e.g., BeautifulSoup): For web scraping additional unstructured data.

Frontend (Optional):

- HTML5, CSS3, JavaScript: For presenting results from the Knowledge Graph queries.

3. Development Methodology

3.1 Phase 1: Data Collection and Parsing

The project began with identifying relevant entities for the Knowledge Graph:

- **Pokémon:** Core information including types, abilities, evolution, and statistics.
- **Moves:** Attributes such as power, accuracy and category.
- **Abilities:** Both regular and hidden abilities associated with Pokémon.
- **Egg Groups:** Grouping of Pokémon for breeding compatibility.
- **Episodes:** Details of Pokémon anime episodes, including titles, numbers, and featured Pokémon.
- **Types:** Classification of Pokémon and moves (e.g., Fire, Water, Electric).
- **Items:** Objects used in the Pokémon universe, such as Poké Balls and potions.

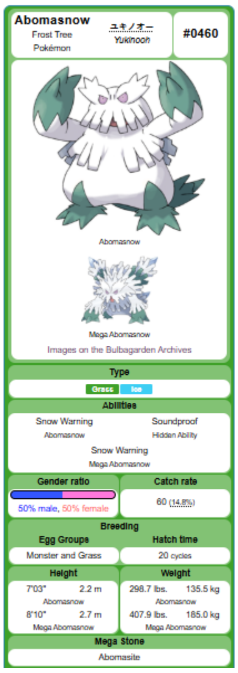
Data extraction followed two methods:

- **MediaWiki API:** Used to extract structured infoboxes.
- **Web Scraping:** For additional details not covered by APIs, leveraging Python scripts for HTML parsing.

3.2 Phase 2: Ontology Design

An ontology was created to model the relationships and attributes of the collected data:

- Key classes included `Pokemon`, `Move`, `Ability`, `EggGroup`, `Type`, `Item`, and `Episode`.
- Properties were defined to link entities, e.g., `hasType`, `hasAbility`, `appearsInEpisode`, `isInEggGroup`, and `usedIn`.
- Multilingual labels (`rdfs:label`) were added for better global accessibility.



Abomasnow ユキノオー #0460
Frost Tree Pokémon Yukinooh

Abomasnow

Mega Abomasnow

Images on the Bulbagarden Archives

Type
Grass Ice

Abilities
Snow Warning (Abomasnow) — Soundproof (Hidden Ability)
Snow Warning (Mega Abomasnow)

Gender ratio	Catch rate
50% male, 50% female	60 (14.8%)

Breeding

Egg Groups	Hatch time
Monster and Grass	20 cycles

Height	Weight
7'03" 2.2 m (Abomasnow)	298.7 lbs. 135.5 kg (Abomasnow)
8'10" 2.7 m (Mega Abomasnow)	407.9 lbs. 185.0 kg (Mega Abomasnow)

Mega Stone
Abomasite

Example Turtle snippet:

```
ex:Abomasnow a ex:Pokemon ;

  rdfs:label "Abomasnow (Pokémon)" ;

  ns1:hasAbility ex:Snow_Warning ;

  ns1:hasEggGroup ex:Grass,

    ex:Monster ;

  ns1:hasHiddenAbility ex:Soundproof ;

  ns1:hasType ex:Grass,

    ex:Ice ;

  ex:category "Frost Tree"^^xsd:string ;

  ex:color "White"^^xsd:string ;

  ex:friendship 70 ;

  ex:hasImage "https://archives.bulbagarden.net/media/upload/1/14/0460Abomasnow.png"^^xsd:anyURI
;

  ex:hatchtime "20"^^xsd:string ;

  ex:height 2.2 ;

  ex:jname "ユキノオー"^^xsd:string ;

  ex:ndex "0460"^^xsd:string ;

  ex:weight 135.5 ;

  schema1:name "Rexblisar"@de,

    "Abomasnow"@en,

    "Yukinooh"@en,

    "Abomasnow"@es,

    "Blizzaroi"@fr,

    "Abomasnow"@it,

    "ユキノオー"@ja,

    "눈설왕"@ko,

    "暴雪王"@zh .
```

```
 ns2:wikiPageExternalLink dbr:Abomasnow,
<https://bulbapedia.bulbagarden.net/wiki/Abomasnow_(Pokémon)>,

        <https://www.wikidata.org/wiki/Abomasnow> ;
```

In the provided RDF snippet, the ontology for `Abomasnow` adheres to the predefined structure by using meaningful ***classes** and **properties**.* `Abomasnow` is identified as an instance of the `ex:Pokemon` class, respecting the domain-specific definition of a Pokémon. Each property, such as `ns1:`*`hasAbility`*, `ns1:`*`hasEggGroup`*, and `ns1:`*`hasType`*, is applied with appropriate domains and ranges as defined in the ontology. For instance, `ns1:`*`hasAbility`* links `Abomasnow` to its abilities, including the hidden ability `ex:Soundproof`, while `ns1:`*`hasType`* associates it with its dual types, `ex:Grass` and `ex:Ice`. Multilingual labels (`schema1:name`) and URIs (e.g., `dbr:Abomasnow`) provide global accessibility and interconnectivity with external knowledge bases like *DBpedia* and Bulbapedia. *Literal values* such as `ex:height` and `ex:weight` use appropriate `xsd` datatypes to ensure semantic precision. This adherence to the ontology ensures consistency, semantic accuracy, and integration with external datasets.

3.3 Phase 3: RDF Triple Generation

The extracted data was transformed into RDF triples:

- **Parsing and Structuring:** Python scripts were developed to parse infoboxes and format data into RDF.
- **Validation:** Data was validated against the ontology to ensure consistency and completeness.

Example Turtle snippet linking an episode and a Pokémon debut:

```
@prefix ep: <http://example.org/episodes/> .

@prefix ex: <http://example.org/pokemon/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ep:DP126 a ep:Episode ;

   ep:hasAnimation "Team Iguchi" ;

   ep:hasEnding "もえよ　ギザみみピチュー！" ;

   ep:hasImage
"https://archives.bulbagarden.net/media/upload/thumb/1/17/DP126.png/267px-DP126.png" ;

   ep:hasJapanReleaseDate "2009-05-07"^^xsd:date ;

   ep:hasOpening "ハイタッチ！" ;

   ep:hasPokemonDebut ex:Abomasnow ;

   ep:hasScreenplay "再学　Atsuhiro Tomioka" ;

   ep:hasStoryboard "大幅秀明 Hideaki Ōba" ;

   ep:hasTitle "DP126" ;

   ep:hasUSReleaseDate "2009-10-03"^^xsd:date .
```

3.4 Phase 4: SPARQL Endpoint Deployment

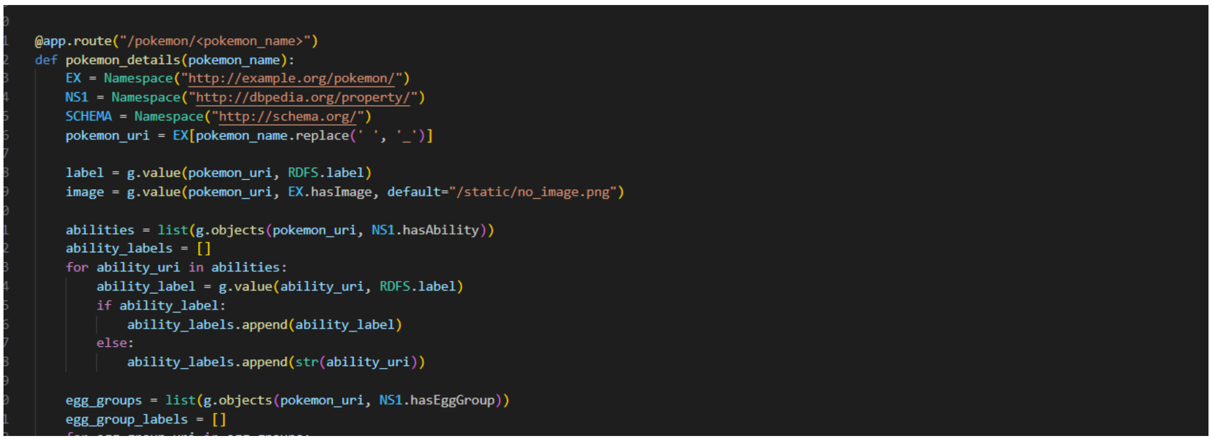
Apache Jena Fuseki was used to host the Knowledge Graph:

- RDF data was uploaded in Turtle format.
- SPARQL queries were tested for retrieving specific data and exploring relationships.

3.5 : Application for Data Extraction and RDF Creation

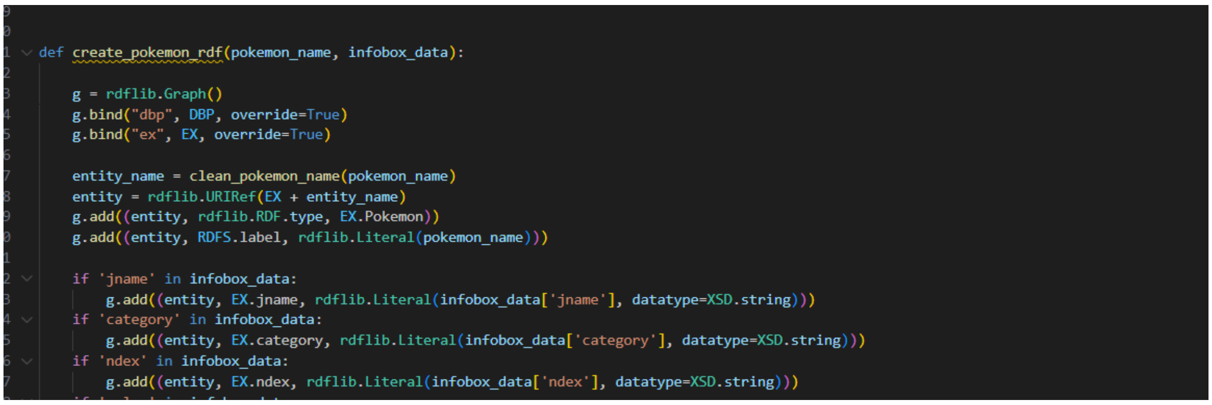
1. **Data Extraction:**
   - Data is extracted from Bulbapedia using both the MediaWiki API and web scraping techniques. The **API** is used to fetch **structured data** from infoboxes, while **web scraping** captures additional **unstructured details** such as image URLs.
2. **RDF Generation:**
   - The extracted data is transformed into RDF triples using the `rdflib` library.
   - Each entity (e.g., Pokémon, Move, Episode) is represented as a URI, and properties like `hasType` and `hasAbility` are assigned based on the ontology.

Flask Application (app.py):



```python
@app.route("/pokemon/<pokemon_name>")
def pokemon_details(pokemon_name):
    EX = Namespace("http://example.org/pokemon/")
    NS1 = Namespace("http://dbpedia.org/property/")
    SCHEMA = Namespace("http://schema.org/")
    pokemon_uri = EX[pokemon_name.replace(' ', '_')]

    label = g.value(pokemon_uri, RDFS.label)
    image = g.value(pokemon_uri, EX.hasImage, default="/static/no_image.png")

    abilities = list(g.objects(pokemon_uri, NS1.hasAbility))
    ability_labels = []
    for ability_uri in abilities:
        ability_label = g.value(ability_uri, RDFS.label)
        if ability_label:
            ability_labels.append(ability_label)
        else:
            ability_labels.append(str(ability_uri))

    egg_groups = list(g.objects(pokemon_uri, NS1.hasEggGroup))
    egg_group_labels = []
```

RDF Creation (pokemon.py):



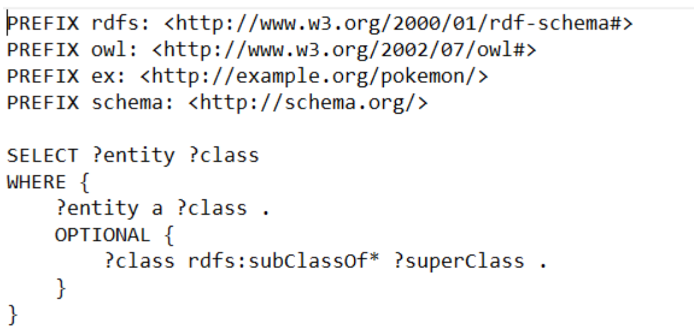
```python
def create_pokemon_rdf(pokemon_name, infobox_data):

    g = rdflib.Graph()
    g.bind("dbp", DBP, override=True)
    g.bind("ex", EX, override=True)

    entity_name = clean_pokemon_name(pokemon_name)
    entity = rdflib.URIRef(EX + entity_name)
    g.add((entity, rdflib.RDF.type, EX.Pokemon))
    g.add((entity, RDFS.label, rdflib.Literal(pokemon_name)))

    if 'jname' in infobox_data:
        g.add((entity, EX.jname, rdflib.Literal(infobox_data['jname'], datatype=XSD.string)))
    if 'category' in infobox_data:
        g.add((entity, EX.category, rdflib.Literal(infobox_data['category'], datatype=XSD.string)))
    if 'ndex' in infobox_data:
        g.add((entity, EX.ndex, rdflib.Literal(infobox_data['ndex'], datatype=XSD.string)))
```

4. Implementation Details

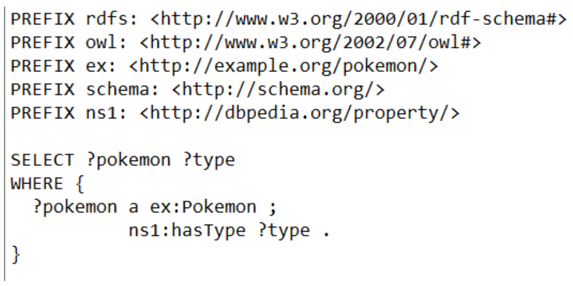
4.1 RDF Modeling and Storage

- RDF triples were stored in Turtle format for readability and compatibility.
- The ontology defined domain and range for each property to ensure semantic accuracy.

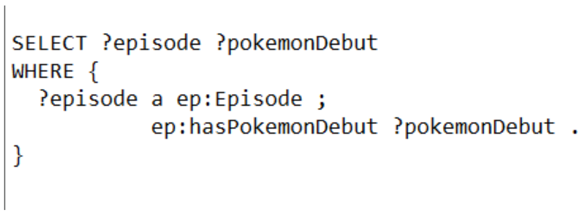
4.2 SPARQL Query Examples



```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX ex: <http://example.org/pokemon/>
PREFIX schema: <http://schema.org/>

SELECT ?entity ?class
WHERE {
    ?entity a ?class .
    OPTIONAL {
        ?class rdfs:subClassOf* ?superClass .
    }
}
```



```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX ex: <http://example.org/pokemon/>
PREFIX schema: <http://schema.org/>
PREFIX ns1: <http://dbpedia.org/property/>

SELECT ?pokemon ?type
WHERE {
  ?pokemon a ex:Pokemon ;
           ns1:hasType ?type .
}
```



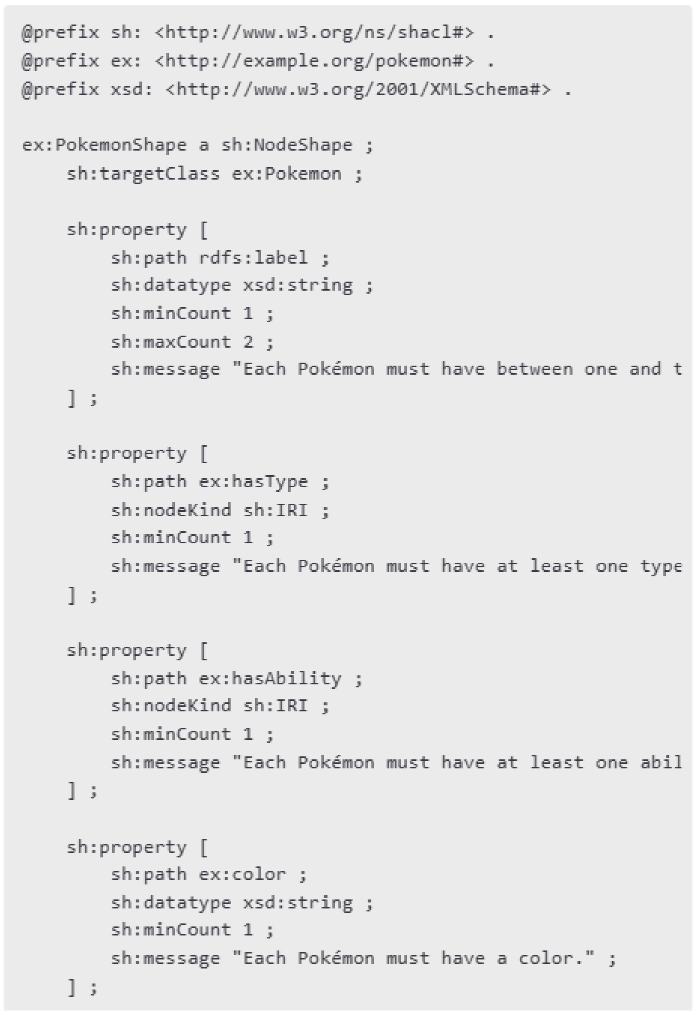
```
SELECT ?episode ?pokemonDebut
WHERE {
  ?episode a ep:Episode ;
           ep:hasPokemonDebut ?pokemonDebut .
}
```

5. SHACL for Validation of RDF Data

To ensure data quality and consistency, **SHACL** (Shapes Constraint Language) was employed to define and validate constraints for RDF data. The use of SHACL guarantees that the Knowledge Graph adheres to the expected structure and semantics, highlighting issues early in the pipeline.

Example SHACL Shapes

1. **Validation for Pokémon Entities:**
   - Ensures that each Pokémon has a valid label, type, ability, egg group, color, height, weight, and national index.
   - Example:

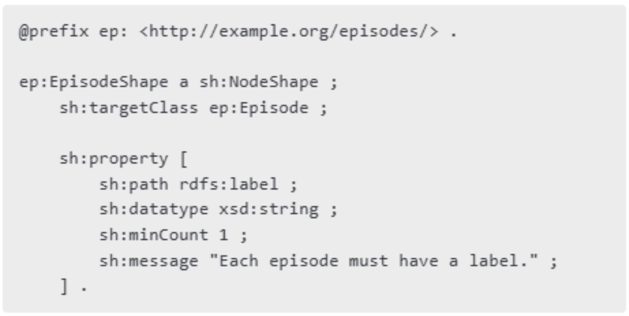


```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://example.org/pokemon#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:PokemonShape a sh:NodeShape ;
    sh:targetClass ex:Pokemon ;

    sh:property [
        sh:path rdfs:label ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 2 ;
        sh:message "Each Pokémon must have between one and t
    ] ;

    sh:property [
        sh:path ex:hasType ;
        sh:nodeKind sh:IRI ;
        sh:minCount 1 ;
        sh:message "Each Pokémon must have at least one type
    ] ;

    sh:property [
        sh:path ex:hasAbility ;
        sh:nodeKind sh:IRI ;
        sh:minCount 1 ;
        sh:message "Each Pokémon must have at least one abil
    ] ;

    sh:property [
        sh:path ex:color ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:message "Each Pokémon must have a color." ;
    ] ;
```

   -

Validation for Episode Entities:

- Ensures that each episode has a label and associated properties like animation team or debut Pokémon.
- Example:



```
@prefix ep: <http://example.org/episodes/> .

ep:EpisodeShape a sh:NodeShape ;
    sh:targetClass ep:Episode ;

    sh:property [
        sh:path rdfs:label ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:message "Each episode must have a label." ;
    ] .
```

By employing SHACL, the project ensures that all RDF data adheres to the desired structure, preventing errors and inconsistencies in the Knowledge Graph.

6. Challenges and Solutions

Data Challenges

Handling Diverse Data Formats:

- **Challenge:** Data extracted from infoboxes varied in structure.
- **Solution:** Implemented custom parsing scripts with error handling to normalize data.

Managing Multilingual Content:

- **Challenge:** Entities like Pokémon and moves had names in multiple languages.
- **Solution:** Used RDF language tags to store labels in various languages for proper localization.

Complex Relationship Mapping:

- **Challenge:** Representing relationships such as evolutions and episode debuts.
- **Solution:** Designed specific predicates to model complex relations like `ep:hasPokemonDebut`.

Technical Challenges

Query Optimization:

- **Challenge:** SPARQL queries were initially slow due to dataset complexity.
- **Solution:** Optimized queries by indexing frequently accessed properties and implementing caching mechanisms.

Data Integration:

- **Challenge:** Combining data from multiple sources (APIs and scraped content).
- **Solution:** Created a standardized pipeline for transforming and merging data into RDF triples.

7. Results

- A functional Knowledge Graph encompassing Pokémon entities, their attributes, and relationships.
- SPARQL endpoint enabling semantic exploration of the data.
- Ontology provides a reusable framework for similar projects.

8. Future Work

- Extend the ontology to include additional entities such as trainers and locations.
- Integrate data from external sources like DBpedia or Wikidata for enriched context.
- Develop a user-friendly frontend to visualize query results interactively.

9. Conclusion

This project demonstrates the potential of Semantic Web technologies in organizing and exploring structured data. By combining RDF, SPARQL, and ontology design, we created a Knowledge Graph that offers insights into the Pokémon universe while serving as a foundation for further exploration and application.

10. References

- RDFLib Documentation
- Apache Jena Fuseki Documentation
- Bulbapedia API
- Semantic Web Best Practices