Technical Report: Pokémon Knowledge Graph

Semantic Web Application Development Project

Aymen BEN ABDALLAH - Saad KEMLANE

Date: January 24, 2025

Executive Summary

This technical report documents the development and implementation of the Pokémon Knowledge Graph project, a semantic web application developed between December 2024 and January 2025. Our project leverages RDF and SPARQL technologies to create an interactive platform for exploring the Pokémon universe through semantic relationships.

Project Timeline

The development process was structured across three main sessions:

- Step1: Initial setup and core architecture
- Step2: Data modeling and RDF implementation
- Step3: Interface development and integration

Technical Architecture

Technology Stack

Our technology stack was carefully selected to create a robust and efficient web application while maintaining simplicity and reliability:

Backend Technologies:

- **Flask (Python 3.8+):** Chosen as our web framework for its lightweight nature and excellent integration with Python's ecosystem
- **RDFLib:** Enables efficient manipulation and processing of our semantic data
- **Apache Jena Fuseki:** Selected as our SPARQL server
- **SPARQLWrapper:** Facilitates communication with our SPARQL endpoint

Frontend Technologies:

- **HTML5:** Provides the structural foundation of our web pages
- **CSS3:** Handles styling and responsive design

- **JavaScript:** Implements interactive features and dynamic content updates

Database Technologies:

- **RDF/SPARQL:** Forms the backbone of our data structure
- **Turtle (.ttl):** Used for storing our RDF data in a human-readable format

Development Methodology

Phase 1: Data Domain Selection and Collection

We began by identifying six key categories that would form the core of our knowledge graph:

1. Pokémon: Core creature data including stats, types, and evolution chains
2. Abilities: Special powers and their effects on Pokémon
3. Items: Tools and objects used in the Pokémon world
4. Trainers: Notable characters and their relationships with Pokémon
5. Moves: Combat techniques and their characteristics
6. Locations: Important places in the Pokémon universe

For each category, we carefully extracted data from infoboxes, which provided structured information in a consistent format.

Phase 2: Data Processing and Transformation

Initial Data Format:

Our starting point was structured infobox data. Here's an example for Abomasnow:

=== BEGIN INFODex ===
—name=Abomasnow
—jname=
—jtranslit=Yukino'ō
—tmname=Yukinooh

—forme=2
—ndex=0460
—type1=Grass
—type2=Ice
—category=Frost Tree
—height-m=2.2
—weight-kg=135.5
—ability1=Snow Warning
—abilityd=Soundproof
=== END INFODex ===

Transformation Process:
We developed a systematic approach to transform this raw data into semantic RDF:

1. **Data Parsing and Categorization**
    - Basic Information (names, numbers)
    - Physical Characteristics (height, weight)
    - Game Mechanics (abilities, experience)
    - Type and Classification information
2. **RDF Schema Definition**
    - Created comprehensive class definitions
    - Established property relationships
    - Defined data types and constraints
3. **Data Enrichment**

Data Enrichment Our enrichment process significantly enhanced the initial RDF data by adding comprehensive information about visualization and presentation. A key aspect of this enrichment was the addition of images and templates. Here's how our final enriched RDF structure evolved:

```
ns1:Abomasnow a ns1:Pokémon ;
    rdfs:label "Rexblisar"@de,
        "Abomasnow"@en,
        "Blizzaroi"@fr,
        ""@ja ;
    ns1:baseExperience 173 ;
```

```
    ns1:hasAbility ns1:Snow_Warning,
        ns1:Soundproof ;
    ns1:hasType ns1:Grass,
        ns1:Ice ;
    ns1:usesTemplate ns1:Template_-,
        ns1:Template_2t,
        ns1:Template_A,
        ns1:Template_AP,
        ns1:Template_Adv ;
    ns2:image <https://bulbapedia.bulbagarden.net/wiki/File:0003MMS.png>,
        <https://bulbapedia.bulbagarden.net/wiki/File:Wulfric_Abomasnow.png>.
```

Phase 3: Web Application Implementation

Service Features

The service provides specialized methods for each data category:

Pokémon Data Retrieval

The service retrieves Pokémon data with a method using SPARQL queries:

```
def get_pokemon_list(self, page=1, per_page=24):
    offset = (page - 1) * per_page
    query = """
    PREFIX ns1: <http://example.org/ontology/pokemon#>
    SELECT ?pokemon ?name
        (GROUP_CONCAT(DISTINCT ?typeName; separator=", ") AS ?types)
        (SAMPLE(?image) AS ?selectedImage)
    WHERE {
        ?pokemon a ns1:Pokémon ;
                ns1:name ?name ;
                ns1:hasType ?type ;
                ns2:image ?image .
        BIND(REPLACE(STR(?type), "^.*#", "") AS ?typeName)
    }
    GROUP BY ?pokemon ?name
    LIMIT {per_page} OFFSET {offset}
    """
```

Ability Information

The service handles ability-related queries with pagination and detail retrieval:

```
def get_ability_list(self, page=1, per_page=24):
    query = f"""
    PREFIX ns1: <http://example.org/ontology/pokemon#>
    SELECT ?ability ?name ?japaneseName
    WHERE {{
        ?ability a ns1:Ability ;
            ns1:name ?name ;
            ns1:japaneseName ?japaneseName .
    }}
    ORDER BY ?name
    LIMIT {per_page} OFFSET {offset}
    """
```

Trainer Information

We implemented comprehensive trainer data retrieval:

```
def get_trainer_list(self, page=1, per_page=24):
    query = f"""
    SELECT ?trainer ?name ?gender ?japaneseName ?relatedTo
        (SAMPLE(?image) AS ?selectedImage)
    WHERE {{
        ?trainer a ns1:Trainer ;
                ns1:name ?name ;
                ns1:gender ?gender ;
                ns1:japaneseName ?japaneseName .
    }}
    """
```

Flask Application Structure

The Flask application handles routing and view rendering, organizing the application's endpoints in a logical manner.

Pokémon Routes

We implemented detailed Pokémon information handling:

```
@app.route('/pokemon/<name>')
def pokemon_detail(name):
    pokemon_details = sparql_service.get_pokemon_details(name)
    details = process_pokemon_details(pokemon_details)
    return render_template('pokemon/detail.html',
                           pokemon_details=details)
```

Key Features Implementation

Data Processing

We implemented detailed data processing for each entity type:

```
details = {
    "name": next((item['value']['value'] for item in pokemon_details
                  if item['property']['value'] == "ns1:name"), None),
    "types": [item['value']['value'].split('#')[-1]
              for item in pokemon_details
              if item['property']['value'] == "ns1:hasType"]
}
```

Image Handling

The application includes sophisticated image handling with fallbacks:

```
FILTER(STRENDS(STR(?image), CONCAT(?name, ".png")))
```

Technical Features

Data Structure and Organization

Our RDF schema was designed to capture relationships:

```
@prefix pokemon: <http://example.org/pokemon/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

pokemon:Pokemon a rdfs:Class .
pokemon:hasType a rdf:Property ;
    rdfs:domain pokemon:Pokemon ;
    rdfs:range pokemon:Type .
```

Search Implementation

The search functionality leverages SPARQL for efficient data retrieval:

```
PREFIX pokemon: <http://example.org/pokemon/>
SELECT ?pokemon ?name ?type
WHERE {
    ?pokemon a pokemon:Pokemon ;
             pokemon:name ?name ;
             pokemon:hasType ?type .
    FILTER(CONTAINS(LCASE(?name), LCASE(?searchTerm)))
}
```

Challenges and Solutions

Data Processing Challenges

Inconsistent Data Formats

- **Challenge**: Varying formats in source data
- **Solution**: Implemented robust parsing with error handling

Multilingual Content

- **Challenge**: Managing multiple language variants
- **Solution**: Utilized RDF language tags for proper localization

Relationship Mapping

- **Challenge**: Representing complex evolutions
- **Solution**: Developed custom predicates for evolutionary chains

Technical Implementation Challenges

Query Performance

- **Challenge**: Complex SPARQL queries were initially slow
- **Solution**: Implemented query optimization and caching

Data Integration

- **Challenge**: Combining data from multiple sources
- **Solution**: Created standardized transformation pipeline

Future Improvements

We identified several potential enhancements:

Technical Enhancements

- Optimize complex queries
- Add real-time updates

Feature Additions

- Add more interactive visualizations
- Implement user accounts

Conclusion

Our implementation successfully meets the project requirements while providing a solid foundation for future extensions. The combination of semantic web technologies with modern web development practices resulted in a robust and user-friendly application.