Título: **WS - Assignment 2**

Autores: **Carina Neves 90451, Eduardo Santos 93107, Pedro Bastos 93150**

Data: **31/05/2023**

Conteúdo

1. Introdução ao tema

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Web Semântica e visa o desenvolvimento de um sistema de informação baseado na Web.

Hoje em dia, os projetos Web lidam com grandes quantidades de dados de diversas fontes. Para efetivamente obter conhecimento sobre esses dados, a incorporação de mecanismos de ontologia e inferência traz várias vantagens. A interoperabilidade semântica garante comunicação e troca de dados de forma mais eficiente, permitindo a junção de informações de diversas fontes.

2. Ontologia

Foi criada uma ontologia que descreve o domínio de dados, representando o conhecimento com vocabulário preciso. Esta é orientada às entidades, ao invés de propriedades, onde foram definidas regras de inferência.

Esta ontologia foi escrita utilizando RDF, RDFS e OWL.

2.1. Classes & Subclasses

As classes são inferidas pelo GraphDB através das propriedades definidas para cada uma. Na tabela seguinte podemos verificar as classes e a maneira como foram inferidas:

Classe	Inferência
Driver	predicados *code, surname, forename, etc*
Team	predicados *name, nationality, signed, etc*
Race	predicados *round, season, name, etc*
Contract	predicados *driver, year, team, etc*
Standing	Todos os standings
Driver_standing	Subclass de Standing, resultados de corridas
Driver_final_standing	Subclass de Standing, resultados finais dos Drivers
Team_final_standing	Subclass de Standing, resultados finais das Equipas
Champion	Drivers com pelo menos 1 campeonato
Veteran	Drivers com pelo menos 4 anos de experiência
Best_teams	Equipas com pelo menos 1 campeonato

2.1.1. Inferência de classes manualmente

As classes *Champion*, *Veteran* e *Best_teams* precisaram de código SPIN extra, pois o Reasoner do GraphDB não era poderoso o suficiente.

Por exemplo, para inferir a classe ”Champion”, idealmente seria criada uma restrição de cardinalidade mínima sobre a propriedade *position* das instâncias da classe *Driver_final_standings*, ligadas ao piloto pela propriedade *finished_in*.

Como o GraphDB não consegue inferir tal regra de forma automática, foi necessário criar uma propriedade *champs*, associada ao piloto, que represente o número de campeonatos ganhos pelo mesmo. Essa propriedade foi definida através de SPIN, como podemos verificar na figura seguinte:

```
PREFIX driver: <http://f1/driver/pred/>
PREFIX driver_final_standings: <http://f1/driver_final_standing/pred/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX f1: <http://f1/>
PREFIX champion: <http://f1/champion/>

INSERT {
    GRAPH champion:graph_temp { ?driver driver:champs ?count }
}
WHERE {
    SELECT ?driver (COUNT(DISTINCT ?season) as ?count) WHERE
    {
        ?driver driver:finished_in ?dfs.
        ?dfs driver_final_standings:season ?season.
        ?dfs driver_final_standings:position ?position.

        FILTER (?position = '1')
    }
    GROUP BY ?driver
};

INSERT { ?driver driver:champs ?count }
WHERE {
    GRAPH champion:graph_temp { ?driver driver:champs ?count }.
};

DROP GRAPH champion:graph_temp;
```

Figura 1: código SPIN para inserir a propriedade *champs*

Assim, já é possível, através do GraphDB, definir a classe *Champion* com cardinalidade mínima no predicado *champs*, definindo Pilotos como *Champion* apenas quando têm pelo menos um campeonato:

```
f1:Champion a owl:Class ;
    rdfs:label "Champion" ;
    rdfs:subClassOf [ a owl:Restriction ;
    owl:intersectionOf (f1:Driver
            [ a owl:Restriction ;
            owl:minCardinality "1"^^<http://www.w3.org/2001/XMLSchema#int>;
            owl:onProperty driver_pred:champs ;
        ])
    ] .
```

Figura 2: definição da classe *Champion* na ontologia

2.2. Propriedades

2.2.1. Propriedades de objeto

De forma a haver relações entre as diferentes entidades, foram definidas propriedades de objeto em que se relacionam 2 entidades diferentes. Por exemplo, para relacionar a entidade *Driver* com a entidade *Contract*, foi definido a propriedade *signed_for*:

```
driver_pred:signed_for rdf:type rdf:Property ;
    rdfs:domain f1:Driver ;
    rdfs:range f1:Contract ;
    rdfs:label "Signed for" .
```

Figura 3: definição da propriedade *signed_for* na ontologia

Como podemos verificar, a propriedade é do domínio da classe dos pilotos, *Driver*, e tem um range da classe dos contratos, *Contract*.

2.2.2. Propriedades de tipo de dados

Estas são as propriedades mais simples da ontologia, onde apenas é definido que a mesma irá ter um valor específico. Por exemplo, para definir a propriedade de *code* na classe *Driver*, basta apenas indicar que será um *Literal*:

```
driver_pred:code rdf:type rdf:Property ;
    rdfs:domain f1:Driver ;
    rdfs:range rdfs:Literal ;
    rdfs:comment "Driver personal code" .
```

Figura 4: definição da propriedade *code* na ontologia

2.2.3. Propriedades simétricas

Este tipo de propriedade é usado em situações mais específicas, onde sabendo que (A) - property - (B), infere-se que (B) - property - (A). No nosso caso, é útil para definir a propriedade *teammate*, na classe *Driver*. Isto porque caso o piloto x seja colega de equipa do piloto y, o simétrico também se aplica, piloto y é colega de equipa do piloto x:

```
driver_pred:teammate rdf:type owl:SymmetricProperty ;
    rdfs:domain f1:Driver ;
    rdfs:range f1:Driver ;
    rdfs:comment "Symetric teammate property" .
```

Figura 5: definição da propriedade simétrica *teammate* na ontologia

2.2.4. Propriedades Inversas

Propriedades inversas também são bastante úteis para inferir uma relação inversa entre entidades diferentes. No nosso caso, o predicado *signed_for* que liga *Driver* com *Contract* já existia. Contudo, se um piloto assina um contrato, o contrato é assinado pelo piloto. Assim, definimos o predicado *is_signed_by* como o inverso do existente:

```
contract_pred:is_signed_by rdf:type owl:ObjectProperty ;
    owl:inverseOf driver_pred:signed_for ;
    rdfs:range f1:Driver ;
    rdf:comment "Inverse of Driver's signed_for" .
```

Figura 6: definição da propriedade inversa *is_signed_by* na ontologia

2.2.5. Propriedades Inferidas com SPIN

Além da necessidade de inferir manualmente algumas das classes anteriormente referidas, foi necessário fazer o mesmo para algumas propriedades mais complexas. No nosso caso, a propriedade *teammate* da classe *Driver* foi inferida com código SPIN, pois o Reasoner do GraphDB não é suficientemente poderoso.

Esta propriedade é adicionada quando, dois pilotos assinam contrato, no mesmo ano, pela mesma equipa, como podemos ver no diagrama:

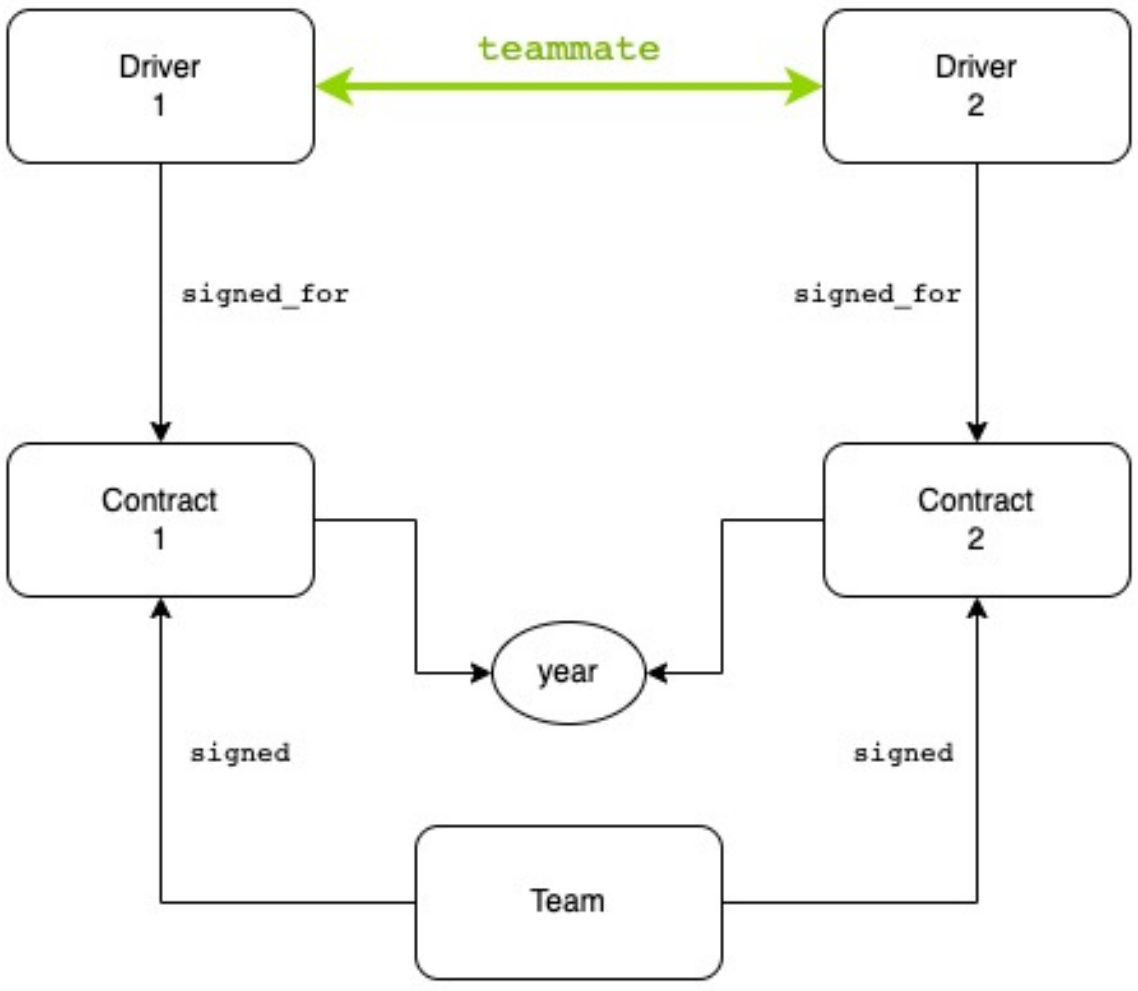

Figura 7: Diagrama de causa da propriedade *teammate*

Como foi dito, foi necessário inferir a propriedade através de código SPIN:

```
PREFIX driver: <http://f1/driver/pred/>
PREFIX contract: <http://f1/contract/pred/>
PREFIX team: <http://f1/team/pred/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX f1: <http://f1/>

INSERT {
    ?d1 driver:teammate ?d2 .
}
WHERE {
    ?d1 rdf:type f1:Driver .
    ?d2 rdf:type f1:Driver .

    ?d1 driver:code ?d1_code .
    ?d2 driver:code ?d2_code .

    ?d1 driver:signed_for ?contract1 .
    ?contract1 rdf:type f1:Contract .
    ?d2 driver:signed_for ?contract2 .
    ?contract2 rdf:type f1:Contract .

    ?contract1 contract:year ?year .
    ?contract2 contract:year ?year .

    ?team rdf:type f1:Team .
    ?team team:signed ?contract1 .
    ?team team:signed ?contract2 .

    FILTER(?d1_code != ?d2_code)
}
```

Figura 8: Código SPIN para inferência da propriedade *teammate*

2.2.6. Vantagens da ontologia e das inferências

Ontologias permitem a interoperabilidade semântica, facilitando a comunicação entre diferentes sistemas e aplicações. Ao fornecer uma compreensão compartilhada do domínio de conhecimento e dos seus relacionamentos, estabelecem um vocabulário comum e preciso.

Além disso, a eficiência das *queries* tende, de forma geral, a ser bastante maior, pois a sua complexidade é menor. Por exemplo, antes de definir a nossa ontologia, para saber se um certo *Driver* era campeão, seria necessário ver todos os predicados *finished_in*, que relacionam o piloto com a classe *Driver_final_standings*, e daí verificar o predicado *position* para perceber se alguma instância tinha o valor igual a "1". Com a ontologia, o conhecimento aumentou, e basta ver se o piloto em questão é da classe *Champion* e, se for, verificar o predicado *champs* para saber quantos campeonatos tem.

3. Queries DBpedia

Outro requisito do projeto era a complementação do conjunto de dados do SI com bases de conhecimento externas. Para isso decidimos usar a *DBpedia*. A DBpedia é um projeto cujo objetivo é extrair conteúdo estruturado das informações da Wikipédia, permitindo-nos obter informações sobre os pilotos, as suas

equipas e as corridas de cada temporada.

3.1. Dados Extraídos

Os dados extraídos da DBpedia foram os seguintes:

- **Piloto**:
  - Idade;
  - Biografia.
- **Equipa**:
  - Informação sobre a equipa.
- **Corrida**:
  - Distância total da corrida;
  - Número de voltas;
  - Informação sobre a corrida.

3.2. Obtenção dos Dados

Para obtermos os dados da wikidata recorremos à biblioteca python *SPARQLWrapper2*. Esta biblioteca permitiu-nos abstrair o estabelecimento de conexão com a *DBpedia*, bastando passar o url para onde tinha de ser feita a query (*https://dbpedia.org/sparql*) e a query em si.

```python
def get_race_info(race, season):
    query = """
    PREFIX db: <http://dbpedia.org/resource/>
        PREFIX dbo: <http://dbpedia.org/ontology/>

        SELECT ?abstract, ?distance, ?distanceLaps
        WHERE {
        db:SEASON_RACE dbo:abstract ?abstract .
        db:SEASON_RACE dbo:distance ?distance .
        db:SEASON_RACE dbo:distanceLaps ?distanceLaps .
        FILTER (LANG(?abstract) = 'en')
        }
        LIMIT 1
    """
    race = race.replace(" ", "_")
    query = query.replace("SEASON_RACE", f"{season}_{race}")

    sparql.setQuery(query)
    results = sparql.query()

    data = results.bindings

    if data:
        if data[0]["abstract"] != []:
            abstract = str(data[0]["abstract"])[15:-2]
        else:
            abstract = "Unknown"

        if data[0]["distance"] != []:
            distance = str(data[0]["distance"])[21:-2]
        else:
            distance = "Unknown"
      You, 16 hours ago • Races' info
        if data[0]["distanceLaps"] != []:
            distanceLaps = str(data[0]["distanceLaps"])[21:-2]
        else:
            distanceLaps = "Unknown"
    else:
            abstract = "Unknown"
            distance = "Unknown"
            distanceLaps = "Unknown"

    return abstract, distance, distanceLaps
```

Figura 9: Query para obter informação sobre uma corrida.

Às vezes, a *DBpedia* não contém a informação que estamos a procurar. Nesses casos, definimos o valor da variável em questão para ”Unknown”, como podemos ver na imagem acima.

Os dados são integrados juntamente com as nossos de forma transparente para o utilizador.

4. Publicação Semântica

A publicação da semântica dos dados era o último requisito deste trabalho. Depois de analisadas as diferentes opções para completar este requisito decidimos implementar o mesmo através de anotações *RDFa*.

Estas anotações foram adicionadas aos templates *HTML*, sendo os ficheiros alterados:

- curiosities.html
- drivers.html
- race-modal.html
- results.html
- teams.html

Recorremos a anotações do tipo *typeof* para identificar o tipo do elemento e o *property* de forma a especificar a propriedade associada ao mesmo. Para validar as mesmas e perceber se os esquemas estavam a ficar como pretendido, utilizámos o *parser online RDFa Play* sugerido pelo docente nos *slides* da Unidade Curricular.

Abaixo, estão dois exemplos do codigo implementado nas classes de drivers e teams, seguido do resultado optido no parser dos mesmos.

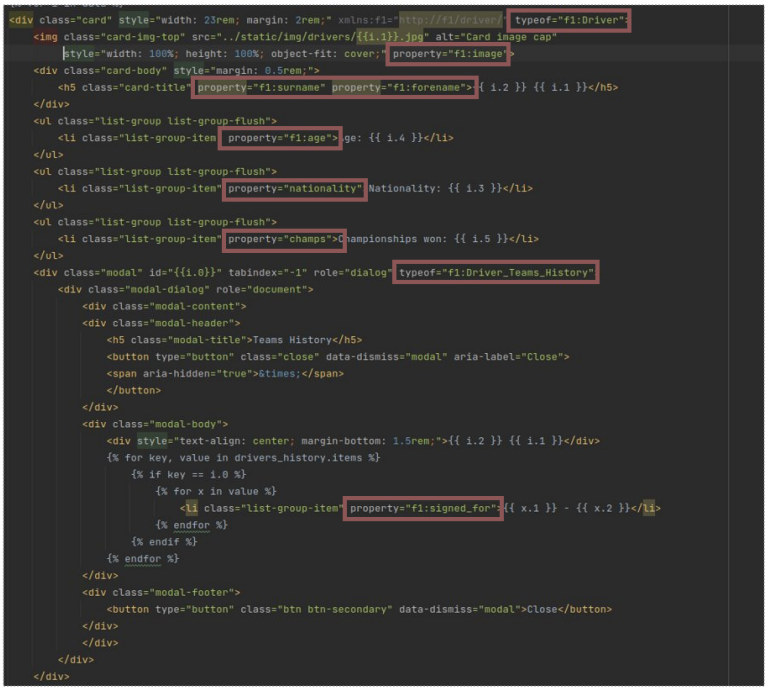

Figura 10: Anotações RDFa em drivers.html

```
<div class="card" style="width: 23rem; margin: 2rem;" xmlns:f1="http://f1/driver/" typeof="f1:Driver">
    <img class="card-img-top" src="../static/img/drivers/{{i.1}}.jpg" alt="Card image cap"
        style="width: 100%; height: 100%; object-fit: cover;" property="f1:image">
    <div class="card-body" style="margin: 0.5rem;">
        <h5 class="card-title" property="f1:surname" property="f1:forename">{{ i.2 }} {{ i.1 }}</h5>
    </div>
    <ul class="list-group list-group-flush">
        <li class="list-group-item" property="f1:age">Age: {{ i.4 }}</li>
    </ul>
    <ul class="list-group list-group-flush">
        <li class="list-group-item" property="nationality">Nationality: {{ i.3 }}</li>
    </ul>
    <ul class="list-group list-group-flush">
        <li class="list-group-item" property="champs">Championships won: {{ i.5 }}</li>
    </ul>
    <div class="modal" id="{{i.0}}" tabindex="-1" role="dialog" typeof="f1:Driver_Teams_History">
        <div class="modal-dialog" role="document">
            <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Teams History</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <div style="text-align: center; margin-bottom: 1.5rem;">{{ i.2 }} {{ i.1 }}</div>
                {% for key, value in drivers_history.items %}
                    {% if key == i.0 %}
                        {% for x in value %}
                            <li class="list-group-item" property="f1:signed_for">{{ x.1 }} - {{ x.2 }}</li>
                        {% endfor %}
                    {% endif %}
                {% endfor %}
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
            </div>
            </div>
        </div>
    </div>
```

Figura 11: Anotações RDFa em teams.html

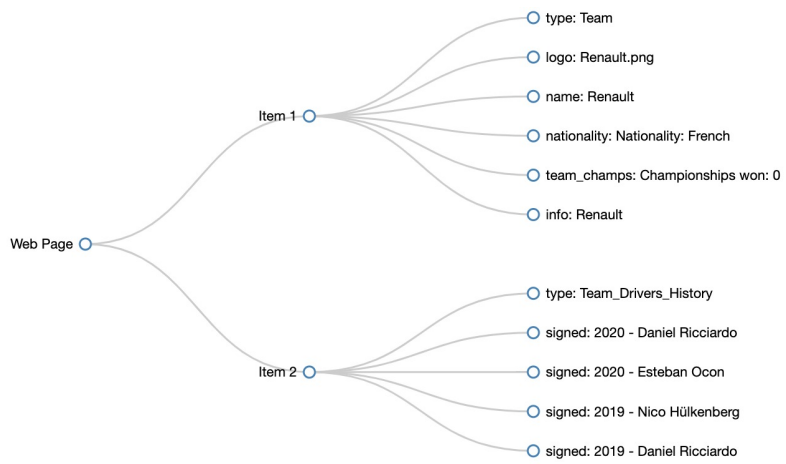

Figura 12: Resultado do parser dos drivers

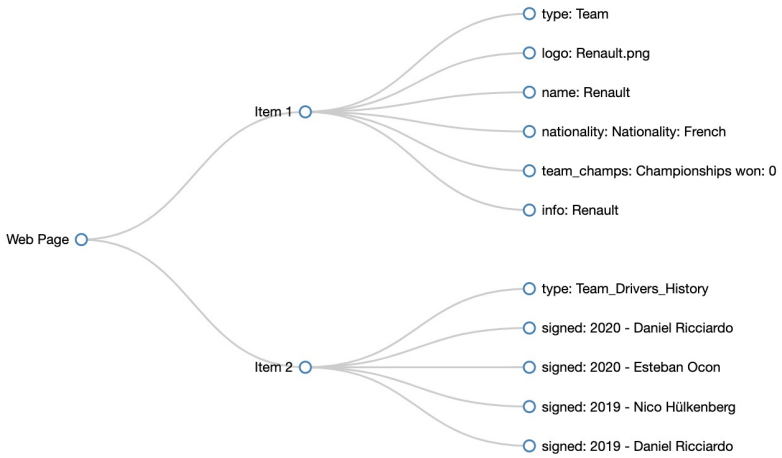

Figura 13: Resultado do parser das teams

Desta forma os motores de busca e outras ferramentas podem compreender facilmente o significado dos dados presentes na página, promovendo a interoperabilidade semântica.

5. Como executar a aplicação

A aplicação pode ser executada seguindo os seguintes passos:

1. Instalar os requirements presentes no ficheiro *requirements.txt*, dentro do diretório */f1_app*.
2. Inicializar o GraphDB, criar um novo repositório *"db"* e importar os ficheiros *f1.nt* e *ontology.n3*. presentes na pasta */datasets*.
3. No diretório */f1_app*, executar o seguinte comando:

```
python manage.py migrate
```

4. Por fim, para iniciar a app, executar o seguinte comando:

```
python3 manage.py runserver
```

5.1. Inferências

Para ter o sistema totalmente funcional, é necessário realizar as inferências.

Para tal, é necessário criar um *user admin*, executando o seguinte comando:

```
python manage.py createsuperuser
```

Depois de criar o admin, basta fazermos login como admin, ir à pagina do admin e clicar no botão das inferências, para que as mesmas sejam geradas.

6. Ideias de trabalho futuro

Como mencionado no inicio deste relatório, o tema escolhido é bastante abrangente e o *dataset* possuía uma vasta quantidade de dados. Como trabalho futuro, os seguintes pontos foram destacados:

- Incluir mais anos nas opções já implementadas;
- Tratar e permitir a visualização de informação referente às corridas de Qualificação e Sprint;
- Inferir as melhores pistas de um dado piloto;
- Inferir os pilotos com mais trocas de equipa, e quais os pares de pilotos mais duradouros;
- Permitir a comparação direta de pilotos e equipas;

7. Conclusão

Concluindo, este trabalho mostrou-se bastante proveitoso para consolidar os conhecimentos adquiros na unidade curricular, relativos à utilização de *Ontologias*, *Resource Description Framework Schema* (RDFS), *Web Ontology Language* (OWL), *GraphDB* e *SPARQL Inferencing Notation* (SPIN).

Permitiu-nos uma perspetiva diferente das Aplicações Web e conhecimento do domínio de dados.

Os principais desafios foram o desenvolvimento da ontologia e a inferência manual de certas relações mais complexas.

Referências

[1] VOPANI. *Formula 1 World Championship (1950 - 2023)*. Dataset Utilizado. URL: https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020?resource=download.