

Segundo Trabalho Prático

Formula 1 Pitstop

João Andrade - 107969

José Gameiro - 108840

Tomás Victal - 109018

Professor Hélder Zagalo

Web Semântica

DETI

Universidade de Aveiro

Junho 2025

Conteúdo

1	Introdução	2
2	Definição da Ontologia	2
2.1	Entidades e Relações	2
2.1.1	Circuit	2
2.1.2	Constructor	3
2.1.3	Driver	3
2.1.4	Race	4
2.1.5	Season	4
2.1.6	Result	4
2.1.7	Status	5
2.1.8	Qualifying	5
2.1.9	Standing	6
2.1.10	ConstructorResult	7
2.2	Propriedades	7
2.2.1	Datatype Properties	7
2.2.2	ObjectProperties	8
2.2.3	Outras Propriedades	9
2.3	Restrições	12
3	Conjunto de Inferências (SPIN)	16
3.1	Regras de Apoio à Inferência	16
3.2	Regras Adicionais	17
4	Alterações nas Operações sobre os dados	19
5	Wikidata e DBpedia	20
6	Micro-formatos e RDFa	21
6.1	Micro-formatos	22
6.2	RDFa	24
7	Novas Funcionalidades da Aplicação	26
7.1	Detalhes para circuitos	26
7.2	Novos detalhes para corrida	26
7.3	Descrição para Circuito	27
7.4	Imagens	27
7.5	Construtores de um piloto	28
8	Conclusão	28
9	Organização do Código	29
10	Instruções de execução	29

1 Introdução

Este trabalho tem como objetivo dar continuidade ao desenvolvimento do sistema de informação semântico iniciado no TP1, aprofundando a sua complexidade e funcionalidades. Para isso, foi criada uma ontologia detalhada que descreve o domínio dos dados, permitindo classificações automáticas e a inferência de novas relações entre entidades. O sistema foi enriquecido com dados externos provenientes da DBpedia e da Wikidata, bem como pela publicação dos dados nas páginas web do próprio sistema. A implementação recorre a tecnologias como **Python/Django**, **GraphDB**, **SPARQL**, **RDF**, **OWL**, **Protégé**, entre outras, promovendo uma forte integração entre os diferentes componentes e uma interface mais funcional para o utilizador.

O nosso grupo manteve o mesmo dataset, que se encontra disponível através deste link, que apresenta dados relacionados com campeonatos de **Formula 1** desde 1950 até 2024.

2 Definição da Ontologia

Para a definição da Ontologia, foi necessário, primeiro identificar quais eram as principais entidades que iriam estar presente na nossa ontologia, quais eram as propriedades de cada uma e como é que elas se relacionam entre si.

2.1 Entidades e Relações

2.1.1 Circuit

Representa os circuitos onde as corridas de Formula 1 se realizaram. Apresenta as seguintes propriedades:

- **circuitId**, um URI único que identifica um determinado circuito
- **circuitRef**, uma string para melhor identificar o circuito;
- **name**, o nome do circuito;
- **location**, a cidade onde o circuito se encontra;
- **country**, o país onde o circuito se encontra localizado;
- **lat**, a latitude da localização do circuito;
- **lng**, a longitude da localização do circuito;
- **url**, um URL para um site exterior com mais informações sobre o circuito.

Este não apresenta nenhuma propriedade que o relacione com outras entidades, no entanto a entidade **Race** apresenta uma propriedade **hasCircuit**, que relaciona uma **Race** com um **Circuit**.

2.1.2 Constructor

Representa todos os construtores existentes de Formula 1. Apresenta as seguintes propriedades:

- **constructorId**, um URI que identifica um determinado construtor;
- **constructorRef**, uma string única para melhor identificar um construtor;
- **name**, o nome do construtor;
- **nationality**, a nacionalidade do construtor;
- **url**, um URL para um site exterior com informações mais detalhadas sobre um construtor.

Tal como a entidade **Circuit**, esta entidade não apresenta nenhuma propriedade que a relacione com outras entidades, mas outras entidades relacionam-se com esta, como **ConstructorResult**, **ConstructorStanding**, **Qualifying** e **Result**, através da propriedade **hasConstructor** que contém um URI de um determinado construtor.

2.1.3 Driver

Representa todos os condutores de Formula 1 existentes no dataset. Apresenta como propriedades:

- **driverId**: um URI que identifica um determinado piloto;
- **driverRef**: uma string única que identifica um piloto;
- **number**: um número único que está associado a um piloto (alguns pilotos podem não conter esta propriedade por causa do dataset escolhido);
- **code**: um código único constituído por caracteres, que está associado a um piloto (alguns pilotos podem não conter esta propriedade também, por causa do dataset escolhido);
- **forename**: o primeiro nome de um piloto;
- **surname**: o sobre nome de um piloto;
- **dob**: data de nascimento de um piloto;
- **nationality**: a nacionalidade de um piloto;
- **url**, um URL para um site exterior com informações mais detalhadas sobre um piloto;

Tal como as duas outras entidades explicadas anteriormente, esta não apresenta nenhuma propriedade que se relacione com outra entidade, mas outras entidades estão relacionadas com esta, como **DriverStanding**, **Qualifying** e **Result** através da propriedade **hasDriver**.

2.1.4 Race

Representa todas as corridas de Formula 1 realizadas. Apresenta as seguintes propriedades:

- **raceId**, um URI único que identifica uma determinada corrida;
- **year**, o ano em que a corrida se realizou;
- **round**, número da ronda da corrida;
- **name**, nome da corrida;
- **date**, data em que a corrida se realizou;
- **totalDuration**, duração total da corrida;
- **url**, URL com informações mais detalhadas sobre uma corrida;
- **hasCircuit**, um identificador único para o Circuito de uma corrida em específico.
- Outras propriedades relacionadas com datas e horários das sessões, que não foram incluídas na ontologia visto que muitos destes campos não apresentam dados relevantes (como `fp1_date`, `quali_time`, entre outros).

2.1.5 Season

Representa todas as épocas de Formula 1. Contém as seguintes propriedades:

- **seasonId**, um URI único que identifica uma determinada época;
- **url**, um URL para uma fonte de dados exterior com informações mais detalhadas sobre uma época;

2.1.6 Result

Representa os resultados que se encontram associados a uma corrida. Apresenta as seguintes propriedades:

- **resultId**, um URI único que identifica um determinado resultado;
- **participatedIn**, um URI único de uma corrida, esta propriedade relaciona um resultado com uma determinada corrida;
- **hasDriver**, um URI único de um piloto, que relaciona os resultados de uma determinada corrida com um piloto;
- **hasConstructor**, um URI único de um construtor, que relaciona os resultados de uma determinada corrida com um construtor;
- **number**, número de um carro de um determinado piloto;
- **grid**, posição de um determinado piloto na grelha de partida;
- **position**, posição final de um piloto na corrida;

- **positionText**, posição final de um piloto na corrida no formato de string;
- **positionOrder**, rank final de um piloto na corrida;
- **points**, número de pontos obtidos por um determinado piloto numa corrida;
- **laps**, número total de voltas de uma corrida;
- **time**, tempo total de uma corrida;
- **milliseconds**, tempo total da corrida em milissegundos;
- **fastestLap**, número da volta em que um piloto foi o mais rápido;
- **rank**, rank em que um piloto se encontrava na sua volta mais rápida;
- **fastestLapTime**, tempo da volta em que um piloto foi mais rápido;
- **fastestLapSpeed**, velocidade alcançada por um piloto na sua volta mais rápida;
- **hasStatus**, URI único que identifica um determinado estado em que um piloto se encontrava na corrida. Esta propriedade relaciona esta entidade com uma outra do tipo **Status**.

2.1.7 Status

Representa diversos estados da Formula 1. Contém as seguintes propriedades:

- **statusId**, URI único que identifica um determinado estado;
- **status**, descrição de um determinado estado.

Esta entidade não apresenta nenhuma propriedade que a relacione com outras, no entanto, a entidade **Result**, encontra-se relacionada com esta através da propriedade **hasStatus**.

2.1.8 Qualifying

Representa as qualificações realizadas antes das corridas de Formula 1. Apresenta as seguintes propriedades:

- **qualifyId**, URI único que identifica uma determinada qualificação;
- **participatedIn**, URI único de uma corrida, que relaciona uma qualificação com uma corrida ;
- **hasDriver**, URI único de um piloto, que relaciona uma qualificação a um piloto;
- **hasConstructor**, URI único de um construtor, que relaciona uma qualificação com um construtor;
- **number**, número do carro de um determinado piloto numa qualificação;
- **position**, posição em que um piloto terminou numa qualificação;

- q1, tempo que um piloto obteve numa determinada qualificação 1;
- q2, tempo que um piloto obteve numa determinada qualificação 2;
- q3, tempo que um piloto obteve numa determinada qualificação 3;

2.1.9 Standing

Representa as qualificações de uma determinada entidade. Esta entidade foi criada com o objetivo de podermos depois adicionar uma regra à nossa ontologia em que disséssemos que as entidades **DriverStanding** e **ConstructorStanding** são sub-classes desta entidade Standing, visto que estas duas entidades apresentam dados semelhantes mas para entidades diferentes (para pilotos e para construtores, respetivamente). Isto foi feito através das seguintes definições:

```
ps:Standing rdf:type owl:Class .
ps:DriverStanding rdf:type owl:Class ;
    rdfs:subClassOf ps:Standing .

ps:ConstructorStanding rdf:type owl:Class ;
    rdfs:subClassOf ps:Standing .

ps:DriverStanding owl:disjointWith ps:ConstructorStanding .
```

Apresenta as seguintes propriedades:

- **standingId**, URI único que identifica uma determinada standing;
- **participatedIn**, URI único de uma corrida que relaciona um standing com uma corrida;
- **obtainedPoints**, pontos obtidos num standing por uma entidade;
- **position**, posição final de uma das duas entidades num standing;
- **positionText**, posição final de uma das duas entidades num formato de uma string;
- **numberOfWins**, número de vitórias de uma das duas entidades.

As propriedades que diferenciam as entidades **DriverStanding** e **ConstructorStanding** é:

- **hasDriver**, para **DriverStanding** que representa um URI único de um piloto e faz com que se crie uma relação entre um **DriverStanding** e um **Driver**;
- **hasConstructor**, para **ConstructorStanding** que representa um URI único de um construtor e faz com que se crie uma relação entre um **ConstructorStanding** e um **Constructor**.

2.1.10 ConstructorResult

Representa todos os resultados das corridas nos campeonatos dos construtores. Contém as seguintes propriedades:

- **constructorResultsId**, URI único que identifica um determinado resultado de um construtor;
- **participatedIn**, URI único de uma corrida, cria uma relação entre um **ConstructorResult** e uma **Race**;
- **hasConstructor**, URI único de um construtor, cria uma relação entre um **ConstructorResult** e um **Constructor**;
- **obtainedPoints**, pontos obtidos numa corrida nos campeonatos dos construtores;
- **hasStatus**, estado de um construtor numa corrida dos campeonatos dos construtores. Embora esta propriedade esteja definida na ontologia, o dataset não apresenta dados para esta propriedade nesta entidade.

2.2 Propriedades

2.2.1 Datatype Properties

As **Datatype Properties** são utilizadas em OWL e RDFS para associar **indivíduos ou instâncias de classes** a valores **literais**, como números, textos ou datas. Nas definições destas propriedades, o nosso grupo incluí sempre um pequeno **comentário**, a explicar melhor em que consiste uma propriedade e um **label** para ser melhor identificável. Na nossa ontologia, existem diversas propriedades que são consideradas **DatatypeProperties**, por isso iremos demonstrar apenas algumas delas no relatório, sendo que depois muitas outras são iguais só que com comentários, labels, domínios e intervalos diferentes.

Exemplo da definição de uma **DatatypeProperty** para a propriedade **location**, pertencente à entidade **Circuit**:

```
pred:location rdf:type owl:DatatypeProperty ;  
  rdfs:domain ps:Circuit ;  
  rdfs:range xsd:string ;  
  rdfs:label "Circuit location" ;  
  rdfs:comment "Place where the circuit is present" .
```

Neste caso, a propriedade **pred:location** é declarada como sendo um valor literal, neste caso **xsd:string**. Esta propriedade tem como domínio a classe **ps:Circuit**, indicando que apenas instâncias da classe **Circuit** podem ter esta propriedade, e como intervalo ou **range** o tipo de dados **string**, ou seja, a localização é expressa em texto. Adicionalmente, a propriedade inclui uma etiqueta (**rdfs:label**) com o nome **Circuit Location** e um comentário explicativo **rdfs:comment** que descreve brevemente a função da propriedade.

Outro exmplo da definição de uma propriedade de dados **year**, pertencente à entidade **Race**:


```

pred:year rdf:type owl:DatatypeProperty ;
  rdfs:domain ps:Race ;
  rdfs:range xsd:integer ;
  rdfs:label "Race Year" ;
  rdfs:comment "Year when the race occurred" .

```

Muito semelhante à definição explicada anteriormente, só que desta vez a propriedade é `pred:year` e o domínio desta é definido como sendo `ps:Race`, ou seja apenas, instâncias da classe **Race** é que podem conter esta propriedade. O **range** também é diferente, sendo que para este caso é o tipo de dados `integer`.

2.2.2 ObjectProperties

As **Object Properties** ou **propriedades de objeto** em OWL e RDFS são utilizadas para representar **relações entre duas instâncias (ou indivíduos)** de classes. Ao contrário das *datatype properties*, que ligam uma instância a um valor literal, as *object properties* ligam uma instância a outra dentro da ontologia.

Na nossa ontologia existe um total de cinco propriedades de objeto:

- **hasDriver**: relaciona uma entidade com um **Driver**, logo o valor desta propriedade tem de ser um URI para um **Driver**;
- **participatedIn**: relaciona uma entidade com uma **Race**, logo o valor desta propriedade tem de ser um URI de uma **Race**;
- **hasConstructor**: relaciona uma entidade com um **Constructor**, logo o valor desta propriedade tem de ser um URI de um **Constructor**;
- **hasStatus**: relaciona uma entidade com um **Status**, logo o valor desta propriedade tem de ser um URI de um **Status**;
- **hasCircuit**: relaciona uma entidade com um **Circuit**, logo o valor desta propriedade tem de ser um URI de um **Circuit**.

O bloco de código seguinte mostra a definição destas cinco propriedades:

```

pred:participatedIn rdf:type owl:ObjectProperty ;
  rdfs:range ps:Race ;
  rdfs:label "Race Identifier" ;
  rdfs:comment "Identifier for a Race that an entity participated
    ↪ in" .

pred:hasConstructor rdf:type owl:ObjectProperty ;
  rdfs:range ps:Constructor ;
  rdfs:label "Associated Constructor" ;
  rdfs:comment "Constructor associated to an entity" .

pred:hasStatus rdf:type owl:ObjectProperty ;
  rdfs:range ps:Status ;
  rdfs:label "Status" ;

```

```

    rdfs:comment "Status of the entity" .

    pred:hasDriver rdf:type owl:ObjectProperty ;
    rdfs:range ps:Driver ;
    rdfs:label "Driver reference" ;
    rdfs:comment "Reference to a driver that participated in an
    ↪ event" .

    pred:hasCircuit rdf:type owl:ObjectProperty ;
    rdfs:domain ps:Race ;
    rdfs:range ps:Circuit ;
    rdfs:label "Race Circuit" ;
    rdfs:comment "Circuit of a specific race" .

```

Podemos observar nas propriedades `participatedIn`, `hasConstructor`, `hasStatus` e `hasDriver` apresentam a definição de qual é o seu **range**, no entanto não existe a definição do domínio visto que estas propriedades são usadas em mais do que uma classe. Por exemplo a propriedade `hasDriver` é usada nas classes **DriverStanding**, **Qualifying** e **Result**, isto leva a um problema que é os **Reasoner**, quer do **GraphDB** quer do **Protégé** não irão conseguir inferir qual é o domínio destas propriedades.

O nosso grupo pensou em definir vários domínios para estas propriedades, onde cada propriedade tinha os domínios necessários, ou seja, definia-se as classes em que esta propriedade estava presente, no entanto esta solução não funcionou como esperávamos pois os Reasoners não conseguiam inferir a que classe é que deviam de associar estas propriedades. Outra solução que pensámos para resolver este problema foi a criação de **Restrições** (que irão ser explicadas mais à frente), contudo mesmo com estas restrições a serem aplicadas e a definição de múltiplos domínios não funcionou do ponto das inferências. Por fim decidimos então tirar esta parte da definição de muitos domínios, visto que nos afetava bastante na parte das inferências, mas conseguimos encontrar uma solução para que os **Reasoners** consigam inferir que uma instância que contenham estas propriedades pertencem a uma determinada classe, que envolvem restrições e regras de SPIN (irão ser descritas mais à frente).

A única propriedade de objeto que contém o domínio definido é `hasCircuit`, pois esta propriedade é apenas utilizada por uma classe **Race**, logo a especificação do domínio foi algo que ajudou os reasoners na inferência.

2.2.3 Outras Propriedades

Como referido na secção anterior existem propriedades que estão presentes em diversas entidades. Na secção anterior referiu-se as propriedades de objetos, no entanto existem também propriedades de dados (**DatatypeProperties**) que estão presentes em diversas entidades, como por exemplo `name`, `url`, `position`, entre outras. Ao todo nove propriedades são usadas em diversas classes. Para resolver este problema tivemos uma abordagem igual à que se explicou na secção das propriedades de objeto, onde primeiro tentámos definir vários domínios e restrições mas não funcionou e por isso decidimos retirar os domínios partilhados.

```

pred:name rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:string ;
  rdfs:label "Entity Name" ;
  rdfs:comment "Full Name of a certain entity" .

pred:url rdf:type owl:DatatypeProperty ;
  rdfs:range ps:anyURI ;
  rdfs:label "Url of a Circuit" ;
  rdfs:comment "URL with more information about a certain Entity"
  ↪ .

pred:position rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:integer ;
  rdfs:label "Position" ;
  rdfs:comment "Position where a certain entity finished in an
  ↪ event" .

```

Foram definidas ainda algumas propriedades extra para conseguirmos criar relações interessantes, como por exemplo a propriedade `time`:

```

pred:time rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:string ;
  rdfs:label "Race Duration" ;
  rdfs:comment "Time it took to complete an event" .

pred:qt rdf:type owl:DatatypeProperty ;
  rdfs:subPropertyOf pred:time ;
  rdfs:domain ps:Qualifying ;
  rdfs:range xsd:string ;
  rdfs:label "Qualifying Time" ;
  rdfs:comment "Qualifying time for a phase" .

pred:totalDuration rdf:type owl:DatatypeProperty ;
  rdfs:subPropertyOf pred:time ;
  rdfs:domain ps:Race ;
  rdfs:label "Total Race Duration" ;
  rdfs:comment "Total duration of a race" .

pred:duration rdf:type owl:DatatypeProperty ;
  rdfs:subPropertyOf pred:time ;
  rdfs:domain ps:Result ;
  rdfs:label "Driver's Race duration" ;
  rdfs:comment "The time a Driver took to finish a Race" .

pred:fastestLapTime rdf:type owl:DatatypeProperty ;
  rdfs:subPropertyOf pred:time ;
  rdfs:domain ps:Result ;
  rdfs:label "Fastest Lap Time" ;

```

```
rdfs:comment "Time of the fastest lap that a driver had" .
```

A criação da propriedade `time` teve como objetivo centralizar e generalizar o conceito de tempo associado a diferentes eventos no domínio das corridas de Formula 1. Ao definir `time` como uma propriedade genérica com `range xsd:string`, foi possível estabelecer uma estrutura comum que agrupa diversas medidas temporais relevantes no sistema. A partir desta, foram definidas várias **subpropriedades** mais específicas, como `qt`, `totalDuration`, `duration` e `fastestLapTime`, cada uma associada a diferentes entidades (**Qualifying**, **Race**, **Result**). Esta abordagem facilita a organização semântica dos dados, promove a reutilização de conceitos e permite que motores de inferência compreendam que todas estas sub-propriedades representam, de alguma forma, um tipo de duração ou tempo, possibilitando consultas e raciocínios mais genéricos sobre o domínio.

Definimos ainda outra propriedade `qt` que embora seja sub-propriedade de `time`, existem outras propriedades que sejam sub-propriedades desta, que são `q1`, `q2` e `q3`.

```
pred:qt rdf:type owl:DatatypeProperty ;
  rdfs:subPropertyOf pred:time ;
  rdfs:domain ps:Qualifying ;
  rdfs:range xsd:string ;
  rdfs:label "Qualifying Time" ;
  rdfs:comment "Qualifying time for a phase" .

pred:q1 rdf:type owl:DatatypeProperty ;
  rdfs:domain ps:Qualifying ;
  rdfs:range xsd:string ;
  rdfs:label "Qualifying time 1" ;
  rdfs:comment "Qualifying time for phase 1" ;
  rdfs:subPropertyOf pred:qt .

pred:q2 rdf:type owl:DatatypeProperty ;
  rdfs:domain ps:Qualifying ;
  rdfs:range xsd:string ;
  rdfs:label "Qualifying time 2" ;
  rdfs:comment "Qualifying time for phase 2" ;
  rdfs:subPropertyOf pred:qt .

pred:q3 rdf:type owl:DatatypeProperty ;
  rdfs:domain ps:Qualifying ;
  rdfs:range xsd:string ;
  rdfs:label "Qualifying time 3" ;
  rdfs:comment "Qualifying time for phase 3" ;
  rdfs:subPropertyOf pred:qt .
```

A propriedade `qt` foi definida como uma sub-propriedade de `time`, com o objetivo de representar genericamente os tempos de qualificação. Para permitir uma descrição mais detalhada das diferentes fases da qualificação, foram ainda criadas as sub-propriedades `q1`, `q2` e `q3`, todas subordinadas a `qt`.

Por fim definimos ainda que algumas propriedades são do tipo `owl:FunctionalProperty`

e `owl:InverseFunctionalProperty`, que são

- `circuitRef`;
- `constructorRef`;
- `driverRef`.

```
pred:circuitRef rdf:type owl:DatatypeProperty,  
  ↪ owl:FunctionalProperty, owl:InverseFunctionalProperty ;  
  rdfs:domain ps:Circuit ;  
  rdfs:range xsd:string ;  
  rdfs:label "Circuit Reference" ;  
  rdfs:comment "Name of a circuit simplified to be referenced" .  
  
pred:constructorRef rdf:type owl:DatatypeProperty,  
  ↪ owl:FunctionalProperty, owl:InverseFunctionalProperty ;  
  rdfs:domain ps:Constructor ;  
  rdfs:range xsd:string ;  
  rdfs:label "Constructor Reference" ;  
  rdfs:comment "Human-readable identifier/reference to a  
  ↪ constructor" .  
  
pred:driverRef rdf:type owl:DatatypeProperty,  
  ↪ owl:FunctionalProperty, owl:InverseFunctionalProperty ;  
  rdfs:domain ps:Driver ;  
  rdfs:range xsd:string ;  
  rdfs:label "Driver Reference" ;  
  rdfs:comment "Human-readable identifier/reference to a driver" .
```

Estas propriedades representam identificadores únicos para as entidades **Circuit**, **Constructor** e **Driver**, respetivamente. Ao serem declaradas como funcionais, garante-se que cada instância possui no máximo um valor para essa propriedade. Por outro lado, ao serem também inversamente funcionais, assegura-se que cada valor dessa propriedade está associado a uma única instância, o que permite identificar de forma inequívoca cada entidade com base no seu "ref".

2.3 Restrições

A última parte da nossa ontologia foca-se em restrições. O nosso grupo criou diversas restrições para que os **Reasoners** conseguissem inferir sobre os nossos dados e identificá-los como fazendo parte de uma determinada entidade. Sem estas restrições os Reasoners não conseguiam fazer estas inferências. Para isto em primeiro lugar o nosso grupo decidiu para cada entidade criar uma restrição onde define uma classe existente como sendo uma **equivalente à interseção** de várias restrições, ou seja, uma instância só é considerada de um determinado tipo e se possuir **pelo menos um valor** para cada uma das propriedades específicas de uma classe. Esta solução surgiu-nos da forma em que se nós restringíssemos que uma classe necessitava de ter todas as propriedades que se encontravam nos ficheiros de CSV. No entanto esta solução não funcionou visto que algumas das propriedades não

tinham o domínio definido, pois várias classes necessitavam delas e estas restrições então não ficaram funcionais.

Exemplo para a classe `Circuit`:

```
ps:Circuit owl:equivalentClass [  
  rdf:type owl:Class;  
  owl:intersectionOf (  
    [  
      rdf:type owl:Restriction ;  
      owl:onProperty pred:name ;  
      owl:someValuesFrom xsd:string  
    ]  
    [  
      rdf:type owl:Restriction ;  
      owl:onProperty pred:circuitRef ;  
      owl:someValuesFrom xsd:string  
    ]  
    [  
      rdf:type owl:Restriction ;  
      owl:onProperty pred:location ;  
      owl:someValuesFrom xsd:string  
    ]  
    [  
      rdf:type owl:Restriction ;  
      owl:onProperty pred:country ;  
      owl:someValuesFrom xsd:string  
    ]  
    [  
      rdf:type owl:Restriction ;  
      owl:onProperty pred:lat ;  
      owl:someValuesFrom xsd:float  
    ]  
    [  
      rdf:type owl:Restriction ;  
      owl:onProperty pred:lng ;  
      owl:someValuesFrom xsd:float  
    ]  
    [  
      rdf:type owl:Restriction ;  
      owl:onProperty pred:alt ;  
      owl:someValuesFrom xsd:integer  
    ]  
  )  
]
```

Com isto, o nosso grupo decidiu então pensar noutra solução que foi a seguinte, para cada entidade verificámos se existiam propriedades que só a entidade em específico continha e verificámos o seguinte:

- A classe **Circuit** contém a propriedade `circuitRef`;
- A classe **Constructor** contém a propriedade `constructorRef`;
- A classe **Driver** contém a propriedade `driverRef`;
- A classe **Qualifying** contém a propriedade `q1`;
- A classe **Race** contém a propriedade `round`;
- A classe **Result** contém a propriedade `milliseconds`;
- A classe **Status** contém a propriedade `status`;
- A classe **Standing** contém a propriedade `numberOfWins`.

Para cada uma destas decidimos criar uma restrição similar à explicada anteriormente, mas que fosse aplicada apenas a esta propriedade única de cada classe. Como por exemplo para a propriedade `constructorRef`:

```
ps:Constructor owl:equivalentClass [
  rdf:type owl:Class ;
  owl:intersectionOf (
    [
      rdf:type owl:Restriction ;
      owl:onProperty pred:constructorRef ;
      owl:someValuesFrom xsd:string
    ]
  )
] .
```

Com isto os motores de inferência conseguiram já inferir que alguns dados pertenciam a algumas classes como podemos observar na imagem seguinte:

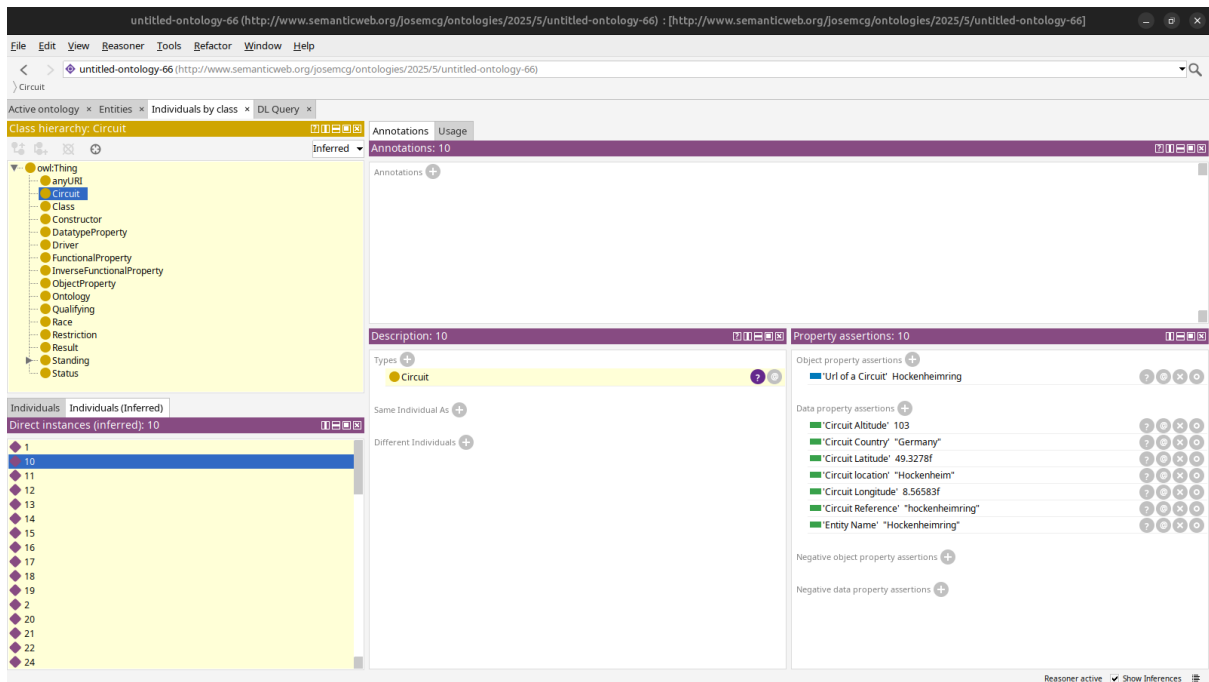


Figura 1: Motor de Inferência do Protégé com as restrições aplicadas

Onde podemos observar que o motor de inferência conseguiu inferir bem os dados do tipo **Driver**.

Contudo tivemos alguns problemas com as classes **ConstructorResults**, **Season** e a separação entre **DriverStanding** e **ConstructorStanding**, pois o motor de inferência conseguiu inferir os dados para o tipo **Standing**, no entanto não conseguia fazer esta separação. Tentámos adicionar a seguinte restrição:

```
ps:DriverStanding owl:equivalentClass [
  rdf:type owl:Class ;
  owl:intersectionOf (
    ps:Standing
    [
      rdf:type owl:Restriction ;
      owl:onProperty pred:hasDriver ;
      owl:someValuesFrom ps:Driver
    ]
  )
]
```

Esta diz que a classe **DriverStanding** é **equivalente** à interseção entre a classe **Standing** e uma restrição que exige que exista **pelo menos um valor** para a propriedade **hasDriver**. Ou seja, uma instância é considerada um **DriverStanding** se for um **Standing** e se estiver associada a um **Driver** através da propriedade **hasDriver**. No entanto esta restrição não funcionou, levando a uma conclusão que foi para conseguirmos ter esta inferência funcional teria de ser através de regras de SPIN.

Outras classe que nos deram problemas foi a **Season**, pois esta apresenta apenas duas propriedades sendo uma delas usada como URI e outra **url** que é partilhada entre outras

classes e a classe **ConstructorResult** que não apresentava nenhuma propriedade única. Decidimos também que para a inferência destas classes seriam necessárias regras de SPIN (que serão explicadas mais à frente no relatório).

3 Conjunto de Inferências (SPIN)

De forma a ultrapassar algumas limitações dos reasoners utilizados (nomeadamente os do GraphDB e do Protégé), desenvolvemos um conjunto de regras de inferência escritas em SPIN (SPARQL Inferencing Notation). Estas regras permitem enriquecer os dados automaticamente, atribuindo classes a entidades quando não é possível inferi-las diretamente, e criando novas relações úteis para análise e visualização.

Todas as regras de SPIN definidas encontram-se no ficheiro `inference_rules.py` que se encontra no diretório `/src/f1-djangoApp/f1App`.

3.1 Regras de Apoio à Inferência

Devido à incapacidade dos reasoners do GraphDB e do Protégé para inferirem corretamente as classes de alguns dados, foi necessário adicionar regras suplementares para ultrapassar estas limitações.

Como tanto os *standings* dos *construtores* como dos *pilotos* pertencem à mesma classe **Standing**, foi necessário criar regras que permitissem distingui-los, sendo que a única diferença entre eles é a presença da propriedade **pred:hasDriver** ou **pred:hasConstructor**.

```
# Driver Standings
INSERT {
    ?s a ps:DriverStanding .
}
WHERE {
    ?s a ps:Standing .
    ?s pred:hasDriver ?c .
}

# Constructor Standings
INSERT {
    ?s a ps:ConstructorStanding .
}
WHERE {
    ?s a ps:Standing .
    ?s pred:hasConstructor ?c .
}
```

Como as temporadas apenas possuem apenas duas propriedades — o **ano** e o **URL** —, sendo o ano utilizado para gerar o seu URI e o URL partilhado por várias entidades, foi necessário criar uma regra SPIN. Esta regra estabelece que qualquer entidade que possua a propriedade **pred:url** e que não tenha nenhuma classe associada, deve ser classificada como pertencente à classe **Season**.

```

INSERT {
    ?entity a ps:Season .
}
WHERE {
    ?entity pred:url ?url .
    FILTER NOT EXISTS {
        ?entity rdf:type ?anyType .
    }
}

```

Para os resultados dos construtores, a situação é semelhante à anterior. A propriedade **pred:obtainedPoints** é partilhada por outras entidades, o que exige a criação de uma regra específica que permita atribuir corretamente a classe **ConstructorResult** às instâncias apropriadas.

```

INSERT {
    ?entity a ps:ConstructorResult .
}
WHERE {
    ?entity pred:obtainedPoints ?points .
    FILTER NOT EXISTS {
        ?entity rdf:type ?anyType .
    }
}

```

3.2 Regras Adicionais

Também criámos algumas regras de inferência que permitem criar mais relações entre as entidades. Estas regras não só enriquecem o nosso conjunto de dados, como também permitem simplificar certas queries, pois adicionam a informação necessária diretamente na triplestore.

Criámos uma regra que adiciona a cada circuito o número de corridas que já ocorreram nesse circuito. Esta informação é útil para sabermos quais são os circuitos mais populares:

```

INSERT {
    ?circuit pred:numberOfRaces ?count .
}
WHERE {
    SELECT ?circuit (COUNT(?race) AS ?count)
    WHERE {{
        ?race a ps:Race ;
        pred:hasCircuit ?circuit .
    }}
    GROUP BY ?circuit
}

```

Criámos uma regra que, em cada corrida, adiciona quem foi o vencedor dessa corrida. Esta informação é bastante útil para queries em que queremos saber quem venceu sem termos de verificar todas as posições de todos os condutores:

```
INSERT {
    ?race pred:wasWonBy ?driver
}
WHERE {
    ?result a ps:Result ;
    pred:hasDriver ?driver ;
    pred:position "1"^^xsd:integer ;
    pred:participatedIn ?race .
}
```

Criámos uma regra que associa a cada corrida a temporada em que esta ocorreu:

```
INSERT {
    ?race pred:partOfSeason ?season .
}
WHERE {
    ?race a ps:Race ;
    pred:year ?year .
    ?season a ps:Season ;
    pred:url ?url .
    FILTER(CONTAINS(STR(?url), STR(?year)))
}
```

Criámos uma regra que adiciona o nome completo a cada condutor, uma vez que anteriormente era necessário concatenar o primeiro e o último nome sempre que se pretendia obter o nome completo:

```
INSERT {
    ?driver pred:fullName ?fullName .
}
WHERE {
    ?driver a ps:Driver ;
    pred:forename ?forename ;
    pred:surname ?surname .
    BIND(CONCAT(?forename, " ", ?surname) AS ?fullName)
}
```

Criámos também uma regra que associa a cada corrida o condutor que fez a volta mais rápida, o que, tal como a regra para o vencedor, facilita as queries que retornam tanto o vencedor como a volta mais rápida:

```
INSERT {
    ?race pred:fastestDriver ?driver .
}
WHERE {
```

```

{
    SELECT ?race (MAX(xsd:decimal(?speedVal)) AS ?maxSpeed)
    WHERE {
        ?result a ps:Result ;
        pred:participatedIn ?race ;
        pred:fastestLapSpeed ?speedRaw .
        BIND(xsd:decimal(?speedRaw) AS ?speedVal)
    }
    GROUP BY ?race
}

?result a ps:Result ;
    pred:participatedIn ?race ;
    pred:hasDriver ?driver ;
    pred:fastestLapSpeed ?driverSpeedRaw .
BIND(xsd:decimal(?driverSpeedRaw) AS ?driverSpeed)
FILTER(?driverSpeed = ?maxSpeed)
}

```

Por fim, criamos também uma regra que adiciona novos triplos indicando a que construtor pertenciam os condutores em cada ano em que participaram em corridas. Esta informação enriquece o nosso frontend, permitindo aos utilizadores visualizar o histórico dos construtores pelos quais os condutores passaram:

```

INSERT {
    << ?driverId pred:wasInConstructor ?constructorId >> pred:year ?year
    ↪ .
}
WHERE {
    ?result a ps:Result ;
        pred:participatedIn ?raceId ;
        pred:hasDriver ?driverId ;
        pred:hasConstructor ?constructorId .

    ?raceId a ps:Race ;
        pred:year ?year .
}

```

4 Alterações nas Operações sobre os dados

Para melhorar os tempos das queries algumas foram alteradas para usar os novos triplos formados por regras de SPIN.

No exemplo abaixo se existir fullname então este é usado como regex em vez de concatenar o forname e surname.

```

SELECT ?driverId ?forename ?surname ?nationality
WHERE {{
    ?driverId a ps:Driver ;
    pred:forename ?forename ;
    pred:surname ?surname ;
    pred:nationality ?nationality

    OPTIONAL {{ ?driverId pred:fullName ?fullName . }}

    BIND(COALESCE(?fullName, CONCAT(?forename, " ", ?surname))
    ↪ AS ?nameToSearch) .

    FILTER regex(?nameToSearch,"{query}", "i") .
}}

```

Neste exemplo de retrieve by race id o OPTIONAL é usado para esta conseguir obter os novos triplos inferidos por SPIN mas caso estes não existem o resto dos triplos são obtidos na mesma. Isto também é usado para quando as regras de SPIN ainda não foram ativadas.

```

OPTIONAL {{
    ?winnerDriverId a ps:Driver ;
    pred:fullName ?winnerDriverName .
}}
OPTIONAL {{
    ?fastestDriverId a ps:Driver ;
    pred:fullName ?fastestDriverFullName .
}}
OPTIONAL {{
    ?result a ps:Result ;
    pred:participatedIn ns:{race_id} ;
    pred:hasDriver ?fastestDriverId ;
    pred:fastestLapSpeed ?fastestLapSpeed ;
    pred:fastestLap ?fastestLap ;
    pred:fastestLapTime ?fastestLapTime ;
    pred:rank ?rank ;
    pred:position ?position .
}}

```

5 Wikidata e DBpedia

Para enriquecer o nosso conjunto de dados, utilizámos o SPARQLwrapper para obter informação através dos endpoints da Wikidata e da DBpedia. Os nossos dados já continham uma boa parte da informação presente nestes websites, mas uma das coisas que nos faltava eram imagens dos condutores e dos circuitos.

Da Wikidata obtivemos as imagens de cada um dos condutores

```
SELECT ?item ?itemLabel ?image WHERE {
  ?item rdfs:label "driver_name"@en.
  ?item wdt:P18 ?image.
}
LIMIT 1
```

Enquanto da DBpedia obtivemos tanto as imagens dos circuitos como uma pequena descrição sobre cada um deles.

```
SELECT ?image ?abstract WHERE {
  OPTIONAL { <http://dbpedia.org/resource/circuit_name> dbo:thumbnail
    ↪ ?image . }
  OPTIONAL { <http://dbpedia.org/resource/circuit_name> dbo:image
    ↪ ?image . }
  OPTIONAL { <http://dbpedia.org/resource/circuit_name> foaf:depiction
    ↪ ?image . }
  OPTIONAL { <http://dbpedia.org/resource/circuit_name> dbo:abstract
    ↪ ?abstract .
    FILTER (lang(?abstract) = "en") }
} LIMIT 1
```

Em ambos os casos, quando é pedida a informação de um condutor ou de um circuito, é feito um pedido aos endpoints dos sites, obtendo-se assim esta informação extra. Quando esta informação é obtida pela primeira vez, é então adicionada à nossa triplestore para poder ser retornada mais rapidamente em pedidos subsequentes.

6 Micro-formatos e RDFa

Para suportar a semântica em nossas páginas, adicionamos microformatos e anotação RDFa no html. Isso proporciona maior facilidade na exportação e no tratamento dos dados, utilizando parsers dedicados a cada um desses formatos. Para facilitar a identificação e o uso dessa funcionalidade, incluímos nas páginas que a possuem (páginas de driver, race e racetrack) um botão que permite copiar a parte do site referente ao conteúdo semântico descrito (Figure 2). Esta copia copia para clipboard um snippet do html que podemos usar no parser.

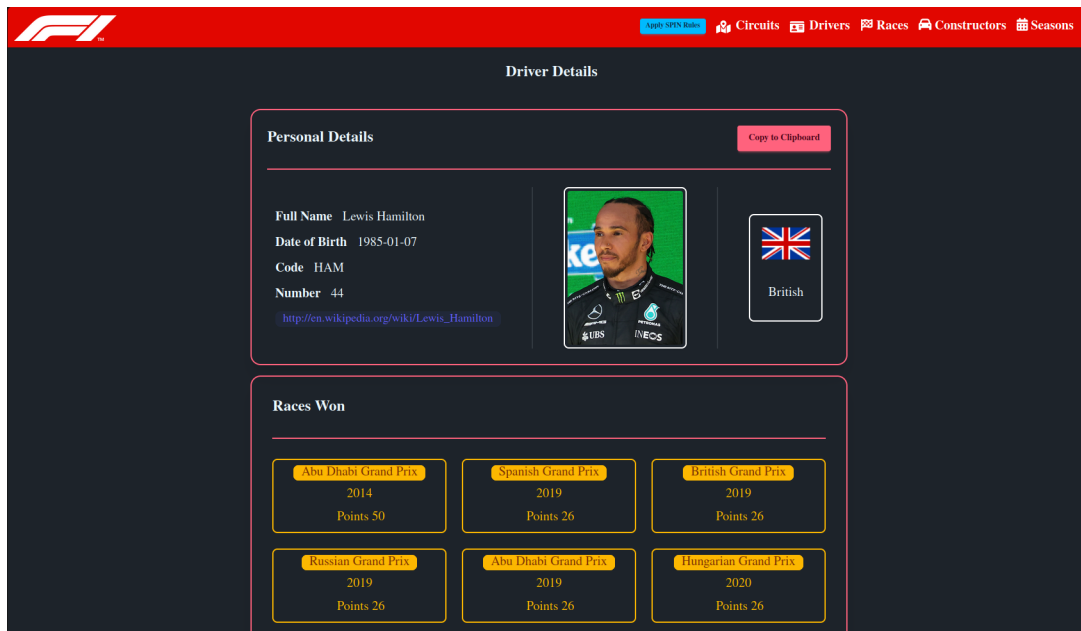


Figura 2: Driver Profile Page

6.1 Micro-formatos

A página do piloto (Figure 2) usa micro-formatos no **div** principal dos detalhes e é utilizado **h-card**. Dentro deste **div** são usadas várias propriedades para aumentar a semântica dos dados presentes no HTML. Exemplos dessas propriedades são **p-given-name**, **p-family-name**, **p-bday** e **p-number**. Por fim, são usados **h-feed** onde estão listas de construtores e lista de vitórias. Cada um destes é do tipo **h-constructor** e **h-event**, respetivamente. **h-constructor** consta de **p-name**, o nome do construtor, e **dt-start**, o ano em que o piloto esteve com o construtor. **h-event** consta de um nome da corrida **p-name**, a data **dt-start** e pontos da corrida **p-note**.

Aqui temos um excerto do resultado dado pelo parser:

```
"items": [
  {
    "type": [
      "h-card"
    ],
    "properties": {
      "given-name": [
        "Lewis Hamilton"
      ],
      "bday": [
        "1985-01-07"
      ],
      "code": [
        "HAM"
      ],
      "number": [
        "44"
      ]
    }
  }
]
```

```

    ],
    "category": [
        "British"
    ],
    "url": [
        "http://en.wikipedia.org/wiki/Lewis_Hamilton"
    ]
},
"id": "card",
"children": [
    {
        "type": [
            "h-feed"
        ],
        "properties": {},
        "children": [
            {
                "type": [
                    "h-event"
                ],
                "properties": {
                    "name": [
                        "Abu Dhabi Grand Prix"
                    ],
                    "points": [
                        "Points 50"
                    ],
                    "start": [
                        "2014"
                    ],
                    "url": [
                        "/races/id/918"
                    ]
                }
            }
        ]
    }
]

```


6.2 RDFa

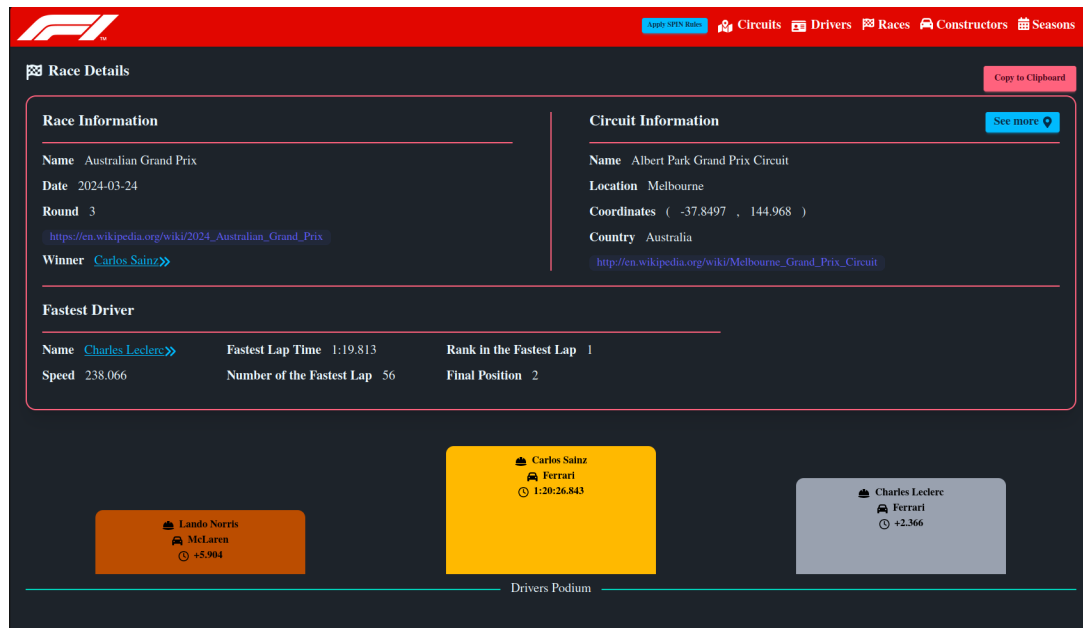


Figura 3: Detalhes da Corrida

A página de detalhes de corrida (Figure 3) usa RDFa (Resource Description Framework in attributes), utilizando o vocabulário <http://schema.org/>. A informação da corrida é marcada por property em cada tag que contém a informação: property="name", property="startDate", property="eventStatus" e property="url". Estas contêm um valor simples.

A propriedade winner, property="winner", contém uma entidade typeof="Person" que contém um nome e uma URL.

Após isto, há mais três secções: Circuit Information, Fastest Driver e Drivers Podium. A Circuit Information está dentro de um bloco marcado com property="location" do tipo typeof="Circuit" e contém propriedades como:

- name
- addressLocality
- latitude e longitude
- addressCountry
- url

O Fastest Driver descreve uma entidade do tipo property="Driver" do tipo typeof="Person" esta contém as seguintes propriedades:

- name
- url

Para além disso, também contém detalhes como velocidade, tempo, lapCount, ranking e position.

O Driver Podium está descrito, cada parte, por property="subEvent" do tipo type="Podium". Todos os três usam uma propriedade driver, igual à descrita anteriormente. Para além disso, usam uma que contém o construtor, que é do tipo Constructor. Por fim, é usada property="extraTimeOfFirst" para o tempo de diferença para o primeiro lugar e property="resultTime" para o mesmo.

Na figura abaixo (Figure 4) podemos ver o resultado dado usando o website rdfa.info/play/.

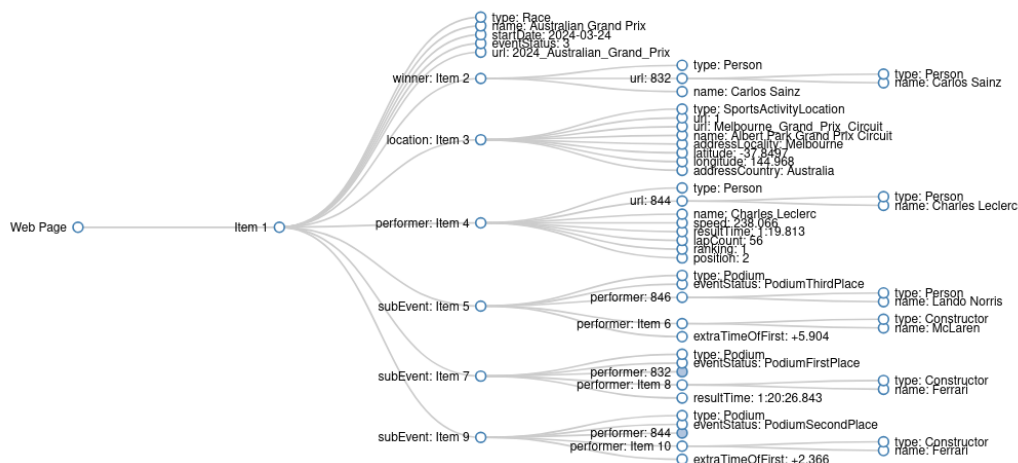


Figura 4: race RDFa

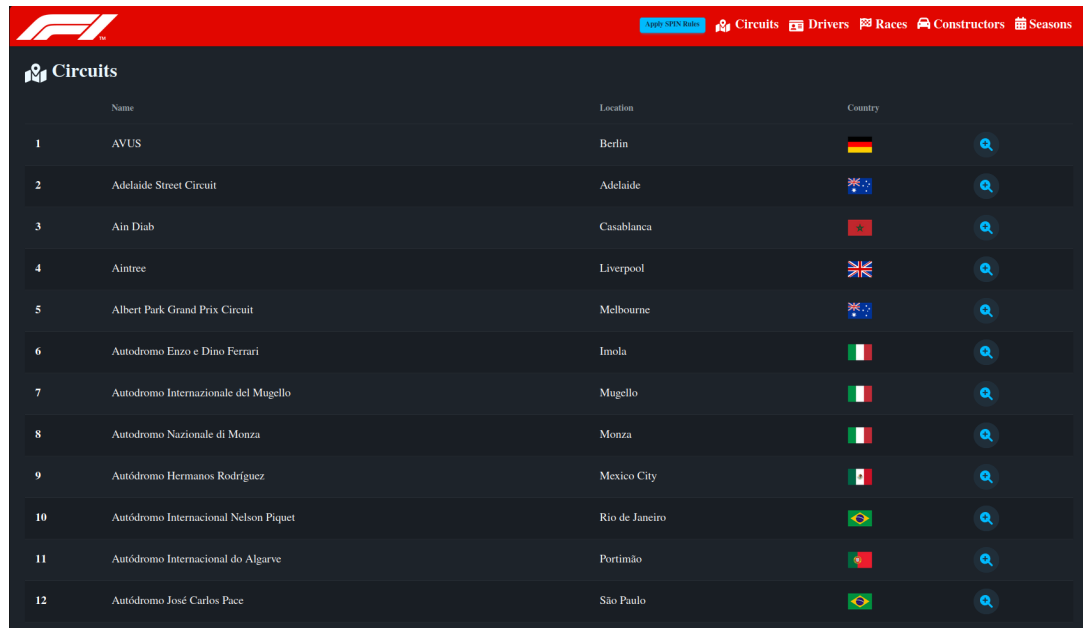
A pagina circuit detail também usa RDFa e na figura abaixo (Figure 5) podemos ver o seu grafico.



Figura 5: circuit RDFa

7 Novas Funcionalidades da Aplicação

7.1 Detalhes para circuitos



























	Name	Location	Country	
1	AVUS	Berlin		
2	Adelaide Street Circuit	Adelaide		
3	Ain Diab	Casablanca		
4	Aintree	Liverpool		
5	Albert Park Grand Prix Circuit	Melbourne		
6	Autodromo Enzo e Dino Ferrari	Imola		
7	Autodromo Internazionale del Mugello	Mugello		
8	Autodromo Nazionale di Monza	Monza		
9	Autódromo Hermanos Rodríguez	Mexico City		
10	Autódromo Internacional Nelson Piquet	Rio de Janeiro		
11	Autódromo Internacional do Algarve	Portimão		
12	Autódromo José Carlos Pace	São Paulo		

Figura 6: Pagina de Detalhes do Circuito

Foram adicionadas duas novas paginas para circuitos uma contendo uma lista dos mesmos e outra contendo os detalhes de um circuito. (Figure 6)(Figure 7)

7.2 Novos detalhes para corrida

Na página dos detalhes da corrida foi adicionado pelas regras de SPIN o vencedor da corrida e o piloto mais rápido na pista ou seja com o melhor lap time dessa corrida. (Figure 3)

7.3 Descrição para Circuito

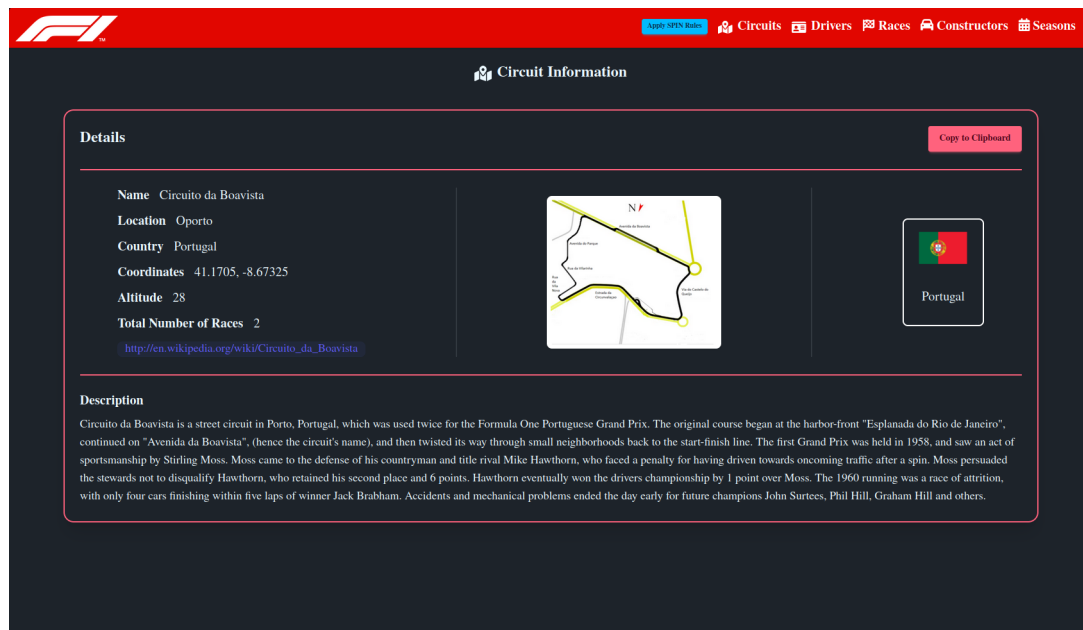


Figura 7: Pagina de Detalhes do Circuito

Nas paginas de Circuito adicionamos a descrição que e obtida através do DBpedia se não existir descrição é colocado um place holder text. (Figure 7)

7.4 Imagens

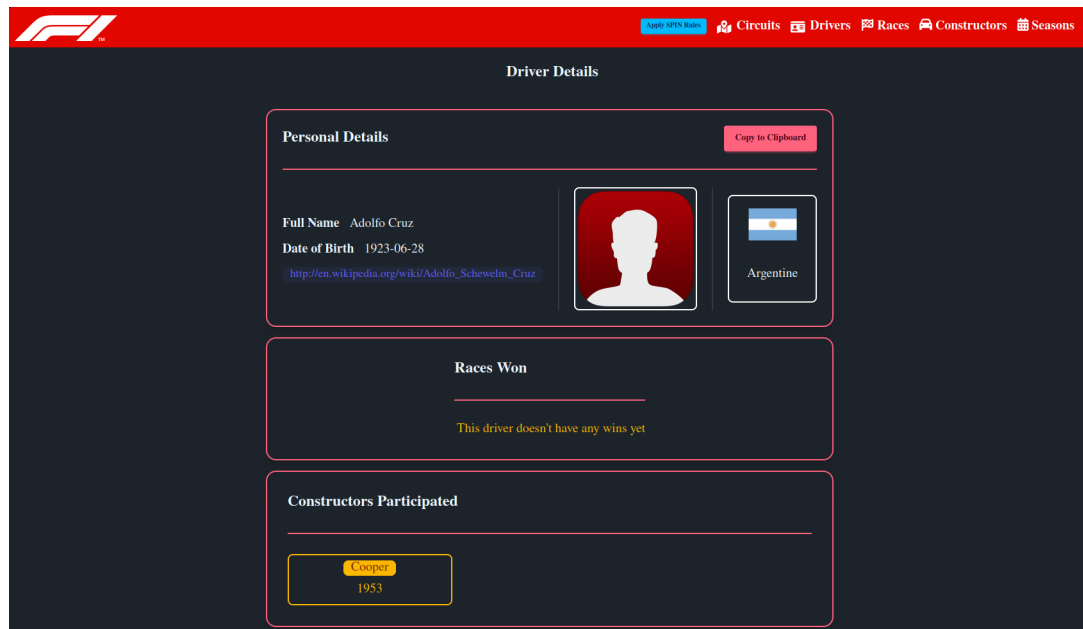


Figura 8: Exemplo de imagem não disponivel

Nas paginas de Driver e Circuit foram adicionadas imagens do Wikidata e do DBpedia respetivamente. Se não for dado imagem é colocado uma imagem default. (Figure 2)

(Figure 7) Exemplo sem imagem Figure 8.

7.5 Construtores de um piloto

Esta nova função disponibiliza uma lista de construtores nos quais os pilotos participaram ao longo do tempo esta lista esta contida na DriverProfile page (Figure 9).

Points 26	Points 26	Points 26
Tuscan Grand Prix 2020 Points 26	Portuguese Grand Prix 2020 Points 26	Emilia Romagna Grand Prix 2020 Points 26
Saudi Arabian Grand Prix 2021	Turkish Grand Prix 2010	Canadian Grand Prix 2010
Constructors Participated		
Mercedes 2024	Mercedes 2023	Mercedes 2022
Mercedes 2021	Mercedes 2020	Mercedes 2019
Mercedes 2018	Mercedes 2017	Mercedes 2016
Mercedes 2015	Mercedes 2014	Mercedes 2013
McLaren 2012	McLaren 2011	McLaren 2010

Figura 9: Construtores na pagina de Driver

8 Conclusão

O presente trabalho prático melhorou o desenvolvimento de um sistema semântico para o domínio da Formula 1, dando continuidade aos conceitos utilizados no TP1. Neste trabalho conseguimos concretizar uma ontologia que inferia maioria das classes como Circuit, Constructor, Driver e Race. E através de regras SPIN foram corrigidos erros dos motores de inferência e também foram feitas novas relações entre os dados podendo aumentar a quantidade de informação por este dada.

Através de sites como o DBpedia e Wikidata aumentamos a quantidade de dados disponíveis para apresentar no website.

Por fim colocamos expressividade nas paginas HTML usando RDFa e microformatos.

Este trabalho permitiu consolidar o que foi abordado nas aulas, explorar algumas das limitações da linguagem OWL, bem como nos deu boas bases de como ir obter informação semântica a outros websites que a disponibilizam.

Achamos que os principais pontos do trabalho foram todos executados no entanto devido ao trabalho colocado para traduzir as paginas de **React** para **Django Templates**, não podemos expandir certas partes do trabalho como uma pagina para detalhar os construtores.

9 Organização do Código

O nosso projeto apresenta 2 grandes diretórios:

- **data**: Contém todos os ficheiros `.csv` que contém os dados que foram arrançados através do kaggle;
- **src**: Contém todos os ficheiros de código implementados. Encontra-se ainda subdividido com outros diretórios:
 - O ficheiro `requirements.txt` contém todas as dependências necessárias para se puder executar o projeto;
 - No diretório `semantic_data` estão presentes 3 ficheiros:
 - * `f1_data.n3`, contém os dados do dataset no formato **N3**;
 - * `f1_ontology.n3`, contém a ontologia definida;
 - * `f1_data_ontology.n3`, contém a ontologia definida e os dados em formato N3, tudo no mesmo ficheiro.
 - O ficheiro `envs.py` que está presente no diretório `src/f1-djangoApp/mysite` contém a string de conexão para o GraphDB e o nome do repositório;
 - O ficheiro `src/f1-djangoApp/mysite/urls.py` contém todos os urls definidos para a nossa aplicação. É nesse ficheiro que é chamada a função `apply_inference_rules` que aplica as regras de SPIN de apoio à ontologia;
 - O ficheiro `src/f1-djangoApp/f1App/views.py` contém todas a implementação para todas views que depois são associadas a um url;
 - O ficheiro `src/f1-djangoApp/f1App/nacionality` contém dicionários para que se consigamos associar nacionalidades a imagens de bandeiras;
 - O diretório `src/f1-djangoApp/f1App/repositories` contém toda a lógica, de query, inserção, update e eliminação de elementos na **GraphDB**, dividido por entidades, ou seja são os componentes responsáveis pela comunicação com a **GraphDB**;
 - O diretório `src/f1-djangoApp/f1App/services` contém toda a lógica de tratamento de dados que foram obtidos pelos repositórios, dividido por entidades, ou seja comunica com os repositórios para organizar melhor os dados obtidos.

10 Instruções de execução

Para executar o projeto são necessario executar os seguintes passos:

1. Instalar os requirements presentes na pasta **src** usando o comando `pip install -r requirements.txt`
2. Ter o GraphDB com um repositório com o nome **f1-pitstop** a correr e carregar primeiro o ficheiro que contém os dados `f1_data.n3` e depois, quando este já tiver sido carregado com sucesso, carregar o ficheiro que contém a ontologia `f1_ontology.n3`
3. Executar o projeto django dentro da pasta **f1-djangoApp** usando o comando `python3 manage.py runserver`. O website estará disponível em `localhost:8000`