FootOntologyPlus

Giacomo Leo Bertuccioli
0001136879

Alex Mazzotti
0001146685

Indice

Capitolo 1

Introduzione

Footology [3] è un knowledge representation system progettato per rappresentare e organizzare le informazioni rilevanti del mondo del calcio. L'ontologia consente di modellare i diversi aspetti e concetti del gioco, come ad esempio partite, squadre di calcio e giocatori. L'idea dietro al progetto è quella di facilitare l'integrazione dei dati, l'interoperabilità e fornire sistemi di interrogazione avanzati per applicazioni riguardanti il calcio.

La nostra estensione FootOntologyPlus mira ad espandere l'originale, modellando meglio alcuni dei concetti inclusi da quest'ultima o aggiungendone di nuovi. L'entità delle nostre modifiche è discussa nei capitoli successivi.

Il nostro progetto è stato realizzato usando l'editor **Protegé**, un programma open source per visualizzare e modificare ontologie.

Capitolo 2

Ontologia di partenza

2.1 Informazioni di base

Footology è disponibile su Github [3], pubblicata dall'utente **arditb1997**, ed è un'ontologia relativamente semplice che modella gli aspetti principali del calcio.

L'ontologia è disponibile in 3 formati diversi: **owl**, **rdf** e **ttl**. **OWL** e **RDF** sono due linguaggi di knowledge representation, mentre il terzo, **Turtle**, è semplicemente una sintassi usata per esprimere ontologie scritte in uno dei due linguaggi precedenti in un formato testuale più facile da leggere.

2.2 Dettagli dell'ontologia

La seguente tabella riassume i contenuti dell'ontologia:

Classi	13
Object properties	26
Data properties	46
Individui	0
Nodi (del grafo)	59
Foglie (del grafo)	62

2.3 Classi dell'ontologia

Le classi contenute nell'ontologia sono:

- `Player`, che rappresenta un calciatore;
- `Team`, che rappresenta una squadra di calcio;
- `Match`, una partita di calcio;

- `PerformanceStats`, statistiche riguardanti un calciatore in una partita o all'intera partita;
- `Tournament`, un torneo di calcio, che ha come sottoclassi `KnockOutTournament`, un torneo piramidale ad eliminazione diretta, e `League`, un torneo in cui ogni squadra affronta tutte le altre e accumula punti in base al risultato delle partite;
- `Stadium`, uno stadio in cui si giocano le partite;
- `Coach`, un allenatore;
- `Referee`, un arbitro;
- `Position`, la posizione di un giocatore in partita;
- `Trophy`, un trofeo vinto da una squadra;
- `Award`, un premio vinto da un giocatore.

2.4 Proprietà importanti

Le proprietà più importanti nell'ontologia sono:

- `playsFor`, che collega un giocatore a una squadra in cui gioca;
- `participatesIn`, che collega un giocatore a una partita a cui partecipa;
- `competesIn`, che collega una squadra a una partita in cui è coinvolta;
- `partOf`, che collega una squadra ad un torneo a cui partecipa;
- `hasStats` e `hasPerformanceStats`, che collegano rispettivamente un giocatore e una partita a delle statistiche;
- `hosts`, che collega uno stadio ad una partita;
- `officiates`, che collega un arbitro ad una partita;
- `manages`, che collega un allenatore ad una squadra.

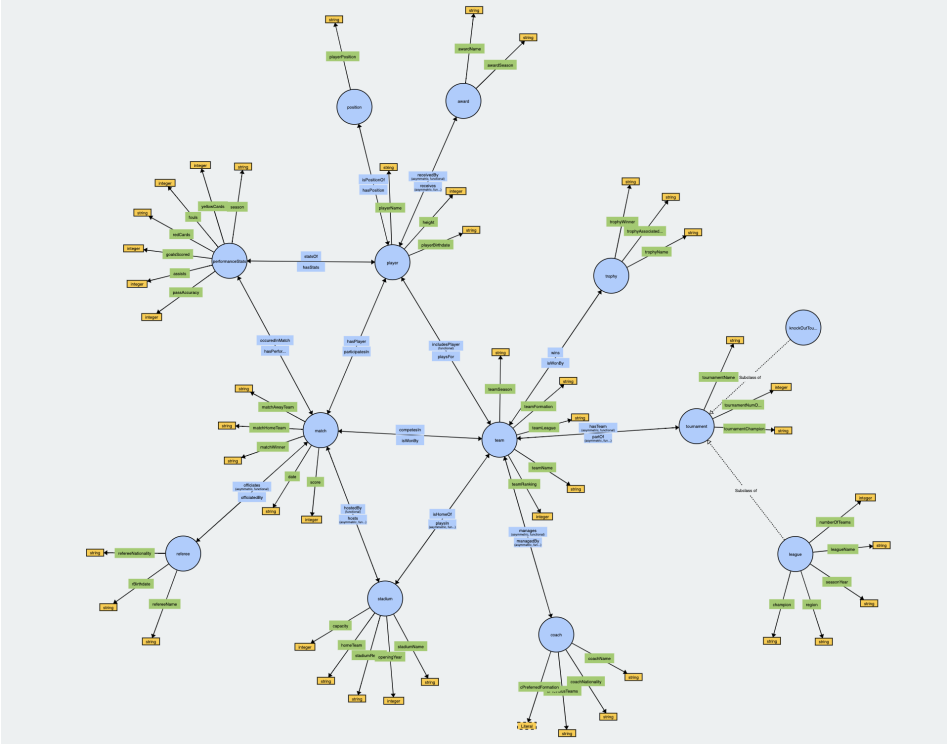

Figura 2.1: Grafico dell'ontologia, realizzato con WebVOWL [5, 3].

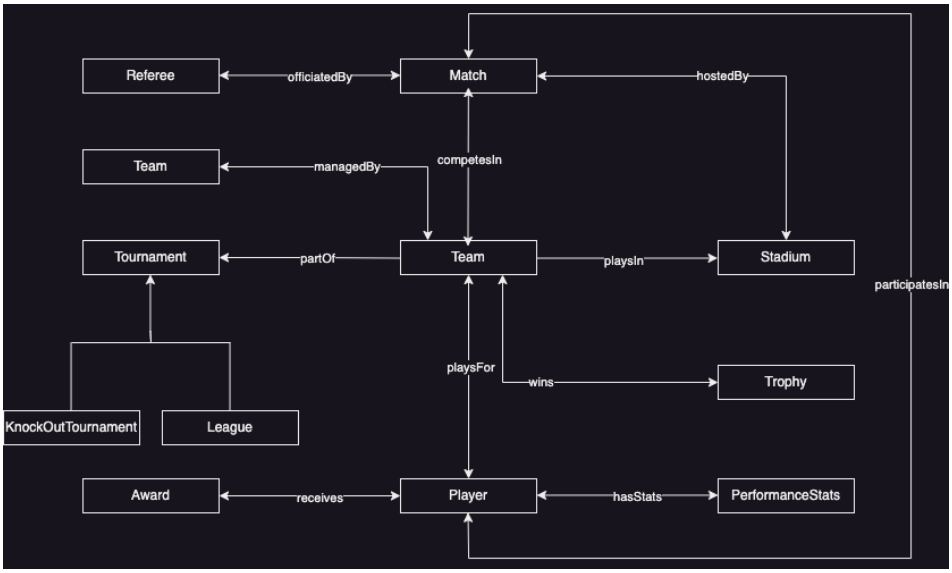

Figura 2.2: Schema dell'ontologia, realizzato con Draw.io [2, 3].

2.5 Ontologie incluse

Per realizzare l'ontologia sono stati utilizzati:

- **RDF** (**Resource Description Language**), un modello per la rappresentazione di dati (e metadati), avente una struttura graph-based nella quale essi sono scritti sotto forma di triple soggetto-predicato-oggetto;
- **RDFS** (**RDF Schema**), un insieme di classi e proprietà utili che estendono il vocabolario di RDF;
- **OWL** (**Web Ontology Language**), un linguaggio standard usato per realizzare ontologie, che estende RDF;
- **DublinCore**, un vocabolario che contiene termini standard per la definizione di metadati (titolo, autore, descrizione, ecc.).

Capitolo 3

Modifiche all'ontologia

L'ontologia iniziale aveva una struttura minimale e mancava di molti concetti. È stato necessario eseguire un lavoro di ristrutturazione, per renderla più completa possibile, e adeguata al contesto del calcio.

In particolare, sono state aggiunte nuove informazioni tramite l'introduzione di classi e relazioni, è stata modificata la struttura complessiva dell'ontologia per migliorarne l'organizzazione, e infine si è cercato di integrarla con parti esistenti di altre ontologie, attraverso operazioni di collegamento (join) tra concetti compatibili.

3.1 Modifica concetti esistenti

- **Match**

  In `Match` è stato aggiunta come sottoclasse la classe `FriendlyMatch` che rappresenta le partite amichevoli, ovvere quelle che non appartengono a un `Tournament`.

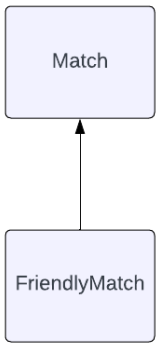

Figura 3.1: UML Match

- **PerformanceStats**

  Invece, in `PerformanceStats` sono state aggiunte due sottoclassi, disgiunte tra di loro, `PlayerPerformanceStats`, e `TeamPerformanceStats`. Questo è stato fatto per distinguere meglio il concetto di statistiche su un `Match`, e statistiche di un singolo `Player`. Per esempio la statistica del possesso palla non è concetto che è associato a un singolo calciatore ma è associata alla squadra.

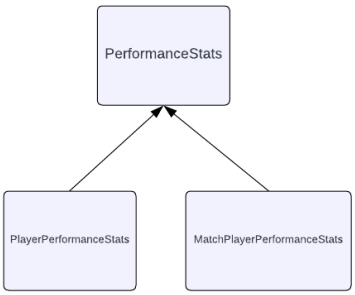

Figura 3.2: UML PerformanceStats

- **Position**

  In `Position`, sono state definite le specifiche posizioni in campo, così da rappresentare in modo preciso e strutturato il ruolo dei giocatori all'interno della formazione. Nell'ontologia originale erano modellate come Data Properties.

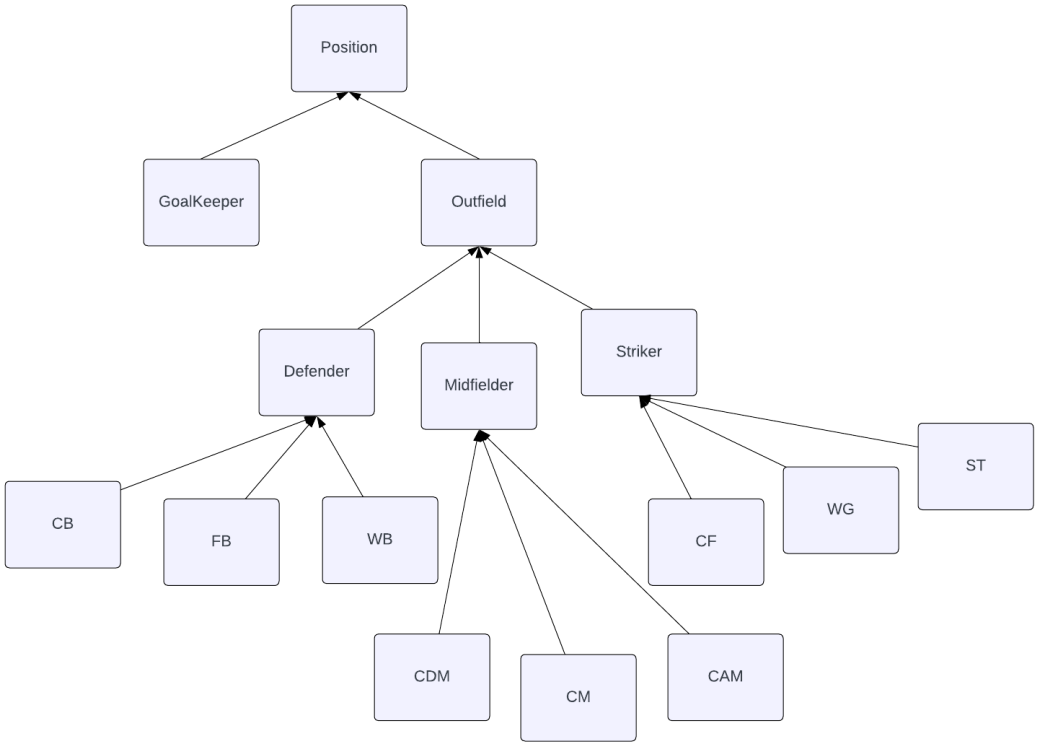

Figura 3.3: UML Position

Le abbreviazioni CB, FB, WB, CDM, etc indicano le specifiche posizioni in campo, per esempio CB significa Central Back, cioè Difensore centrale.

3.2 Join con altre ontologie

Come richiesto dalla consegna, è stato necessario integrare l'ontologia con altre ontologie esistenti, facendo operazioni di join per favorire l'interoperabilità dei concetti.
In particolare, sono stati importati:

- La classe `Person` dall'ontologia *schema.org* [4], utilizzata per rappresentare entità individuali come giocatori, allenatori e arbitri.

  Nella nostra ontologia, `Person` è stata inserita come superclasse delle classi `Player`, `Coach` e `Referee`. Il join è stato eseguito in modo da includere solo le Object Properties che ci interessavano, escludendo quelle non rilevanti per il nostro dominio.

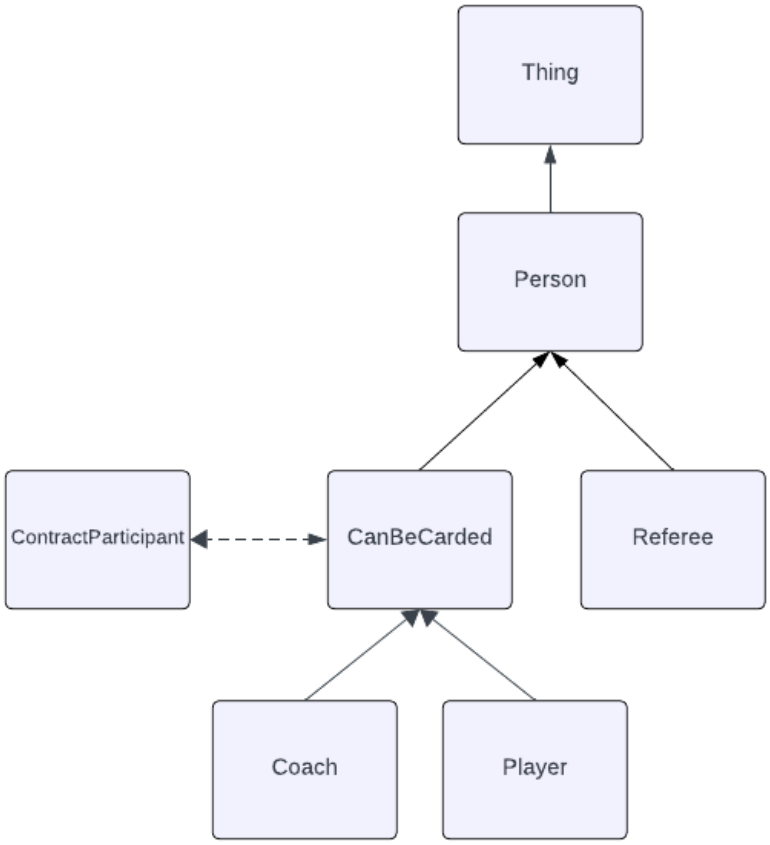

Figura 3.4: UML Person

- Le classi `City` e `Country` da *dbpedia.org* [1], adottate per modellare i concetti geografici.

  Anche in questo caso è stato effettuato un join selettivo: `City` e `Country` sono state collegate alla nostra ontologia per rappresentare, rispettivamente, la città dello stadio, la nazione della lega, e la nazionalità delle persone.

  La relazione `hasNationality` tra `Country` e `Person`, è stata definita direttamente sulla superclasse `Person`, in modo da poter utilizzare tale relazione per tutte le sue sottoclassi, ovvero `Player`, `Referee` e `Coach`.

3.3 Classi aggiunte

Come detto a inizio capitolo l'ontologia di partenza era incompleta su vari aspetti. Di seguito, si mostrano i nuovi concetti inseriti.

- `Contract`, con sottoclassi `CoachContract` e `PlayerContract`

  I contratti sono utili per mettere in relazione giocatori e allenatori alle squadre, tenendo conto anche di una storicizzazione con i contratti terminati. Per esempio un contratto terminato, indica che un certo giocatore/allenatore ha giocato per una certa squadra.

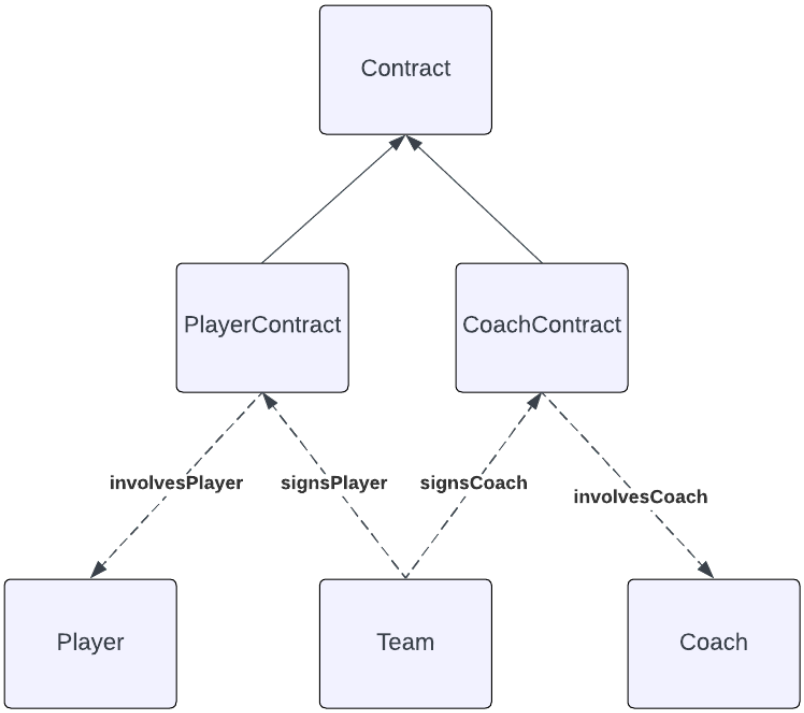

Figura 3.5: UML contratti in relazione con Team, Coach e Player

Nella figura 3.5 sono mostrate solo alcune relazioni che legano i contratti con giocatori e allenatori. Per esempio non sono mostrate le inverse, o la relazione padre, `Person` - `Contract`, e `Team` - `Contract`.

- `Formation`

  Inizialmente, i moduli adottati dalle squadre nei match erano rappresentati tramite Data Properties. Tuttavia, poiché l'insieme dei moduli più comuni è ben definito (es. 4-3-3, 3-5-2, ecc.), abbiamo preferito modellare i moduli come sottoclassi di `Formation`.

- `MatchEvent`

  Gli eventi dei match non erano presenti, quindi li abbiamo inseriti attraverso una gerarchia di classi.

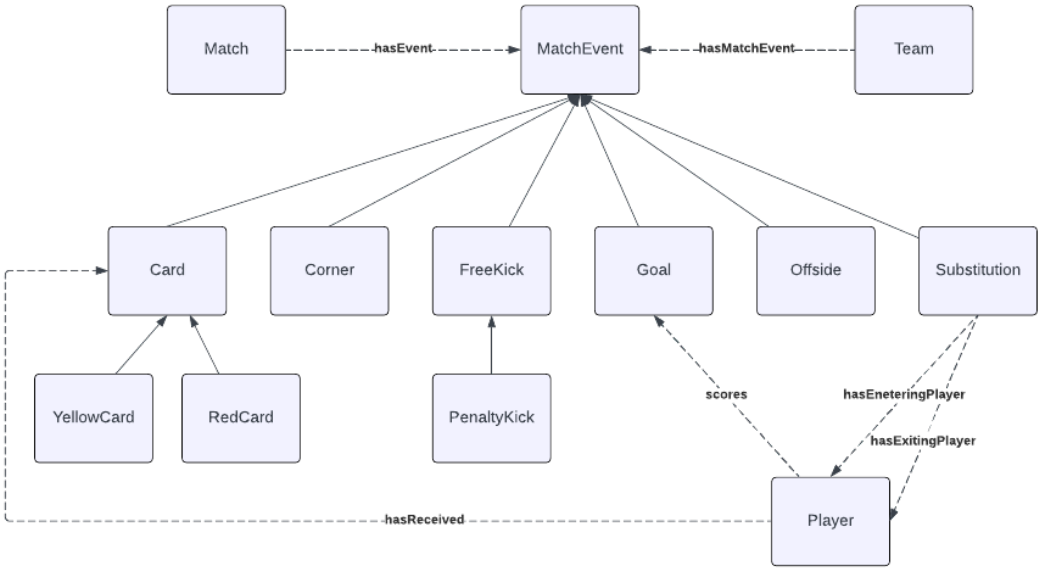

Figura 3.6: UML eventi Match

Nella figura 3.6 sono mostrate anche come gli eventi sono messi in relazione ai `Match`; è stata inserita anche la relazione ai `Team`, per modellare di quale squadra è l'evento. Inoltre alcuni eventi sono associati ai `Player`.

3.3.1 Classi d'errore

Per gestire eventuali errori dovuti a incompatibilità nei dati, rilevati tramite regole SWRL, è stata introdotta una classe `Error`. Questa classe raccoglie diverse sottoclassi, ciascuna delle quali rappresenta un tipo specifico di errore semantico o logico. In questo modo, è possibile classificare e tracciare gli errori direttamente all'interno dell'ontologia, facilitando il controllo di consistenza durante la validazione dei dati.
Le sottoclassi di `Error` sono:

- `FriendlyMatchInTournamentError`
- `InconsistentBallPossessionInMatchError`
- `InconsistentContractDatesError`
- `InconsistentMatchScoresError`
- `InconsistentPlayerPassesError`
- `InvalidLeagueMatchFormatError`
- `InvalidMatchError`
- `InvalidSubstitutionError`
  - `DuplicateSubstitutedError`
  - `DuplicateSubstituteError`
  - `InvalidReserveReplacementError`
  - `InvalidSubstitutionPlayersError`
  - `StarterAsSubstituteError`
- `MultipleRedCardsError`
- `NonGoalKeeperWithSavesError`
- `TeamNotInMatchError`
- `UncontractedPlayerInMatchError`

Vedi capitolo su regole SWRL per l'utilizzo di queste classi.

3.4 Proprietà aggiunte

Di seguito una lista delle proprietà più importanti aggiunte.
Object Properties:

- `hasEvent` (da `Match` a `MatchEvent`): associa a una partita i suoi eventi (goal, falli, sostituzioni, ecc.).
- `hasFormation` (da `Team` a `Formation`): specifica la formazione adottata da una squadra.
- `hasMatchEvent` (da `Team` a `MatchEvent`): collega una squadra agli eventi della partita.
- `hasPlayer` (da `Match` a `Player`): elenca i giocatori che partecipano a una determinata partita.
- `hasPlayerPerformanceStats` (da `Match` a `PlayerPerformanceStats`): collega una partita alle statistiche individuali dei giocatori.
- `hasPosInFormation` (da `Formation` a `Position`): definisce le posizioni previste all'interno di una formazione.
- `hasTeamFormation` (da `Match` a `Formation`): specifica le formazioni usate dalle squadre in una partita.
- `hasWinner` (da *Match* a *Team*): indica la squadra vincitrice della partita.
- `includes` (da `Match` a `Team`): rappresenta le squadre che partecipano a una partita; ha come sotto-proprietà:
  - `hasHomeTeam`: indica la squadra di casa;
  - `hasAwayTeam`: indica la squadra ospite.
- `involves` (da `Contract` a `Player` o `Coach`): collega un contratto con un giocatore o un allenatore.
- `partecipatesIn` (da `Player` a `Match`): indica le partite a cui ha partecipato un giocatore.
- `scores` (da `Player` a `Goal`): rappresenta i goal segnati da un giocatore.
- `signs` (da `Team` a `Contract`): indica i contratti firmati da una squadra.
- `statsIn` (da `PerformanceStats` a `Match`): collega le statistiche a una specifica partita.

Data Properties:

- `BallPossession` (su `TeamPerformanceStats`): indica la percentuale di possesso palla di una squadra in una partita.
- `BirthDate` (su `Person`): rappresenta la data di nascita di una persona.
- `IsFriendlyMatch` (su `Match`): specifica se una partita è amichevole.
- `MatchAwayTeamScore` (su `Match`): indica il numero di goal segnati dalla squadra ospite.
- `MatchHomeTeamScore` (su `Match`): indica il numero di goal segnati dalla squadra di casa.
- `MatchDate` (su `Match`): rappresenta la data in cui si è svolta la partita.
- `MinuteOfEvent` (su `MatchEvent`): indica il minuto della partita in cui si è verificato un evento.
- `NumberOfDefenders` (su `Formation`): specifica il numero di difensori nella formazione.
- `NumberOfMidfielders` (su `Formation`): specifica il numero di centrocampisti nella formazione.
- `NumberOfStrikers` (su `Formation`): specifica il numero di attaccanti nella formazione.

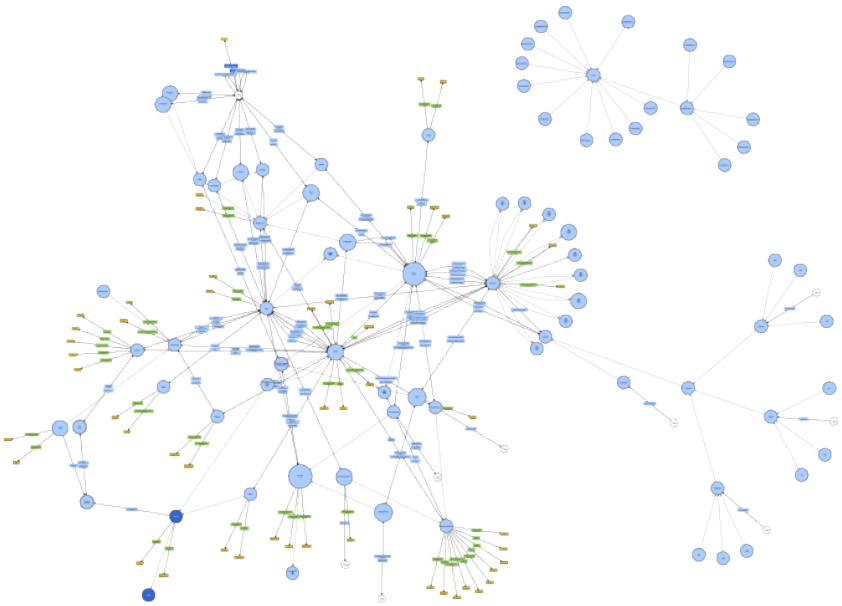

Figura 3.7: Grafico completo dell'ontologia modificata, realizzato con WebVOWL [5, 3].

Capitolo 4

Inserimento di individui

La fase successiva ha riguardato l'inserimento degli individui nelle varie classi. L'obiettivo era quello di aggiungere esempi concreti di istanze all'interno dell'ontologia, così da renderla effettivamente utilizzabile per interrogazioni **SPARQL**.

Creare tutti gli individui per ogni classe e definire tutte le relative relazioni è un'attività molto lunga e ripetitiva. Per questo motivo, ci siamo limitati a creare una sottoparte degli individui, sufficiente per mostrare il funzionamento delle query.

Per velocizzare questa fase, abbiamo utilizzato **Cellfie**, un plugin di **Protegé** che consente di generare individui a partire da un file Excel. Nel nostro caso, ogni riga del file Excel rappresenta un'istanza (individuo) da creare, mentre le colonne contengono le informazioni associate a questo. In particolare:

- la prima colonna contiene il nome dell'individuo;
- le altre colonne rappresentano i valori associati a una o più Object Properties e/o Data Properties

4.1 Esempio inserimento con Cellfie

Di seguito è riportato come abbiamo inserito molti contratti, presenti in un file Excel, attraverso il plugin **Cellfie** e come abbiamo impostato le regole di trasformazione necessarie a trasformare i dati grezzi in individui e relazioni vere e proprie.

Nella Figura 4.1 è mostrata la tabella da inserire, mentre nell'immagine 4.2 si può osservare come sono utilizzati i campi di ogni colonna. In questo caso abbiamo:

- Colonna A: rappresenta il nome dell'individuo
- Colonna B: Data Properties per l'inizio del contratto
- Colonna C: Data Properties per la fine del contratto (se il contratto è ancora attivo allora non è presenta la data di fine contratto.)
- Colonna D: Data Properties per il valore del contratto
- Colonna E: Object Properties della squadra a cui è associato
- Colonna F: Object Properties del giocatore a cui è associato

Foglio1

	A	B	C	D	E	F
1	ContractID	Start Date	End Date	Value	Team	Player
2	Contract1	2018-06-05T00:00:00		3.4	Team1	Player1
3	Contract2	2014-12-30T00:00:00		1.8	Team1	Player2
4	Contract3	2018-12-27T00:00:00		2.7	Team1	Player3
5	Contract4	2021-10-07T00:00:00		4.5	Team1	Player4
6	Contract5	2017-01-09T00:00:00		4.6	Team1	Player5
7	Contract6	2016-04-26T00:00:00		4.8	Team1	Player6
8	Contract7	2015-09-13T00:00:00		2.5	Team1	Player7
9	Contract8	2018-01-21T00:00:00		4.1	Team1	Player8
10	Contract9	2019-10-06T00:00:00		3.9	Team1	Player9
11	Contract10	2014-11-28T00:00:00		3.6	Team1	Player10
12	Contract11	2023-10-28T00:00:00		2.0	Team1	Player11
13	Contract12	2022-01-01T00:00:00		4.3	Team2	Player12
14	Contract13	2015-02-22T00:00:00		2.5	Team2	Player13

Figura 4.1: Foglio Excel dei contratti

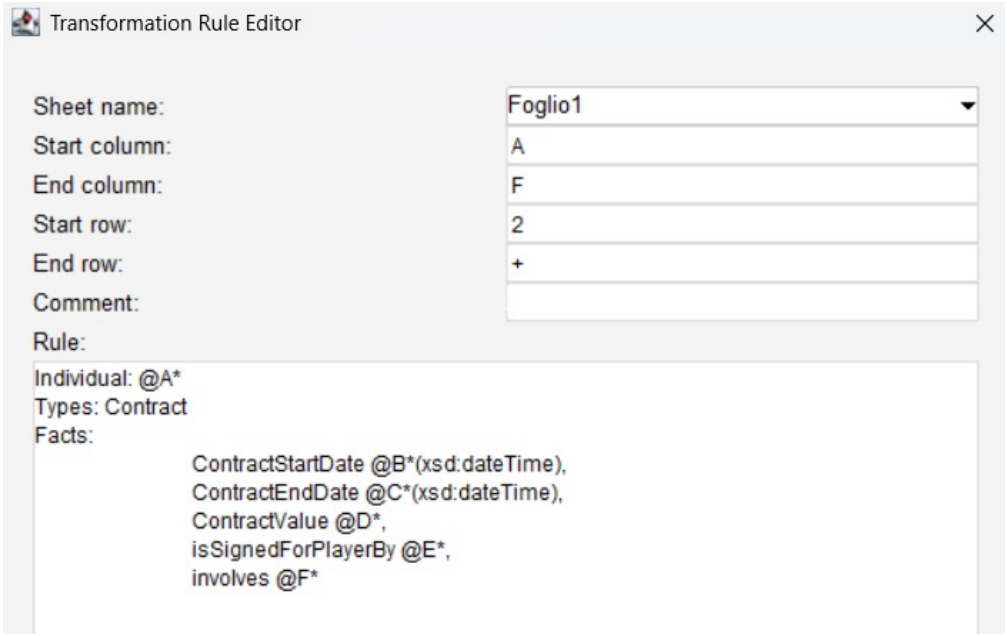

Figura 4.2: Regola di trasformazione per creare i contratti presenti nel foglio Excel

Capitolo 5

Regole SWRL

SWRL (**Semantic Web Rule Language**) è un linguaggio usato nell'ambito del Semantic Web per esprimere regole di inferenza e vincoli logici più complessi di quelli realizzabili usando il solo OWL.

Nella nostra ontologia abbiamo inserito numerose regole, utili a:

1. inferire nuovi collegamenti tra i dati;
2. individuare casi di errore sui dati.

Le sezioni che seguono riportano tali regole, non in un ordine particolare.

5.1 HomeFormationRule

Questa regola "raffina" il collegamento tra una formazione e una partita, esplicitando che la prima è usata dalla squadra in casa. Questo perché `isAwayFormationIn` è una sottoproprietà di `isFormationIn`.

```
isHomeTeam(?t, ?m) ^ hasFormation(?t, ?f) ^ isFormationIn(?f, ?m) ->
    isHomeFormationIn(?f, ?m)
```

5.2 AwayFormationRule

Questa regola funziona in modo analogo alla precedente, ma riguardo la squadra in trasferta.

```
isAwayTeam(?t, ?m) ^ hasFormation(?t, ?f) ^ isFormationIn(?f, ?m) ->
    isAwayFormationIn(?f, ?m)
```

5.3 DuplicateSubstituteRule

Questa regola verifica se un giocatore segnato come riserva partecipa a multiple sostituzioni come giocatore entrante, e in tale caso gli assegna una classe di errore.

```
hasReservePlayer(?f, ?p) ^ hasEnteringPlayer(?s1, ?p) ^
    hasEnteringPlayer(?s2, ?p) ^ differentFrom(?s1, ?s2) ^
    hasTeamFormation(?m, ?f) ^ hasSubstitution(?m, ?s1) ^
    hasSubstitution(?m, ?s2) -> DuplicateSubstituteError(?p)
```

5.4 DuplicateSubstitutedRule

Il funzionamento di questa regola è analogo alla precedente, ma riguardante il giocatore uscente. Questa regola copre però solo i giocatori titolari (ovvero i giocatori che partecipano alla partita dall'inizio).

```
hasStarterPlayer(?f, ?p) ^ hasExitingPlayer(?s1, ?p) ^
    hasExitingPlayer(?s2, ?p) ^ differentFrom(?s1, ?s2) ^
    hasTeamFormation(?m, ?f) ^ hasSubstitution(?m, ?s1) ^
    hasSubstitution(?m, ?s2) -> DuplicateSubstitutedError(?p)
```

5.5 StarterAsSubstituteRule

Questa regola verifica se un giocatore titolare è il giocatore entrante in una sostituzione, e in tale caso gli assegna una classe di errore.

```
hasStarterPlayer(?f, ?p) ^ hasEnteringPlayer(?s, ?p) ^
    hasTeamFormation(?m, ?t) ^ hasSubstitution(?m, ?s) ->
    StarterAsSubstituteError(?p)
```

5.6 FriendlyMatchInTournamentRule

Non è corretto assegnare una partita amichevole ad un torneo, qualsiasi sia il suo tipo, e se ciò viene fatto allora una classe di errore è assegnata alla partita.

```
includedInTournament(?m, ?t) ^ FriendlyMatch(?m) ->
    FriendlyMatchInTournamentError(?m)
```

5.7 GoalScoredByTeamRule

Questa regola collega un goal ad una squadra, sapendo che il goal è stato fatto da un giocatore che giocava in tale squadra nella partita.

```
scores(?p, ?g) ^ hasEvent(?m, ?g) ^ isPlayerInFormation(?p, ?f) ^
    hasFormation(?t, ?f) ^ competesIn(?t, ?m) -> scoredByTeam(?g, ?t)
```

5.8 InconsistentBallPossessionInMatchRule

Questa regola controlla se la somma del possesso palla indicata nelle statistiche di performance per le squadre che hanno giocato una partita non è 100%, e in tale caso assegna una classe di errore alla partita e alle statistiche.

```
teamStatsIn(?ps1, ?m) ^ teamStatsIn(?ps2, ?m) ^ differentFrom(?ps1,
    ?ps2) ^ BallPossession(?ps1, ?p1) ^ BallPossession(?ps2, ?p2) ^
    swrlb:add(?r, ?p1, ?p2) ^ swrlb:notEqual(?r, 100) ->
    InconsistentBallPossessionInMatchError(?m) ^
    InconsistentBallPossessionInMatchError(?ps1) ^
    InconsistentBallPossessionInMatchError(?ps2)
```

5.9 InconsistentContractDatesRule

Questa regola verifica se le date di un contratto sono inconsistenti (data di inizio futura alla data di fine).

```
ContractStartDate(?c, ?sd) ^ ContractEndDate(?c, ?ed) ^
    temporal:before(?ed, ?sd) -> InconsistentContractDatesError(?c)
```

5.10 InconsistentMatchScoresHome/AwayRule

Queste due regole verificano se il vincitore di una partita non è consistente con i punteggi assegnati alle due squadre. Sono state necessarie due regole in quanto i punteggi delle squadre sono legati alla partita e non direttamente alle squadre.

```
isHomeTeam(?t1, ?m) ^ isAwayTeam(?t2, ?m) ^ hasWinner(?m, ?t1) ^
    MatchHomeTeamScore(?m, ?s1) ^ MatchAwayTeamScore(?m, ?s2) ^
    swrlb:lessThanOrEqual(?s1, ?s2) -> InconsistentMatchScoresError(?m)
```

```
isHomeTeam(?t1, ?m) ^ isAwayTeam(?t2, ?m) ^ hasWinner(?m, ?t2) ^
    MatchHomeTeamScore(?m, ?s1) ^ MatchAwayTeamScore(?m, ?s2) ^
    swrlb:lessThanOrEqual(?s2, ?s1) -> InconsistentMatchScoresError(?m)
```

5.11 InvalidLeagueMatchFormatExtraTime/PenaltyRule

Le partite che fanno parte di una lega non possono avere supplementari o rigori, e queste due regole servono a identificare tali inconsistenze.

```
includedInTournament(?m, ?t) ^ League(?t) ^ ExtraTimePlayed(?m, true) ->
    InvalidLeagueMatchFormatError(?m)
```

```
includedInTournament(?m, ?t) ^ League(?t) ^ PenaltyShootoutPlayed(?m,
    true) -> InvalidLeagueMatchFormatError(?m)
```

5.12 InvalidMatchRule

Questa regola verifica se una squadra sta giocando contro sé stessa in una partita.

```
hasHomeTeam(?m, ?t) ^ hasAwayTeam(?m, ?t) -> InvalidMatchError(?m)
```

5.13 MultipleRedCardsRule

Questa regola controlla se un giocatore ha ricevuto multipli cartellini rossi in una partita.

```
hasReceived(?p, ?c1) ^ hasReceived(?p, ?c2) ^ RedCard(?c1) ^
    RedCard(?c2) ^ differentFrom(?c1, ?c2) ^ hasEvent(?m, ?c1) ^
    hasEvent(?m, ?c2) -> MultipleRedCardsError(?p)
```

5.14 NonGoalKeeperWithSavesRule

Questa regola verifica se le statistiche di un giocatore indicano il numero di parate sebbene esso non abbia giocato come portiere nella partita.

```
playsInPosition(?pl, ?p) ^ Outfield(?p) ^ hasPlayerStats(?pl, ?ps) ^
    Saves(?ps, ?s) ^ hasPlayer(?m, ?p) ^ hasPlayerPerformanceStats(?m,
    ?ps) -> NonGoalKeeperWithSavesError(?pl)
```

5.15 PlayerInMatchRule

Questa regola collega un giocatore ad una partita, sapendo che il giocatore era in una formazione utilizzata in tale partita.

```
hasPlayerInFormation(?f, ?p) ^ isFormationIn(?f, ?m) -> hasPlayer(?m, ?p)
```

5.16 TeamNotInMatchRule

Questa regola controlla se un evento in una partita è legato ad una squadra che non gioca in tale partita.

```
hasEvent(?m, ?e) ^ hasMatchEvent(?t, ?e) ^ isHomeTeam(?t1, ?m) ^
    isAwayTeam(?t2, ?m) ^ differentFrom(?t, ?t1) ^ differentFrom(?t, ?t2)
    -> TeamNotInMatchError(?t)
```

5.17 UncontractedPlayerInMatchRule

Questa regola verifica se un giocatore ha partecipato ad una partita senza far parte di una delle due squadre coinvolte, in quanto non si trovava sotto contratto con una delle due squadre al momento della partita. La regola copre soltanto i casi in cui il giocatore si trovava sotto contratto con un'altra squadra e se il contratto è terminato.

```
signs(?t1, ?c) ^ involvesPlayer(?c, ?p) ^ participatesIn(?p, ?m) ^
    includes(?m, ?t2) ^ includes(?m, ?t3) ^ differentFrom(?t1, ?t2) ^
    differentFrom(?t1, ?t3) ^ ContractStartDate(?c, ?sd) ^
    ContractEndDate(?c, ?ed) ^ MatchDate(?m, ?d) ^ temporal:before(?sd,
    ?d) ^ temporal:after(?ed, ?d) -> UncontractedPlayerInMatchError(?p)
```

Capitolo 6

Interrogazioni SPARQL

L'ultima parte del progetto è stata la realizzazione di interrogazioni usando **SPARQL** (**SPARQL Protocol and RDF Query Language**), un linguaggio simile a SQL per poter estrarre dati dall'ontologia.

Le interrogazioni che abbiamo scritto sono esempi che producono risultati sulla base degli individui d'esempio che abbiamo predisposto. Per realizzarle abbiamo usato le viste **SPARQL Query** e **Snap SPARQL Query** di **Protegé**.

6.1 Interrogazione sui contratti di un giocatore

Questa interrogazione recupera i contratti firmati da un giocatore (nell'esempio `Player1`) e i nomi dei team per cui i contratti sono stati firmati.

```
PREFIX : <http://visualdataweb.org/FootOntologyPlus/>

SELECT ?contract ?teamName
WHERE {
    ?team :signsPlayer ?contract ;
          :TeamName ?teamName .
    ?contract :involvesPlayer :Player1 .
}
```

?contract	?teamName
:Contract1	Milan^^xsd:string
:Contract61	Milan^^xsd:string

Figura 6.1: Risultato della query sulla nostra ontologia.

6.2 Interrogazione sui contratti non terminati

Questa interrogazione recupera tutti i contratti che non hanno una data di terminazione. Si assume in questo caso che tali contratti siano ancora attivi, ma in generale vige la "Open World assumption" riguardo alle informazioni mancanti.

```
PREFIX : <http://visualdataweb.org/FootOntologyPlus/>

SELECT ?player ?team ?contract ?startDate
WHERE {
    ?contract :isSignedForPlayerBy ?team ;
        :involvesPlayer ?player ;
        :ContractStartDate ?startDate .
    FILTER NOT EXISTS { ?contract :ContractEndDate ?endDate }
}
ORDER BY ASC (?player)
```

player	team	contract	startDate
Player1	Team1	Contract1	"2018-06-05T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player10	Team1	Contract10	"2014-11-28T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player11	Team1	Contract11	"2023-10-28T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player12	Team2	Contract12	"2022-01-01T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player13	Team2	Contract13	"2015-03-22T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player14	Team2	Contract14	"2015-04-25T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player15	Team2	Contract15	"2022-11-26T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player16	Team2	Contract16	"2015-06-28T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player17	Team2	Contract17	"2016-05-30T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player18	Team2	Contract18	"2019-05-15T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player19	Team2	Contract19	"2014-11-17T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player2	Team1	Contract2	"2014-12-30T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player20	Team2	Contract20	"2015-11-25T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player21	Team2	Contract21	"2015-03-03T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player22	Team2	Contract22	"2017-04-15T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
Player23	Team3	Contract23	"2017-08-22T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>

Figura 6.2: Parte del risultato della query sulla nostra ontologia.

6.3 Interrogazione sui cartellini ricevuti dai giocatori

Questa interrogazione recupera le informazioni riguardanti i cartellini ricevuti dai giocatori, nello specifico il tipo di cartellino, la partita e il minuto di gioco in cui è stato ricevuto.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://visualdataweb.org/FootOntologyPlus/>

SELECT ?player ?cardType ?match ?minute
WHERE {
    ?player :hasReceived ?card .
    ?card :MinuteOfEvent ?minute ;
        :isEventIn ?match ;
        rdf:type ?cardType .
    ?cardType rdfs:subClassOf :Card .
    FILTER (?cardType != :Card) .
}
ORDER BY ASC (?player)
```

?player	?cardType	?match	?minute
:Player17	:RedCard	:Match1	53
:Player5	:YellowCard	:Match1	23

Figura 6.3: Risultato della query sulla nostra ontologia.

6.4 Interrogazione sulle sostituzioni effettuate in una partita

Questa interrogazione le informazioni riguardanti le sostituzioni effettuate in una partita (nell'esempio `Match1`), nello specifico il nome dei giocatori coinvolti e il minuto di gioco.

```
PREFIX : <http://visualdataweb.org/FootOntologyPlus/>

SELECT ?enteringName ?exitingName ?minute
WHERE {
    ?sub :isSubstitutionIn :Match1 ;
        :hasEnteringPlayer ?entering ;
        :hasExitingPlayer ?exiting ;
        :MinuteOfEvent ?minute .
    ?entering :FullName ?enteringName .
    ?exiting :FullName ?exitingName .
}
ORDER BY ASC (?minute)
```

?enteringName	?exitingName	?minute
Kirill Volkov^^xsd:string	Mohamed Khalifa^^xsd:string	55
Thiago Oliveira^^xsd:string	Jean Dubois^^xsd:string	75

Figura 6.4: Risultato della query sulla nostra ontologia.

6.5 Interrogazione sui goal segnati dai giocatori

Questa interrogazione recupera il numero di goal segnati da ogni giocatore in un torneo (nell'esemptio `League1`).

```
PREFIX : <http://visualdataweb.org/FootOntologyPlus/>

SELECT ?player ?playerName (COUNT(?goal) as ?totalGoals)
WHERE {
    ?match :includedInTournament :League1 ;
        :hasGoal ?goal .
    ?goal :scoredByPlayer ?player .
    ?player :FullName ?playerName
}
GROUP BY ?player ?playerName
ORDER BY DESC (?totalGoals)
```

?player	?playerName	?totalGoals
:Player14	David Brown^^xsd:string	1
:Player19	Jan Kowalski^^xsd:string	1
:Player7	João Silva^^xsd:string	1

Figura 6.5: Risultato della query sulla nostra ontologia. Il nome dell'individuo giocatore è incluso in caso di omonimi.

Capitolo 7

Confronto metriche

Di seguito un confronto delle metriche dell'ontologia di partenza "Footology" e quella finale "FootOntologyPlus".

Metrica	**Footology**	**FootOntologyPlus**
Axioms	496	2.160
Logical axiom count	180	1.530
Declaration axioms count	88	437
Class count	13	81
Object property count	26	123
Data property count	46	51
Individual count	0	180
Annotation Property count	5	7

Tabella 7.1: Confronto metriche

Osservando la tabella si può vedere il lavoro di ristrutturazione e sopratutto di estensione dell'ontologia di partenza.

Capitolo 8

Conclusioni

In conclusione, il nostro progetto ci ha permesso di lavorare concretamente su una vera e propria ontologia, mettendo in pratica i concetti appresi durante il corso. L'obiettivo principale è stato quello di rendere l'ontologia il più possibile completa, coerente e adatta a rappresentare in modo accurato il dominio del calcio.

L'ontologia di partenza era molto limitata e incompleta, il che ci ha dato l'opportunità di intervenire in modo significativo con numerose modifiche e aggiunte. Come mostrano le metriche del capitolo precedenti, il numero di classi è passato da 13 a 81, le Object Properties da 26 a 123, etc... questo ci dice quanto è stata ampliata l'ontologia.

Questo progetto ci ha inoltre permesso di comprendere l'importanza delle ontologie come strumenti fondamentali per strutturare e interrogare grandi quantità di dati in modo efficiente, verificabile e interoperabile.

Sebbene l'espansione realizzata sia focalizzata su uno scenario accademico, il lavoro svolto costituisce una base solida per possibili estensioni future e per l'utilizzo dell'ontologia in applicazioni reali, come sistemi di analisi calcistica, motori di raccomandazione, o strumenti per la gestione e validazione di dataset sportivi complessi.

Bibliografia

[1] *DBpedia*. URL: https://www.dbpedia.org/.

[2] *Draw.io*. URL: https://app.diagrams.net/.

[3] *Footology*. licensed under CC BY 4.0. URL: https://github.com/arditb1997/footology.

[4] *Schema.org/Person*. URL: https://schema.org/Person.

[5] *WebVOWL*. URL: https://service.tib.eu/webvowl/.