Football Team Ontology Project

Leveraging Semantic Web Technologies for Football Data Management

Team Members:

- Amar Yasser 21-00479
- Kareem Ahmed 21-00839
- Ahmed Esam 21-00328
- Ahmed Ehab 21-00345
- Ahmed Seleem 21-00325
- Mohamed Abd Alazeeem 21-01149
- Abdelrahman Helel 21-00303

Introduction - The Vision

What is this project about?

- Java-based application to load, explore, and analyze a football team ontology
- Goal: Model football team domain using semantic web technologies
- Focus on players, coaches, matches, and relationships
- Machine-readable, flexible data representation supporting complex queries

Core Technology:

- OWL (Web Ontology Language)
- OWL API

Core Problem & Solution

Why is this project needed?

- Traditional relational databases are cumbersome for complex football data relationships
- Ontology-based approach offers greater flexibility
- Semantic web technologies enable:
  - Inference of new knowledge
  - Complex querying capabilities
  - More effective data integration

Benefits:

- Enhanced data integration across multiple sources
- Improved querying capabilities
- Knowledge inference for discovering implicit relationships

The Football Ontology (RES.owl) - Key Concepts

Building Blocks of Our Football World

Key Classes:

- FootballTeam: Represents a football club or team
- Player: Represents an individual player
- Coach: Represents the team coach
- Stadium: Represents match venues
- Position: Represents player positions (Forward, Defender, etc.)
- Match: Represents a football game
- Referee: Represents match officials

```xml
<owl:Class rdf:about="#FootballTeam" />
<owl:Class rdf:about="#Player" />
```

Defining Relationships (Object Properties)

How Entities are Connected

Examples of Object Properties:

- hasPlayer (Domain: FootballTeam, Range: Player)
- hasCoach (Domain: FootballTeam, Range: Coach)
- playsIn (Domain: FootballTeam, Range: Stadium)
- hasPosition (Domain: Player, Range: Position)

```xml
<owl:ObjectProperty rdf:about="#hasPlayer">
      <rdfs:domain rdf:resource="#FootballTeam" />
      <rdfs:range rdf:resource="#Player" />
</owl:ObjectProperty>
```

Describing Entities (Data Properties) & Individuals

Attributes and Specific Instances

Data Properties Examples:

- teamName (for FootballTeam)
- playerName (for Player)
- playerAge (for Player)

Individuals:

- Specific instances of classes (e.g., "LiverpoolFC" or "Mohamed Salah")

```xml
<owl:NamedIndividual rdf:about="#MohamedSalah">
      <rdf:type rdf:resource="#Player" />
      <playerName rdf:datatype="xsd:string">Mohamed Salah</playerName>
      <playerAge rdf:datatype="xsd:integer">31</playerAge>
      <hasPosition rdf:resource="#Forward" />
</owl:NamedIndividual>
```

The Java Application - Bringing the Ontology to Life

How the Program Interacts with the Ontology

Features:

- Loads the RES.owl ontology using OWL API
- Displays and manipulates ontology components
- Performs logical consistency checks using OWL reasoners

```java
// Load Ontology
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
OWLOntology ontology = manager.loadOntologyFromOntologyDocument(inputStream);
System.out.println("Ontology Loaded: " + ontology.getOntologyID());

// Display Classes
System.out.println("\nClasses in the ontology:");
for (OWLClass cls : ontology.getClassesInSignature()) {
    System.out.println(" * " + cls.getIRI().getShortForm());
}
```

Ensuring Quality - Ontology Consistency & Reasoning

Making Sure Our Model is Sound

Key Concepts:

- Ontology Consistency: Ensures no logical errors
- Reasoning: Using OWL reasoners to infer new facts and detect inconsistencies

Example: A football team must have at least one player, and a player must have exactly one position.

```java
// Consistency Check
OWLReasonerFactory reasonerFactory = StructuralReasonerFactory();
OWLReasoner reasoner = reasonerFactory.createReasoner(ontology);
System.out.println("\nIs ontology consistent? " + reasoner.isConsistent());
```

Ontology Development & Interaction with Protégé

Authoring, Visualizing, and Querying the Ontology

Features of Protégé:

- Rule Creation: Using Semantic Web Rule Language (SWRL)
- Ontology Visualization: OntoGraf and VOWL
- SPARQL Querying: Test and validate ontology axioms

Example Query: "Find all players in 'LiverpoolFC' who are 'Forwards'."

Project Setup - Maven Configuration

Maven POM File Structure:

- Project identification: `com.example.football_team`
- Java 17 environment
- Dependencies: OWL API Distribution (v5.1.19)

xml

```xml
<dependencies>
    <dependency>
        <groupId>net.sourceforge.owlapi</groupId>
        <artifactId>owlapi-distribution</artifactId>
        <version>5.1.19</version>
    </dependency>
</dependencies>
```

Build Configuration:

- Exec Maven Plugin for executing the application
- Main class: `com.example.football_team.Main`
- Simplifies building and running the application with Maven commands

---

Future Work

Enhancing the Ontology and Application

Expanding the Ontology:

- Add more classes and properties (tournaments, transfers, fan engagement)
- Incorporate detailed player statistics

Advanced Reasoning:

- Complex rules for additional inferences
- Machine learning integration for match prediction

Integration with External Data:

- Real-time data sources
- External football APIs

User Interface Development:

- Web or mobile interface for ontology interaction