Universite de Montpellier

Rapport de projet
Ingénierie des Connaissances

Belkassim BOUZIDI
Chakib ELHOUITI
Massili KEZZOUL

22 mars 2021

1 Introduction

Dans le cadre de ce projet, il est demandé, pour un domaine applicatif, de proposer une illustration de mise en œuvre des techniques et technologies de l'Ingénierie des Connaissances.

2 Définition des objectifs d'application

2.1 Contexte métier et objectifs

Le domaine applicatif choisi est le pari sportif. L'objectif est de définir une base de connaissance qui permettra de classer des matches de football en plusieurs niveaux de risques d'un point de vu d'un parieur. Afin de simplifier la modélisation et l'implémentation pour une première version, nous ne considérons ici que la Premier League, le championnat d'Angleterre de football.

À partir de cette base de connaissance, le but est d'exécuter des requêtes SPARQL et/ou DL afin de trouver les équipes pour lesquelles parier avec un risque qui doit être le plus faible possible.

Selon la précision qui découlera de l'implémentation, ce projet peut-être utilisé pour de l'aide à la décision pour de parieurs professionnels ou de simples recommandations pour des parieurs amateurs.

2.2 Quelques scénarios d'utilisation

Un scénario type d'utilisation est de demander, via une requête SPARQL par exemple, de trouver une relation entre une équipe et un match à venir, tel que cette équipe à de fortes chances de gagner.

Cette relation représentera donc, d'une façon, la probabilité que cette équipe gagne ce match. Cela se fera en définissant cette relation comme étant une conjonction de certains critères qui favorisent la victoire d'une équipe à un match. Voici une liste non-exhaustive de ces critères :

— Cette équipe joue à domicile ;
— Elle a une meilleur forme que son adversaire ;
— Elle a un meilleur classement que son adversaire dans la ligue ;
— Et bien d'autres ...

3 La mise en œuvre

Après avoir défini le contexte métier ainsi que les objectifs voulus, nous avons commencé à mètre en œuvre notre ontologie.

3.1 Modélisation

La première étape consiste en la définition de la composante terminologique (la TBOX).

3.1.1 Les classes et propriétés de base

Team On a commencé par la plus évidente, une équipe de football.

Propriètes de la classe **Team**				
Détail	Nom de la propriétés	Domaine	Co-domaine	Notes
le nom d'une équipe	rdfs :label	X	X	
Joue un match	PlayedAGame	Team	Game	
A un entraîneur	ManagedBy	Team	Manager	
Classement Actuel	rank	Team	Manager	
Nombre de victoire	nWin	Team	int	
Nombre de match Nul	nDraw	Team	int	
Nombre de défaite	nLose	Team	int	
La forme Actuel	currentFrom	Team	int	La forme est représentée par la somme des point acquis lors des 5 derniers matchs.
Joue ce match à domicile	playGameHome	Team	Game	
Joue ce match à l'extérieur	playGameAway	Team	Game	
A gagner ce match	winner	Team	GamePlayed	
A perdu ce match	looser	Team	GamePlayed	
A joué contre cette équipe	playedAgainst	Team	Team	

Manager Chaque équipe a un entraîneur. Le but de cette classe est de pouvoir définir quelques critères (Voir Quelques scénarios d'utilisation) associés à la forme de l'entraîneur et son historique avec l'équipe qu'il va affronter.

Propriètes de la classe **Manager**				
Détail	Nom de la propriétés	Domaine	Co-domaine	Notes
Nom de l'entraîneur	rdfs :label	X	X	
Entraîne une équipe	Manage	Manager	Team	

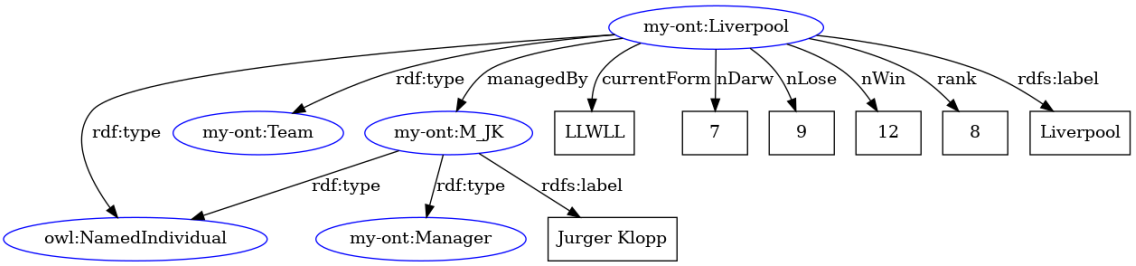

Schéma 1 – Représenation d'une équipe et son entraîneur

Referee Chaque match de football se dispute sous le contrôle d'un arbitre central, disposant de toute l'autorité nécessaire pour veiller à l'application des Lois du Jeu dans le cadre du match qu'il est appelé à diriger. Bien qu'il est tenu d'être le plus neutre possible, ils influent souvent sur la stratégie choisie par les entraîneurs.

Propriètes de la classe **Referee**				
Détail	Nom de la propriétés	Domaine	Co-domaine	Notes
Nom	rdfs :label	X	X	
Arbitre un match	referee	Referee	Game	contraire de redereedBy
Nombre de HomeWIn	homeWin	Referee	int	Nombre de match que l'arbitre a fait gagner à l'équipe jouant à domicile
Nombre de AwayWin	awayWin	Referee	int	Nombre de match que l'arbitre a fait gagner à l'équipe jouant à l'extérieur
Nombre de matchs nuls	refereeDraw	Referee	int	

NB : Les propriétés *homeWin*, *awayWin* et *refereeDraw* sont utiles pour voir si un arbitre est plutôt avantageux pour les équipes jouant à domicile ou à l'extérieur.

Game On arrive ici au cœur du problème. Il s'agit de définir un match et ses propriétés les plus utiles pour atteindre nos objectifs. On a décidé qu'il fallait différencier les matchs qui se sont déjà joués des matches qui arrivent. La classe *Game* a donc deux sous-classes : *GamePlayed* et *GameUpComing*. Les propriétés associées à ces classes sont les suivantes :

Propriètes de la classe **Game** et ses sous-classes				
Détail	Nom de la propriétés	Domaine	Co-domaine	Notes
Date du match	dateGame	Game	string	...
L'équipe qui joue à domicile	homeTeam	Game	Team	
L'équipe qui joue à l'extérieur	awayTeam	Game	Team	
Nombre de but domicile	fullTimeHomeGoal	GamePlayed	int	...
Nombre de but extérieur	fullTimeAwayGoal	GamePlayed	int	...
L'arbitre du match	refereedBy	Game	Referee	Contraire de referee
Vainqueur	winner	GamePlayed	Team	Le vainqueur d'un match
Perdant	looser	GamePlayed	Team	Le perdant d'un match

3.1.2 Les propriétés inférées

Certaines des ces propriétés ne sont pas à définir explicitement dans la **ABOX**, car elles seront inférées automatiquement par le raisonneur de Protégé ou déduites par des requêtes SPARQL.

manager et managedBy *manager* et *managedBy* sont un exemple des propriétés inférées par le raisonneur. En effet, il suffit d'ajouter à la **ABOX** que M *manage* T alors il déduira automatiquement que T *managedBy* M.

winner et looser Le cas de *winner* et *looser* est plus particulier. Pour qu'une équipe soit la vainqueur d'un match alors elle doit participer à ce match et marquer plus de buts que son adversaire. Donc on peut l'écrire de cette façon :

$$winner(g,t) \equiv \exists a, \exists h fullTimeHomeGoal(g,h) \ \wedge \ fullTimeAwayGoal(g,a) \wedge (homeTeam(g,t) \ \wedge \ h > a) \vee (awayTeam(g,t) \ \wedge \ a > h)$$

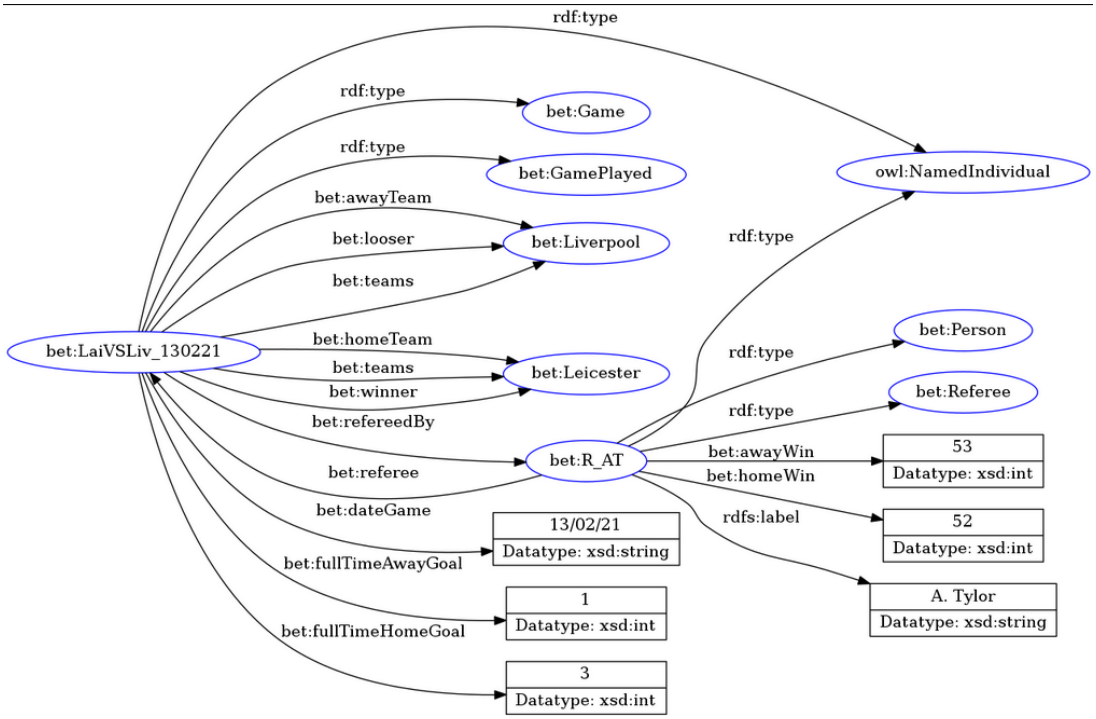

Schéma 2 – Représenation d'un match et son arbitre

Mais ce genre d'équivalence ne peut être déduite par le raisonneur de Protégé. Pour remédier à cela nous avons écrit une requête SPARQL pour déduire automatiquement ces relations.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bet: <http://www.semanticweb.org/massy/ontologies/2021/2/untitled-ontology-5#>

# This SPARQL query add winner, looser propreties to the KB

CONSTRUCT {
    ?game bet:winner ?winner.
    ?game bet:looser ?looser.
}
WHERE {
    ?winner rdf:type bet:Team.
    ?looser rdf:type bet:Team.
    ?game rdf:type bet:GamePlayed.
    {
        ?game bet:fullTimeHomeGoal ?h.
        ?game bet:fullTimeAwayGoal ?a.
        ?game bet:homeTeam ?winner.
        ?game bet:awayTeam ?looser.
        FILTER(?h > ?a).
    }
    UNION
    {
        ?game bet:fullTimeHomeGoal ?h.
        ?game bet:fullTimeAwayGoal ?a.
        ?game bet:awayTeam ?winner.
        ?game bet:homeTeam ?looser.
        FILTER(?h < ?a).
    }
}
```

Listing 1 – Requête qui déduit les propriétés *winner* et *looser*

3.1.3 Les contraintes favorisant la victoire

L'idée générale est de déduire des relations, entre une équipe et un match à venir, qui indiques que cette équipe a des chances de gagner ce match. Pour cela, on définit les propriétés suivantes :

Détail	Nom de la propriétés	Domaine	Co-domaine
l'équipe a de forte chances de gagner ce match	highProb	Team	GameUpComing
l'équipe a des chances de gagner ce match	medProb	Team	GameUpComing
l'équipe a peu de chances de gagner ce match	lowProb	Team	GameUpComing

Avant de définir concrétement ces propriétés, on doit d'abord définir les facteurs qui favorisent la victoire d'une équipe.

Contraintes de base Elles prendront la forme d'une propriété dont le domaine est *Team*.

Détail	Nom de la propriétés	Domaine	Co-domaine
L'équipe joue à domicile	playGameHome	Team	Game
L'équipe a un meilleur classement que son adversaire	bestRankThan	Team	Team
Le classement de l'équipe est largement mieux que son adversaire	best10RankThan	Team	Team
l'équipe a une meilleur forme que son adversaire	bestFormThan	Team	Team

Généralisation des contraintes Une fois les contraintes de base définit et afin de simplifier la définition des relations finales, nous passons par cette étape de généralisation des contraintes de base. Cela consiste en la transformation des propriétés précédemment définies en propriétés qui ont toutes comme domaine *Team* et comme co-domaine *GameUpComing*.

Par exemple, la propriété *Team bestRankThan Team* va devenir *Team c2 GameUpComing*. La sémantique associée à *c2* sera : «une équipe a un meilleur classement que l'équipe contre laquelle elle va jouer à ce match.»

```
CONSTRUCT {
    ?team bet:c2 ?game.
}
WHERE {
    ?team rdf:type bet:Team.
    ?t rdf:type bet:Team.
    ?game rdf:type bet:GameUpComing.

    ?team bet:bestRankThan ?t.

    ?game bet:teams ?t.
    ?game bet:teams ?team.
    FILTER(?t != ?team)
}
```

Listing 2 – Requête qui transforme *bestRankThan* en *c2*

On aura donc ce qui suit :

c1 associée a *playGameHome*

c2 associée a *bestRankThan*

c3 associée a *best10RankThan*

c4 associée a *bestFormThan*

Conjonctions des contraintes Enfin, les propriétés finales ne seront que des conjonctions des contraintes généralisées.

highProb est définie comme étant la conjonction de toutes les contraintes.

$$highProb(t, g) \equiv c_1(t, g) \wedge c_2(t, g) \wedge c_3(t, g) \wedge c_4(t, g)$$

medProb est définie comme étant la conjonction de 2 ou 3 contraintes.

lowProb est définie comme étant le conjonction d'une seule contrainte.

3.2 Constitution de la ABOX

Une fois la **TBOX** définie et construite, nous devons récupérer les assertions qui constitueront les faits de notre **ABOX**. Pour cela, nous avons extrait, via un script python [1], les matches qui se sont déroulés à ce jour [2] ainsi que le classement actuel [3] de la *Premier league*.

```
,Unnamed: 0,rank,team,played,win,draw,loose,currentForm
0,0,1,Man City,30,22,5,3,12
1,1,2,Man United,29,16,9,4,11
2,2,3,Leicester,29,17,5,7,10
3,3,4,Chelsea,29,14,9,6,9
4,4,5,West Ham,28,14,6,8,9
5,5,6,Liverpool,29,13,7,9,6
6,6,7,Everton,28,14,4,10,9
7,7,8,Tottenham,28,13,6,9,9
8,8,9,Aston Villa,27,12,5,10,5
9,9,10,Arsenal,28,12,5,11,10
10,10,11,Crystal Palace,29,10,7,12,8
11,11,12,Leeds,28,11,3,14,4
12,12,13,Wolves,29,9,8,12,5
13,13,14,Southampton,29,9,6,14,3
14,14,15,Burnley,29,8,9,12,6
15,15,16,Brighton,28,6,11,11,4
16,16,17,Newcastle,28,7,7,14,3
17,17,18,Fulham,29,5,11,13,7
18,18,19,West Brom,29,3,9,17,5
19,19,20,Sheffield United,29,4,2,23,3
```

Schéma 3 – Classement actuel des équipes

```
,Date,HomeTeam,AwayTeam,FTHG,FTAG,Referee
0,12/09/2020,Fulham,Arsenal,0,3,C Kavanagh
1,12/09/2020,Crystal Palace,Southampton,1,0,J Moss
2,12/09/2020,Liverpool,Leeds,4,3,M Oliver
3,12/09/2020,West Ham,Newcastle,0,2,S Attwell
4,13/09/2020,West Brom,Leicester,0,3,A Taylor
5,13/09/2020,Tottenham,Everton,0,1,M Atkinson
6,14/09/2020,Brighton,Chelsea,1,3,C Pawson
7,14/09/2020,Sheffield United,Wolves,0,2,M Dean
8,19/09/2020,Everton,West Brom,5,2,M Dean
```

Schéma 4 – Quelques matches joués

À ce stade, nous avons donc toutes les informations dont on a besoin sous un format CSV. On a ensuite définis des scripts python, à l'aide de la librairie RDFLib, afin de générer les faits constituant la **ABOX**.

Il faut noter que, à ce stade, les faits ne contiennent que certaines propriétés de base. Le reste sera inférées par un raisonneur et des requêtes SPARQL.

1. Vous trouverez le script ici : update.py
2. https://www.football-data.co.uk/mmz4281/2021/E0.csv
3. https://www.msn.com/en-au/sport/football/epl/ladder

3.3 Autres composants

Avant de pouvoir générer les propriétés finales (ex : *highProb*), nous devons d'abord construire quelques propriétés intermédiaires. Pour cela, nous faisons appel à plusieurs programmes.

3.3.1 Raisonneur

Le premier est un raisonneur qui permet de déduire les propriétés à partir de celles disponible dans la ABOX. Nous avons utilisé le raisonneur fourni avec l'API Java de JENA [4].

Le programme prend en paramètre deux fichiers OWL, le schéma qui correspond à la **TBOX** ainsi que les données qui forme la **ABOX**, et retourne un fichier OWL qui contient le tout avec les propriétés déduites.

3.3.2 Requêtes SPARQL

Comme il est mentionné plus tôt, certaines propriétés ne peuvent être déduites par un raisonneur (Voir winner et looser). Il est donc indispensable de les générer d'une autre manière. Pour cela, nous avons écrit des requêtes **SPARQL** [5] qui permet à partir des propriétés de base de générer toutes les autres (y compris les requêtes représentant les contraintes).

Ensuite, un script Python englobe le tout. Il exécute le raisonneur, charge le résultat et exécute dessus les requêtes **SPARQL**.

Enfin, nous avons défini les propriétés finales en **SPARQL**. Il suffit donc d'exécuter ces requêtes pour avoir le résultat.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX bet: <http://www.semanticweb.org/massy/ontologies/2021/2/untitled-ontology-5#>


SELECT ?team ?game
WHERE{
    ?team rdf:type bet:Team.
    ?game rdf:type bet:GameUpComing.

    ?team bet:c1 ?game.
    ?team bet:c2 ?game.
    ?team bet:c3 ?game.
    ?team bet:c4 ?game.
}
```

Listing 3 – Requête qui retourne les équipes qui ont une forte chance de gagner un match

4 Illustration d'utilisation

Charger les données On commence par charger les données en CSV et les transformer en RDF. Cela est fait en exécutant le script abox.py. Le résultat est le fichier OWL data.owl.

Inférer les propriétés Ensuite, on exécute le raisonneur et les requêtes **SPARQL**. Cela est fait en exécutant le script main.py. Le résultat est le fichier OWL final.owl.

4. Le code est disponble ici : JenaApi.java
5. Disponible ici sparql-construct/ et là sparql-generalization/

```
executing ../ontologies/sparql-construct/winner-looser.sql
438  element added to the ontologie.
executing ../ontologies/sparql-construct/winAgainst.sql
374  element added to the ontologie.
executing ../ontologies/sparql-construct/drawAgainst.sql
122  element added to the ontologie.
executing ../ontologies/sparql-construct/playGameHome.sql
10  element added to the ontologie.
executing ../ontologies/sparql-construct/bestRankThan.sql
190  element added to the ontologie.
executing ../ontologies/sparql-construct/best10RankThan.sql
45  element added to the ontologie.
executing ../ontologies/sparql-construct/betterFormThan.sql
175  element added to the ontologie.
executing ../ontologies/sparql-generalization/c1.sql
10  element added to the ontologie.
```

Schéma 5 – Affichage du script *main.py*

Exécuer les requêtes finales Enfin, on exécute le dernier script (query.py) pour exécuter les requêtes finales. Ce qui nous donne ce qui suit :

```
Executing: ../ontologies/sparql-final/highProb.sql
2 element found
Executing: ../ontologies/sparql-final/medProb.sql
8 element found
Executing: ../ontologies/sparql-final/lowProb.sql
10 element found
Result for High probabilities :
     Man United -> ManUnitedVSBrighton04042021
     Chelsea -> ChelseaVSWestBrom03042021
Result for Medium probabilities :
     Leeds -> LeedsVSSheffieldUnited03042021
     Southampton -> SouthamptonVSBurnley04042021
     Arsenal -> ArsenalVSLiverpool04042021
     Aston Villa -> AstonVillaVSFulham03042021
     Man City -> LeicesterVSManCity03042021
     Everton -> EvertonVSCrystalPalace05042021
     Tottenham -> NewcastleVSTottenham04042021
     West Ham -> WolvesVSWestHam05042021
Result for Low probabilities :
     Sheffield United -> LeedsVSSheffieldUnited03042021
     Burnley -> SouthamptonVSBurnley04042021
     Brighton -> ManUnitedVSBrighton04042021
     West Brom -> ChelseaVSWestBrom03042021
     Liverpool -> ArsenalVSLiverpool04042021
     Fulham -> AstonVillaVSFulham03042021
     Leicester -> LeicesterVSManCity03042021
     Crystal Palace -> EvertonVSCrystalPalace05042021
     Newcastle -> NewcastleVSTottenham04042021
     Wolves -> WolvesVSWestHam05042021
```

Schéma 6 – Affichage du script *query.py*

On voit ici que *Chelsea* a de très forte chance de gagner son match contre *West Brom*. Ce qui s'explique par le fait que *Chelsea* satisfait toutes les contraintes défini plus haut.

5 Résultat et perspectives

5.1 Résultat

On estime que pour une première version avec assez peu de données, on a réussi à construire un système à base de connaissances qui prédit **relativement** [6] bien le gagnant d'un match.

5.2 Perspectives

On pourra réaliser plusieurs améliorations sur ce projet. On peut notamment coupler notre vocabulaire avec un vocabulaire externe (tel que `http://fr.dbpedia.org/page/Football`) pour pouvoir récupérer plus d'informations sur les équipes et les matches. Ce qui nous permettra de définir des contraintes plus complexe, plus ou moins forte, ce qui implique d'ajouter une sorte de coefficient pour chaque contrainte.

Enfin, nous pourrons ajouter plusieurs classes de probabilité afin d'affiner le résultat final.

5.3 Avantages et limites de l'Ingénierie des Connaissances

Durant la réalisation du projet, nous nous sommes rendu compte que la partie modélisation du problème est assez intuitive. En effet, il suffit de bien connaitre le domaine considérer et de traduire les connaissances en tuples RDF. Ce qui a été très facile. Par ailleur, peu de données suffisent à avoir un système qui peut déduire quelques connaissances utiles.

Par contre, nous avons rencontrés quelques problèmes. Notamment, lors de la phase de le constitution de la **ABOX**. En effet, le processus de récupération des données n'a pas été aussi facile qu'on le penser. Malgré un domaine qui est assez grand public, les données n'ont pas été facile à retrouver dans un format CSV ou équivalent. On a dû écrire quelques scripts pour récupérer les données qui manquent.

6 Intérêt des ontologies par rapport aux bases de données relationnelles

À la fin du projet, on s'est rendu compte de quelques différences notables des systèmes à base de connaissances par rapport aux classiques bases de données relationnelles. On va mettre en avant dans ce qui suit les limites des BDR et les réponses apportés par les ontologies.

Séparation entre la structure et les données Dans une base de données relationnelles, la structure des données est séparée des données elle-même. Si on extrait les données de la base, il faut donc aussi en extraire la structure pour comprendre à quoi correspond chaque donnée. Dans les ontologies, les données sont décrites sous la forme de graphe. La structure fait donc partie de la donnée.

Indépendances des données Dans une base de données relationnelles, les données ne sont pas indépendantes les unes des autres. Elles se conçoivent dans le contexte de la base, d'un champ et d'un enregistrement. Par contre, dans une ontologie, les données sont décrite selon la logique des prédicat du premier ordre, sous la forme de triplets (*subject - propreties - object*). Chaque triplet représente une donnée et est indépendant du contexte. De plus une propriété peut-être répétable pour une même ressource. Ce qui nécessite un nouveau champ ou une nouvelle table dans une base de données relationnelles.

6. Prédictions plutôt naïve pour l'instant

Multilinguisme Si on veut exprimer, dans un schéma relationnel, une données dans plusieurs langues, alors on doit définir un champ pour chaque langue. Ce qui donne plusieurs champs qui représentent la même donnée alors que la signification de celui-ci reste exactement la même. Par contre, dans les ontologies, il est possible d'attribuer un tag pour la donnée qui définit la langue dans laquelle elle est décrite.

Exemple :

```
dbp:Football rdfs:label "Football"@fr
dbp:Football rdfs:label "Soccer"@en
```

Une seule propriété définit le nom de la ressource (ici *rdfs :label*), alors que dans un schéma relationnel, il aurait fallu définir deux champs (par exemple *label-fr* et *label-en*).

Portabilité et normalisation des données L'une des limites principale des bases de données relationnelles est le manque de portabilité et de normalisation des données. C'est-à-dire, il n'est pas possible d'identifier nativement deux ressources équivalentes entre deux bases différentes. C'est une possibilité offerte par les ontologies au travers de **OWL**[7] qui fournit les moyens pour définir des ontologies web structurées. Puisqu'une ressource est identifiée par un identificateur unique (URI[8]), il possible de dire qu'une certaine ressource, de part la propriété *owl :sameAs*, est équivalente à une autre, et cela, même si les deux ressources ne sont pas dans la même base de connaissances. Ce qui donne aux ontologies une portabilité et une universalisation des ressources et données qu'elles expriment. De plus, avec cette possibilité de créer des relations sémantique entre les ressources, il est possible d'inférer sur les connaissances déjà établies pour pouvoir en déduire de nouvelles.

Exemple : Étant donnée ces connaissances

```
bet:entraineur_X bet:manage bet:equipe_X
bet:manage owl:inverseOf bet:managedBy
```

Alors on peut déduire automatiquement cette connaissance

```
bet:equipe_X bet:managedBy bet:entraineur_X
```

Pour finir, on voit bien que le modèle relationnel impose une structure **rigide, difficile à faire évoluer et indépendante** des données elles-mêmes alors que les ontologies offrent un modèle **souple, universel, et compatible** avec le fonctionnement du Web.

7. Web Ontology Language est un langage de représentation des connaissances construit sur le modèle de données de RDF.
8. Uniform Resource Identifier