AN ONTOLOGY-BASED RETRIEVAL SYSTEM USING SEMANTIC INDEXING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SONER KARA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JULY 2010

Approval of the thesis:

AN ONTOLOGY-BASED RETRIEVAL SYSTEM USING SEMANTIC INDEXING

submitted by **SONER KARA** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Nihan K. Çiçekli _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Özgür Ulusoy
Computer Engineering Dept., Bilkent Univ. _____

Assoc. Prof. Dr. Nihan K. Çiçekli
Computer Engineering Dept., METU _____

Assoc. Prof. Ferda N. Alpaslan
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU _____

M.Sc. Özgür Alan
Orbim Corp., METU Technopolis _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SONER KARA

Signature :

ABSTRACT

AN ONTOLOGY-BASED RETRIEVAL SYSTEM USING SEMANTIC INDEXING

Kara, Soner
M.Sc., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Nihan K. Çiçekli

July 2010, 57 pages

In this thesis, we present an ontology-based information extraction and retrieval system and its application to soccer domain. In general, we deal with three issues in semantic search, namely, usability, scalability and retrieval performance. We propose a keyword-based semantic retrieval approach. The performance of the system is improved considerably using domain-specific information extraction, inference and rules. Scalability is achieved by adapting a semantic indexing approach. The system is implemented using the state-of-the-art technologies in Semantic Web and its performance is evaluated against traditional systems as well as the query expansion methods. Furthermore, a detailed evaluation is provided to observe the performance gain due to domain-specific information extraction and inference. Finally, we show how we use semantic indexing to solve simple structural ambiguities.

Keywords: Semantic Web, Keyword-based Querying, Semantic Indexing, Ontology, Inference

ÖZ

ANLAMSAL INDEKSLEME KULLANARAK ONTOLOJİ TABANLI SORGULAMA SİSTEMİ

Kara, Soner
Yüksek Lisans, Bilgisayar Mühendisliğ Bölümü
Tez Yöneticisi : Doç. Dr. Nihan K. Çiçekli

Temmuz 2010, 57 sayfa

Bu tezde, ontoloji tabanlı bilgi çıkarma ve sorgulama sistemi ve bu sistemin futbol alanına uygulanması anlatılmaktadır. Genel olarak, anlamsal sorgulama alanındaki üç sorunla ilgilenilmiştir: kullanılabilirlik, ölçeklenebilirlik ve sorgulama performansı. Çözüm olarak, kelime-tabanlı bir anlamsal sorgulama yöntemi sunulmaktadır. Alana özgü bilgi çıkarma, anlamsal çıkarım yapma ve kurallar ile performans büyük ölçüde arttırılmıştır. Ölçeklenebilirlik ise, anlamsal indeksleme yöntemiyle elde edilmiştir. Anlamsal Web'in en güncel teknolojileri kullanılarak gerçekleştirilen sistem, geleneksel yaklaşımlar ve sorgu genişletme yöntemleriyle karşılaştırılmıştır. Detaylı deneyler ile alana özgü bilgi çıkarma ve anlamsal çıkarımın sistem performansına etkileri gözlemlenmiştir. Son olarak, basit yapısal belirsizliklerin anlamsal indeksleme yöntemiyle nasıl çözülebileceği gösterilmiştir.

Anahtar Kelimeler: Anlamsal Web, Kelime-tabanlı Sorgulama, Anlamsal İndeksleme, Ontoloji, Anlamsal Çıkarım

To my family

ACKNOWLEDGMENTS

I would like to express my sincere gratitude and appreciation to my supervisor Assoc. Prof. Dr. Nihan Kesim Çiçekli for her guidance, advice, criticism, encouragements and insight throughout the research.

The technical assistance and guidance of Özgür Alan, Orkunt Sabuncu and Samet Akpınar is gratefully acknowledged.

I would like to thank to Onur Deniz for implementing the core of the Web Crawler module.

I am deeply grateful to my parents for their love and support. Without them, this work could not have been completed.

Finally, I would like to thank to the Scientific and Technical Research Council of Turkey (TÜBİTAK) for the financial support in accordance with the 2228-scholarship program. This thesis is also partially supported by TÜBİTAK Grant EEEAG 107E234.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

The huge increase in the amount and complexity of reachable information in the World Wide Web caused an excessive demand for tools and techniques that can handle data semantically. The current practice in information retrieval mostly relies on keyword-based search over full-text data, which is modeled with bag-of-words. However, such a model misses the actual semantic information in text. To deal with this issue, ontologies are proposed [5] for knowledge representation, which are nowadays the backbone of semantic web applications. Both the information extraction and retrieval processes can benefit from such metadata, which gives semantics to plain text.

Having obtained the semantic knowledge and represented them via ontologies, the next step is querying the semantic data, also known as semantic search. There are several query languages designed for semantic querying. Currently, SPARQL[1] is the state-of-the-art query language for the Semantic Web. Unfortunately, these formal query languages are not meant to be used by the end-users. Formulating a query using such languages requires the knowledge of the domain ontology as well as the syntax of the language. Therefore, Semantic Web community works on simplifying the process of query formulating for the end-user. The current studies on semantic query interfaces are carried in four categories, namely, keyword-based, form-based, view-based and natural language-based systems as reviewed in [18]. Out of these, keyword-based query interfaces are the most user-friendly ones and people are already used to use such interfaces easily, thanks to Google.

Combining the usability of keyword-based interfaces with the power of semantic technologies is one of the most challenging areas in semantic searching. According to our vision of

[1] http://www.w3.org/TR/rdf-sparql-query/ (last visited on 06/07/2010)

Semantic Web, all the efforts towards increasing the retrieval performance while preserving the user-friendliness will eventually come to the point of improving semantic searching with keyword-based interfaces. This is a challenging task as it requires complex queries to be answered with only a few keywords. Furthermore, it should allow the inferred knowledge to be retrieved easily and provide a ranking mechanism to reflect semantics and ontological importance.

In this thesis, we present a complete ontology-based framework for the extraction and retrieval of semantic information in limited domains, that we partially presented in [8]. We applied the framework in soccer domain and observed the improvements over classical keyword-based approaches. The system consists of an automated information extraction module, an ontology populator module, an inference module, and a keyword-based semantic query interface. Our main concern, in this study, is achieving a high retrieval performance while preserving the user-friendliness. We show that our system can handle even the very complex queries a user can ask in soccer domain. Furthermore, we evaluate and report the effects of information extraction and inference on the query performance.

The rest of the thesis is organized as follows: We begin with some basic background information in Chapter 2. A brief discussion about the related work is given in Chapter 3. In Chapter 4, we give the details of the components of the system, namely information extraction (IE), ontology population, inference and information retrieval (IR). In Chapter 5, we give the evaluation results. Further analysis and a brief discussion are provided in Chapter 6. In Chapter 7 we give some implementation details and Chapter 8 concludes the thesis with some remarks for future work.

CHAPTER 2

BACKGROUND INFORMATION

This chapter provides some basic background information about the terms and technologies used throughout this thesis. We begin with the core discipline of this thesis, i.e. the Information Retrieval. Then, we continue with the important aspects of the Semantic Web, such as ontologies, query languages, inference and rules.

2.1 Information Retrieval

Information retrieval (IR) is one of the oldest research areas in the information science. The goal of the IR discipline is to search and retrieve the most *relevant* documents to the information needs of the user. Therefore a good IR system should retrieve only those documents that satisfy the user needs, not a bunch of unnecessary data.

2.1.1 The General IR Process

Information retrieval systems have been evolved and improved a lot since their first emergence in 1950s. However, the core process hardly changes, which can be seen in Figure 2.1. The important aspects of the IR process are described below.

2.1.1.1 User Interface

Although it seems insignificant at first, the user interface is one of the most important aspects in IR. The design of the user interface brings the tradeoff between user-friendliness

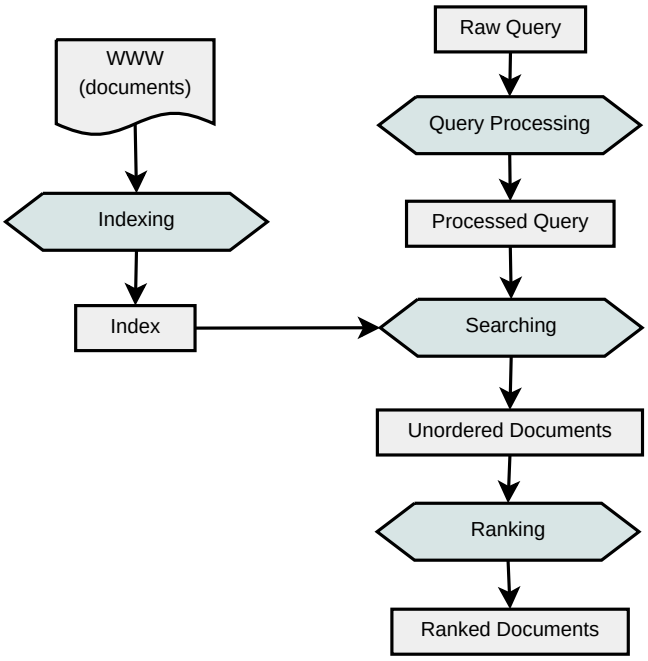


Figure 2.1: The General IR Process

and performance. Simple and relaxed interfaces are easier to use at the cost of resulting ambiguous queries. Complex and powerful interfaces, however, provide more detailed and precise query formulation, while they are cumbersome and time-consuming for the end-user. Some of the widely used user-interface methods are described in [1] for traditional IR and in [18] for semantic retrieval. Keyword-based, natural language-based, form-based and graph-based interfaces are some of the commonly used interface methods in the literature. We use keyword-based interface in this thesis to achieve maximum usability for the end-user.

2.1.1.2 Query Processor

The raw query submitted by the user should be processed before searching. Usually, the query is transformed into an internal form that the system can interpret. Furthermore, several processing tasks can be involved such as stopword elimination, stemming and other application specific tasks. These tasks should also be done in the indexing phase for consistent matching.

2.1.1.3 Indexing

Indexing is an essential part of the IR systems for two reasons. First, it optimizes the query performance and improves the respond times considerably by storing terms in an inverted

file structure. Basically, it stores the text positions for occurrences of each term. Secondly, a number of processing tasks are carried out during the indexing phase similar to the query processing phase, which further improves the performance.

2.1.1.4 Searching

In this phase, the query terms are searched against the inverted index. All the documents that contain the occurrences of the query terms are retrieved. Depending on the application, the retrieval can be done even for the partially matched documents.

2.1.1.5 Ranking

The documents retrieved in the previous step are given scores according to the matching quality between the query terms and the documents. The documents are sorted according to this score, so that the most relevant documents are presented to the user on top of the retrieval list. Ranking process is highly dependent to the IR model. As we will explain in the following sections, some IR models does not support ranking, where all the retrieved documents are considered to be equally important. Finally, the choice of evaluation metric is critical, since only a few of them can utilize the ranking information for the evaluation score as we mention in Section 2.1.3.

2.1.2 IR Models

IR systems can be categorized into three main groups in terms of the model they use for document and query representation, which have a great influence on the retrieval performance.

Boolean Model This is a simple model based on the set theory and the Boolean algebra [6]. The documents are represented as sets of terms and queries are just Boolean expressions. A document is retrieved if it contains all the terms expressed by the Boolean expression. Although it is intuitive and easy to implement, it needs exact matching in the retrieval, which causes the "too many or nothing" problem.

Vector Space Model (VSM) This model represents both the documents and queries as vec-

tors of weighted terms. The documents are retrieved according to the degree of similarity with the query vector. In contrast to the Boolean model, vector space model can also retrieve partially matched documents. VSM is coined by G. Salton [14].

Probabilistic Model In this model, a set of relevant documents to the query is assumed. The retrieval is done according to the probabilities of belonging to this set. The probabilities are usually derived using the Bayes' Theorem [19].

Having described different IR models, we want to mention briefly about the IR model that is used in Apache Lucene[1], an open-source indexer and searcher, that we use in this thesis. Lucene uses a combination of VSM and Boolean model for the retrieval. Basically, VSM is used to score the relevant documents and Boolean model is used to select the matching documents according to the user query, so it is essentially a VSM with Boolean capabilities.

2.1.3 Evaluation Metrics

The performance of IR systems can be measured using several evaluation metrics. To be able to use one of these metrics, one should prepare a ground-truth for each query by manually categorizing each retrieved document as relevant or irrelevant with respect to the query. In this section we describe some of the important metrics used in the evaluation of IR systems.

2.1.3.1 Precision

Precision is one the most commonly used metrics in the IR world. It basically measures how precisely the system picks the related documents among all documents. More specifically, it is the proportion of the related documents in the retrieved documents (true positives) to the total number of retrieved documents (Eq. 2.1). Precision, on its own, does not give much information about the actual performance of the system, since it does not consider whether or not all the related documents are retrieved.

$$Precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{2.1}$$

[1] http://lucene.apache.org/ (last visited on 06/07/2010)

2.1.3.2 Recall

Recall is another widely used IR metric. It is the proportion of the retrieved related documents to the total number of related documents that should have been retrieved. Similar to precision, it is not much meaningful on its own, because it does not takes into account the unrelated documents retrieved (Eq. 2.2).

$$Recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{2.2}$$

2.1.3.3 F-Measure

Precision and recall should be used together to obtain a more complete evaluation metric. F-Measure is developed for this very reason. It is basically the harmonic mean of precision and recall, thus it provides a more robust evaluation criteria (Eq. 2.3).

$$F\_measure = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.3}$$

2.1.3.4 Mean Average Precision (MAP)

When used together, precision and recall give a solid idea about the retrieval performance. However, they do not take the order of retrieved documents, i.e. the ranking, into account. Ranking is important, since the user likes the most relevant documents to be on top of the retrieval list. MAP fills this gap by adding the order of documents into consideration. Furthermore, MAP has shown to have a good discrimination and stability among evaluation metrics [10]. The average precision (AP) value for a single query is calculated by taking the average of the precision values obtained for the set of top k documents existing after each relevant document is retrieved. This value is then averaged over the entire query set to find the MAP value (Eq. 2.4).

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{2.4}$$

In this formula, m_j is the number of related documents, $Precision(R_{jk})$ is the precision at recall level k for query j, and $|Q|$ is the total number of queries. Due to the reasons stated above we use MAP as our evaluation metric in this thesis.

2.2 Semantic Web

In this section, we describe what the Semantic Web is, the main goal of Semantic Web and the technologies involved.

2.2.1 What is Semantic Web?

The Semantic Web (SW) is defined[2] as a Web of data. Its vision is to represent the whole Web as a globally linked database. SW is an extension to the current World Wide Web, not a complete replacement of the current standards. It brings a number of new tools and technologies to model, annotate, search and integrate data.

2.2.2 The Goal of Semantic Web

The current Web consists of pages and documents that are connected to each other via hyperlinks. A user can easily reach the desired information by following these links. It is easy for us, because the content is rendered by the browser so that the human-being can understand and interpret it. However, from a machine or a software agent perspective, a web page is nothing but pure html code, which does not give any clue about the *meaning* of the content (Figure 2.2). Thus, the automatic agents cannot browse the Web and collect information as easily as we do. Semantic Web is proposed to overcome these difficulties. It gives meaning to documents and the entities in that document. Entities can be uniquely identified and have their own set of properties. Two entities can be related to each other via these properties allowing data integration, data reuse and automation.

[2] http://www.w3.org/standards/semanticweb/ (last visited on 06/07/2010)

Figure 2.2: The human can see and interpret a rendered webpage (left), but a machine can only see the HTML code of that page (right).

2.2.3 Semantic Knowledge Representation

Since HTML documents are not capable of holding semantic data, new standards for the Semantic Web have been developed. We describe some of the important ones, namely RDF[3], RDF-S[4] and OWL[5].

2.2.3.1 The Semantic Data Model

The foundation of semantic knowledge representation is the RDF (Resource Description Framework) standard. RDF is a simple data model, which describes the objects (resources) and the relations among them with a triple syntax (subject-predicate-object). A knowledge-base consisting of these RDF statements is essentially a labeled and directed graph with the nodes being resources while the edges represent the properties. Figure 2.3 illustrates this with an example. With the introduction of ontologies, a number of extensions to RDF are emerged, such as RDFS and OWL.

[3] http://www.w3.org/TR/PR-rdf-syntax/ (last visited on 06/07/2010)
[4] http://www.w3.org/TR/rdf-schema/ (last visited on 06/07/2010)
[5] http://www.w3.org/TR/owl-guide/ (last visited on 06/07/2010)

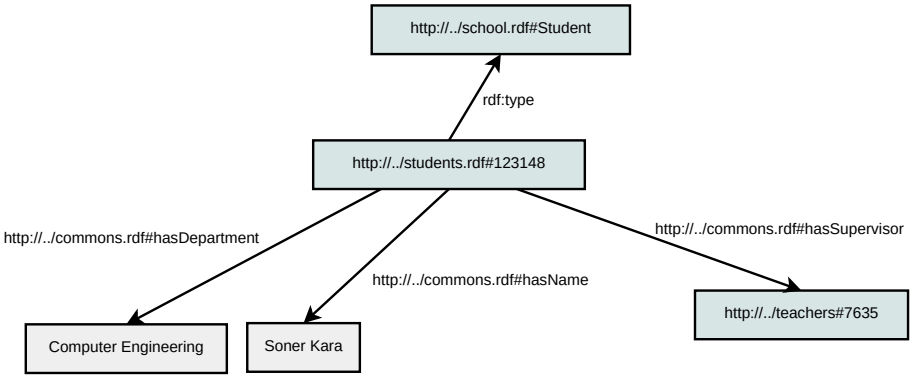


Figure 2.3: A Simple Example to RDF Graphs

2.2.3.2 Ontology Languages

Ontologies are used to model real word entities and relations among them in a taxonomic structure. They are nowadays the backbone for the Semantic Web applications. Several languages are developed for the formal representation of ontologies. RDF Schema (RDFS) was the first attempt towards developing an ontology language and it became a W3C recommendation in 2004. RDFS was built upon RDF. It extends the RDF vocabulary with additional classes and properties such as rdfs:Class and rdfs:subClassOf.

The latest W3C recommendation for ontology languages is the Web Ontology Language (OWL). OWL further extends RDFS by providing additional features such as cardinality constraints, equality, disjoint classes, efficient reasoning support and much more. OWL language has three sublanguages, which are OWL-Lite, OWL-DL and OWL-Full in the order of increasing expressibility. OWL-Lite and OWL-Full are not widely used, because the former is too restricted and the latter does not guarantee efficient reasoning. OWL-DL provides maximum expressibility with a complete and decidable reasoning support.

2.2.4 Semantic Search

By representing the data in RDF or OWL format, the Semantic Web allows more intelligent search engines to be developed. These search engines can make use of the metadata associated with the entities to improve search quality. Semantic relations defined in ontologies allow very complex queries to be answered which are not possible otherwise. For example, semantic search engines can easily answer queries such as “find locations of hospitals which contain

more than 20 operation rooms". Obviously, that much detail cannot be handled by traditional methods.

2.2.4.1 Query Languages

Several query languages have been developed to search over RDF documents. RDQL[6], SeRQL[7], and SPARQL are only a few of them. Since SPARQL became an official W3C recommendation in 2008[8], it is currently the most widely used semantic query language. Basically, a SPARQL query consists of conjunctions and disjunctions of triple patterns similar to RDF triples. A simple SPARQL query to search "the goals scored by Roberto Carlos" is shown in Figure 2.4. Despite its simplicity, the usability of SPARQL is limited for the end-user. First of all, formulating a query requires considerable time and effort even for the simplest query. Secondly, the domain knowledge is required, i.e. the exact names of classes and properties need to be known in advance. Therefore, researchers work on the easy-to-use alternatives to SPARQL as we mentioned in Section 2.1.1.1.

```
PREFIX pre: <http://isl.ceng.metu.edu.tr/futbol_yeni3.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX players: <http://www.oyuncular.com/oyuncular2.owl#>
SELECT DISTINCT ?goal ?match ?minute ?date
WHERE
{
   ?goal a pre:Goal.
   ?goal pre:scorerPlayer players:Roberto_Carlos.
   ?goal pre:inMatch ?match.
   ?goal pre:inMinute ?minute.
   ?match pre:atDate ?date.
}
```

Figure 2.4: An Example SPARQL Query to Search "Goals scored by Roberto Carlos"

2.2.5 Inference

Inference (a.k.a reasoning) is a method for knowledge acquisition or expansion. It expands the knowledgebase with additional information using the existing data, metadata, and rules. We mentioned about the efficient reasoning support of OWL-DL in Section 2.2.3. With the development of OWL-DL, a number of sound, complete and terminating DL reasoners, such

[6] http://www.w3.org/Submission/RDQL/ (last visited on 06/07/2010)
[7] http://www.openrdf.org/doc/sesame2/users/ch09.html (last visited on 06/07/2010)
[8] http://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation (last visited on 06/07/2010)

as RACER[9], FaCT++[10] and Pellet[11], have been arisen. These tools make use of the ontological information and restrictions to classify, realize and verify the semantic data. Classification builds the complete class hierarchy. Realization finds direct types of each individual and verification checks the data against contradictions.

Besides these reasoning tasks, one can develop domain-specific rules to infer more complex and interesting information that cannot be inferred with standard OWL features. SWRL[12] and Jena Rules[13] are two widely used rule languages in the literature. SWRL is the W3C recommendation for the rule language of the Semantic Web. Jena Rules is a powerful alternative to SWRL. The idea is the same in both languages: when the conditions on the body of the rule are satisfied then the clauses on the head are executed. A very simple semantic rule can be seen in Figure 2.5.

```
[uncleRule: (?f isFather ?a) (?u isBrother ?f) -> (?u isUncle ?a)]
                                          body -> head
```

Figure 2.5: A Classical Example to Semantic Rules

[9] http://www.sts.tu-harburg.de/~r.f.moeller/racer/ (last visited on 06/07/2010)
[10] http://owl.man.ac.uk/factplusplus/ (last visited on 06/07/2010)
[11] http://clarkparsia.com/pellet/ (last visited on 06/07/2010)
[12] http://www.w3.org/Submission/SWRL/ (last visited on 06/07/2010)
[13] http://jena.sourceforge.net/inference/#rules (last visited on 06/07/2010)

CHAPTER 3

RELATED WORK

In this chapter, we give brief information about the studies related to our thesis work. We begin with the simplest IR method, the traditional approach, and continue with methods in the order of increasing relatedness to our study, namely the query expansion methods, semantic approaches and semantic indexing. Finally, we state the differences between the related work and our approach.

3.1 Traditional Approaches

The classical or traditional keyword-based information retrieval approaches are based on the IR models described in Section 2.1.2. In the area of text retrieval, these models are usually accompanied with the bag-of-words model to represent the documents. Therefore, no extraction or annotation tasks are involved and the meanings of the sentences are ignored. There are many studies that can be put in this category. We review some of them here.

The studies of G. Salton et al. had a great impact on the field of information retrieval. The SMART Information Retrieval System [12] was one of the most important studies in the IR field. The main emphasis of SMART project was automatic text retrieval. During the development of SMART, G. Salton et al. came up with many new concepts such as vector space models and relevance feedback. Moreover, a number of experiments are done to show the effects of various actions such as stemming, statistical phrase selection and weighting.

Another important study presented by G. Salton et al. is the extended boolean model [13], which improves the classical boolean model by adjusting the strictness of AND and OR operators, so that it supports ranking and partial matching.

The study presented in [7] proposes a weighting schema for retrieval systems. The author suggests that the terms should be weighted according to collection frequency, so that more specific terms has greater value on matching.

3.2 Query and Index Expansion Methods

The first step towards semantic retrieval was using WordNet synonym sets (synsets) for word semantics. The main idea is expanding both the indices and queries with the semantics of the words to achieve better recall and precision.

The study described in [4] tries to find out how much WordNet improves the retrieval performance over a standard vector space based retrieval system. The authors show that expanding the index with word senses or synsets can improve the performance if a successful Word Sense Disambiguation (WSD) method is applied. They provide further experiments to show the effects of WSD on retrieval performance with different levels of error rates. They conclude that the performance greatly depends on the quality of WSD, thus a WordNet expansion combined with a poor WSD will cause degradation in performance.

The system described in [11] extends the Boolean IR model by adding word semantics to free-text index. It expands both the queries and the indices with WordNet synsets, part-of-speech tags and word stems. To find the correct sense of terms, the authors implement a semi-complete WSD algorithm with 92% accuracy. They evaluate the system and observe an improvement over the traditional approach.

So, we conclude that an effective WSD algorithm is needed in order to benefit from this approach. Another drawback is that the semantics are limited to the relations defined in the WordNet.

3.3 Semantic Approaches

With the introduction of semantic web technologies, knowledge representation has become more structured and sophisticated, which requires more advanced extraction and retrieval methods to be implemented. The general approach is storing the extracted data in RDF or OWL format, and querying with RDF query languages such as RDQL or SPARQL. Although

this approach offers the ultimate precision and recall performance, it is far from useful since it requires a relatively complex query language.

To overcome the difficulties of learning a formal query language, a number of query interface methods are proposed [18]. As we stated earlier, our main focus is keyword-based interfaces. There are several approaches to implement keyword-based querying. We review some of them in this section.

SPARK [21] is a retrieval system that can query semantic data with SPARQL queries generated from keywords. It consists of three main steps: term mapping, query graph construction and query ranking. The term mapping step tries to match query terms to ontological resources. In the query construction step, the user query is analyzed and the terms are linked together with the missing relations to obtain SPARQL queries. Usually more than one query is constructed due to the ambiguities. Therefore in the last step, the queries are evaluated with a probabilistic ranking model so that the more likely SPARQL queries get higher rates.

Q2Semantic [20] tries to bridge the gap between keyword queries and formal queries. The authors deal with three problems to achieve their goal, namely term matching, ranking and scalability. The first problem is tackled by enriching the user queries with the terms extracted from Wikipedia, so that query terms are easily matched with the entities in the ontology. To solve the second problem, they implement a ranking mechanism that consider many factors such as query length, the relevance and importance of ontology elements, etc. Finally, for the scalability problem they propose a clustered graph structure to represent summaries of RDF graphs. But they still need the costly graph construction and traversal procedures at run-time.

SemSearch [9] is a similar system that generates formal queries from the keywords. It differs from the systems above in the way it construct the queries. It uses a number of predefined templates that are filled with various combinations of query terms. Since every keyword in the query can be a class, an instance or a property, the number of combinations is quite large, which affect query respond times negatively. When the queries are generated, they are used directly for searching the semantic data. The ranking is done after the results of all queries are retrieved.

Although these systems improve traditional approach, they have a common disadvantage: given a keyword query, they generate more than one formal query and search the same knowl-

edgebase many times using these queries. If we also consider the cost of term matching and query generation, we can clearly see the scalability issues of these methods.

3.4 Semantic Indexing Methods

A scalable alternative to query construction from keywords is semantic indexing. In this approach, semantic data in RDF knowledge-bases are indexed in a structured way and made directly available to use with keyword queries.

OWLIR [16] is a semantic retrieval system that adapts a similar approach. It consists of an information extraction module, an inference module and a retrieval module. They use the AeroText[1] system to extract key phrases from free-text and represent them as RDF triples. This data is enriched with reasoning and semantic rules to obtain inferred RDF triples. Finally, these triples are indexed along with the corresponding free-text. The details of indexing mechanism and ranking are omitted. In the evaluation part, they show the improvements by comparing the three indices: free-text, free-text with RDF triples and free-text with inferred RDF triples,

QuizRDF [3] combines keyword-based querying with RDF browsing, so that users can initiate a query with simple keywords and continue their search by browsing the semantic knowledge-base until they reach the desired information. The knowledgebase is again built by indexing the RDF triples in addition to the free-text. Since an indexed document contains only one property associated with the entry, the retrieval system does not support complex queries containing more than one property of the searched entity. No evaluation results are provided in this study.

Squiggle [2] is a retrieval framework that uses semantic indexing for efficient access to semantic knowledge. Instead of using the whole knowledgebase for searching, it indexes and searches only those entities that are recognized during the semantic annotation phase. The unit of indexing is again triples. Unlike the systems described above, Squiggle keeps free-text in a separate index and the query submitted by the user is searched in both indices. The retrieval is done primarily using the free-text index and the semantic index is used to make suggestions to the user according to the ontological knowledge.

[1] http://www.lockheedmartin.com/products/AeroText/index.html (last visited on 06/07/2010)

To sum up, the systems described above have a simple indexing mechanism that store all the RDF triples together with the corresponding free text to enhance traditional approach. Since they use very basic extraction methods, such a naive indexing seems feasible. However, complex semantics cannot be captured from the indices containing only subject-predicate-object triples as index terms. If a retrieval system should answer complex queries involving extracted and inferred knowledge, the index must be designed and enriched accordingly.

3.5 Comments

Our literature survey revealed that current studies on the keyword-based semantic searching are not mature enough: either they are not scalable to large knowledgebases or they cannot capture all the semantics in the queries. Our aim is to fill this gap by implementing a keyword-based semantic retrieval system using the semantic indexing approach. In other words, we try to implement a system that performs at least as good as traditional approaches and improves the performance and usability of semantic querying. We tested our system in soccer domain to see the effectiveness of semantic searching over traditional approaches and observed a remarkable increase in precision and recall. Moreover we noted that our system can answer complex semantic queries, which is not possible with the traditional methods. The study presented in this thesis can be extended to other domains as well by modifying the current ontology and the information extraction module as described in [17].

CHAPTER 4

OUR APPROACH TO SEMANTIC RETRIEVAL

Within the scope of this thesis we have developed a fully fledged semantic application which a) contains all aspects of Semantic Web from information extraction to information retrieval and b) uses all the cutting-edge technologies such as OWL-DL, inference, rules, RDF repositories and semantic indexing. The overall diagram of the framework can be seen in Figure 4.1. The following sections describe the important aspects of the system starting with a summary of the overall process.

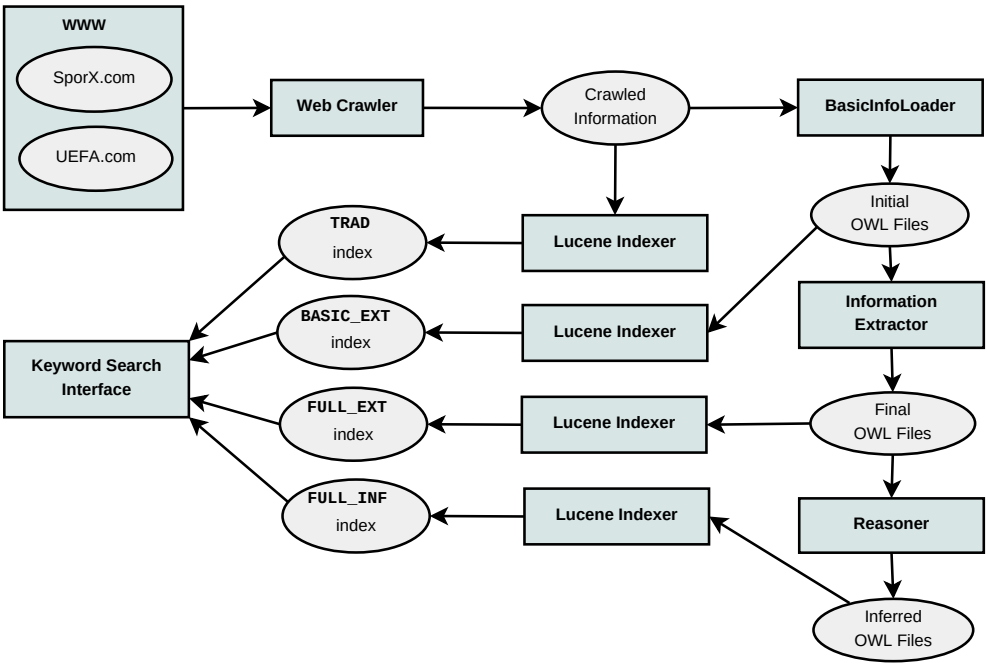


Figure 4.1: Overall System Diagram

4.1 Overall Process

Before giving the details of each module, we find it helpful first to see the overall flow of the system that we adapted for the soccer domain. In the following, we describe the steps we take until the system becomes ready for semantic querying and evaluation.

1. The usable information from web sites such as UEFA[1] and SporX[2] are crawled and put in a temporary storage. The crawled information contains some basic information such as teams, players, goals, substitutions and the stadium of each match as well as the minute-by-minute narrations of that match in free-text format.

2. Using only these free-text narrations we create an index, `TRAD`, for the traditional keyword search.

3. Using the basic information and narrations we populate the initial OWL files (Section 4.4).

4. From the initial OWL files, we create our second index, `BASIC_EXT`, that contains both the basic information and the narrations. This index is created for evaluation purposes only. (i.e, to compare it with the index created after the information extraction, `FULL_EXT`)

5. The OWL files created in the previous step are read by the information extractor module. This module populates the OWL files with the extracted events from the narrations such as offsides, fouls, corners, etc. to obtain the final OWL files (Section 4.3).

6. These OWL files are read and indexed to build `FULL_EXT` (Section 4.6.1).

7. We run the reasoner over these files and obtain new OWL files containing the inferred information (Section 4.5).

8. Finally, we build the index, `FULL_INF`, using these inferred OWLs, which is the final index used in semantic querying.

Although we have focused on the soccer domain in this study, the methodology described above can be adapted to any domain given its ontology. The most domain-specific part of

[1] http://www.uefa.com (last visited on 06/07/2010)
[2] http://www.sporx.com (last visited on 06/07/2010)

the system is the IE module, which should be extended to deal with other domains. The rest of the thesis describes the details of the major components of the system. We start with the design of the domain ontology, then continue with IE and IR components. Finally we report the evaluation results.

4.2 Ontology Design

Ontologies are specifications of concepts and relations among them. They play a central role in semantic web applications by providing a shared knowledge about the objects in real world, which promotes reusability and interoperability among different modules. Therefore the quality of the ontology should be the first concern in any semantic application.

For this study, we designed a central soccer ontology, which is utilized by every aspect of the system, especially in the information extraction, inference and retrieval phases. We followed an iterative development process in the ontology engineering phase. First, we started with a core ontology including the basic concepts and a simple hierarchy. Then, we experimented with this ontology and fix the issues in reasoning and searching. These steps were repeated until we end up with a stable ontology containing 79 classes and 95 properties in soccer domain. The full class hierarchy can be seen in Figure 4.2.

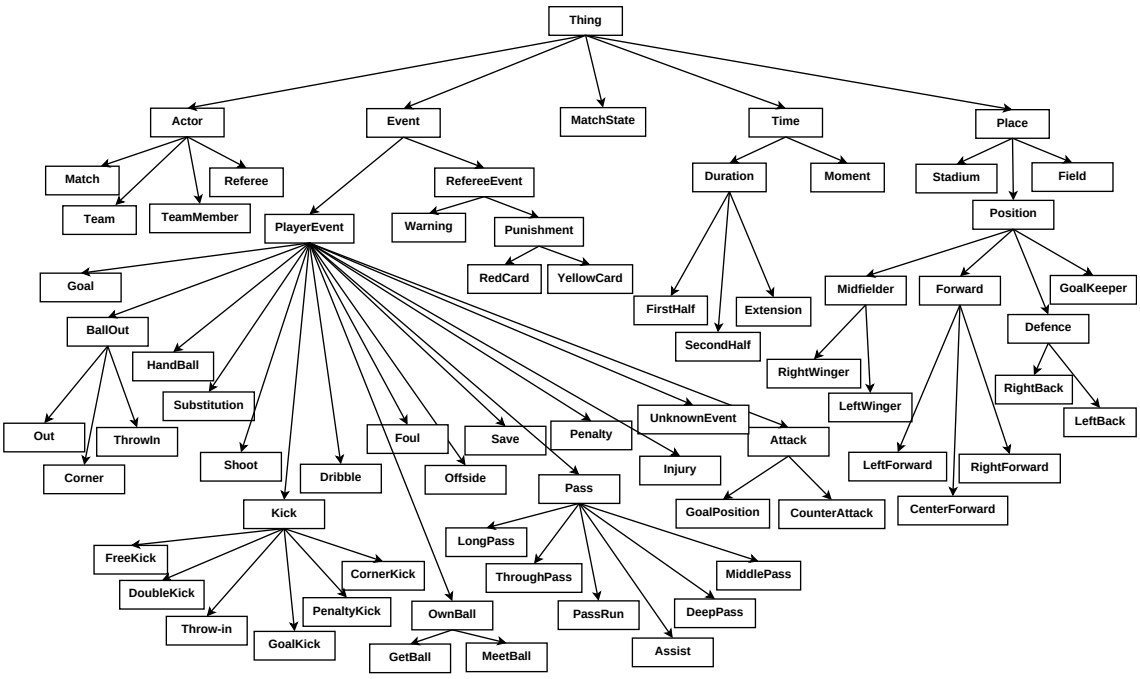


Figure 4.2: Domain Ontology, Class Hierarchy

4.3 Information Extraction (IE)

Information extraction is one of the most important parts of ontology-based semantic web applications. It is the process of adding structured information to the knowledgebase by processing unstructured resources. In this phase, we use the data crawled from the Web sites such as UEFA and SporX. What we obtain as the output of the web crawler is some basic information specific to a match (teams, players, goals, stadium, etc.) and natural language texts (minute-by-minute narrations of that match). The basic information and the narrations are used as input to our information extraction module. The details of this module are reported in [17]. In this thesis, we give just an overview of its functionality.

Basically, it is a template-based IE approach for specific domains. Unlike other automated approaches it does not use linguistic tools such as part-of-speech taggers, parsers or phrase chunkers. Therefore our approach can be applied to any domain or any language without using any linguistic tool, although there is the drawback of high effort spent for crafting the templates. The details of the IE module are summarized in two parts: a named entity recognizer and a lexical analyzer.

4.3.1 Named Entity Recognizer (NER)

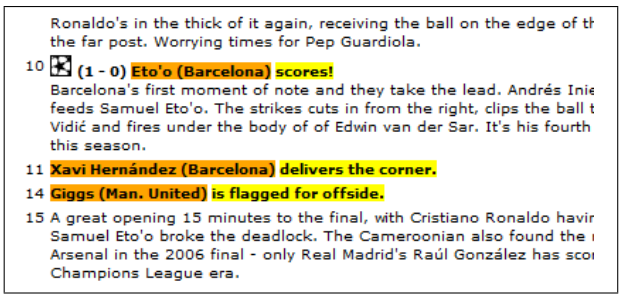
As mentioned earlier, IE module takes some basic information such as teams, players, etc., as input in addition to the text narrations. This information is used for recognizing and tagging the named entities in narrations. After running NER, the team and player names are replaced by tags of the form <team1>, <team2>, <team1player5>, etc. For example, the sentence "Iniesta scores!" may become "<team2player11> scores!", if Iniesta is the 11th player of the away team.

4.3.2 Two Level Lexical Analysis

This is the most critical part in our IE module, where complex semantic entities and relations are extracted according to the predefined templates. The first level recognizes the defined keywords/phrases and discards the rest. The second level takes the output of the first level as input and extracts information according to the pre-defined templates. According to our

survey, most of the studies in Semantic Web lack this kind of extraction as they are usually content with the annotation using only NER.

As reported in [17], we can achieve 100% success rate in UEFA narrations thanks to the language used in the UEFA web-site, which is highly structured and error-free. Figure 4.3 gives an idea about the information we can extract from the UEFA web-site. The integration of this module to the system is done in a loosely coupled fashion, so we can use it in semantic applications for any language or domain.


Ronaldo's in the thick of it again, receiving the ball on the edge of th
the far post. Worrying times for Pep Guardiola.

10 (1 - 0) Eto'o (Barcelona) scores!
Barcelona's first moment of note and they take the lead. Andrés Inie
feeds Samuel Eto'o. The strikes cuts in from the right, clips the ball t
Vidić and fires under the body of of Edwin van der Sar. It's his fourth
this season.

11 Xavi Hernández (Barcelona) delivers the corner.

14 Giggs (Man. United) is flagged for offside.

15 A great opening 15 minutes to the final, with Cristiano Ronaldo havir
Samuel Eto'o broke the deadlock. The Cameroonian also found the
Arsenal in the 2006 final - only Real Madrid's Raúl González has scor
Champions League era.

Figure 4.3: Example extractions from UEFA narrations

4.4 Ontology Population

Ontology population is the process of knowledge acquisition by transforming or mapping unstructured, semi-structured and structured data into ontology instances [15]. Our information extractor module [17] already does most of the labor by extracting structured information from unstructured text narrations. For example, from the narration "Keita commits a foul after challenging Belletti" we obtain a foul object, more specifically `FOUL(Keita, Belletti)`.

Having the output of the IE module, the ontology population process becomes creating an OWL individual for each object extracted during IE. This is the phase, where the IE module is integrated with the rest of the system. We tried to keep this integration as loose as possible. We deal with this issue at the ontology level by defining some high level attributes. Normally, every event in the ontology has its own set of attributes. For example, a `Foul` instance has a `punishedPlayer` attribute, while a goal instance has a `scorerPlayer` attribute to refer to the subject of the event. The problem is filling the right attribute of each event by using the information received from the IE module. Our solution to this problem is to define four generic properties for each event in our ontology, namely `subjectPlayer`,

objectPlayer, subjectTeam and objectTeam. These properties are generic and have sub-properties special to each event. For example Goal event has the property scorerPlayer, which is in turn a sub-property of subjectPlayer propery. In this way, we can automatically fill the scorerPlayer attribute of a Goal event by using the subject of the event. Similarly injuredPlayer property of an Injury event will be filled with the object of the event, since injuredPlayer property is defined as a sub-property of objectPlayer property in the ontology.

If the IE module cannot extract some attribute of an event, we still create an instance with empty properties. Thus, the recall performance for simple queries will not be affected even if the IE fails to extract some details of the event. Moreover, if no event is detected in a narration, an instance with the type UnknownEvent is created for that narration. Unknown events are not discarded because of the reasons described in Section 4.6.1. Figure 4.4 shows the process of ontology population starting from UEFA narrations ending with OWL instances.

Ontology population is not restricted with the events extracted from the IE module. As mentioned earlier, the crawled information also contains some basic information about the match including players, teams, referees, stadium etc. This information is also added to the ontology by creating an OWL individual for each of them if they do not already exist in the knowledgebase.

4.5 Inference and Rules

The formal specification of Web Ontology Language, OWL, is highly influenced by Description Logics (DLs). OWL-DL is designed to be computationally complete and decidable version of OWL, thus it benefits from a wide range of sound, complete and terminating DL reasoners. For our inference module, we use Pellet[3], an open-source DL-reasoner, which supports all the standard inference services such as consistency checking, concept satisfiability, classification and realization.

Consistency checking ensures that there is no contradictory assertion in the ontology. In order to benefit from this feature, we specify some property restrictions during the ontology development. There are two kinds of restrictions in OWL: value constraints and cardinality

[3] http://clarkparsia.com/pellet (last visited on 06/07/2010)

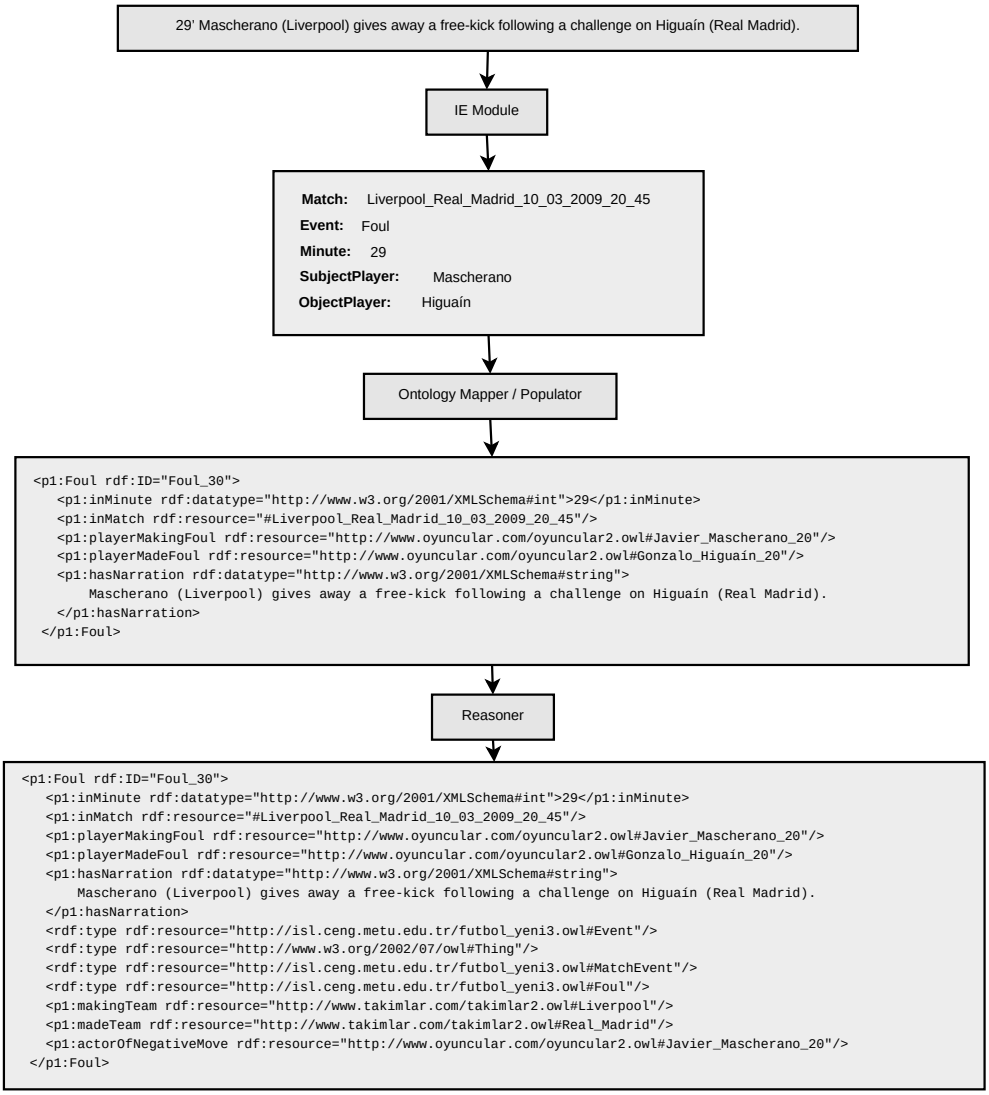


Figure 4.4: Information Extraction and Ontology Population

constraints. We use value constraints, for example, to state that only the goalkeepers (a subset of players) are allowed in the position of goalkeeping and using a cardinality constraint, we can say that only one goalkeeper is allowed in the game. These restrictions not only are useful in consistency checking but also allow new information to be inferred. For example, we could infer the type of an individual if it is the value of a property whose range is restricted to a certain class.

Using classification reasoning we obtain the whole class hierarchy according to class-subclass definitions in the ontology. Inferring new knowledge through classification is a domain independent process and its contribution to the knowledgebase is trivial. A simple example can be seen in Figure 4.5, where the class hierarchy of “Long Pass” is inferred.

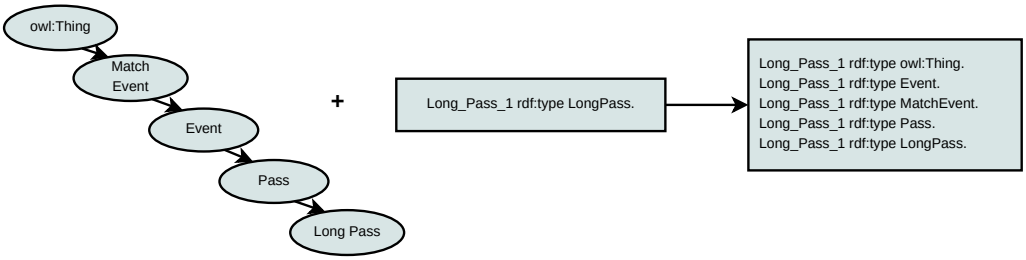


Figure 4.5: Inferring Class Hierarchy of Long Pass

In order to infer more interesting information, we use Jena[4] Rules. To illustrate the power of Jena rules, we give the example of inferring an "Assist" event. Using the Jena rule shown in Figure 4.6, we are able to add a new "Assist" instance to our knowledgebase. The rule simply looks for two events, namely a "Goal" and a "Pass", that happened in the same match in the same minute and the receiver of the pass is the same person with the scorer. If this is the case, then an "Assist" instance is created and added to the knowledgebase. More examples of Jena Rules can be seen in Appendix A.

```
noValue(?pass rdf:type pre:Assist)
(?pass rdf:type pre:Pass)
(?pass pre:passingPlayer ?passer)
(?pass pre:passReceiver ?receiver)
(?pass pre:inMatch ?match)
(?pass pre:inMinute ?minute)
(?goal pre:inMatch ?match)
(?goal pre:inMinute ?minute)
(?goal pre:scorerPlayer ?receiver)
makeTemp(?tmp)

-> (?tmp rdf:type pre:Assist)
   (?tmp pre:inMatch ?match)
   (?tmp pre:inMinute ?minute)
   (?tmp pre:passingPlayer ?passer)
   (?tmp pre:passReceiver ?receiver)
```

Figure 4.6: An Example for Jena Rules (Assist rule)

Inference is a costly process, especially if the number of assertions (ABox) is large. This problem should be handled carefully to maintain the scalability of the system. Therefore we take some measures in order to deal with this issue. First of all, we keep each match separate from each other and run the inference separately. Then, we disjunctively add the inferred information to the knowledgebase. So, the time needed for the inference of a match becomes independent of the total number of matches. Secondly, all reasoning tasks, including classification and rule inference, are done offline, i.e. prior to querying. This improves scalability as

[4] http://jena.sourceforge.net/ (last visited on 06/07/2010)

well as the query efficiency since no online reasoning is needed at runtime.

4.6 Semantic Indexing and Retrieval

There are many methods to query a semantic knowledgebase and retrieve the results. These methods are reviewed by [18] in four categories, namely keyword-based, natural language-based, view-based and form-based semantic querying. Out of these, keyword-based interfaces provide the most comfortable and relaxed way of querying for the end-user. Other methods, although they allow more precise queries to be formulated, require more user interaction depending on the size of the domain. Keyword-based interfaces have their own disadvantages such as ambiguities, however there are ways to minimize them as we will mention. Now, having decided on keyword-based querying, the next question is how we achieve high retrieval performance and scalability. The answer is, simply, semantic indexing.

As we mentioned in our literature survey, current keyword-based approaches either do real-time traversals in large RDF graphs or make simple RDF triple indexing. In other words, they do not focus on both scalability and retrieval performance. We propose a model, called semantic indexing, that extends traditional keyword-search with extracted and inferred information using domain ontology. The indexing mechanism is built upon Apache Lucene, which is essentially designed for free-text search. Semantic retrieval is achieved by implementing a custom ranking for Lucene indices so that documents containing ontological information get higher rates. The details of the index structure and ranking are given in Section 4.6.1 and 4.6.2 respectively.

4.6.1 Index Structure

The structure of the semantic index has utmost importance in the retrieval performance. We constructed a Lucene index such that each entry represents a soccer event. As we have mentioned in the previous sections, each event has its own properties associated with it, such as subjects and objects. That information is also included with each event. We also include full-text narrations associated with events to the index. This is especially important if the event type is unknown (an event which is not recognized by the information extractor). Adding full-text narrations to the index tolerates the incomplete event information, thus ensures at

least the recall values of the traditional full-text search. The index structure can be seen with an example entry in Table 4.1.

Table 4.1: Index Structure (Simplified for better understanding)

Field	**Value**
docNo	7
event	Foul
match	Chelsea_Barcelona_06_05_2009_20_45
team1	Chelsea
team2	Barcelona
date	2009-05-06
minute	43
subjectPlayer	Michael Ballack
subjectTeam	–
objectPlayer	Sergio Busquets
objectTeam	–
narration	Ballack gives away a free-kick following a challenge on Busquets

For the inferred OWL files, we build an extended version of this index. In addition to the basic information contained in this index, the inferred index also contains a field for all the inferred types of an event, a field for inferred player properties and a field to keep inferred information due to semantic rules. The additional information appended to the inferred index can be seen in Table 4.2. Note that the `subjectTeam` and `objectTeam` fields are also filled due to the semantic rules shown in Appendix A.

Table 4.2: Additional Information in Inferred Index

Field	**Value**
event	Negative Event Foul
subjectPlayerProp	Left Back Defence Player
subjectTeam	Chelsea
objectPlayerProp	Center Forward Player
objectTeam	Barcelona
fromRules	–

4.6.2 Searching and Ranking

In the traditional keyword search, the indexed documents usually contain nothing but raw text associated with that document. Lucene can easily handle such indices and its default ranking gives usually good results. However, complex indices should be handled carefully. In order to take the advantages of our ontology-aided index structure, we slightly modified the default querying and ranking mechanism of Lucene. First of all, we boosted the ranking

of fields containing the extracted and inferred information to stress the importance of them. Secondly, these fields are re-ranked according to their importance. For example, the "event" field is given the highest ranking. This approach prevents misleading stemming from ambiguous words in full-text. For example, suppose a narration contains "Ronaldo misses a goal". Searching for a "goal" in a traditional search may return this document in the first place, which is a false positive. However, in the ontology-aided index, the events whose type is `Goal` will have higher ranks. Since the type of the event above is a `Miss`, it will have a lower rank.

CHAPTER 5

EVALUATION

5.1 Test Data

In order to evaluate the retrieval performance of our system, we have crawled 10 UEFA matches, containing a total of 1182 narrations. Out of these narrations, our IE module was able to extract 902 events. Using these data, we constructed 4 Lucene indices for detailed comparisons. First, we built a traditional full-text index, `TRAD`, using only the narrations of the UEFA matches. This index is used as the baseline for the performance of other methods. Then, we built two indices for the ontology aided semantic search, namely `BASIC_EXT` and `FULL_EXT`, where the former contains only the basic information available in the UEFA crawl and the latter contains the extracted information in addition to the basic information. Finally, we built an index, `FULL_INF`, which is the expanded version of `FULL_EXT` with the inferred knowledge. Note that we built the indices in a cumulative manner starting from `TRAD` up to `FULL_INF`, i.e. the information contained in an index is also carried to the indices built after it. This is important, because it allows us to observe the improvements stemming from different actions, such as information extraction and inference.

5.2 Evaluation Process

We started the evaluation process by preparing the 10 queries shown in Table 5.1. Next to each query, we also put the corresponding keyword query (in boldface) which was actually used in the evaluation. Then, we prepared the ground-truth, i.e. the correct number of documents that should be retrieved, for each query. Finally, we run the queries for all indices and calculated the performance using the Mean Average Precision metric as mentioned in Section 2.1.3.

Table 5.1: Evaluation Queries

Q-1	Find all goals **(query: goal)**
Q-2	Find all goals scored by Barcelona **(query: barcelona goal)**
Q-3	Find all goals scored by Messi at Barcelona **(query: messi barcelona goal)**
Q-4	Find all punishments **(query: punishment)**
Q-5	Find all yellow cards received by Alex **(query: alex yellow card)**
Q-6	Find all goals scored to Casillas **(query: goal scored to casillas)**
Q-7	Find all negative moves of Henry **(query: henry negative moves)**
Q-8	Find all events involving Ronaldo **(query: ronaldo)**
Q-9	Find all saves done by the goalkeeper of Barcelona **(query: save goalkeeper barcelona)**
Q-10	Find all shoots delivered by defence players **(query: shoot defence players)**

5.2.1 A Closer Look to Evaluation Queries

Before starting to analyze the results, we want to clarify the evaluation queries for those readers who have little or no knowledge about the soccer domain.

Queries 1, 2 and 3 are used to retrieve goals, which are the most important events in soccer that every user will definitely search for. By executing the Query-1, a user should access all the goals in the knowledgebase. Query-2 limits the retrieved goals by specifying a team (Barcelona in this case) that scores the goal. Query-3 further limits the retrieval by specifying both a team (Barcelona again) and a player of that team (Messi) that scores the goal.

Query-4 is a tough one, in that the user wants to see all kinds of punishments, which are not explicitly stated. In soccer domain, we have two types of punishments: yellow cards and red cards. Therefore, the system is expected to retrieve those cards to the user.

Query-5 is a more specific version of Query-4, where both the type of the punishment (Yellow Card) and the receiver of the card (Alex) are explicitly given.

Query-6 can also be tricky for the retrieval system. It should retrieve all the goals scored to a specific goalkeeper (Casillas). Since the UEFA narrations do not state explicitly to which goalkeeper a goal is scored, the system should deduce that information somehow.

Query-7 is similar to Query-4. It requires the system to find all kinds of negative moves of a player (Henry). There are several negative moves in soccer domain, such as foul, off-side, warning and punishments.

Query-8 consists of a single player name (Ronaldo), which means that the user wants to see all the events involving Ronaldo. The only problem with this query can be the false positives

caused by those narrations that contain the word “Ronaldo”, while Ronaldo has nothing to do with the mentioned event. For example the sentence “The coach should take some precautions to stop Ronaldo” can cause a false positive.

Query-9 can be used to retrieve all the saves done by the goalkeeper of a team (Barcelona). “Saving” is the act of preventing a goal by the goalkeeper. The tricky part of this query is that the name of the goalkeeper is not given explicitly, while the narrations contain only the name of the goalkeeper as in the sentence “Casillas makes a save”.

Query-10 can be used to find shoots delivered by defence players. However, UEFA narrations do not contain the positions of players. So, the system should find out that implicit information in order to retrieve correct events.

5.2.2 An Example Average Precision (AP) calculation

The main part of our evaluation process is calculating the AP values of all queries. These AP values are then averaged over the queries to find MAP values of each index. Therefore it is important to clarify this process with an example calculation.

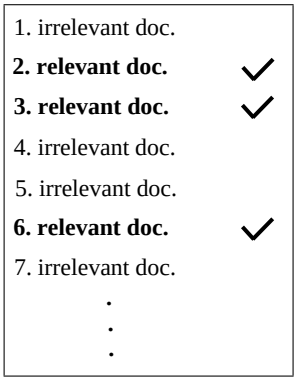


Figure 5.1: An Example Retrieval Scenario

Suppose that we have a query that should retrieve 3 documents (i.e. the set of relevant documents is of size 3). After executing the query, assume that we get the results shown in Figure 5.1. One of the related documents is returned in the 2nd place of the list, another related document is returned in the 3rd place and the final document is returned in the 6th place. Using this information we can calculate the AP value as shown in 5.1.

$$AP = \frac{1/2 + 2/3 + 3/6}{3} = 0.55 = 55\% \tag{5.1}$$

In this equation, 1/2 is the ”precision at recall level 2” and 2/3 is the ”precision at recall level 3”, etc. Note that the AP value will be 100% only if all the relevant documents are retrieved on top of the list.

5.3 Analysis of Results

In this section we give the detailed analysis of the evaluation results. For the purpose of clearness, first we analyze the queries in groups and then give the full results in Table 5.5. So, the improvements gained with basic extraction, domain-specific information extraction and inference can be easily understood.

5.3.1 Effects of Basic Extraction

The performance of `BASIC_EXT` has a great importance as it shows how the free-text search can be improved using the basic structured information in the crawled pages. To see the effects of basic extraction, let us consider the first three queries (Table 5.2). There is a considerable difference between `TRAD` and the other indices. The reason is that UEFA narrations use the phrase “P scores!” when the player P scores a goal. Since they omit the word “goal” in narrations, the traditional index is not able to retrieve all the goals with the keyword query: “goal”. However, the crawled data contains the goal information in a structured way and we can index it as a document with its `eventType` field filled as “goal”. Thus, the improved index can answer both the queries “goal” and “scores” successfully. That is the reason why `BASIC_EXT` and the rest have very high precision rates. The same argument is valid for the queries 5 and 6.

5.3.2 Effects of Domain-Specific Information Extraction

The improvement provided by the information extraction module can be seen clearly by looking at the difference between `BASIC_EXT` and `FULL_EXT` in 9th and 10th queries (Table 5.3). The difference stems from the extracted events such as shoots and goalkeeper saves.

Table 5.2: Effects of Basic Extraction

	TRAD		BASIC_EXT	
Q-1	0.5/35	1.4%	35/35	100%
Q-2	0.4/7	5.7%	5.3/7	75.7%
Q-3	0.7/3	23.3%	3/3	100%
Q-4	0/43	0%	0/43	0%
Q-5	1.1/2	55%	2/2	100%
Q-6	0.1/9	1.1%	5.7/9	63.3%
Q-7	2.2/7	31.4%	1.9/7	27.1%
Q-8	7.9/11	71.8%	8.6/11	78.1%
Q-9	5.1/8	63.7%	4.5/8	56.2%
Q-10	0/83	0%	0/83	0%

Table 5.3: Effects of Domain-Specific Information Extraction

	BASIC_EXT		FULL_EXT	
Q-1	35/35	100%	35/35	100%
Q-2	5.3/7	75.7%	5.3/7	75.7%
Q-3	3/3	100%	3/3	100%
Q-4	0/43	0%	0/43	0%
Q-5	2/2	100%	2/2	100%
Q-6	5.7/9	63.3%	5.6/9	62.2%
Q-7	1.9/7	27.1%	2.3/7	32.8%
Q-8	8.6/11	78.1%	8.5/11	77.2%
Q-9	4.5/8	56.2%	6.3/8	78.7%
Q-10	0/83	0%	21.9/83	26.4%

5.3.3 Effects of Inference

The improvements stemming from the inference can be observed by looking at the queries 4, 7 and 10. In these queries, `FULL_INF` index performs much better than other indices, because it contains additional information due to ontological inference and classification. For example, the 4th query exploits the inferred knowledge about the fact that red cards and yellow cards are also known as punishments. Similarly, the 10th query benefits from the inferred defence players through classification. Finally, the 7th query uses the knowledge obtained from the property hierarchies defined in the ontology. This means, the system can recognize the properties such as `actorOfMissedGoal`, `actorOfOffside`, and `actorOfRedCard` as `actorOfNegativeMove`. Moreover, in the 6th query, we can see the effect of Jena rules. Here, according to one of the rules we defined, we can infer the implicit knowledge of which goal is scored to which goalkeeper, even if that knowledge does not exist explicitly.

Table 5.4: Effects of Inference

	FULL_EXT		FULL_INF	
Q-1	35/35	100%	35/35	100%
Q-2	5.3/7	75.7%	5.3/7	75.7%
Q-3	3/3	100%	3/3	100%
Q-4	0/43	0%	43/43	100%
Q-5	2/2	100%	2/2	100%
Q-6	5.6/9	62.2%	9/9	100%
Q-7	2.3/7	32.8%	6.3/7	90.0%
Q-8	8.5/11	77.2%	7.4/11	75.9%
Q-9	6.3/8	78.7%	7.5/8	93.7%
Q-10	21.9/83	26.4%	81.4/83	98.1%

5.4 Worst Case

If we look at the query 8, we can see that all the four indices perform nearly the same. The reason is that, it is a simple query with a single player name (ronaldo). So, it does not contain much information that the semantic indexing can make use of. However, even in such queries, the performance does not drop below the traditional approach, because the full-text narrations are not discarded but preserved in a separate field. In other words, our approach guarantees at least the performance of the traditional approach in the worst case.

5.5 Summary of Evaluation

Our evaluation results show that starting from the `BASIC_EXT`, each index makes a solid improvement over its predecessor and we ultimately reach the desired performance in `FULL_INF`. This can be seen clearly by looking at the Mean Average Precisions in Table 5.6. Note that the performances of `BASIC_EXT` and `FULL_EXT` are also satisfactory. Especially the huge gap between the `TRAD` and `BASIC_EXT` is important, because it shows that even the basic information provided in the crawled data can make a great difference. However, when the queries get more and more complex, we need domain-specific information extraction, rules and inference to handle them. So, with this framework, we provide a set of opportunities for the developer to tweak their system according to the user needs. In any case, the framework guarantees to provide the same scalability and user-friendliness.

Having showed the improvements over the traditional methods, we provide more experiments in the next chapter to compare our system against simple query expansion methods.

Table 5.5: Full Evaluation Results (Average Precision)

	TRAD		**BASIC_EXT**		**FULL_EXT**		**FULL_INF**	
Q-1	0.5/35	1.4%	35/35	100%	35/35	100%	35/35	100%
Q-2	0.4/7	5.7%	5.3/7	75.7%	5.3/7	75.7%	5.3/7	75.7%
Q-3	0.7/3	23.3%	3/3	100%	3/3	100%	3/3	100%
Q-4	0/43	0%	0/43	0%	0/43	0%	43/43	100%
Q-5	1.1/2	55%	2/2	100%	2/2	100%	2/2	100%
Q-6	0.1/9	1.1%	5.7/9	63.3%	5.6/9	62.2%	9/9	100%
Q-7	2.2/7	31.4%	1.9/7	27.1%	2.3/7	32.8%	6.3/7	90.0%
Q-8	7.9/11	71.8%	8.6/11	78.1%	8.5/11	77.2%	7.4/11	75.9%
Q-9	5.1/8	63.7%	4.5/8	56.2%	6.3/8	78.7%	7.5/8	93.7%
Q-10	0/83	0%	0/83	0%	21.9/83	26.4%	81.4/83	98.1%

Table 5.6: Final Results (Mean Average Precision)

TRAD	**BASIC_EXT**	**FULL_EXT**	**FULL_INF**
25.34%	60.04%	65.3%	93.34%

CHAPTER 6

FURTHER ANALYSIS AND DISCUSSION

6.1 Comparison With Query Expansion

In Section 5 we have compared our solution with traditional approaches and observed a remarkable improvement. However, one can still ask whether this result could be achieved by using only the query expansion methods. Therefore, a final experiment showing the difference between our solution and the query expansion methods is needed. To fill this gap we implemented a prototype for the query expansion which uses the domain terms to extend queries. For example, a query containing the word "goal" is expanded with the verbs "score", "miss" and their derivatives. Ontological information is also used for query expansion, thus the query "punishment" is augmented with its subclasses such as "yellow card" and "red card" as well as the verb "book"[†] and its derivatives. The expanded queries are run directly on free-text, so we can clearly see the improvements stemming from the query expansion and compare it with our solution.

Table 6.1: Comparison With Query Expansion

	TRAD	**QUERY_EXP**	**FULL_INF**
Q-1	1.4%	30.1%	100%
Q-2	5.7%	16.4%	75.7%
Q-3	23.3%	49.0%	100%
Q-4	0%	63.6%	100%
Q-5	55%	51.5%	100%
Q-6	1.1%	11.5%	100%
Q-7	31.4%	27.16	90.0%
Q-8	71.8%	71.8%	75.9%
Q-9	63.7%	62.5%	93.7%
Q-10	0%	4.3%	98.1%

[†]the verb "book" means "kart göstermek" in Turkish

The results of the experiment are shown in Table 6.1. At a glance, we can easily say that the performance of the query expansion method resides between the traditional approach and the semantic indexing as we expected. The effects of the query expansion can be seen clearly by looking at the queries 1, 2, 3 and 4. The first three queries are improved by the expansion with the term "scores". The huge increase in the 4th query stems from the ontological expansion of the query term "punishment". Still, the performance is not close to our solution with semantic indexing. The rest of the queries are not much affected by the query expansion method due to the absence of suitable expansion terms. Some queries are even deteriorated by this method because of the false positives introduced by the extra query terms.

So, we conclude that although the query expansion method can improve the performance of traditional approaches, it cannot exceed the performance of semantic indexing, because it is unable to capture the actual semantics of the query words.

6.2 Adding Phrasal Expression Support

We have mentioned in Section 4.6 that the ambiguity problem is the most important issue in keyword-based search interfaces. Ambiguities arise usually in two forms: lexical and structural. Lexical ambiguities are caused by homonyms and structural ambiguities occur when a sentence or phrase imply more than one meaning due to the multiple assignments of the same word. We can solve the lexical ambiguities only if the information extraction module recognizes them since we did not implement a word disambiguation method. Structural ambiguities, however, can be solved in the semantic indexing phase. For demonstration purposes, we solve simple structural ambiguities by adding phrasal expression support to our current implementation.

Suppose a user tries to find the fouls that Alex made to Ronaldo and types the query "foul Alex Ronaldo". The system may also retrieve the fouls by Ronaldo to Alex because both of the players can act either as a subject or an object. To solve this structural ambiguity, we introduce simple phrasal expressions such as "to X", "by X", "of X", so that the system can understand the intended object and subject of the query. The cost of the implementation was negligible as it is achieved just by adding two new fields, one for the subject and one for the object. The content of each field is the concatenation of the name of the object or subject with

the corresponding preposition.

We compared the performance of the new index (`PHR_EXP`) with the original one (`FULL_INF`). We prepared three artificial queries for this purpose. The first query measures the subject discrimination performance (Daniel is the maker of foul). The second query adds an object (Florent) to the first query and finally the object and subject positions are swapped in the third query. The results are shown in Table 6.2. Note that the old index has difficulties with handling the ambiguities. It cannot discriminate which player is the object and which player is the subject of the query, but constantly assumes Florent as the subject player without any reason. The new index, on the other hand, can successfully discriminate the object and the subject in every condition.

Table 6.2: Effects of Phrasal Expressions

Query	**FULL_INF**	**PHR_EXP**
Foul by daniel	48.2%	100%
Foul by daniel to florent	47.7%	100%
Foul by florent to daniel	100%	100%

6.3 Discussion

In this thesis, we show how we achieve semantic querying without compromising from scalability and user-friendliness. There is another important benefit of our approach that we did not cover in this thesis, which is flexibility, i.e. how easily the system responds to data updates or modifications. We claim that the use of the semantic index as an upper layer above ontology makes the knowledgebase much more flexible.

Today, most of the semantic applications use ontologies as the main data structure of the system, i.e. all the heavy read/write operations are run directly over the ontology instances. However ontologies and ontology instances are not meant to be used in such conditions. By definition, they are strict, formal representations of knowledge and assertions. In other words, they are immutable by nature and are not optimized for rapid changes or updates. Therefore we recommend an efficient secondary data storage for these tasks, such as semantic indexing as our solution does.

To illustrate how the semantic indexing improves the flexibility, we can give the following example. Suppose we want to add support for another language in the query interface, so

that we can query the same knowledgebase with two different languages. To achieve this with ontology instances, one has to either duplicate the instances for the second language or duplicate the properties with the translated values. Either solution is impractical when the number of instances is too large. With the semantic indexing, however, it is as easy as adding the translated value next to its original value for each field. Expanding the index terms with WordNet synonyms, natural language phrases or even word stems are other examples that can be achieved easily with semantic indexing.

CHAPTER 7

IMPLEMENTATION

In this chapter, we briefly mention about an example application that we implemented using the framework described in this thesis. Then, we describe the tools that we used for the implementation.

7.1 Realization of the Framework

We followed a service-oriented approach to realize the system and implemented two interfaces to control it: a main control interface and a query interface. These interfaces interact with several services, such as crawler, information extractor, Solr and Protege Server to accomplish user requests. The overall architecture can be seen in Figure 7.1.

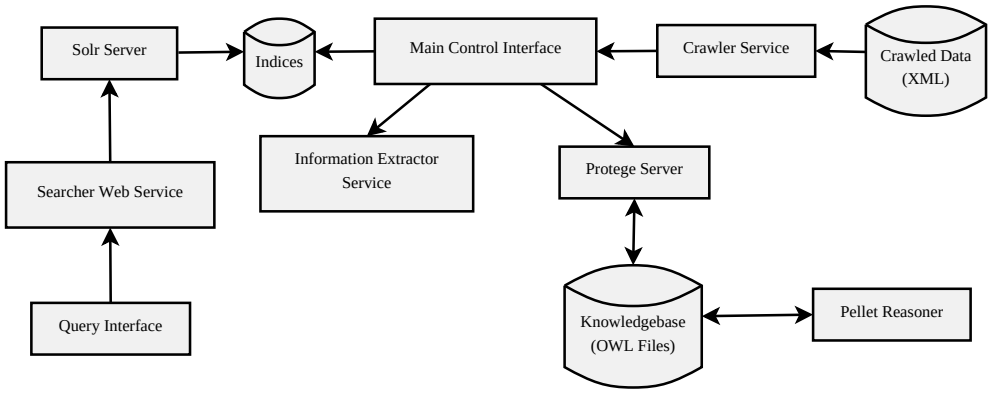


Figure 7.1: Service-Oriented Architecture

7.1.1 Main Control Interface

The main control interface is designed to ease the task of adding new matches to the system. It takes the XML files created by the crawler as input. An example input can be seen in Appendix B. After selecting the match to add, one has to decide the index to which the match information will be stored and press the ”Add Match to Index” button. Depending on the selected index, the system will call various services such as ontology populator, information extractor, reasoner and indexer, so that the data becomes ready for searching. A screenshot of the main control interface can be seen in Figure 7.2.

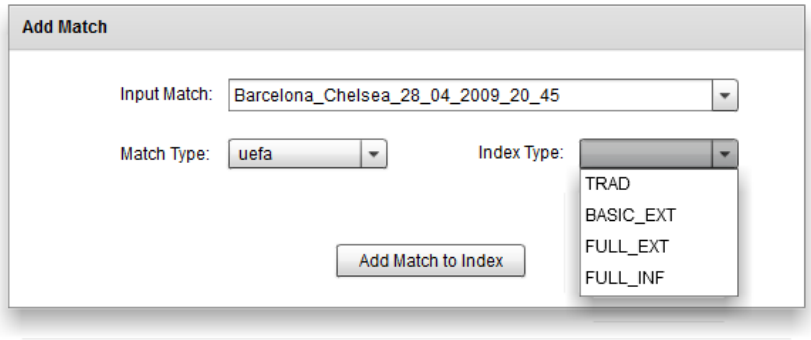


Figure 7.2: Add Match Screen

7.1.2 Query Interface

We want multiple users to be able to run their queries remotely and concurrently on the system. To achieve this, we implemented a web service that directs the user request to a Solr server that we describe in Section 7.2.4 and returns the obtained results back to the caller.

The user interacts with this service using a simple interface that we implemented using Adobe Flash Builder[1]. The results are presented to user within a table structure and ranked according to their matching score. The interface can also help the user to type the player and team names so that typos are prevented. A screenshot of the interface can be seen in Figure 7.3.

7.2 Tools Used for Implementation

In this section, we describe important tools that we used during the implementation.

[1] http://www.adobe.com/products/flashbuilder/ (last visited on 06/07/2010)

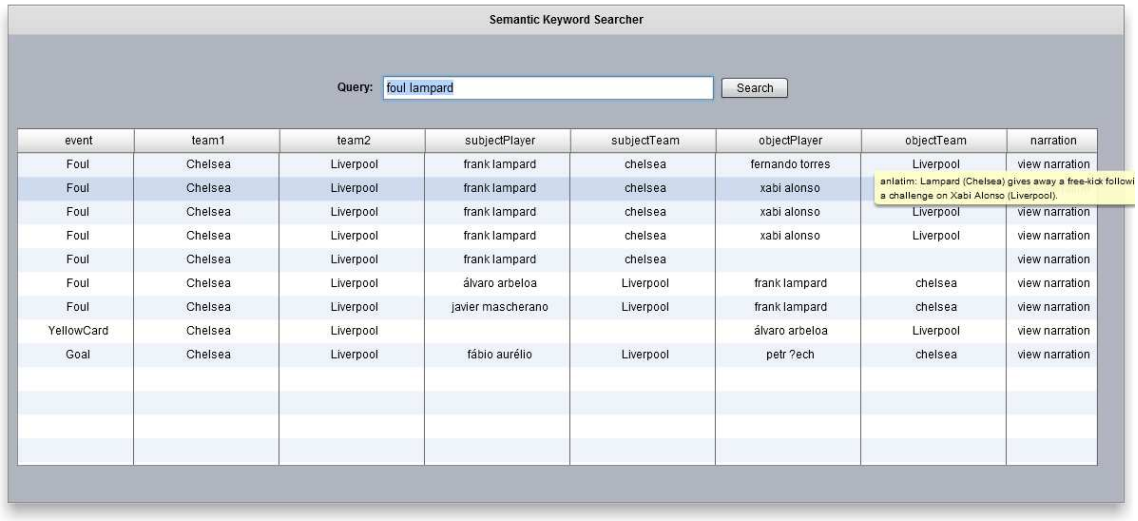


Figure 7.3: Query Interface

7.2.1 Protege Ontology Editor

Protege[2] is an open-source framework for knowledge-base modeling and ontology editing. It contains a frame-based and an OWL-based ontology editor. We used the latter for our ontology modeling task.

Protege OWL editor is actually an extension (plug-in) for the Protege, which allows building and editing ontologies in OWL ontology language. A screenshot of the main window of the editor can be seen in Figure 7.4. Using the OWL editor, we built a class hierarchy (taxonomy) for the soccer domain and defined the properties for individuals.

7.2.1.1 Classes and Restrictions

OWL classes are the concepts representing the actual individuals. For example the "Player" class represents all the player individuals such as Ronaldo, Messi, etc. and the "Goal" class represents all the goal events. These classes are organized in a hierarchical way, i.e. with subclass-superclass relations. This information is then used by the reasoning engine for classification and realization tasks.

Protege OWL editor also allows defining restrictions for classes. Restrictions are used to restrict the individuals belonging to a class. We can define three types of restrictions in OWL:

[2] http://protege.stanford.edu/ (last visited on 06/07/2010)

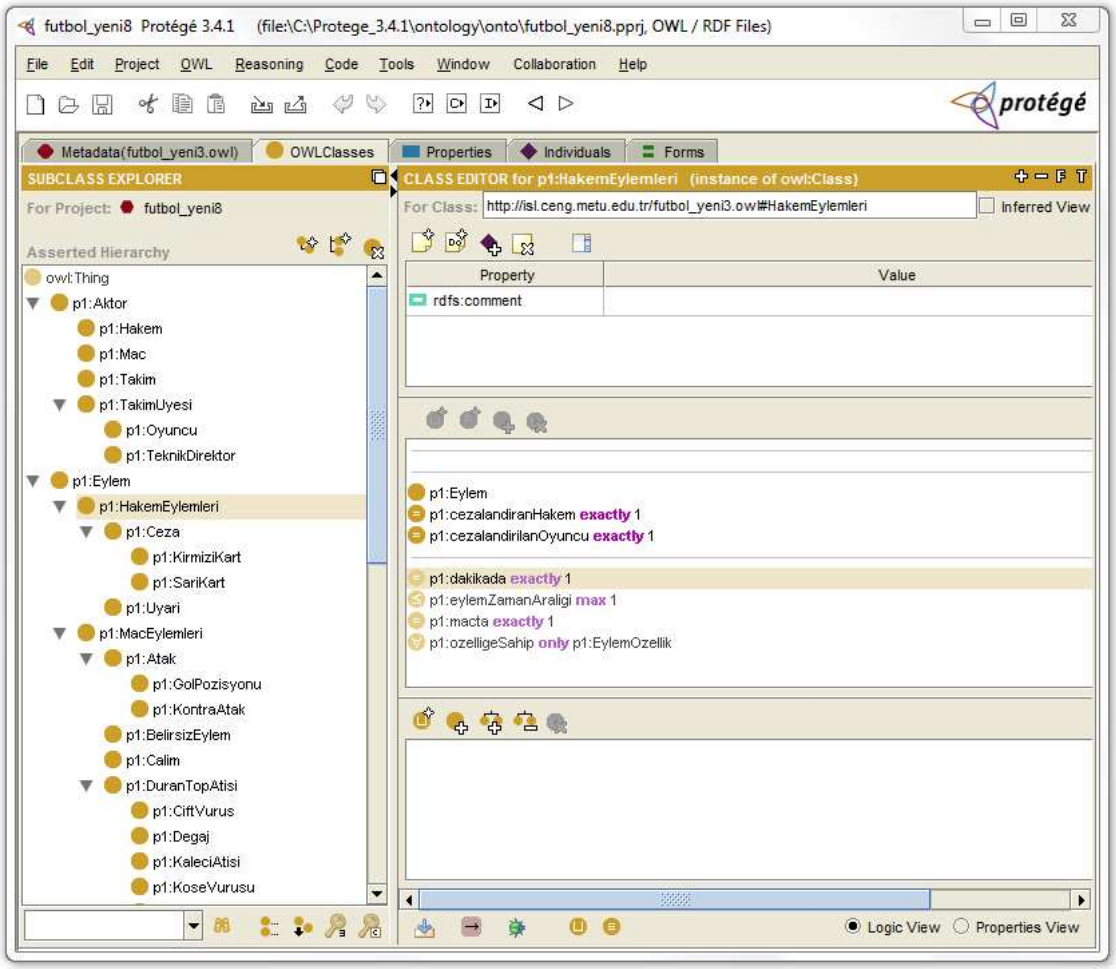


Figure 7.4: Protege Ontology Editor

quantifier restrictions, cardinality restrictions and hasValue restrictions.

Quantifier restrictions (∀ **and** ∃) can be used to select *only* those individuals having a specific property or those individuals having *at least* one specific property.

Cardinality restrictions are used to select those individual that have at least N, at most N, or exactly N relations of a given type. For example, we defined a cardinality restriction for the number of players of a team with the following definition in Protege: (”Team”, hasPlayer, min 11), which says a team has to have at least 11 players.

hasValue restriction is used to select the set of individuals that have at least one specified relation with a *specific value*. So, it is similar to ∃ restriction except that it specifies the value of the given relation.

7.2.1.2 Properties

Properties are binary relations defined between two ontology resources. There are two types of properties defined in OWL language: Object properties and Datatype properties.

Object properties are used to link two individuals. For example `hasTeam` property can be used to bind a player individual with a team individual.

Datatype properties are used to link an individual with a literal, such as string, integer, date, etc. For example `hasName` property can be used to bind a player individual with his name.

OWL properties can be defined as transitive, symmetric or functional. Properties can also have inverses. For example, we defined the `hasTeam` property as the inverse of the `hasMember` property. This allows the reasoning engine to infer a new statement if its inverse is known.

7.2.1.3 Individuals

OWL individuals (instances) are actual objects of a class. They are also called assertions. In our system, the individuals are created automatically during ontology population process; therefore we do not use Protege for this task.

7.2.2 Protege Server

Protege comes with a built-in client-server architecture to administrate OWL files remotely. We used this architecture for storing, accessing and modifying OWL files. Another benefit of this approach is that it allows concurrent access to the knowledgebase.

We organized the knowledgebase so that the common individuals such as player, teams, stadiums, etc. are completely separated from the terminology (T-Box). In other words, we keep separate OWL files for players, teams, stadiums, referees and the main ontology. Each match in the knowledgebase is also kept in a separate OWL file, which imports the common things stated above. The organization of the knowledgebase can be seen in Figure 7.5. Clients connect to Protege Server to retrieve or modify OWL content using some defined setter and getter functions that we implemented.

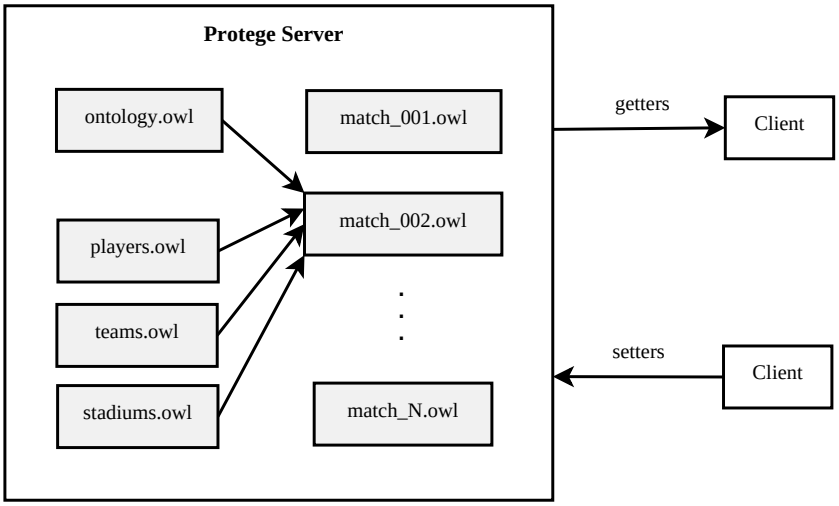


Figure 7.5: Knowledgebase Organization with Protege Server

Setter functions are required to add/remove individuals like players, teams, events, etc. or modify the properties of existing individuals. Getters are used to retrieve the values of properties of individuals such as the value of the `scorerPlayer` property of a `Goal` event. Here, clients can be a human or an automated agent, such as the IE module or indexer module.

7.2.3 Jena & Pellet

Jena is an open-source framework for building Java applications with semantic capabilities. It contains a comprehensive API to work with RDF, RDFS, OWL and SPARQL. Moreover, it comes with a powerful rule engine, which is one of the main reasons that we chose Jena in this thesis. Another reason is that it is fully compatible with the Pellet reasoner.

Pellet is a sound and complete OWL-DL reasoner. As mentioned in Section 4.5, we use Pellet for our classification, realization and verification tasks.

7.2.4 Solr

Solr[3] is an open-source enterprise search server based on Apache Lucene. It provides a highly scalable, distributed and easily manageable searching platform. Our primary reason for using Solr instead of Lucene was to implement our search engine as a web service that can be easily

[3] http://lucene.apache.org/solr/ (last visited on 06/07/2010)

managed, accessed and modified remotely and concurrently by many users and agents. Solr needs a servlet container such as Apache Tomcat[4] to run. The indices are the same as in the Lucene. The structure of our search service can be seen in Figure 7.6.

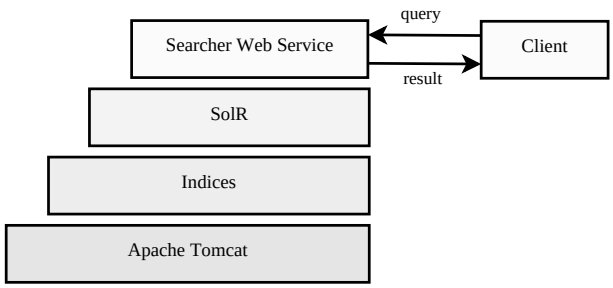


Figure 7.6: The Structure of Search Service

[4] http://tomcat.apache.org/ (last visited on 06/07/2010)

CHAPTER 8

CONCLUSION AND FUTURE WORK

We have presented a novel semantic retrieval framework and its application to soccer domain, which includes all the aspects of Semantic Web, namely, ontology development, information extraction, ontology population, inference, semantic rules, semantic indexing and retrieval. When these technologies are combined with the comfort of keyword-based search interface, we obtain one of the most user-friendly, high performance and scalable semantic retrieval system to the best of our knowledge. Evaluation results show that our approach can easily outperform both the traditional approach and the query expansion methods. Moreover, we observed that the system can answer complex semantic queries without needing formal queries such as SPARQL. Although we did not compare our solution with other semantic search approaches, we observed that the system can get close to the performance of SPARQL, which is the best that can be achieved with semantic querying. Finally, we show how the structural ambiguities can be resolved easily using semantic indexing.

The current implementation can be extended and improved in many ways. First of all, we are planning to enrich the knowledgebase to support multiple languages as we mentioned in Section 6.3. The performance will be further improved by implementing a word disambiguation module for lexical ambiguities. Finally, a mechanism that expands the index automatically according to the user feedback is one of our future goals.

REFERENCES

[1] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] Irene Celino, Emanuele Della Valle, Dario Cerizza, and Andrea Turati. Squiggle: An experience in model-driven development of real-world semantic search engines. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, *ICWE*, volume 4607 of *Lecture Notes in Computer Science*, pages 485–490. Springer, 2007.

[3] John Davies and Richard Weeks. Quizrdf: Search technology for the semantic web. *Hawaii International Conference on System Sciences*, 4:40112+, 2004.

[4] Julio Gonzalo, Felisa Verdejo, Irina Chugur, and Juan Cigarrin. Indexing with wordnet synsets can improve text retrieval. pages 38–44, 1998.

[5] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[6] H. S. Heaps. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., Orlando, FL, USA, 1978.

[7] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[8] Soner Kara, Özgür Alan, Orkunt Sabuncu, Samet Akpınar, Nihan K. Çiçekli, and Ferda N. Alpaslan. An ontology-based retrieval system using semantic indexing. In *1st International Workshop on Data Engineering meets the Semantic Web (DESWeb'2010) (co-located with ICDE'2010)*, November 2010.

[9] Yuangui Lei, Victoria S. Uren, and Enrico Motta. Semsearch: A search engine for the semantic web. In *EKAW*, pages 238–245, 2006.

[10] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[11] Rada Mihalcea and Dan Moldovan. Semantic indexing using wordnet senses. In *Proceedings of the ACL-2000 workshop on Recent advances in natural language processing and information retrieval*, pages 35–45, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[12] Gerard Salton. *The SMART Retrieval System; Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[13] Gerard Salton, Edward A. Fox, and Harry Wu. Extended boolean information retrieval. *Commun. ACM*, 26(11):1022–1036, 1983.

[14] Gerard Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[15] The semantic web wiki, http://semanticweb.org, 2008.

[16] Urvi Shah, Tim Finin, Anupam Joshi, R. Scott Cost, and James Matfield. Information retrieval on the semantic web. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 461–468, New York, NY, USA, 2002. ACM.

[17] Doruk Tunaoglu, Ozgur Alan, Orkunt Sabuncu, Samet Akpinar, Nihan K. Cicekli, and Ferda N. Alpaslan. Event extraction from turkish football web-casting texts using hand-crafted templates. In *In Proc. of Third IEEE Inter. Conf. on Semantic Computing (ICSC) (in press)*, 2009.

[18] Victoria Uren, Yuangui Lei, Vanessa Lopez, Haiming Liu, Enrico Motta, and Marina Giordanino. The usability of semantic search tools: A review. *Knowl. Eng. Rev.*, 22(4):361–377, 2007.

[19] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[20] Haofen Wang, Kang Zhang, Qiaoling Liu, Thanh Tran, and Yong Yu. Q2semantic: A lightweight keyword interface to semantic search. In *ESWC*, pages 584–598, 2008.

[21] Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. Spark: Adapting keyword query to semantic search. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, volume 4825 of *LNCS*, pages 687–700, Berlin, Heidelberg, November 2007. Springer Verlag.

Appendix A

SEMANTIC RULES

Assist Rule

```
noValue(?pass rdf:type pre:Assist)
(?pass rdf:type pre:Pass)
(?pass pre:passingPlayer ?passer)
(?pass pre:passReceiver ?receiver)
(?pass pre:inMatch ?match)
(?pass pre:inMinute ?minute)
(?goal pre:inMatch ?match)
(?goal pre:inMinute ?minute)
(?goal pre:scorerPlayer ?receiver)
makeTemp(?tmp)

-> (?tmp rdf:type pre:Assist)
   (?tmp pre:inMatch ?match)
   (?tmp pre:inMinute ?minute)
   (?tmp pre:passingPlayer ?passer)
   (?tmp pre:passReceiver ?receiver)
```

Goal To Goalkeeper Rule

```
noValue(?gol pre:goalToKeeper ?goalKeeper)
(?goal rdf:type pre:Goal)
(?goal pre:scorerPlayer ?scorer)
(?goal pre:scoredTeam ?team)
(?goalKeeper rdf:type pre:Oyuncu)
(?goalKeeper pre:position pre:GoalKeeper)
(?goalKeeper pre:playerOfTeam ?team)

-> (?scorer pre:goalToKeeper ?goalKeeper)
```

Subject Team Rule

```
noValue(?event pre:makingTeam ?team)
(?event rdf:type pre:Event)
(?event pre:makingPlayer ?player)
(?player pre:playerOfTeam ?team)

-> (?event pre:makingTeam ?team)
```

Object Team Rule

```
noValue(?event pre:madeTeam ?team)
(?event rdf:type pre:Event)
(?event pre:madePlayer ?player)
(?player pre:playerOfTeam ?team)

-> (?event pre:madeTeam ?team)
```

Own Goal Rule

```
noValue(?goal pre:isOwnGoal "true"^^xsd:boolean)
(?goal rdf:type pre:Goal)
(?goal pre:scorerPlayer ?scorer)
(?goal pre:scoredTeam ?scoredTeam)
(?scorer pre:playerOfTeam ?scoredTeam)

-> (?goal pre:isOwnGoal "true"^^xsd:boolean)
```

Steady Ball Rule

```
noValue(?goal pre:goalReason ?steadyBall)
(?goal rdf:type pre:Goal)
(?goal pre:inMatch ?match)
(?goal pre:inMinute ?goalMinute)
(?steadyBall rdf:type pre:SteadyBall)
(?steadyBall pre:inMatch ?match)
(?steadyBall pre:inMinute ?steadyBallMinute)
le(?steadyBallMinute, ?goalMinute)
difference(?goalMinute, ?steadyBallMinute, ?diff)
lessThan(?diff, 2)

-> (?gol pre:goalReason ?steadyBall)
```

Derby Rule

```
noValue(?mac pre:isDerby "true"^^xsd:boolean)
(?match rdf:type pre:Match)
(?match pre:hasTeam teams:Fenerbahce)
(?match pre:hasTeam teams:Galatasaray)

-> (?match pre:isDerby "true"^^xsd:boolean)
```

Transitivity Rule

```
(?x rdfs:subClassOf ?y)
(?a rdf:type ?x)

-> (?a rdf:type ?y)
```

Appendix B

CRAWLED DATA STRUCTURE

```
<match>
    <matchTimeLocation>
        <dateHour>
            <date>12/03/2007</date>
            <hour>20:45</hour>
        </dateHour>
        <stad>Anfield</stad>
    </matchTimeLocation>
    <team1>
        <name>Leverkusen</name>
        <score>1</score>
    </team1>
    <team2>
        <name>Real Madrid</name>
        <score>2</score>
    </team2>
    <stad>Hampden Park</stad>
    <referee>
        <name>Urs Meier </name>
    </referee>
    <team1coach>
        <name>Klaus Toppmöller</name>
    </team1coach>
    <team2coach>
```

```
        <name>Vicente Del Bosque</name>
    </team2coach>
    <team1players>
        <player>
            <number>1</number>
            <name>Jörg Butt</name>
        </player>
        <player>
            <number>6</number>
            <name>Boris Zivkovic</name>
        </player>
        <player>
            <number>10</number>
            <name>Yıldıray Baştürk¡/name¿
        </player>
        <player>
            <number>13</number>
            <name>Michael Ballack</name>
        </player>
        <player>
            <number>19</number>
            <name>Lucio</name>
        </player>
        <player>
            <number>23</number>
            <name>Thomas Brdaric</name>
        </player>
        <player>
            <number>25</number>
            <name>Bernd Schneider</name>
        </player>
        <player>
            <number>26</number>
```

```
            <name>Zoltán Sebescen</name>
        </player>
        <player>
            <number>27</number>
            <name>Oliver Neuville</name>
        </player>
        <player>
            <number>28</number>
            <name>Carsten Ramelow</name>
        </player>
        <player>
            <number>35</number>
            <name>Diego Placente</name>
        </player>
    </team1players>
    <team2players>
        ...
    </team2players>
    <team1subs>
        ...
    </team1subs>
    <team2subs>
        ...
    </team2subs>
    <basicEvents>
        <event>
            <min>8</min>
            <eventName>Goal</eventName>
            <player1No>7</player1No>
            <player1Name>Raúl González</player1Name>
            <team>Real Madrid</team>
        </event>
        <event>
```

<min>13</min>

<eventName>Goal</eventName>

<player1No>19</player1No>

<player1Name>Lucio</player1Name>

<team>Leverkusen</team>

</event>

<event>

<min>39</min>

<eventName>Substitution</eventName>

<player1No>23</player1No>

<player2No>12</player2No>

<player1Name>Thomas Brdaric </player1Name>

<player2Name>Dimitar Berbatov</player2Name>

<team>Leverkusen</team>

</event>

<event>

<min>45</min>

<eventName>Goal</eventName>

<player1No>5</player1No>

<player1Name>Zinédine Zidane</player1Name>

<team>Real Madrid</team>

</event>

<event>

<min>45 + 2</min>

<eventName>Yellow card</eventName>

<player1No>2</player1No>

<player1Name>Míchel Salgado¡/player1Name¿

<team>Real Madrid</team>

</event>

<event>

<min>61</min>

<eventName>Substitution</eventName>

<player1No>10</player1No>

<player2No>8</player2No>

<player1Name>Luís Figo ¡/player1Name¿

<player2Name>Steve McManaman</player2Name>

<team>Real Madrid</team>

</event>

...

</basicEvents>

<minBymin>

<event>

<minute>1’</minute>

<text>Real Madrid CF, hot favourites to lift the trophy tonight,

kick off in their famous all-white strip.

</text>

</event>

<event>

<minute>1’</minute>

<text>Lucio (Leverkusen) commits a foul after challenging Raúl (Real Madrid).

</text>

</event>

<event>

<minute>2’</minute>

<text>Schneider (Leverkusen) gives away a free-kick following a challenge

on Roberto Carlos (Real Madrid).

</text>

</event>

<event>

<minute>2’</minute>

<text>Bernd Schneider brings down Roberto Carlos inside the opening minute,

but Madrid waste the resulting free-kick and Leverkusen have a goal kick.

</text>

</event>

<event>

<minute>2’</minute>

```
            <text>Makelele (Real Madrid) fouls.</text>
        </event>
        ...
        <event>
            <minute>90'+7'</minute>
            <text>(Leverkusen) delivers the corner.</text>
        </event>
        <event>
            <minute>90'+7'</minute>
            <text>Berbatov (Leverkusen) has an effort on goal.</text>
        </event>
        <event>
            <minute>90'+7'</minute>
            <text>Leverkusen are throwing absolutely everything forward and Berbatov is only
                denied by a marvellous sprawling save by Casillas. Madrid hearts are firmly
                in mouths and they are struggling to hold on.
            </text>
        </event>
        <event>
            <minute>90'+7'</minute>
            <text>The final whistle sounds and Real Madrid CF have hung on to win the UEFA
                Champions League final, despite a brave fight by Bayer 04 Leverkusen. Zidane's
                wonderful volley proved to be the winner and Madrid congatulate each
                other on a rain-soaked Hampden Park pitch.
            </text>
        </event>
    </minBymin>
</match>
```