NATURAL LANGUAGE QUERY PROCESSING IN ONTOLOGY BASED MULTIMEDIA DATABASES

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

BY

FİLİZ ALACA AYGÜL

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

APRIL 2010

Approval of the thesis:

NATURAL LANGUAGE QUERY PROCESSING IN ONTOLOGY BASED MULTIMEDIA DATABASES

submitted by **FİLİZ ALACA AYGÜL** 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 Kesim Çiçekli
Supervisor, **Computer Engineering Dept., METU** ________________

Assoc. Prof. Dr. İlyas Çiçekli
Co-supervisor, **Computer Engineering Dept., BİLKENT** ________________

Examining Committee Members:

Assoc. Prof. Dr. Cem Bozşahin
Computer Engineering Dept., METU ________________

Assoc. Prof. Dr. Nihan Kesim Çiçekli
Computer Engineering Dept., METU ________________

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU ________________

Dr. Ayşenur Birtürk
Computer Engineering Dept., METU ________________

Özgür Alan
Orbim, TEKNOKENT, METU ________________

Date: **28.04.2010**

ABSTRACT

NATURAL LANGUAGE QUERY PROCESSING IN ONTOLOGY BASED MULTIMEDIA DATABASES

Aygül, Filiz Alaca
M.S., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Nihan Kesim Çiçekli
Co-Supervisor : Assoc. Prof. Dr. İlyas Çiçekli

April 2010, 92 pages

In this thesis a natural language query interface is developed for semantic and spatio-temporal querying of MPEG-7 based domain ontologies. The underlying ontology is created by attaching domain ontologies to the core Rhizomik MPEG-7 ontology. The user can pose concept, complex concept (objects connected with an “AND” or “OR” connector), spatial (left, right . . .), temporal (before, after, at least 10 minutes before, 5 minutes after . . .), object trajectory and directional trajectory (east, west, southeast . . . , left, right, upwards . . .) queries to the system. Furthermore, the system handles the negative meaning in the user input. When the user enters a natural language (NL) input, it is parsed with the link parser. According to query type, the objects, attributes, spatial relation, temporal relation, trajectory relation, time filter and time information are extracted from the parser output by using predefined rules. After the information extraction, SPARQL queries are generated, and executed against the ontology by using an RDF API. Results are retrieved and they are used to calculate spatial, temporal, and trajectory relations between objects. The results satisfying the required relations are displayed in a tabular format and user can navigate through the multimedia content.

Keywords: Natural Language Querying, Spatio-Temporal Querying, MPEG-7 Ontology, Link Parser, SPARQL

ÖZ

ONTOLOJİ TABANLI MULTİMEDYA VERİTABANLARINDA DOĞAL DİL SORGU İŞLEME

Aygül, Filiz Alaca
Yüksek Lisans, Bilgisayar Mühendisliği
Tez Yöneticisi : Doç. Dr. Nihan Kesim Çiçekli
Ortak Tez Yöneticisi : Doç. Dr. İlyas Çiçekli

Nisan 2010, 92 sayfa

Bu tez kapsamında, MPEG-7 tabanlı alan ontolojilerini anlamsal ve uzay-zamansal sorgulamak için bir doğal dil sorgu arayüzü geliştirilmiştir. Sistemde sorgulanan temel ontoloji, alan ontolojilerini Rhizomik MPEG-7 ontolojisine ekleyerek oluşturulur. Kullanıcı sistemde kavramsal, kompleks nesne (“VE”, “VEYA” ile bağlanmış birden fazla nesne), uzaysal (sağ, sol ...), zamansal (önce, sonra, en az 10 dakika önce, 5 dakika sonra ...), nesnesel yörünge ve yönsel yörünge (doğu, batı, güneydoğu ..., sağ, sol, yukarı ...) sorguları yapabilir. Bununla birlikte, kullanıcının negatif anlam taşıyan sorgu yapabilmesini sağlamak amacıyla, doğal dil sorgu cümlesindeki negatif anlamlar tespit edilir. Kullanıcı sisteme doğal dilde bir sorgu cümlesi girdiğinde, bu cümle link ayrıştırıcı kullanılarak çözümlenir. Sorgu tipine göre, sistemde önceden tanımlı olan kurallar kullanılarak, sorgu cümlesinden nesneler, nesne özellikleri, uzaysal, zamansal, ve yörüngesel ilişkiler, zaman filtresi ile zaman bilgileri çıkarılır. Çıkarılan bilgiler kullanılarak, doğal dil sorgu cümlesi, SPARQL sorgu cümlesine çevrilir. SPARQL sorgusu ontoloji üzerinde çalıştırılır. Elde edilen sorgu sonuçları, nesneler arasında uzaysal, zamansal, ve yörüngesel ilişkiyi hesaplamak için kullanılır. Sorgu cümlesinde sorulan ilişkiyi sağlayan sonuçlar kullanıcıya gösterilir. Kullanıcı, sorgu sonuçlarını kullanarak multimedia içeriği üzerinde gezinebilir.

Anahtar Kelimeler: Doğal Dil Sorgulama, Uzay-Zamansal Sorgulama, MPEG-7 Ontoloji, Link Ayrıştırıcı, SPARQL

To my dear dad

and

to my husband, İbrahim...

ACKNOWLEDGMENTS

I would like to express my appreciation and indebtedness to both my supervisor Assoc. Prof. Dr. Nihan Kesim Çiçekli and my co-supervisor Assoc. Prof. Dr. İlyas Çiçekli for their expert guidance, supportive and constructive approach throughout my masters study and their efforts during supervision of the thesis.

I would like to show my gratitude to my thesis jury members Assoc. Prof. Dr. Cem Bozşahin, Assoc. Prof. Dr. Ahmet Coşar, Dr. Ayşenur Birtürk, and Özgür Alan for reviewing and evaluating my thesis.

I also want to thank to Scientific and Technical Research Council of Turkey (TÜBİTAK) for its support to the project with grant number EEEAG 107E234.

I would also like to thank my supervisors from TÜBİTAK UEKAE - İLTAREN for their understanding and support during my academic studies.

I would also express my special thanks to my dear friend Yeliz Şimşek for her invaluable friendship, encouragement and support throughout my life. She put up with my whims much.

I owe much thanks to my family, making me who I am now with their love, trust, and support throughout my life.

I am also grateful to my dear dad for bringing me up with his endless love and care, and his encouraging and supportive approach throughout my education life. I wish he had been with us.

Finally, my deepest thanks are to my husband, İbrahim, for his endless patience, encouragement and support during this thesis. I cannot forget his altruism.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

API	Application Programming Interface
GUI	Graphical User Interface
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
KB	Knowledge Base
MBR	Minimum Bounding Rectangle
NL	Natural Language
NLP	Natural Language Processing
OWL	Web Ontology Language
POS	Part of Speech
RDF	Resource Description Framework
SeRQL	Sesame Rdf Query Language
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

1.1 Overview

Due to the increasing number of video files, storing and querying semantic contents of multimedia data becomes more important. In order to standardize the description of semantic contents of video files, MPEG-7 standard is proposed. It is implemented by XML-Schemas; as a result, it does not have a reasoning capability. Also, its XML based architecture prevents interoperability between multimedia management systems [40]. In order to overcome these deficiencies of MPEG-7 standard, four MPEG-7 based multimedia ontologies, represented in OWL, are developed [41]: Jane Hunter ontology, Tsinaraki (DS-MIRF) ontology, Rhizomik ontology, and COMM ontology. In this thesis, the Rhizomik MPEG-7 ontology is used as the underlying ontology where the domain ontologies can be attached.

Spatio-temporal querying can be useful in many fields such as surveillance, automation, health and medical systems [36]. For example, in medical field, it can be used to recognize the cancer cells. Another example can be given from surveillance videos: By querying spatio-temporal properties of objects, the suspects can be found in a burglary event. Also, it can be used to analyze tennis strokes in sport videos [36].

In order to query the data stored in ontologies or knowledge bases, there exists some formal query languages such as SPARQL [11] and SeRQL [2]. Since these query languages have quite complex syntax and require a good understanding of both the underlying data schema and the language syntax, it is not acceptable by casual end users [39]. Five different methods are proposed to ease the query construction over ontologies: keyword-based, graph-based, form-based, view-based, and natural language based [29].

End users are most familiar with keyword-based approach due to the search engines like Google. SPARK [44], Q2Semantic [42], SemSearch [34], and GuNQ [20] construct queries using keyword-based methods. The common problem with these methods is that they produce lots of results and it is hard to choose the most relevant result.

In the graph-based methods [37, 25, 21], the user construct the SPARQL query by drawing an RDF graph. This approach is not feasible for end users either; because the user should be familiar with the SPARQL syntax and the RDF Schema.

Form-based query interfaces are easy to use for end users. The form displays the ontology entities in a user friendly manner but the details of the underlying ontology are hidden from the user. The query is being generated while the user fills the form. This approach is suitable for only small-scale ontologies because for each query type to be answered, a form should be created.

By using view-based interfaces [18, 22, 28], the user can browse an ontology and select the subjects, the properties and the objects to generate the query. It can be a little tricky and time-consuming job for non-experienced users.

The most sophisticated method for querying semantic data is natural language interfaces [19, 31, 32, 39]. According to [30], 48 end users evaluate four types of query language interfaces; namely keyword-based, natural language-based, menu-guided, and graph-based. They judged that the full natural language is the most functional and most-liked query interface. They found keyword-based interface too relaxed. They also concluded that menu-guided approach is too restrictive and it provides few sentence structures. They found the graph based interface too complex, and they need much training to construct a query.

In this thesis, a natural language interface is developed for querying ontologies which contain multimedia data. The underlying ontology is created by attaching domain ontologies to the core Rhizomik MPEG-7 ontology. The ontology stores the semantic content of a video file. For each object in the ontology, the frame intervals during which the object is seen in the video, and the minimum bounding rectangles which specify the position of the object are annotated.

By using the natural language (NL) interface, the user can pose concept, complex concept (objects connected to each other with an “AND” or “OR” connector), spatial (left, right . . .), temporal (before, after, at least 10 minutes before, 5 minutes after ...), object trajectory and directional trajectory (east, west, southeast ..., left, right, upwards ...) queries. Also, the system can capture the negative meaning in the user input. When the user enters a natural language input, it is parsed with the link parser and the query type is found by using the predefined rules. After finding the query type, the object(s), the attribute(s), the relation between object(s), and the start and end times are extracted from the user input by using the rules. Then, SPARQL queries are generated. By using SemWeb which is an RDF API, the queries are executed against the ontology. Afterwards, the query results are used to calculate the spatial, temporal, and trajectory relations. Finally, the results satisfying the relations are displayed to the user.

1.2 Contribution of the Thesis

This thesis is conducted as a part of a research project partially supported by TUBITAK (TUBITAK - EEEAG 107E234). In this project, a multimedia information management system is being developed for managing personal videos. There are some studies carried out for this project. In [40], a framework is developed and the semantic annotation of the multimedia content is achieved. Also MPEG-7 based ontologies are integrated to the system. However, its querying capability is limited; it only supports concept querying. In order to extend the querying capabilities of the framework, Şimşek [27] added spatio-temporal and trajectory query types to the system. As he states, adding a natural language query interface to the system would be a better solution and will enhance the system's querying ability. For this reason, a NL query interface is developed which enables the users to pose semantic, spatio-temporal, and trajectory queries in English.

The contributions of this thesis can be listed as follows:

- The main contribution of this thesis is providing NL query interface for concept, spatio-temporal, and trajectory queries over the MPEG-7 based domain ontologies.
- The proposed solution is a generic solution; any domain ontology can be queried with this system.
- MPEG-7 based domain ontologies are used in this study, as a result we benefit from the reasoning mechanism and handle the semantic interoperability problem.

- The user can pose queries to the system without struggling with the complex syntax of formal ontology query languages.
- The details of the underlying ontology are hidden from the user; this provides some level of abstraction.
- The semantic objects can be queried with their attributes.
- Complex concept querying is provided to the user.

1.3 Organization

The rest of the thesis is organized as follows. Chapter 2 gives background information about MPEG-7 ontology, RDF and SPARQL. In Chapter 3 we present the related work. The natural language interfaces for both ontologies and video databases are discussed. The main differences between these systems and this thesis are given. Chapter 4 describes the system architecture and presents the query types supported by the system. In Chapter 5, we explain the details of mapping NL input to SPARQL query. Chapter 6 discusses the details of query processing. Finally, Chapter 7 concludes the thesis and discusses some possible future works.

CHAPTER 2

BACKGROUND

2.1 MPEG-7 Ontology

MPEG-7, developed by **M**oving **P**icture **E**xperts **G**roup (MPEG), is an ISO / IEC standard for describing the structural and semantic features of multimedia content [41]. These features include visual features (e.g. texture, camera motion) and audio features (e.g. spectrum, harmonicity), or semantic objects (e.g. places, actors, events, objects).

MPEG-7 is implemented by XML Schemas. As a result, a great part of the semantics remains implicit [26]. In order to overcome the problem of lacking formal semantics in MPEG-7, four MPEG-7 based multimedia ontologies, represented in OWL, are developed [41]: Jane Hunter ontology, Tsinaraki (DS-MIRF) ontology, Rhizomik ontology, and COMM ontology. Rhizomik MPEG-7 ontology is used as the underlying ontology in this thesis.

In order to capture the formal semantics, there are several attempts to move metadata from the XML Schema to the Semantic Web. The Rhizomik ontology was proposed by Garcia and Celma in 2005, in order to map MPEG-7 Schema to OWL. It is based on a generic XML Schema to OWL mapping and provides a complete and automatic mapping of the whole MPEG-7 standard to OWL [26]. XSD2OWL mapping is preferred as the translation approach. It aims to capture the implicit semantics of XML Schema. The mapping is done by translating the XML Schema constructs into the OWL constructs. The transformation details can be seen in Table 2.1 [26].

The XSD2OWL mapping converts the MEPG-7 XML Schemas into MPEG-7 ontologies that make the semantics of the XML Schemas explicit. The MPEG-7 ontology has 2372 classes and 975 properties, and it is intended to be an upper-ontology where other domain ontologies

Table 2.1: XSD2OWL translation from the XML Schema Constructs to OWL Constructs

XML Schema	OWL	Shared informal semantics
element | attribute	rdf:Property owl:DatatypeProperty owl:ObjectProperty	Named relation between nodes or nodes and values
element@substitutionGroup	rdfs:subPropertyOf	Relation can appear in place of a more general one
element@type	rdfs:range	The relation range kind
complexType | group | attributeGroup	owl:Class	Relations and contextual restrictions package
complexType//element	owl:Restriction	Contextualized restriction of a relation
extension@base | restriction@base	rdfs:subClassOf	Package concretizes the base package
@maxOccurs | @minOccurs	owl:maxCardinality owl:minCardinality	Restrict the number of occurrences of a relation
sequence | choice	owl:intersectionOf owl:unionOf	Combination of relations in a context

can be added [26].

2.2 Resource Description Framework

The **R**esource **D**escription **F**ramework (RDF) is a general-purpose language for conceptual description or modeling of information in web resources [7]. In the RDF data model, statements about resources, especially web resources, are included. An RDF statement is a triple which is composed of a subject, a predicate and an object. An RDF graph is a collection of RDF statements.

The subject of an RDF statement denotes the resource. It is either a **U**niform **R**esource **I**dentifier (URI) or a blank node. The predicate is a URI which represents a relationship between the subject and the object. The object can be a URI, blank node or a Unicode string literal [7].

Some example RDF statements are given as the N-triples form of RDF in the following:

```
<http://iltaren.org/users/JohnSmith> <http://iltaren.org/userProp/fullName> "John Smith".
<http://iltaren.org/users/JohnSmith> <http://iltaren.org/userProp/company> "Iltaren".
<http://iltaren.org/users/JohnSmith> <http://iltaren.org/userProp/position> "Researcher".
```

“http://iltaren.org/users/JohnSmit” represents a particular resource, which has a full name “John Smith”, works for company Iltaren in the position of “Researcher”. The RDF / XML form of the statements are as follows:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:up="http://iltaren.org/userProp/">
    <rdf:Description rdf:about="http://iltaren.org/users/JohnSmith">
        <up:fullName>John Smith</up:fullName>
        <up:company>Iltaren</up:company>
        <up:position>Researcher</up:position>
    </rdf:Description>
</rdf:RDF>
```

2.3 SPARQL

SPARQL is a RDF query language, stands for the recursive acronym **S**PARQL **P**rotocol *a*nd *R*DF **Q**uery **L**anguage [11]. It is developed by RDF **D**ata **A**ccess **W**orking **G**roup (DAWG) of the World Wide Web Consortium (W3C) as a query language and a protocol for accessing RDF [11, 12]. It is considered a key semantic web technology and became an official W3C Recommendation in 2008 [11].

SPARQL is “data-oriented” which means that there is no inference mechanism. It can only query the information that resides in the RDF graph. To include an inference mechanism in the queries, some SPARQL implementations can be used such as ARQ and SemWeb. ARQ is the implementation of SPARQL query language for Jena, and it provides OWL reasoning [12]. SemWeb is an RDF library written in C# for Microsoft’s .NET 1.1/2.0 [8]. It includes RDFS reasoning and general reasoning with the Euler engine.

SPARQL by Examples:
The basic SPARQL query has two parts:

- The variables which will appear in the query result are declared in the SELECT clause.
- In the WHERE clause, the graph patterns are stated. These triple patterns are matched against the triples in the RDF graph.

Basic SPARQL Query:

The following example shows a SPARQL query to find the company of a person from the given RDF graph.

```
SELECT ?company
WHERE
{
    <http://iltaren.org/users/JohnSmith>
    <http://iltaren.org/userProp/company> ?company .
}
```

In this example, the subject and the predicate have fixed values, and the object is a variable. As a result, the pattern matches any triples with these subject and predicate values, and solutions for company variable are generated.

String Matching:

SPARQL provides a method to restrict the values in a solution. It has an operation to test strings, based on regular expressions [12]. The operation is similar to that of "LIKE" operator in SQL.

```
PREFIX up: <http://iltaren.org/userProp/>
SELECT ?name
WHERE
{
    ?person up:fullName ?name .
    FILTER regex(?name, "j", "i")
}
```

This example query finds the full names containing "j" or "J". A case-insensitive pattern match is done by the usage of the flag "i". It is an optional argument. If it is not stated in the query, the full names with only "j" will be returned.

Testing Values:

The result of a query can be restricted by using a filter on the value of a variable. The following query finds the names of people who are older than 30.

```
PREFIX up: <http://iltaren.org/userProp/>
SELECT ?name
WHERE
{
    ?person up:fullName ?name .
    ?person up:age ?age.
    FILTER regex(?age > 30)
}
```

Solution Modifiers:

A set of solutions is produced as the result of query. It can be modified in one of the following ways:

- Projection: Only selected variables are returned in the result set.
- OFFSET: A set of solutions is returned from the beginning of the specified start index (i.e the offset).
- LIMIT: The number of solutions to be returned is specified.
- ORDER BY: The result set is sorted according to the given variables.
- DISTINCT: One row for one combination of variables and values is returned in the result set.

CHAPTER 3

RELATED WORK

There are several approaches to access the data contained in ontologies. These are keyword-based, natural language based, graph based, form based and view based approaches [29]. In this thesis a natural language query interface is developed for ontology based multimedia databases. Therefore, throughout this chapter, existing natural language interfaces for both ontologies and video databases are discussed. A comparison between the existing work and this thesis is given.

3.1 Natural Language Interfaces Over Ontologies

3.1.1 Ginseng

Ginseng is a **g**uided **i**nput **n**atural language **s**earch **eng**ine for querying ontologies [19]. When it is started, it loads the ontologies from a search path, and constructs its own vocabulary from these ontologies. While the user is typing the query, it guides the user with possible completions of the sentence using the constructed vocabulary. Thus, it prevents non-grammatical queries. After the user enters the input sentence, Ginseng converts it to a SPARQL query, and executes the query against the ontology. The generated SPARQL query and the results of the query are displayed to the user.

Ginseng is composed of three components: a multi-level grammar, an incremental parser, and an ontology-access layer [19]. Its architecture is given in Figure 3.1. Ginseng's grammar has two parts: static grammar and dynamic grammar. The static grammar includes rules that define the basic sentence structure for English questions. There are about 120 domain independent rules in the static part. The dynamic grammar contains rules which are generated for each class, individual, object property and data type property of the loaded ontologies. The static grammar and the full grammar form the full grammar. Full grammar has the information of the set of all parsable sentences and information about how to map the user input to SPARQL queries. The incremental parser uses the full grammar to construct the SPARQL query from the complete parse tree. Finally, Jena SPARQL engine executes the query.

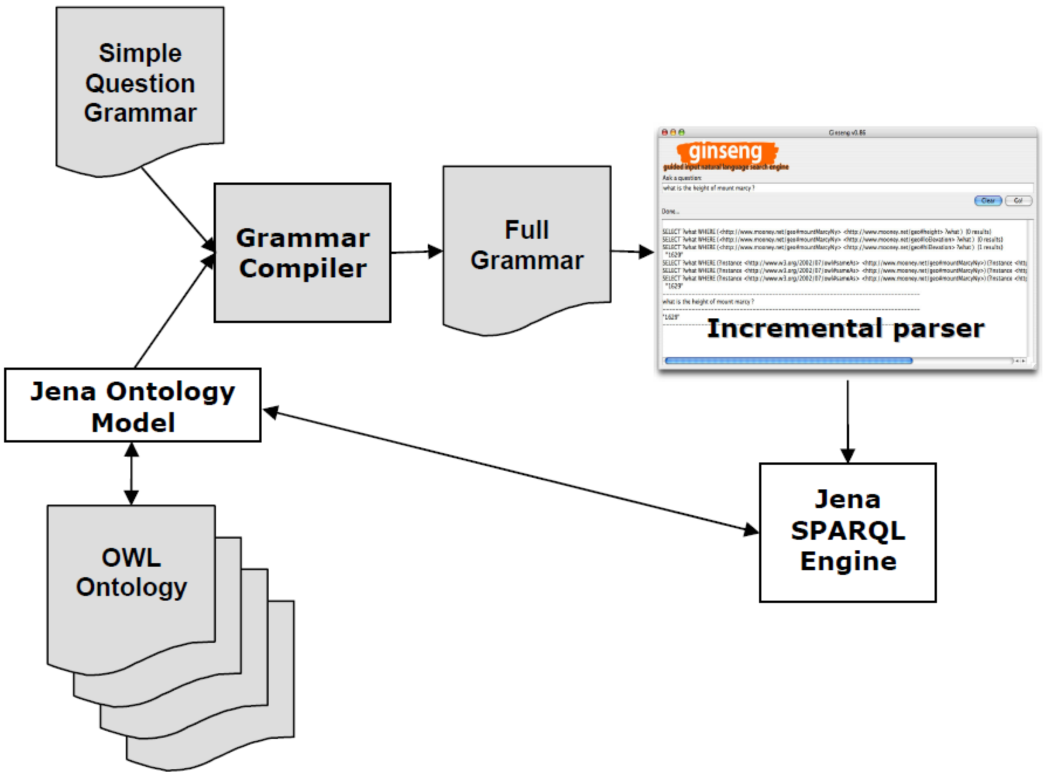


Figure 3.1: Ginseng's Architecture

The main differences between Ginseng and this thesis are summarized as follows:

- Ginseng does not use any predefined vocabulary, however, the link grammar parser used in this thesis has its own dictionary.
- Ginseng uses a static grammar to provide the basic sentence structure of English. In this thesis, it is not necessary to specify the sentence structure because the link parser handles it.
- Ginseng guides the user with possible queries. In ours, the possible completions of queries are not offered to the user.

- In Ginseng, a complete parse tree of the input is used to generate the SPARQL query, whereas a parse tree is not needed in the construction of the SPARQL query in this study.

3.1.2 NLP-Reduce

NLP-Reduce is a domain independent natural language query interface for ontologies [31]. In order to make the interface robust to non-complete and ill-formed inputs, it uses a small set of NLP operators such as stemming and synonym expansion.

NLP-Reduce has five main parts: a user interface, a lexicon, a query processor, a SPARQL query generator, and an ontology access layer [31]. From the user interface, the user can enter a NL query, sentence fragment or keywords. The lexicon is constructed when the ontology is loaded into the system. It contains all explicit and implicit subject-predicate-object triples. It is also expanded by the synonym of the subject, predicate, and object labels using WordNet. In addition, Porter Stemmer [14] is used to get the stem of each word.

The query processor removes the stop words and punctuation marks from the input sentence and then finds the stem of each word. The SPARQL query generator matches the stemmed words with the triples in the lexicon. Then, the query is generated for the matches. NLP-Reduce employs Jena as ontology access layer and Pellet as reasoner. After the query is executed, the results and some execution statistics are displayed to the user.

This thesis differs from NLP-Reduce in the following ways:

- NLP-Reduce does not make use of a parser, it uses synonym expansion and stemming, whereas link parser is used in this study.
- NLP-Reduce can handle sentence fragments and keywords. However, we handle only full sentences.
- In NLP-Reduce, WordNet and Porter Stemmer are used to expand the lexicon. We find the singular forms of the words to increase the matches between query words and the words in the ontology.

3.1.3 Querix

Querix is a domain independent natural language interface for querying ontologies. The following components are utilized in the system: a user interface, an ontology manager, a query analyzer, a matching center, a query generator, a dialog component, and an ontology access layer [32].

The user chooses the ontology and enters a NL query from the user interface. When the ontology is loaded into the system, the labels of the resources and their synonyms from WordNet are stored by the ontology manager. The query analyzer component parses the user input with Standford parser [15]. Using the parse tree, main word categories, e.g. wh-word (Q), noun (N), verb (V), and preposition (P), are extracted. Then, a query skeleton is generated from these categories. Moreover, WordNet is utilized to find the synonyms of all nouns and verbs in the query sentence.

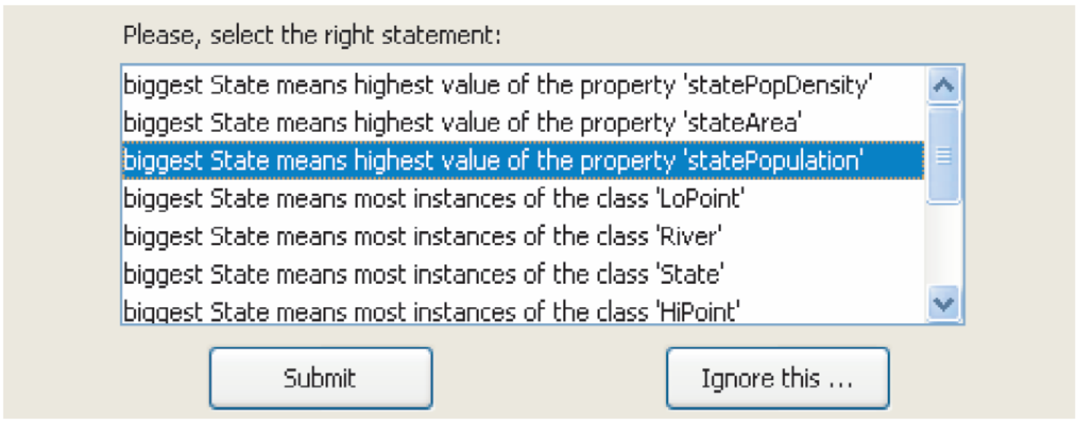
The matching center tries to map the query skeleton with the triples in the ontology. There are three steps in matching. First, the subject-predicate-object patterns in the query text are extracted from the skeleton, such as Q-V-N (what is the height) or N-P-N (height of the mountains). In the second step, expanded nouns and verbs of the input sentence are matched with the expanded resources of the ontology. Finally, the patterns found in step 1 and the resources that are matched with the user input in step 2 are matched. Then, according to the matching results, the query generator produces the SPARQL queries.

If Querix faces with ambiguities (i.e. one query word matched with one or more ontology resources) then it asks the user for clarification. The user selects the intended meaning, and the results are retrieved accordingly. An example clarification dialog can be seen in Figure 3.2 [32].

The generated SPARQL queries are executed with Jena access layer and Pellet is used as the reasoner. The results of the query and the SPARQL query are displayed to the user.

The differences between Querix and this thesis are listed below:

- In Querix, the Standford parser is used. In this work, link parser is employed.
- Querix uses WordNet to enhance ontology resources. But we do not make use of Word-


Please, select the right statement:

biggest State means highest value of the property 'statePopDensity'
biggest State means highest value of the property 'stateArea'
biggest State means highest value of the property 'statePopulation'
biggest State means most instances of the class 'LoPoint'
biggest State means most instances of the class 'River'
biggest State means most instances of the class 'State'
biggest State means most instances of the class 'HiPoint'

Figure 3.2: Querix Clarification Dialog

Net.

- Querix asks for clarification in case of ambiguities; however ambiguities are not handled in this study.

3.1.4 QuestIO

Question-based **I**nterface to **O**ntologies (QuestIO) is a natural language interface for querying ontologies [39]. It is domain-independent and does not have a predefined vocabulary. Rather, it constructs the vocabulary when the ontology is loaded into the system. QuestIO uses Sesame [9] as the RDF repository; thus the NL input is converted to SeRQL queries. It is designed to be robust; it handles language ambiguities, and incomplete or deficient inputs. It accepts sentence fragments and short queries as input.

When the knowledge base (KB) is loaded into the system, a gazetteer is constructed with the lexicalizations in the KB. Since different orthography conventions can be used in an ontology, normalization is needed for the words of the ontology. Normalization process includes splitting *CamelCase* words, splitting underscored words, and finding the non-inflected form of words. Also, a morphological normalization is needed for the words in the input query. Hence, the matching is independent of the inflection.

When a user enters a query, a linguistic analysis is applied to the input text. The input sentence is separated into tokens, and then a part-of-speech tagger is used to find the POS tag of each token. Finally, the noninflected form of each word is discovered by a morphologic analyzer.

After the linguistic analysis, the words of the query is tried to be matched with the words in the ontology. If a query word is matched with a class in the ontology, it is used directly in the query. If a reference to an instance is found, then the class of the instance is required. To add property values to the query, the associated instance and its class is needed. Once, there are two classes, a list of possible properties between these classes are formed. The candidate interpretations are scored with the weighted sum of string similarity score, specificity score, and distance score metrics. A SeRQL query is created from the list of interpretations and executed against the ontology. The results are displayed to the user.

The main differences between QuestIO and this work are:

- SeRQL is used in QuestIO. In this study, we use SPARQL as the query language.
- QuestIO handles ungrammatical text and sentence fragments. We handle only full sentences.
- In QuestIO, a POS tagger and a morphological analyzer are used in linguistic analysis. In this thesis, the link parser is used.
- In QuestIO, string similarity algorithms are used to improve the matching between the user input and the ontology. In ours, we make use of the singular forms of query words to increase the matches between query words and the words in the ontology.

3.2 Natural Language Interfaces Over Video Databases

3.2.1 BilVideo

BilVideo [23] is a video database system which supports spatio-temporal, semantic and low level feature queries . In order to extend the BilVideo's querying capabilities, a natural language query interface is developed. In NL querying of BilVideo, three types of queries are supported: object, spatial, and similarity-based object trajectory queries. Object queries deals with the occurrence of objects in video segments. With spatial queries, the spatial relationship between two objects is investigated. Spatial relations can be divided into three parts: topological relations (disjoint, touch, inside, etc.), directional relations (north, south, left, right, etc.), and 3D relations (infrontof, behind, samelevel, etc.). For each query type, patterns are defined according to the POS tag information. In Figure 3.3 [23], spatial relations, query examples, results of the tagging of the example and possible tag ordering are shown.

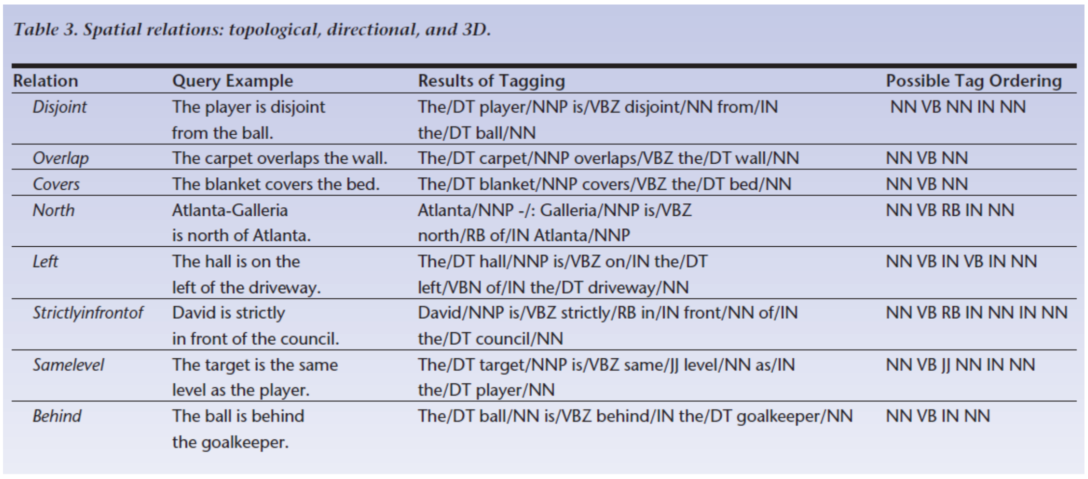


Table 3. Spatial relations: topological, directional, and 3D.

Relation	Query Example	Results of Tagging	Possible Tag Ordering
Disjoint	The player is disjoint from the ball.	The/DT player/NNP is/VBZ disjoint/NN from/IN the/DT ball/NN	NN VB NN IN NN
Overlap	The carpet overlaps the wall.	The/DT carpet/NNP overlaps/VBZ the/DT wall/NN	NN VB NN
Covers	The blanket covers the bed.	The/DT blanket/NNP covers/VBZ the/DT bed/NN	NN VB NN
North	Atlanta-Galleria is north of Atlanta.	Atlanta/NNP -/: Galleria/NNP is/VBZ north/RB of/IN Atlanta/NNP	NN VB RB IN NN
Left	The hall is on the left of the driveway.	The/DT hall/NNP is/VBZ on/IN the/DT left/VBN of/IN the/DT driveway/NN	NN VB IN VB IN NN
Strictlyinfrontof	David is strictly in front of the council.	David/NNP is/VBZ strictly/RB in/IN front/NN of/IN the/DT council/NN	NN VB RB IN NN IN NN
Samelevel	The target is the same level as the player.	The/DT target/NNP is/VBZ same/JJ level/NN as/IN the/DT player/NN	NN VB JJ NN IN NN
Behind	The ball is behind the goalkeeper.	The/DT ball/NN is/VBZ behind/IN the/DT goalkeeper/NN	NN VB IN NN

Figure 3.3: Spatial Relations in BilVideo

When the user enters the NL input, BilVideo uses MontyTagger to assign POS tags to each word in the input sentence. The MontyTagger is a rule-based POS tagger and uses the Penn Treebank tag set. After the sentence is tagged, BilVideo merges the proper nouns. The POS tags such as determiners and possessive pronouns are discarded. Whereas, nouns, verbs, prepositions, adverbs, and conjunctions are essential to match the query pattern with the predefined patterns. When the query pattern is found, the system creates queries in the form of Prolog facts. The results of the query are sent to Web clients.

This thesis differs from BilVideo in the following ways:

- BilVideo handles NL queries for video databases, whereas in this study the target data storage is ontology.
- In BilVideo, a POS tagger is used to find the pattern of the query. However, we prefer to use link parser to parse the user input.
- BilVideo does not deal with temporal queries.

3.2.2 Natural Language Querying for Video Databases

In [24], a natural language query interface is developed for querying video databases. The system architecture can be seen in Figure 3.4 [24]. The approach is domain independent, i.e. it is applicable to new domains easily. In the video data model, there are entities such as objects, activities, and events. These entities are linked to the frame sequences that they occur. Occurrence, spatial, temporal, and trajectory query types are supported in the system.

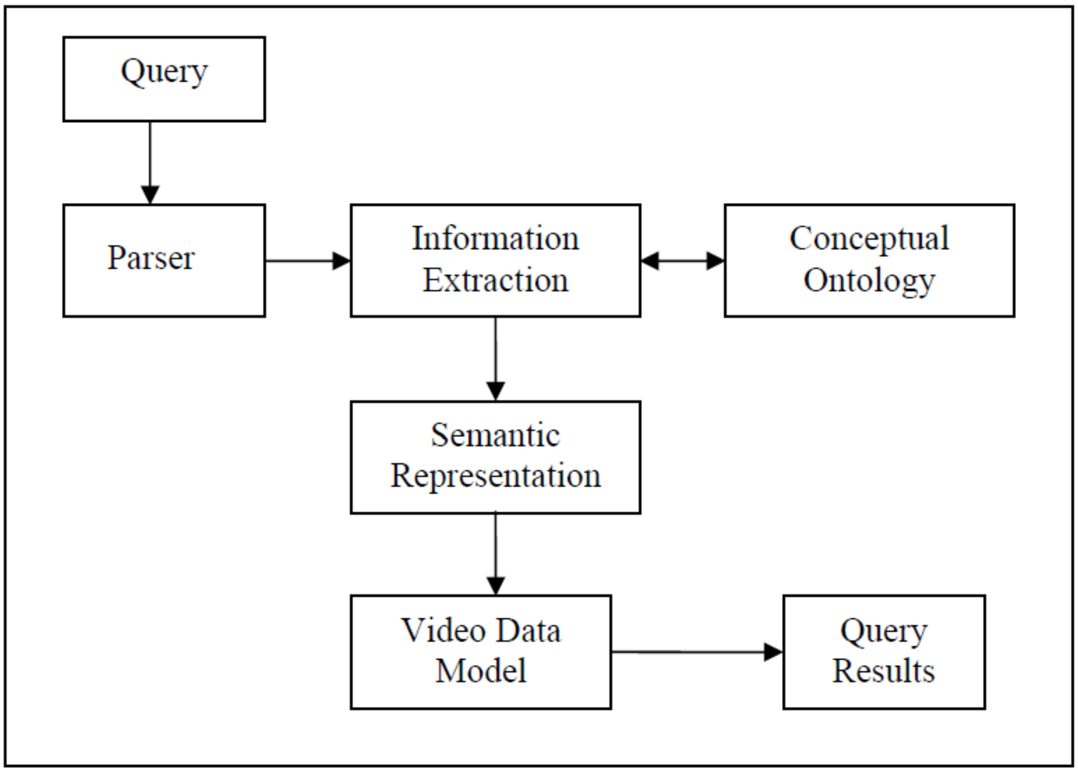


Figure 3.4: The main structure of the natural language interface of the video data model

In order to parse the given NL query, link parser is used. By using the link parser, the words or word groups that are crucial for the system are found. There are rules which are defined to extract information from the link parser output. The semantic representations of the queries are obtained by using these rules along with the parser output.

In the video database, all objects and activities are annotated with nouns and verbs respectively. So, in the query sentence, at least one noun or one verb is expected. If the query word matches with an annotation word in the video database, then this is an exact match. With the purpose of extending the matching, WordNet [16] and a conceptual similarity algorithm is used. To measure the semantic similarity between the query word and the video object word, Wu and Palmer's algorithm [43], a distance based similarity method, is preferred. This algorithm uses the noun and verb hierarchies of WordNet. The semantic representations are expanded with semantically similar words found by this algorithm. Therefore, not only the exact matches, but also approximate matches are retrieved from the video database.

The study discussed above is one of the bases for this thesis. The main differences between our approach and [24] can be summarized as follows:

- Their query interface is for video databases, whereas in this study we developed an NL interface for querying ontologies.
- They do not handle attributes in the queries, however, we can handle one or more attributes.
- The negative meaning in the queries is captured in this thesis.
- Complex object querying is not supported in their work, whereas the user can query one or more objects at a time in our study.
- They use WordNet to enable ontological reasoning. In this thesis, ontological reasoning is already achieved by storing the target data in the ontology.
- Their work handle fuzzy queries by using threshold value between [0, 1]. Fuzzy queries are not included in this study.

3.3 Ontology-Based Spatio-Temporal Video Management System

OntoVMS [27] is a video data management system, which extends spatio-temporal querying by integrating ontologies into the system infrastructure. In order to standardize the ontologies, it attaches the domain ontologies to the core Rhizomik MPEG-7 ontology. As a result, a generic framework is achieved; any domain ontology can be used in the system.

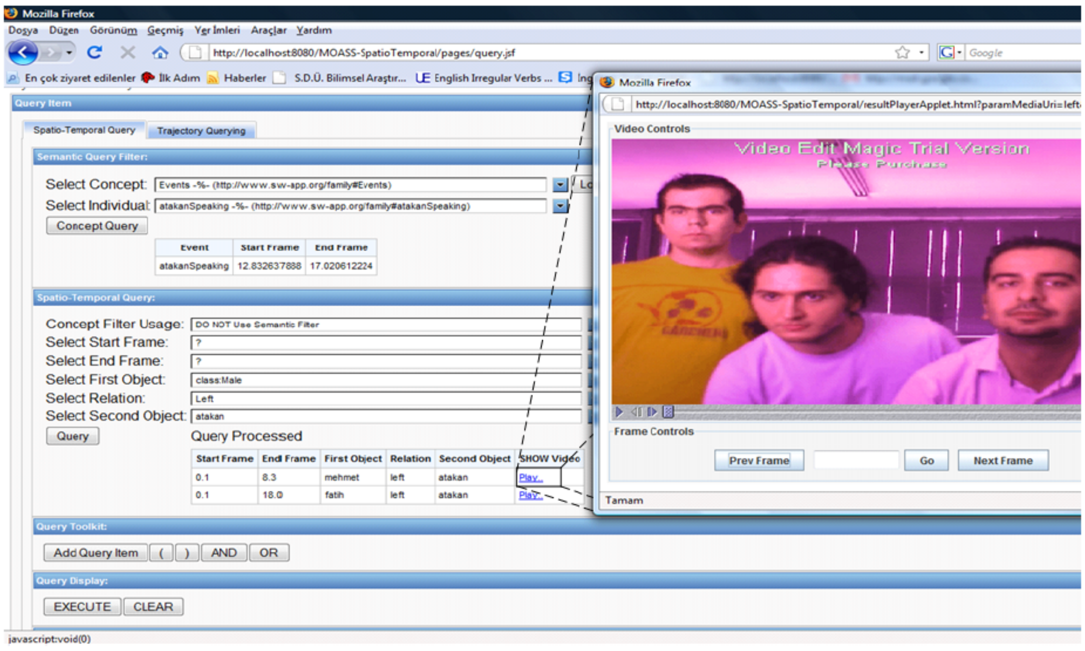
In order to make semantic and spatio-temporal querying possible on the video data, the video data must be annotated first. In the annotation process the attached domain ontology is used, which implies that ontological reasoning is available in the querying phase. By using OntoVMS, individuals of the classes in the domain ontology can be defined for further use in annotation and querying steps. In the semantic annotation, time intervals for events are defined. The spatial and temporal properties of objects are essential to make spatio-temporal querying, so the minimum bounding rectangle (MBR) should be defined for the objects in the ontology for each frame they occur. Also, a face tracker tool, which can track the human faces in a video, is used to make semi-automatic annotation.

In OntoVMS, spatial relations can be defined as the objects relative to other objects, and temporal relations describe the occurrence of two objects relative to each other. Spatial relations consist of strict directional (north, south, . . .), mixed directional (northeast, northwest, . . .), positional relations (left, right, . . .), distance relations (near, far) and topological relations (inside, touch, . . .). Temporal relations include precedes, meets, overlaps, finished by, contains, equal, and etc. Allen's [1] temporal interval algebra is used to manage temporal relations.

There is no need to store all of the spatial and temporal relations in the system, because some of the relations can be inferred from the others. As a result, a rule based modeling, which is composed of basic facts and inference rules, is preferred. Basic facts and inference rules are stored in knowledge bases in permanent storage. A fact pruning algorithm is used to decide which relations can be inferred from the others, and the required ones for inference are stored in the basic facts KB. Besides, the inference rules are formalized by using Allen's temporal algebra and spatial logic.

OntoVMs supports concept, spatial, temporal, and trajectory queries. Also compound queries are provided with "AND" and "OR" operators. The query interface is designed to give the user the ability of selecting the ontology classes and the instances, and the predefined spatial and temporal relations to be used in the query. SPARQL is used as the ontology query language and Jena ontology API is used to access the ontology. As mentioned above, the facts are pruned in the annotation phase. In the querying phase, if the user makes a query about the non-stored facts, then inference rules are used to get the results. In Figure 3.5 [27], spatial query screen of OntoVMS can be seen.

The work mentioned above is chosen as one of the bases for this thesis. The output ontology files of OntoVMS are used as input to our system. The output ontology file contains all annotated objects, the frame intervals and the MBRs of these objects. In this thesis, a natural language query interface is developed for providing concept, spatial, temporal, and trajectory


Mozilla Firefox
Dosya Düzen Görünüm Geçmiş Yer İmleri Araçlar Yardım
http://localhost:8080/MOASS-SpatioTemporal/pages/query.jsf
En çok ziyaret edilenler İlk Adım Haberler S.D.Ü. Bilimsel Araştır... English Irregular Verbs ...
Query Item
Spatio-Temporal Query
Trajectory Querying
Semantic Query Filter:
Concept Query
Spatio-Temporal Query:
Query
Query Toolkit:
Add Query Item
(
)
AND
OR
Query Display:
EXECUTE
CLEAR
javascript:void(0)
Mozilla Firefox
http://localhost:8080/MOASS-SpatioTemporal/resultPlayerApplet.html?paramMediaUri=left
Video Controls
Video Edit Magic Trial Version
Please Purchase
Frame Controls
Prev Frame
Go
Next Frame
Tamam

Select Concept:	Events -%- (http://www.sw-app.org/family#Events)
Select Individual:	atakanSpeaking -%- (http://www.sw-app.org/family#atakanSpeaking)

Event	Start Frame	End Frame
atakanSpeaking	12.832637888	17.020612224

Concept Filter Usage:	DO NOT Use Semantic Filter
Select Start Frame:	?
Select End Frame:	?
Select First Object:	class:Male
Select Relation:	Left
Select Second Object:	atakan

Query Processed

Start Frame	End Frame	First Object	Relation	Second Object	SHOW Video
0.1	8.3	mehmet	left	atakan	Play...
0.1	18.0	fatih	left	atakan	Play...

Figure 3.5: Simple Spatial Query Screen of OntoVMS

queries. The main differences between OntoVMS and this thesis are listed below:

- OntoVMS provides form-based querying, whereas our system is an NL-based querying system.
- In OntoVMS, object attributes are not included in queries, but in this study, one or more attributes of objects can be used in queries.
- In this thesis, a subset of OntoVMS spatial relations (left, right, above and below) and temporal relations (after and before) are supported.
- The source-destination trajectory type is not provided in our system, since it can not be stated with the natural language.
- The knowledge base structures and the inference rules are not needed in this work, because the calculations of the spatial and temporal relations are done on the results of the ontology query.

CHAPTER 4

SYSTEM ARCHITECTURE AND THE SUPPORTED QUERY TYPES

In this thesis, a system is developed for querying ontologies based on multimedia data. The system accepts natural language input form the user, maps the input to SPARQL queries, and then executes the SPARQL queries against the ontology. The query results are processed to calculate spatial, temporal and trajectory relationships according to the type of the query.

This chapter introduces the system architecture and the query types supported by the system. In Section 4.1 the general architecture of the system is given. The query types are explained in Section 4.2. Finally, in Section 4.3 the internal representations of the query types are detailed.

4.1 General Properties of the System

In this thesis, a natural language interface is developed for querying multimedia domain ontologies. Domain ontologies are based on the core Rhizomik MPEG-7 Ontology. They are simply formed by attaching the domain ontology itself to the Rhizomik MPEG-7 Ontology. The system accepts the natural language input from the user, parses it with the link parser, generates the SPARQL query according to the query type, and finally executes the query against the ontology. The supported query types are given in Section 4.2.

4.1.1 System Specifications

In this study, a windows application is developed in Microsoft Visual Studio 2005 using .NET Framework 2.0. Also, the application uses the following libraries:

- **Link Grammar .NET 2.0:** A wrapper library [6] for using Link Grammar Parser [5] in .NET platform.
- **SemWeb:** A Semantic Web/RDF library [8] written in C# for .NET Framework 1.1/2.0.
- **ActiveMovie COM Component:** It is used to play a video file (such as an MPEG, an AVI, or a WMV file) in a Windows Forms application [3]. This component is included with Windows Media Player.

4.1.2 System Architecture

The general architecture of the system is presented in Figure 4.1.

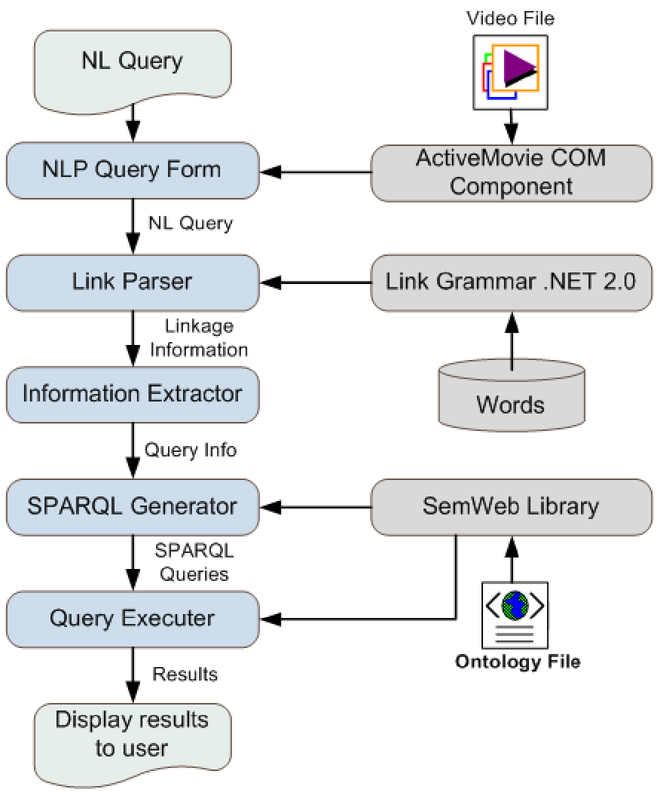


Figure 4.1: System Architecture

Main modules of the system are:

- NLP Query Form: This is the main form of the application. The user chooses the ontology to be queried and enters the natural language input by using this form. When the user selects the ontology file, the name of the video file is extracted from the ontology. Then, by using ActiveMovie COM component, the video file is loaded into the system and the duration of the video is found. The user can view the results of the query are displayed in a tabular format in the form. By double clicking on a result row, the user can view the video file content that corresponds to the result.

- Link Parser: This module is used to parse the natural language input. It uses Link Grammar .NET 2.0 library [6]. It parses the NL input and produces the linkage information of the input query.

- Information Extractor: This module uses the output of the Link Parser. It constructs Query Info from the linkage information. Initially, it finds the query type. According to the query type, it extracts objects, attributes, spatial relation, temporal relation, trajectory relation, and time information by the usage of linkage information.

- SPARQL Generator: This module is responsible for creating the SPARQL queries from the query information. It uses SemWeb library [8]. When the ontology file is chosen from the NLP Query Form user interface, it loads the data from the ontology into the system. For further use, it extracts the attributes in the ontology which have restricted data values. It constitutes a class list whose elements are the classes in the ontology. An object property list and a data type property list are also populated from the entries in the ontology. It gets the video file name from the ontology. While generating the SPARQL queries, it checks if the query information exists in the ontology by using the extracted information.

- Query Executer: This module is used to execute the generated SPARQL queries. It uses SemWeb library [8] and executes the queries against the ontology. Query results are processed according to the query type for calculating spatial, temporal, and trajectory relations. Finally, the results are displayed in the NLP Query Form.

4.2 Supported Query Types

Ontologies which are based on Mpeg-7 ontologies are being queried in this system. Consequently, spatio-temporal querying should be offered to the users. The system supports five types of queries, namely: *Concept*, *Complex Concept*, *Spatial*, *Temporal*, and *Trajectory*.

1. **Concept Query**

   This query type is used when selecting the objects of interest. Moreover, objects can be queried by specifying one or more attributes. For example, the system can answer queries such as: "return all frames where a woman is seen", "in which frames John Locke is seen" or "find all intervals where the white building is seen".

   In addition, the negative meaning in the NL input is also captured. Thus, users can view the time intervals where the specified object does not appear in the video. For instance, the user can enter the query "show me all frames where a male is not seen" to the system.

2. **Complex Concept Query**

   By using this type of query, users can select more than one object that are connected with an "AND" or "OR" connector. Also, attributes can be used to describe these objects. Thus, a complex concept query can be thought as a collection of concept queries, along with a connector parameter. When handling the "AND" connector in the query, the intersection of the results for each concept query is found. If the "OR" connector is given in the NL query, the result is the union of the results for each concept query. For instance, "Return frames where John and Mary appear" and "Find the intervals where a female or a male is seen" are examples of complex concept query.

   The negative meaning is also captured in the complex concept query. When the user enters a query such as "Return frames where Jack and Kate are not seen.", the frame intervals in which neither Jack nor Kate is seen will be returned.

3. **Spatial Query**

   Spatial queries are used to find the positions of objects relative to each other. A spatial relation exists between two objects that appear in the same frame intervals. Four types of positional relations are supported in the system: *Left*, *Right*, *Above*, and *Below*. For instance "Retrieve all frames where a player is seen to the right of the goalkeeper." and "which objects appear above the yellow car?" are spatial queries that can be answered in our system. The two objects can also be described with attributes.

4. **Temporal Query**

   In temporal queries, the relative appearances of objects in a time sequence can be investigated. The relations "Before" and "After" of Allen's interval algebra are supported.

The algebra deals with possible relations between time intervals [1]. For example, "Before" relation holds between objects X and Y, if the end time of object X is earlier than the start time of object Y [1, 17]. "Return the objects that appear before the police man" and "show all frames where John appears after the car" are example of temporal queries.

5. **Trajectory Query**

   Users can query the paths of the objects of interest. There are two types of trajectory queries: *Object* trajectory, and *Directional* trajectory. In object trajectory, the paths of the objects are queried. "Return the path of the dogs" and "return the trajectory of the balls" are examples of object trajectory. In directional trajectory, the objects that go to the specified direction are searched. For example, "which objects go to the north?", "return all players who go to the east" are directional trajectory queries. In directional trajectory, there are 8 directions: *East, West, North, South, Northeast, Northwest, Southeast,* and *Southwest*. *Right, Left, Upwards (Upward), Downwards (Downward), Upper Right, Upper Left, Lower Right,* and *Lower Left* can be used interchangeably with the 8 directions above respectively.

4.3 Internal Representations of Query Types

Each of the supported query types in the system has an internal representation. When the natural language input is parsed, its output is mapped to its internal representation. The internal representation of each query type is given in the following.

1. **Concept Query Representation**

   ConceptQuery:

   Concept (ObjectName, AttributeList)

   QueryStartTime

   QueryEndTime

   IsNegationQuery

   VideoDuration

   ObjectName is the name of the object in the query such as *Ali, dog, prime minister, John Smith* etc. The *AttributeList* contains the attributes of the object if stated in the query sentence. The object can have one or more attributes like *big, black, rectangular, cold* etc. Any number of attributes are accepted in the system. The *Concept* is composed of the *ObjectName* and the *AttributeList*. *QueryStartTime* and *QueryEndTime* fields are used to retrieve the concepts in the specified time interval. *IsNegationQuery* is a boolean flag, which is set to true if negative meaning exists in the NL input. For example, in the query "Return frames where a dog is seen", *IsNegationQuery* is set to false. When the query "Show intervals where Tom is not seen" is given as input, then the negative meaning in the query is captured and *IsNegationQuery* is set to true. *VideoDuration* is the total duration of the video. It is used to calculate the time intervals when the query is a negation concept query. The details of the calculation process are given in section 6.2.1

Example: Show all intervals where a white cat appears from 5 to 10 minutes.

ConceptQuery:

Concept:

ObjectName = cat

AttributeList = {white}

QueryStartTime = 300

QueryEndTime = 600

IsNegationQuery = false

VideoDuration = 1200

Example: Return frames where a policeman is not seen in the video.

ConceptQuery:

Concept:

ObjectName = policeman

IsNegationQuery = true

VideoDuration = 1800

As it can be seen from the examples, the *QueryStartTime*, *QueryEndTime*, and *Video-Duration* is represented in seconds. If *QueryStartTime* and *QueryEndTime* are not shown in the representation, then the query is not restricted to a specific time interval, rather the scope of the query is considered to be the whole video.

2. **Complex Concept Query Representation**

ComplexConceptQuery:

List of ConceptQuery

Connector

QueryStartTime

QueryEndTime

In this query type, the objects that are connected together with "AND" or "OR" can be queried. *List of ConceptQuery* contains more than one *ConceptQuery*. Each *ConceptQuery* is represented as explained in the above section. The Connector can be "AND" or "OR". *QueryStartTime* and *QueryEndTime* fields are used to retrieve the concepts in the *ConceptQuery* list in the specified time interval.

Example: Find the intervals where a female or a male is seen.

ComplexConceptQuery:

ConceptQuery:

Concept:

ObjectName = female

IsNegationQuery = false

VideoDuration = 1800

ConceptQuery:

Concept:

ObjectName = male

IsNegationQuery = false

VideoDuration = 1800

Connector = OR

3. Spatial Query Representation

SpatialQuery:

First Concept (ObjectName, AttributeList)

Second Concept (ObjectName, AttributeList)

SpatialRelation

QueryStartTime

QueryEndTime

In spatial queries, the spatial relation between two objects is queried. Therefore, there are two *Concepts* and a *SpatialRelation* in the representation. Each concept has an object name, and can have any number of attributes. The *SpatialRelation* can be one of *Left, Right, Above,* and *Below*. *QueryStartTime* and *QueryEndTime* fields are the beginning and end times that will be used as filter in the query.

Example: Retrieve all frames where a player is seen to the right of the goalkeeper.
SpatialQuery :

    Concept firstObject:

        ObjectName = player

    Concept secondObject:

        ObjectName = goalkeeper

    Spatial Relation: RIGHT

4. Temporal Query Representation

TemporalQuery :

    First Concept (ObjectName, AttributeList)

    Second Concept (ObjectName, AttributeList)

    TemporalRelation

    QueryStartTime

    QueryEndTime

    TimeFilter (FilterType, MinOrSec, FilterAmount, Start, End)

In temporal queries, the temporal relation between two objects is queried. Therefore, there are two *Concepts* and a *TemporalRelation* in the representation. Each concept has an object name, and can have any number of attributes. The *TemporalRelation* can be one of *Before* and *After*. *QueryStartTime* and *QueryEndTime* fields are used to declare the time interval and the results in that interval will be returned. *TimeFilter* is used to restrict the *TemporalRelation*. Five types of filter are supported: EXACT, LEAST, MOST, INTERVAL, and STANDARD. When the *FilterType* is one of EXACT, LEAST and MOST, then *FilterAmount* is used to define the time restriction. When the *FilterType* is INTERVAL, then *Start* and *End* are used to define the time interval. *MinOrSec* specifies whether the filter is in minutes or in seconds. Below, an example sentence is given for each filter type.

FilterType = EXACT
Return the objects that appear **5 minutes after** the bus.
FilterType = LEAST
Return the humans that are seen **at least 10 minutes before** the police.

FilterType = MOST

Return the animals that appear **at most 3 minutes after** the cat.

FilterType = INTERVAL

Return the objects that appear **from 5 to 10 minutes before** the crash.

FilterType = STANDARD

Return the males that are seen **before** John Locke.

Example: Show all frames where John appears after the car.

TemporalQuery :

  Concept firstObject:

    ObjectName = John

  Concept secondObject:

    ObjectName = car

  Temporal Relation = AFTER

  TimeFilter

    FilterType = STANDARD

Example: Return the objects that appear from 5 to 10 minutes before the crash.

TemporalQuery :

  Concept firstObject:

    ObjectName = objects

  Concept secondObject:

    ObjectName = crash

  Temporal Relation = BEFORE

  TimeFilter

    FilterType = INTERVAL

    MinOrSec = MINUTES

    Start = 5

    End = 10

5. **Trajectory Query Representation**

TrajectoryQuery :

  TrajectoryType

  Concept (ObjectName, AttributeList)

  Direction

QueryStartTime

QueryEndTime

There are two types of trajectory queries: *Object* and *Directional*. *TrajectoryType* defines the type of the trajectory. In object trajectory queries, the paths of the objects are queried. The *Concept* is the composition of the object name and attributes.

Example: Return the trajectory of the bus from 10 to 20 minutes.

TrajectoryQuery:

TrajectoryType = Object

Concept

ObjectName = bus

QueryStartTime = 600

QueryEndTime = 1200

If the trajectory type is directional, then the objects which go to the specified direction are being queried. Hence, along with the *Concept*, the *Direction* is used. Direction can be one of *East, West, North, South, Northeast, Northwest, Southeast, Southwest, Right, Left, Upwards (Upward), Downwards (Downward), Upper Right, Upper Left, Lower Right,* and *Lower Left*. The first eight can be used interchangeably with the last eight respectively.

Example: Return the players that go to the right.

TrajectoryQuery:

TrajectoryType = Directional

Concept

ObjectName = player

Direction = Right

CHAPTER 5

MAPPING NATURAL LANGUAGE INPUT TO SPARQL QUERY

The backbone of the system is to correctly map the NL input to SPARQL query. The used parsing algorithm has considerable importance in this process. Extracting information from the user input depends on some predefined rules, which are defined according to the parser output. In order to get data from the ontology, the SPARQL queries are generated from the extracted information.

This chapter is organized as follows: Section 5.1 explains how to parse the natural language query and how to extract information from the parser output. The mapping of the parser output to SPARQL queries is presented in Section 5.2.

5.1 Parsing Natural Language Queries

Parsing the natural language input efficiently has a significant importance on the system accuracy. In this thesis, the specific parts of the input query are being investigated. An appropriate parser should be selected for this purpose. Deep parsers construct the whole parse tree for an input, so it is not appropriate in our case. Shallow parsers do not need to find the parse tree of a sentence, instead they identify the groups of words such as nouns, adjectives, and verbs in the input sentence [10]. As a result, it is concluded that a light parser is more suitable for our system, and a link grammar parser is used in the system development. In Section 5.1.1 link grammar parser is explained in detail, and in Section 5.1.2 how the information is extracted from the natural language input is described.

5.1.1 Link Grammar Parser

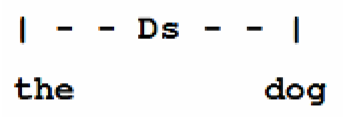
The Link Grammar Parser is a light syntactic parser of English, which builds relations between pairs of words in a sentence [4, 5]. It constructs the syntactic structure of a given sentence by assigning links to word pairs. A sentence has a valid syntactic structure, if it satisfies the following conditions when it is parsed by the link grammar [38]:

Planarity: The links in the sentence do not cross each other.

Connectivity: All words of the sentence are connected together via links.

Satisfaction: For each word, linking requirements should be satisfied.

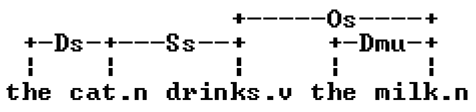
A link grammar has a dictionary which consists of a set of words, and linking requirements for each word [38]. For example, a determiner, such *as a, an, the*, requires a D connector to its right. A verb requires an S connector to its left. Here, D and S are link types, and a word can be the left or right connector of a link type. It can be better explained with the following example.



```
| - - Ds - - |
the        dog
```

D is a link type. It connects determiners to nouns. The subscript placed on D link dictates the number agreement. In this case, the determiner “the” is followed by a singular noun “dog”. As a result, an “s” subscript, stands for singular, is attached to the D link. The links are represented in capitals and the subscripts are represented with small letters.

A linkage is the set of links for a valid sentence in the language. The link parser can produce more than one linkage for a sentence because it tries every way of forming links between the word pairs. The linkage information for the sentence “The cat drinks the milk” is illustrated in the following example.



```
                 +-----Os----+
 +-Ds-+---Ss--+      +-Dmu-+
 |    |       |      |     |
the cat.n drinks.v the milk.n
```

Ds, Ss, Os and Dmu links constitute the linkage. The Ss link connects singular nouns to singular verb forms and Os link marks nouns as being singular.

In the link parser's dictionary, there are about 60000 words [5]. The link grammar parser has nearly seven hundred definitions which deal with noun-verb agreement, questions, imperatives, complex and irregular verbs, different types of nouns, past- or present-participles in noun phrases, commas, a variety of adjective types, prepositions, adverbs, relative clauses, and possessives [38].

The parser is robust; which means that it is not required to fully understand the given sentence. Instead, it can assign the syntactic structure to the portion it understands. One of the reasons why the link parser is chosen is the robustness of the parser. We investigate specific portions in the sentence, such as objects, attributes, spatial relation, temporal relation, trajectory relation, and time information. In order to find these groups of words, the proper link or link sequences are selected. In Section 5.1.2, rules to extract the needed word groups are explained. In Appendix A, the link types used in the thesis are listed.

5.1.2 Extracting Information from a Natural Language Query

The aim of parsing the natural language input is extracting the information being queried. In order to achieve this, the structure of the input should be analyzed and rules must be generated according to both the input structure and the link information. Link types and link orders are used to define a rule. An example rule which is used to find an object is given below:

1. Search for Cs link.
2. If one of Ss, Sp, Spx links follows the Cs link, and a Pv link follows one of these links, then query is a concept query and the object is the right word of the Cs link.

When a natural language query is entered to the system, the type of the query is found first by using rules. One or more rules are defined for each query type. The complete list of the rules is given in Appendix B. Then, according to the query type, the corresponding rules are used to map the input to the internal representations defined for objects, attributes, spatial relation, temporal relation, trajectory relation, and time information. The output of the link parser is searched for the links used in the rule. If all links in the rule appear in the output of the parser with the same order, and the output satisfies the constraints defined in the rule, then the pattern is found. By using the mapping information in the rule, the query is mapped to the internal representation. Let's explain the flow with an example. The natural language input is:

- Show all frames where a red car is seen.

The link parser output is given as follows.

```
    +----------------------------Xp---------------------------+
    |        +--------MVp-------+------Cs-----+               |
    |        +-----Op----+      |   +----Ds---+               |
    +---Wi---+    +--Dmc-+      |   |   +--A--+--Ss-+--Pv-+   |
    |        |    |      |      |   |   |     |     |     |   |
LEFT-WALL show.v all frames.n where a red.a car.n is.v seen.v .
```

For this input, the query type is found by using the rule given above. Since Cs link is followed by an Ss link, and a Pv link follows the Ss link, the query type is found as concept query. The object is the right word of the Cs link, so it is “car”. It is seen that, the noun “car” is a right connector of the link A. The A link is a connector between pre-noun adjectives and nouns. When the A link appears in the output, it means that the left word of A link is an attribute of the object. As a result, “red” is extracted as the attribute of the “car”. Any number of adjectives can be used before a noun and each of them are connected to the noun with A link.

5.2 Mapping NLP Output to SPARQL Query

In order to be able to execute the query on a given ontology, the natural language input should be mapped to a SPARQL query. There are two steps to do this mapping. Initially, the internal representations should be extracted from the given sentence which is explained in Section 5.2.1. Then, these internal representations should be converted to SPARQL queries. This process is given in Section 5.2.2.

5.2.1 Mapping NLP Output to Internal Representations

When the user enters an NL input, it is parsed using the link parser and the query type is extracted. According to the query type, the portions of the given sentence to used for mapping are determined. For example, if the query type is *Concept*, then the rules given in Appendix B.2 are used, and the extracted data is mapped to the *ConceptQuery* class.

The following examples illustrate the mappings:

Example 1:

NL Input: Return all frames where a male appears in the video.

```
    +--------------------------------Xp---------------------------------+
    |         +--------MVp--------+                                     |
    |         +-----Op-----+      +---Cs---+              +---Jp---+    |
    +----Wi---+     +--Dmc-+      |   +-Ds-+---Ss--+--MVp-+  +-D*u-+    |
    |         |     |      |      |   |    |       |      |  |     |    |
LEFT-WALL return.v all frames.n where a male.n appears.v in the video.n .
```

The query type is found as *Concept* for this sentence. Using *Rule 1* given in Appendix B.2, it can be observed that an *Ss* link follows a *Cs* link, and the right word of the *Ss* link is "appears". So, the object is the right word of the *Cs* link and it is extracted as "male". Since no attributes are found, the internal representation of this query is:

ConceptQuery :

  Concept :

    ObjectName = male

    AttributeList = { }

Example 2:

NL Input: Return all frames where John is seen to the right of the red car.

```
    +-----------------------------------------Xp-----------------------------------------+
    |         +--------MVp--------+                                    +------Js-----+   |
    |         +-----Op-----+      |                     +---Js---+     |  +----Ds----+   |
    +----Wi---+     +--Dmc-+      +--Cs-+-Ss-+--Pv-+-MVp+  +--Ds-+--Mp-+  |    +--A--+   |
    |         |     |      |      |     |    |     |    |  |     |     |  |    |     |   |
LEFT-WALL return.v all frames.n where John is.v seen.v on the right.n of the red.a car.n .
```

The query type of this sentence is *Spatial*. Using *Rule 1* given in Appendix B.5, it can be seen that *MVp* + *Js* + *Mp* + *Js* link sequence follows the *Ss* link. As a result, the first object is the left word of the *Ss* link and it is found as "John". The relation is the right word of the first *Js* and it is extracted as "right". And finally, the second object is the right word of second *Js* and it is found as "car". Also, *Rule 1* in Appendix B.10 is used to find the attributes. The "car" is found as an object and it is the right connector of *A* link. So, the left word of the *A* link is an attribute of the object. Then "red" is extracted as an attribute of the object "car". The internal representation of this query is:

SpatialQuery :

  Concept firstObject :

    ObjectName = John

    AttributeList = { }

  Concept secondObject :

    ObjectName = car

    AttributeList = {red}

  Spatial Relation : RIGHT

5.2.1.1 Mapping Time Information to Internal Representations

To extract time information from the given input, the rules in Appendix B.9 are used. Video duration in seconds is required to calculate the query start and end times. It is obtained when the ontology file to be queried is selected. There are three cases in time information extraction:

a. If the sequences “in the first X minutes” or “in the first X seconds” are found in the query, then:

```
QueryStartTime: 0
IF minutes
    QueryEndTime: X * 60
ELSE IF seconds
    QueryEndTime: X
```

b. If the sequences “in the last X minutes” or “in the last X seconds” are found in the query, then:

```
IF minutes
    QueryStartTime: Video Duration - X * 60
ELSE IF seconds
    QueryStartTime: Video Duration - X
QueryEndTime: Video Duration
```

c. If the sequences “from X to Y minutes” or “from X to Y seconds” are found in the query, then:

```
IF minutes
    QueryStartTime: X * 60
    QueryEndTime: Y * 60
ELSE IF seconds
    QueryStartTime: X
    QueryEndTime: Y
```

5.2.1.2 Mapping Time Filter Information to Internal Representations

To find out the time filter information from the user input, the rules in Appendix B.7 are employed. There are five cases in extracting the time filter information:

a. If the sequences “X minutes / seconds after” or “X minutes / seconds before” are found in the query, then:
FilterType: EXACT
MinOrSec: Minutes or Seconds
FilterAmount: X

b. If the sequences “at least X minutes / seconds after” or “at least X minutes / seconds before” are found in the query, then:
FilterType: LEAST
MinOrSec: Minutes or Seconds
FilterAmount: X

c. If the sequences “at most X minutes / seconds after” or “at most X minutes / seconds before” are found in the query, then:
FilterType: MOST
MinOrSec: Minutes or Seconds
FilterAmount: X

d. If the sequences “from X to Y minutes / seconds after” or “from X to Y minutes / seconds before” are found in the query, then:
FilterType: INTERVAL
MinOrSec: Minutes or Seconds
Start: X
End: Y

e. If “before” or “after” is found in the query, then:
FilterType: STANDARD

5.2.2 Mapping Internal Representations to SPARQL Query

The SPARQLManager module is responsible for constructing the SPARQL queries for the internal representations. Each concept in the internal representations is retrieved from the ontology. For example, in the *TemporalQuery*, there are two concepts, and a SPARQL query is constructed for each concept. Concept is a class that has an object name, and an attribute list.

There are several important issues when creating queries for concepts. First, an entity with the given object name must exist in the ontology in order to generate a SPARQL query. Otherwise, no SPARQL query is generated. The queries are constructed differently according to the type of the entity in the ontology. If the entity is of type class, then a triple should be added to the query which specifies that the type of the object is the given class. For example, if "human" is the concept to be queried and it is defined as class in the ontology, then, the following triple indicates that the type of the variable "?object" is "human" class.

```
?object <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.sw-app.org/family#human>.
```

If the entity is of type instance, no need to add the triple above. The instance name itself is used instead of the variable "?object". The query is produced for the instance only. So, the information about one instance exists in the result set. Whereas, the result set of queries for class type includes information about all instances of that class.

The start time and duration properties of the objects are queried in *Concept* type and *Temporal* type queries. Besides these properties, the polygon property is also queried in *Spatial* and *Trajectory* queries.

5.2.2.1 Mapping Attributes to SPARQL

In this thesis, the attributes are mapped to SPARQL queries by using three approaches.

First Approach:
The attribute name is specified in the query text as in the following sentence:
NL Input: *Find the intervals where a car with red color is seen.*

ConceptQuery :

    Concept :

        ObjectName = car

        AttributeList = {color-red}

The attribute name "color" and attribute value "red" is given in the query. So, we store both the attribute name and the value, separated with a dash, in the attribute list. If the "color" attribute and the "car" object are found in the ontology, and the domain of the "color" attribute is the "car" class, then SPARQL text is generated for the attribute.

```
?object <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.sw-app.org/family\#car>.
?object <http://www.sw-app.org/family#color> "red"^^<http://www.w3.org/2001/XMLSchema\#string>.
```

If the "color" attribute and the "car" object are found in the ontology, and the domain of "color" attribute is one of the parent classes of the "car" class, again SPARQL text is generated for the attribute. To illustrate, in the following code segment, the "car" class is a sub class of the "vehicle" class and the domain of "color" attribute is the "vehicle" class. Thus, the above SPARQL text is used for the attribute.

```
<rdfs:Class rdf:about="http://www.sw-app.org/family#car">
    <rdfs:subClassOf rdf:resource="http://www.sw-app.org/family#vehicle"/>
</rdfs:Class>

<owl:DatatypeProperty rdf:about="http://www.sw-app.org/family#color">
    <rdfs:domain rdf:resource="http://www.sw-app.org/family#vehicle"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

Second Approach:

The attribute name is not given in the query text; rather the attribute value is extracted from the sentence, and this attribute value is in the restricted list of an attribute in the ontology. To illustrate:

NL Input: *Show all frames where a red car appears.*

ConceptQuery :

    Concept :

        ObjectName = car

        AttributeList = {red}

As seen from the example, only the attribute value “red” is given in the sentence. In fact, we do not know the attribute to which “red” refers. So, the ontology is searched if there exists an attribute which includes the “red” value in its DataRange values. If such an attribute is found, then the domain of the attribute is extracted. If the domain matches with the “car” class, then SPARQL text is generated for the attribute.

```
?object <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.sw-app.org/family\#car>.
?object <http://www.sw-app.org/family#color> "red"^^<http://www.w3.org/2001/XMLSchema\#string>.
```

The data range values of an attribute are specified in the ontology as follows:

```
<owl:DatatypeProperty rdf:about="http://www.sw-app.org/family#color">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
            Red
          </rdf:first>
        </rdf:rest>
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
          Blue
        </rdf:first>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
  <rdfs:domain rdf:resource="http://www.sw-app.org/family#car"/>
</owl:DatatypeProperty>
```

Third Approach:
The attribute name is not given in the query text; rather the attribute value is extracted from the sentence, and this attribute value is in one of the triples in the ontology. A triple is composed of a subject, a predicate, and an object. Below, an example triple is given:

```
<car rdf:about="http://www.sw-app.org/family#BMW">
  <color rdf:datatype="http://www.w3.org/2001/XMLSchema#string">red</color>
</car>
```

We will explain the third approach with an example:

NL Input: *Find all intervals where a red car is seen.*

ConceptQuery :

Concept :

ObjectName = car

AttributeList = {red}

The attribute value “red” and the object “car” are given in the sentence. Whenever an object value matching “red” is found, the predicate of the triple is used to find the domain of the attribute. In the above example, the predicate is “color”, so we will investigate the domains of “color”. If the domain of the “color” attribute is the “car” class or one of the parent classes of the “car” class, then the predicate value is used as the attribute name and the SPARQL text is constructed for the attribute.

```
?object <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.sw-app.org/family\#car>.
?object <http://www.sw-app.org/family#color> "red"^^<http://www.w3.org/2001/XMLSchema\#string>.
```

5.2.2.2 Mapping Time Information to SPARQL

If the start and end times are given in the sentence, they will be used to filter the results. The entities in the ontology have *SpatioTemporalLocator* property, and it has *MediaIncrDuration* and *MediaRelIncrTimePoint* properties, which indicate the start time and duration of the *SpatioTemporalLocator* property. These properties will be explained in Section 6.1.1. By using the start time and the duration of the entities, it should be figured out which entities’ time interval overlaps with the given time interval. The example below shows how the time information is mapped to the SPARQL query. The start and end variables are start and end times that are extracted from the NL input.

```
FILTER ((<http://www.w3.org/2001/XMLSchema#double>(?startTime) +
   <http://www.w3.org/2001/XMLSchema#double>(?duration)) >= start &&
   <http://www.w3.org/2001/XMLSchema#double>(?startTime) <= end).
```

5.2.3 Examples of Mapping NL Input to SPARQL Query

The following examples clarify the mapping of NL input to SPARQL query:

Example 1:
Return all frames where the big yellow ball is seen.

- **Link Parser Output:**

```
    +-----------------------------------Xp----------------------------------+
    |                             +------------Cs-----------+               |
    |         +--------MVp--------+    +---------Ds---------+               |
    |         +-----Op-----+      |    |    +-------A-------+               |
    +----Wi---+     +--Dmc-+      |    |    |       +---A---+--Ss-+--Pv-+   |
    |         |     |      |      |    |    |       |       |     |     |   |
LEFT-WALL return.v all frames.n where the big.a yellow.a ball.n is.v seen.v .
```

- **Internal Representation:**

  ConceptQuery:

  Concept:

  ObjectName = ball

  AttributeList = {big, yellow}

- **SPARQL Query:**

```
SELECT ?object ?startTime ?duration
WHERE {
?object <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://www.sw-app.org/family\#ball>.
?object <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator>
  ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>.
?spatioTemporalLocatorType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType.
?mediaTimeType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>.
?mediaTimeType <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration>
  ?duration.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaRelIncrTimePoint> ?startTime.
?object <http://www.sw-app.org/family#color>
  "yellow"^^<http://www.w3.org/2001/XMLSchema#string>.
?object <http://www.sw-app.org/family\#size>
  "big"^^<http://www.w3.org/2001/XMLSchema\#string> .}
ORDER BY <http://www.w3.org/2001/XMLSchema#double>(?startTime)
```

Example 2:

Show all intervals where Mary appears from 20 to 80 seconds?

- **Link Parser Output:**

```
    +------------------------------------Xp-----------------------------------+
    |        +----------QI---------+                                          |
    |        +-----Op-----+        |            +------MVp-----+---Jp---+     |
    +---Wi---+    +--Dmc--+        +--Cs-+--Ss--+--MVp--+-Jp+  |  +-Dmcn+     |
    |        |    |       |        |     |      |       |   |  |  |     |     |
LEFT-WALL show.v all intervals.n where Mary appears.v from 20 to 80 seconds.n .
```

- **Internal Representation:**

  ConceptQuery:

    Concept:

      ObjectName = Mary

      AttributeList = { }

    QueryStartTime = 20

    QueryEndTime = 80

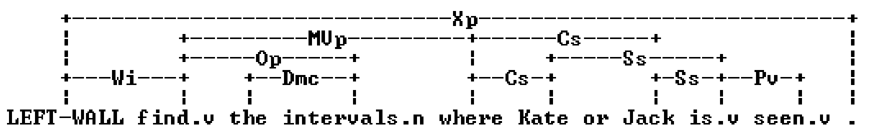
- **SPARQL Query:**

```
SELECT ?startTime ?duration
WHERE {
<http://www.sw-app.org/family#Mary>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator>
  ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>.
?spatioTemporalLocatorType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType.
?mediaTimeType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>.
?mediaTimeType <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration>
  ?duration.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaRelIncrTimePoint>
  ?startTime.
FILTER ( (<http://www.w3.org/2001/XMLSchema#double>(?startTime) +
    <http://www.w3.org/2001/XMLSchema#double>(?duration) ) >=  20 &&
    <http://www.w3.org/2001/XMLSchema#double>(?startTime) <= 80 ) .}
ORDER BY <http://www.w3.org/2001/XMLSchema#double>(?startTime)
```

Example 3:

Find the intervals where Kate or Jack is seen.

- **Link Parser Output:**



```
    +----------------------------Xp-----------------------------+
    |        +---------MVp---------+------Cs-----+              |
    |        +-----Op-----+        |     +-----Ss-----+         |
    +---Wi---+    +--Dmc--+        +--Cs-+       +-Ss-+--Pv-+   |
    |        |    |       |        |     |       |    |     |   |
LEFT-WALL find.v the intervals.n where Kate or Jack is.v seen.v .
```

- **Conjunction 1:**

```
    +-----------------------------Xp----------------------------+
    |        +---------MVp---------+                            |
    |        +-----Op-----+        |                            |
    +---Wi---+    +--Dmc--+        +--Cs-+-----Ss-----+--Pv-+   |
    |        |    |       |        |     |            |     |   |
LEFT-WALL find.v the intervals.n where Kate or Jack is.v seen.v .
```

- **Conjunction 2:**

```
    +-----------------------------Xp----------------------------+
    |        +---------MVp---------+                            |
    |        +-----Op-----+        |                            |
    +---Wi---+    +--Dmc--+        +------Cs-----+-Ss-+--Pv-+   |
    |        |    |       |        |             |    |     |   |
LEFT-WALL find.v the intervals.n where Kate or Jack is.v seen.v .
```

- **Internal Representation:**

  ComplexConceptQuery:

  ConceptQuery:

  Concept:

  ObjectName = Kate

  IsNegationQuery = false

  VideoDuration = 1800

  ConceptQuery:

  Concept:

  ObjectName = Jack

  IsNegationQuery = false

  VideoDuration = 1800

  Connector = OR

- **SPARQL Queries:**

```
SELECT ?startTime ?duration
WHERE {
<http://www.sw-app.org/lost#Kate>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator>
  ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>.
?spatioTemporalLocatorType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType.
?mediaTimeType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration> ?duration.
?mediaTimeType
```

```
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaRelIncrTimePoint>
  ?startTime.}
ORDER BY <http://www.w3.org/2001/XMLSchema#double>(?startTime)

SELECT ?startTime ?duration
WHERE {
<http://www.sw-app.org/lost#Jack>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator>
  ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>.
?spatioTemporalLocatorType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType.
?mediaTimeType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration> ?duration.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaRelIncrTimePoint>
  ?startTime.}
ORDER BY <http://www.w3.org/2001/XMLSchema#double>(?startTime)
```

Example 4:

Retrieve all frames where a male is seen to the right of a female.

- **Link Parser Output:**

```
    +----------------------------------------Xp-------------------------------------------+
    |          +---------MVp--------+                                                     |
    |          +------Op-----+      +---Cs---+                +---Js---+     +---Js--+    |
    +----Wi----+      +--Dmc-+      |   +-Ds-+--Ss-+--Pv-+-MVp+  +--Ds-+--Mp-+ +--Ds-+    |
    |          |      |      |      |   |    |     |     |    |  |     |     | |     |    |
LEFT-WALL retrieve.v all frames.n where a male.n is.v seen.v on the right.n of a female.n .
```

- **Internal Representation:**

  SpatialQuery :

  Concept firstObject:

  ObjectName = male

  Concept secondObject:

  ObjectName = female

  Spatial Relation: RIGHT

- **SPARQL Queries:**

```
SELECT ?object ?startTime ?duration ?coordRef
WHERE {
?object
```

```
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.sw-app.org/lost#male>.
?object <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator>
  ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>.
?spatioTemporalLocatorType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType.
?mediaTimeType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration> ?duration.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaRelIncrTimePoint> ?startTime.
?spatioTemporalLocatorType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#CoordRef> ?coordRef.}
ORDER BY <http://www.w3.org/2001/XMLSchema#double>(?startTime)

SELECT ?object ?startTime ?duration ?coordRef
WHERE {
?object
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.sw-app.org/lost#female>.
?object <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator>
  ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>.
?spatioTemporalLocatorType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType.
?mediaTimeType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration> ?duration.
?mediaTimeType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaRelIncrTimePoint> ?startTime.
?spatioTemporalLocatorType
  <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#CoordRef> ?coordRef.}
ORDER BY <http://www.w3.org/2001/XMLSchema#double>(?startTime)
```

CHAPTER 6

QUERY PROCESSING

In this thesis, concept, complex concept, spatial, temporal, and trajectory query types are supported. A user friendly GUI is designed so that the user can enter NL input and view the query results from the GUI. The query processing procedure includes creating SPARQL queries from the NL input, executing the SPARQL queries, and managing the query results. Throughout this chapter, the details of the query processing are explained. Section 6.1 presents how to run the SPARQL query on the given ontology. Section 6.2 explains the query execution process in detail.

6.1 Running SPARQL Query

After the NL input is mapped to SPARQL query, it is executed on the given ontology by the SemWeb library. The structure of the ontology file is explained in Section 6.1.1 and a sample ontology is given in Appendix C. The functionalities of SemWeb used in the thesis are given in Section 6.1.2.

6.1.1 MPEG-7 Descriptors Used in the System

In the ontology, the following MPEG-7 elements are used:

```
<SpatioTemporalLocator>
  <SpatioTemporalLocatorType>
    <CoordRef/>
    <MediaTime>
      <MediaTimeType>
        <MediaIncrDuration/>
        <MediaRelIncrTimePoint/>
      </MediaTimeType>
    </MediaTime>
  </SpatioTemporalLocatorType>
</SpatioTemporalLocator>
```

SpatioTemporalLocatorType is a spatio-temporal datatype which is used to specify the spatial data changing over time [35]. It is constructed from primitive temporal datatypes such as *MediaTime* and spatial datatypes such as *CoordRef*. It is associated with the video objects and defines the location of the object during the time interval.
CoordRef is used to define the location of the video object. Its format is *"x1 y1 x2 y2"*. (*x1, y1*) is the top-left point, and (*x2, y2*) is the bottom-right point of the rectangle.
MediaTimeType is an MPEG-7 time datatype which used to define a time interval. It contains a *MediaIncrDuration* and a *MediaRelIncrTimePoint*.
MediaIncrDuration is a descriptor which specifies the duration of a media time period.
MediaRelIncrTimePoint specifies a media time point which denotes the start time.

6.1.2 RDF API

SemWeb is a .NET library for reading, writing, manipulating, and querying RDF data [8]. It is used throughout the thesis. The following utilities of SemWeb are used:

- The RDF / XML file is loaded into the MemoryStore.
  ```
  MemoryStore data = new MemoryStore();
  data.Import(new RdfXmlReader("C:\test.owl"));
  ```
- SemWeb provides two inferencing mechanisms: RDFS reasoning and general reasoning with Euler engine. RDFS reasoning is used as the inference method in the thesis. The reasoning engine is initialized with axioms that contain subClassOf relations between classes and subPropertyOf relations between properties. After the initialization step, the engine can make reasoning on any target data model.
  ```
  RDFS engine = new RDFS();
  engine.LoadSchema(new RdfXmlReader("C:\test.owl"));
  data.AddReasoner(engine);
  ```
- The SPARQL query is executed against the ontology by using SemWeb library. To run a query, a *Query* object and a *QueryResultSink* object must be created. When the query

is run, the results of the query will be returned in the *QueryResultSink* object.

```
Query query = new SparqlEngine(queryStr);
MemoryStream mStream = new MemoryStream();
XmlWriter writer = XmlWriter.Create(mStream);
SparqlXmlQuerySink sink = new SparqlXmlQuerySink(writer);
query.Run(data, sink);
```

6.2 Query Execution

The query execution can be divided into two steps: running the SPARQL query and managing the query results. In the first step, the SPARQL queries are executed on the given ontology by using the SemWeb library. In the second step, the results of the first step are prepared for display in the user interface. If the query type is different from the concept type, the second step includes calculating the spatial, temporal, and trajectory relations.

The execution process changes depending on the query type. For example, if the query type is *Concept* or *Trajectory*, then only one SPARQL query is executed. If the type of the query is *Temporal*, two SPARQL queries are executed, and their results along with the *Temporal-RelationType* are used to identify which results satisfy the temporal relationship. Then, the satisfying results and the temporal relation are added to the query result set.

To capture the differences in the execution process of each query type, they are explained in the following subsections separately.

6.2.1 Execution of Concept Query

The concept query type is used when selecting the objects of interest. Also, users can specify one or more attributes for objects to get exact matches or restrict the result set. Below, some example concept queries are given:

Show all frames where a male is seen.
Which objects are seen in the first 5 minutes?
Return scenes where a male is not seen.
Find the intervals that Kate does not appear.

When the user enters an NL input, the query type is extracted first. If the query type is *Concept*, the object name is extracted by using the rules given in Appendix B.2, and the object attributes are found by the rules in Appendix B.10. If start time and end time information exist in the NL query, the rules in Appendix B.9 are used to get the time information. Furthermore, the user input must be checked whether it has negative meaning or not. Then, the *Concept-Query* internal representation is populated with the object name, attributes, start time and end time values, and the boolean flag that shows the negative meaning in the input. The next step is mapping the internal representation into the SPARQL query. The generated SPARQL query is executed against the ontology by using the SemWeb library. If *IsNegationQuery* in the internal representation is false, then it means that no negative meaning can be found in the sentence. Consequently, the results of the query execution, the SPARQL query text, and the concept query information are displayed to the user. For instance, in Table 6.1, the execution results of the query sentence "Show all frames where a male is seen" is shown. In the table, the males that appear in the video and the time intervals in which they appear can be seen.

Table 6.1: The execution results of a concept query.

Jack	6.5	8.6
Jack	8.6	16.7
Hurley	14	39
Jin	39.5	48.2
Jin	48.7	57.3
Boone	72.5	74.7
Sawyer	78	86.4

If *IsNegationQuery* in the internal representation is true, then it means that negative meaning is captured in the NL input. Thus, the inverse of the results of query execution must be found. To illustrate, if the input is *"Show all frames where a male is not seen"*, then the same SPARQL query will be generated. Hence, the results of query execution will be same as Table 6.1. However, since negative meaning exists in the input, we conclude that the user is aiming to search the time intervals where no male is seen. For this reason, we calculate the inverse of Table 6.1 and display it in Table 6.2.

Table 6.1 is ordered by the start time. If the start time of the first row is different than zero, then an entry starting from zero and ending with the start time of the first row is added to the result. The remaining rows of the result table are scanned to find the intervals that a male is

Table 6.2: The inverse of the execution results of a concept query.

Male	0	6.5
Male	57.3	72.5
Male	74.7	78
Male	86.4	128.6

not seen. Finally, if the end time of the last row is different than the video duration, a row starting with the end time of the last row and ending with the video duration is added to the Table 6.2. As it can be seen from Table 6.2, the object name is "Male". If negative meaning is captured in the query and an ontological class is being queried, then the class name is used while calculating the inverse. If an ontological object is being queried, the object name is used. For instance, in the inverse of "Show all frames where Jack is not seen." query result, "Jack" is used as the object name.

Finally, the inverse of query execution results, the SPARQL query text, and the concept query information are displayed in the user interface.

6.2.2 Execution of Complex Concept Query

The complex concept query type includes more than one object that are connected with an "AND" or "OR" connector. Also, these objects can be described with attributes. Examples of the complex concept query are given below:

Retrieve all frames where Jack and Kate appear.
Show me the intervals where a male or a female is seen.
Return the frames where a cat and a dog are not seen.

When an NL input is entered to the system, the query type is found first. If the query type is *Complex Concept*, then the rules in Appendix B.4 are used and the user input is mapped to the *ComplexConceptQuery* internal representation. A complex concept query can be thought as a collection of concept queries, along with a connector parameter. As a result, a SPARQL query is generated for each concept query and each SPARQL query is executed against the ontology individually.

If the connector is an "AND" connector, the intersection of the execution results of concept queries is found. Initially the intersection of the first two result sets is found. Then the obtained result is intersected with the third result set. And this process goes on until all of the concept query results are used. In Figure 6.1, this process is clarified.

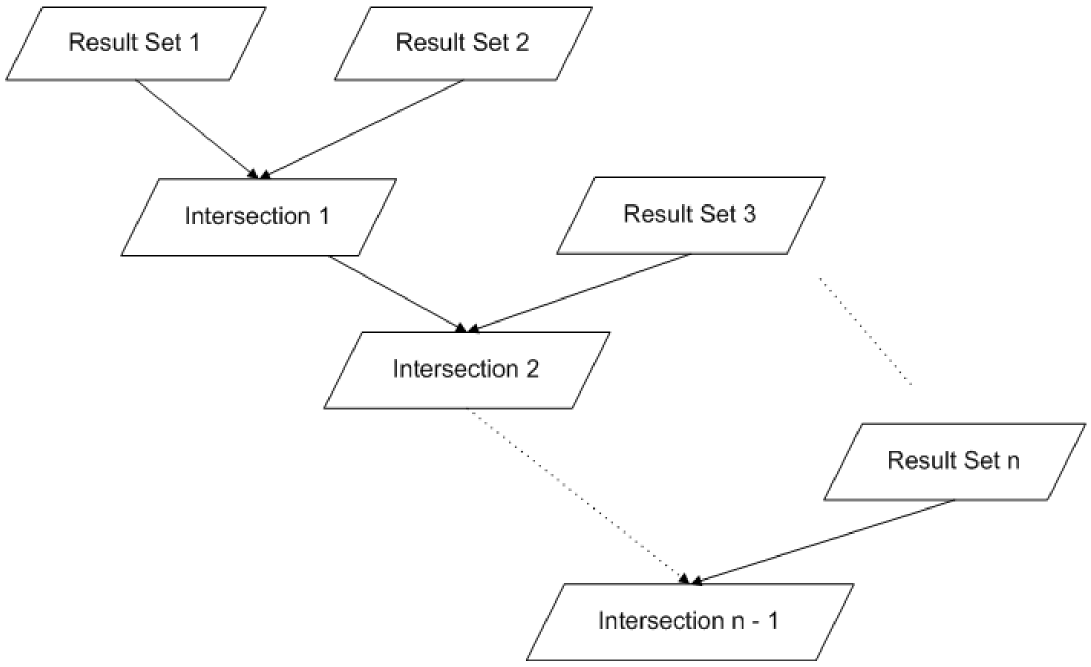


Figure 6.1: The intersection of the execution results of concept queries.

To find the intersection of two result sets, each interval in the first result set is compared with every interval in the second result set. If the first time interval overlaps with the second time interval, then the overlapping interval is found and added to the intersection result. The overlap conditions of two time intervals are given in Figure 6.2. The overlapping regions are displayed as shaded. S1 and E1 stand for start and end times of the first interval, and S2 and E2 stand for the start and end times of the second interval.

If the connector is an “OR” connector, the result is the union of the execution results of concept queries. While finding the union, initially the union of the first two result set is found. Then the union result is joined together with the third result set. And this process goes on until all of the concept query results are used. Again, Figure 6.1 can be used to exemplify the process if we replace “Intersection” with “Union”.

To find the union of two result sets, each interval in the first result set is compared with every interval in the second result set. If the first time interval overlaps with the second time interval,

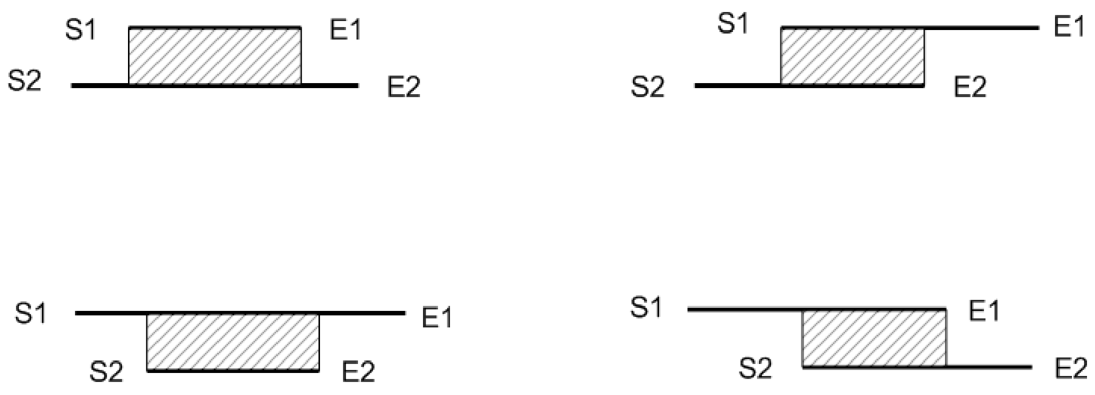


Figure 6.2: Overlapping conditions for two time intervals and their overlapping regions.

then the union of two intervals is found and added to the union result. The overlap conditions of two time intervals are given, and the union of regions is displayed as shaded in Figure 6.3. S1 and E1 stand for start and end times of the first interval, and S2 and E2 stand for start and end times of the second interval.

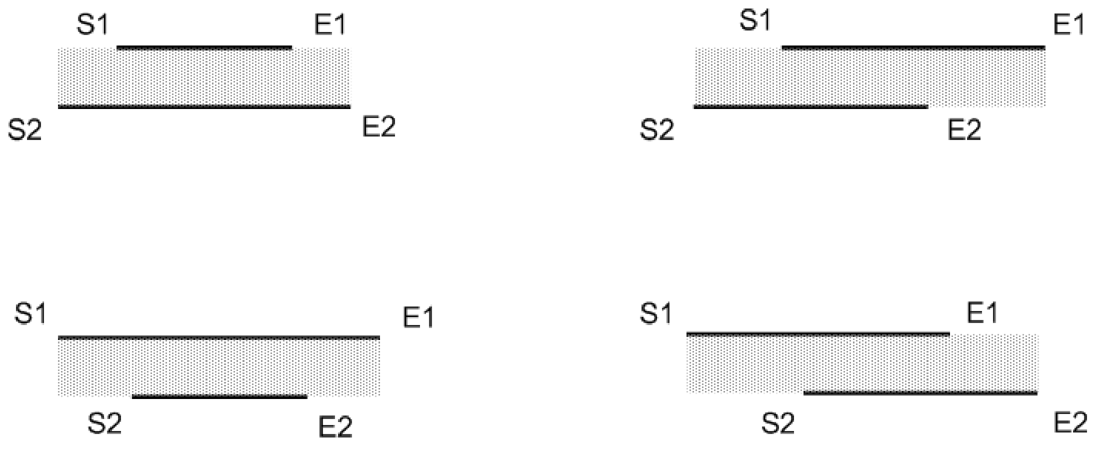


Figure 6.3: Overlapping conditions for two time intervals and the union of these two intervals.

If negative meaning exists in a complex concept query, it will be handled separately in each of the concept queries as explained in section 6.2.1. Then, the union or the intersection of the results of the concept queries will be found depending on the connector type.

6.2.3 Execution of Spatial Query

Spatial queries are used to query the positions of objects relative to each other. A spatial relation exists between two objects that appear in the same frame intervals. There are four positional relations supported in the system: *Left, Right, Above,* and *Below*. Some spatial query examples are as follows:

Return all objects that are on the left of the red car.
Find all frames where the ball appears above a player.

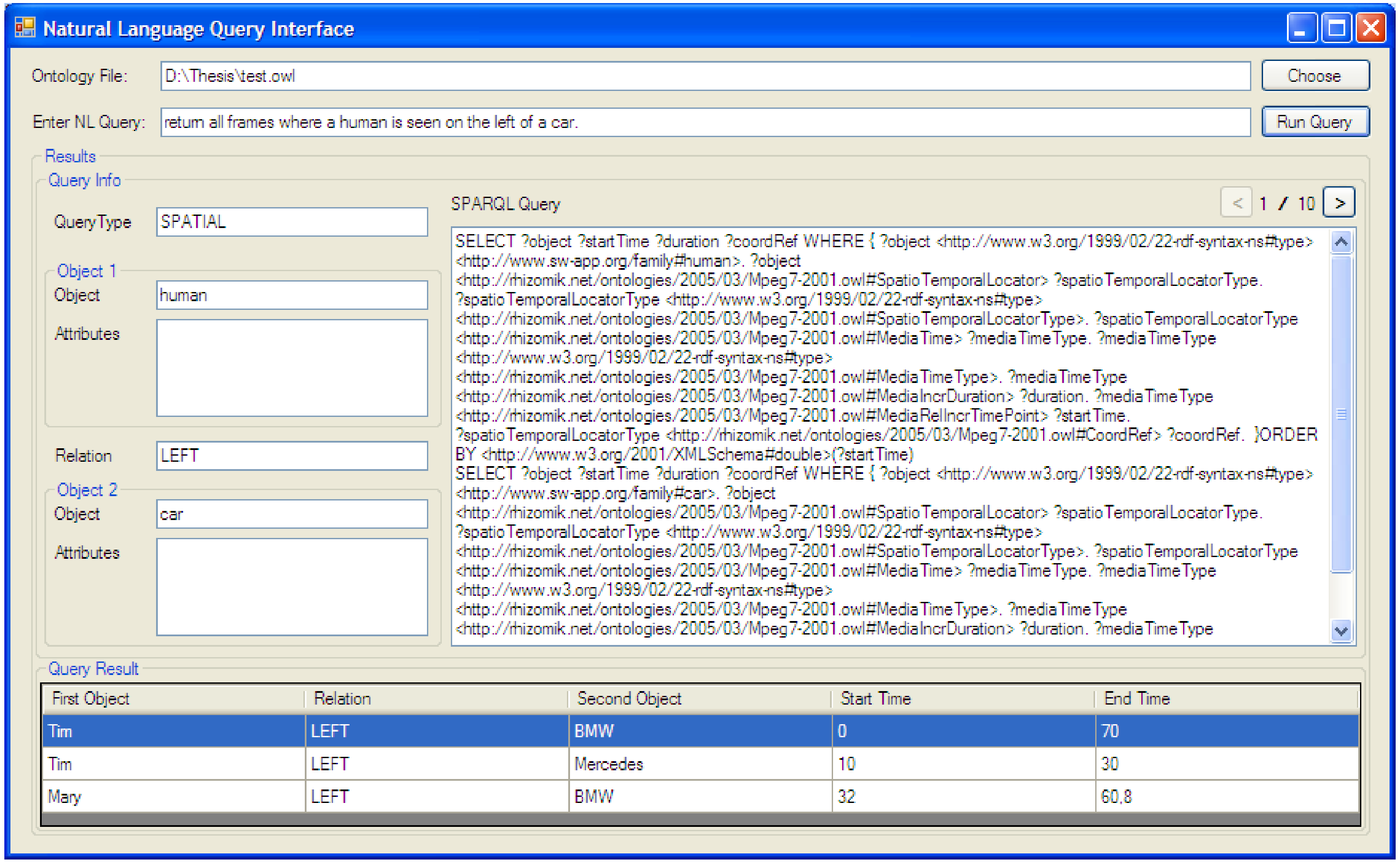
User can describe the objects with attributes in spatial queries. When the user enters an NL query in the system, the query type is found first. If the query type is Spatial, then by using the rules in B.5, the NL input is mapped to the *SpatialQuery* internal representation. There are two concepts and a spatial relation in spatial queries. A SPARQL query is generated for each concept. As a result, two SPARQL queries are executed against the ontology. The results of the two SPARQL queries are joined to see which result couple satisfies the spatial relation. The first condition to satisfy the spatial relation is that the result couple must have overlapping time interval. If this condition holds, then the spatial relation between two objects will be calculated as explained later in this section. The result set contains the objects satisfying the spatial relation, the spatial relation itself, the start time and the end time. Start time and end time are found as in Figure 6.2. Finally, the result set, the texts of two SPARQL queries, and the spatial query information are displayed to the user.

The natural language query interface developed throughout this thesis is shown in Figure 6.4. The query in the figure is *"return all frames where a human is seen on the left of a car"*. The *SpatialQuery* information, the SPARQL queries, and the results of the query execution are displayed in the query interface.

Calculating Spatial Relations:

In order to find the spatial relation between the two objects, the center points of the MBRs covering the objects are used. For instance, let A and B are two objects, and they are represented with their center points as follows:

A (*x1, y1*)
B (*x2, y2*)



Ontology File: D:\Thesis\test.owl

Enter NL Query: return all frames where a human is seen on the left of a car.

Results

Query Info

QueryType: SPATIAL

Object 1

Object: human

Attributes:

Relation: LEFT

Object 2

Object: car

Attributes:

SPARQL Query 1 / 10

```
SELECT ?object ?startTime ?duration ?coordRef WHERE { ?object <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.sw-app.org/family#human>. ?object
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator> ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>. ?spatioTemporalLocatorType
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType. ?mediaTimeType
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>. ?mediaTimeType
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration> ?duration. ?mediaTimeType
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaRelIncrTimePoint> ?startTime.
?spatioTemporalLocatorType <http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#CoordRef> ?coordRef. }ORDER
BY <http://www.w3.org/2001/XMLSchema#double>(?startTime)
SELECT ?object ?startTime ?duration ?coordRef WHERE { ?object <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.sw-app.org/family#car>. ?object
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator> ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>. ?spatioTemporalLocatorType
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType. ?mediaTimeType
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>. ?mediaTimeType
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration> ?duration. ?mediaTimeType
```

Query Result

First Object	Relation	Second Object	Start Time	End Time
Tim	LEFT	BMW	0	70
Tim	LEFT	Mercedes	10	30
Mary	LEFT	BMW	32	60.8

Figure 6.4: Natural language query interface showing the results for “return all frames where a human is seen on the left of a car”

The left and right relation between two objects is determined by using the center values in the x-axis.

```
IF $x1 < x2$
    A **LEFT** B
ELSE IF $x2 < x1$
    A **RIGHT** B
```

The above and below relation between two objects is determined by using the center values in the y-axis.

```
IF $y1 < y2$
    A **BELOW** B
ELSE IF $y2 < y1$
    A **ABOVE** B
```

6.2.4 Execution of Temporal Query

By using temporal queries, the relative appearances of objects in a time sequence can be queried. There are two temporal relations supported in the system: *Before* and *After*. Some examples of temporal queries are as follows:

Return the objects that appear before the policeman.
Show all frames where John appears after the car.
Return the objects that appear from 5 to 10 minutes before the crash.
Return frames where a female appears at least 1 minute before a male.

The user can describe the objects with attributes in temporal queries. When the user enters an NL query in the system, the query type is found first. If the query type is *Temporal*, then by using the rules in Appendix B.6, the NL input is mapped to the *TemporalQuery* internal representation. Moreover, rules in Appendix B.7 are used to find the temporal time filter information. There are two concepts, a temporal relation, and a temporal time filter in temporal queries. A SPARQL query is generated for each concept. As a result, two SPARQL queries are executed against the ontology. The results of the two SPARQL queries are joined to see which result couple satisfies the temporal relation and the time filter. The result set contains the objects satisfying the temporal relation, the temporal relation itself, the first object's start and end times, and the second object's start and end times. Finally, the result set, the texts of two SPARQL queries, and the temporal query information are displayed to the user.

Calculating Temporal Relations:

Temporal relations between two objects are calculated according to the appearance order of objects. During calculation, Allen's interval algebra [1] is used. For instance, let A and B are two objects. According to the Allen's algebra, *Before* relation holds between objects A and B, if the end time of object A is earlier than the start time of object B. *After* relation is inverse of Before. The *Before* and *After* relations are illustrated in Figure 6.5. In the figure, object A appears before object B, as a result object B appears after object A.

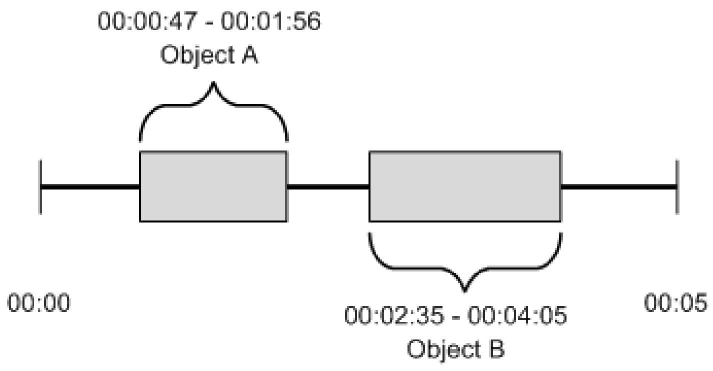


Figure 6.5: Appearances of objects A and B in timeline.

In order to include a time filter in computing the temporal relations, for each interval in the first result set, we calculate a time interval according to the filter information and the time interval of the result. The following pseudocode describes how this interval is calculated.

```
Foreach result in First Result Set
firstObjEndTime = result.EndTime
intervalStartTime = 0
intervalEndTime = 0
isMinute = TimeFilter.MinOrSec
videoDuration = GetVideoDuration

if isMinute
    filterAmount = TimeFilter.FilterAmount * 60
    filterStart TimeFilter.Start * 60
    filterEnd = TimeFilter.End * 60
```

```
switch TimeFilter.FilterType
case FilterType.EXACT
    if isMinute
      intervalStartTime = firstObjEndTime + (filterAmount - 30)
      intervalEndTime = firstObjEndTime + (filterAmount + 30)
    else
      intervalStartTime = firstObjEndTime + (filterAmount - 0.5)
      intervalEndTime = firstObjEndTime + (filterAmount + 0.5)
case FilterType.INTERVAL
      intervalStartTime = firstObjEndTime + filterStart
      intervalEndTime = firstObjEndTime + filterEnd
case FilterType.LEAST:
      intervalStartTime = firstObjEndTime + filterAmount
      intervalEndTime = videoDuration
case FilterType.MOST:
      intervalStartTime = firstObjEndTime;
      intervalEndTime = firstObjEndTime + filterAmount
case FilterType.STANDARD:
      intervalStartTime = firstObjEndTime;
      intervalEndTime = videoDuration;
```

For each object in the first result set, every object of the second result set is examined whether the second object time interval overlaps with the calculated interval. If these two intervals overlap, it means that the two results satisfy the temporal relation with the specified time filter values.

6.2.5 Execution of Trajectory Query

Trajectory queries are used to query the paths of the objects of interest. Two types of trajectory queries, namely object trajectory and directional trajectory, are supported in the system. In object trajectory, the paths of the objects are queried. The following examples can be given for object trajectory:

Return the path of all dogs.
Return the trajectory of the balls.

In directional trajectory, the user can query the objects which make a directional motion. The user can query *East, West, North, South, Northeast, Northwest, Southeast, and Southwest, Right, Left, Upwards (Upward), Downwards (Downward), Upper Right, Upper Left, Lower*

Right, and *Lower Left* directions. Directional trajectory examples are given below:

Which objects go to the southwest?
Return all players who go to the east.
Return objects that go to the left.

The user can query the paths of the objects with their attributes. When the user enters an NL query in the system, the query type is found first. If the query type is *Trajectory*, then by using the rules in Appendix B.8, the NL input is mapped to the *TrajectoryQuery* internal representation. A SPARQL query is generated for the queried concept and it is executed against the ontology by the SemWeb library. If the trajectory type is object, then the paths of the objects in the result set are calculated and displayed to the user. If the trajectory type is directional, the motion paths of the objects in the result set are calculated and the results that satisfy the trajectory direction are shown to user.

In Figure 6.6, the query interface is shown with input "return the trajectory of a human.". The TrajectoryQuery information, the SPARQL query, and the results of the query execution are displayed in the query interface.

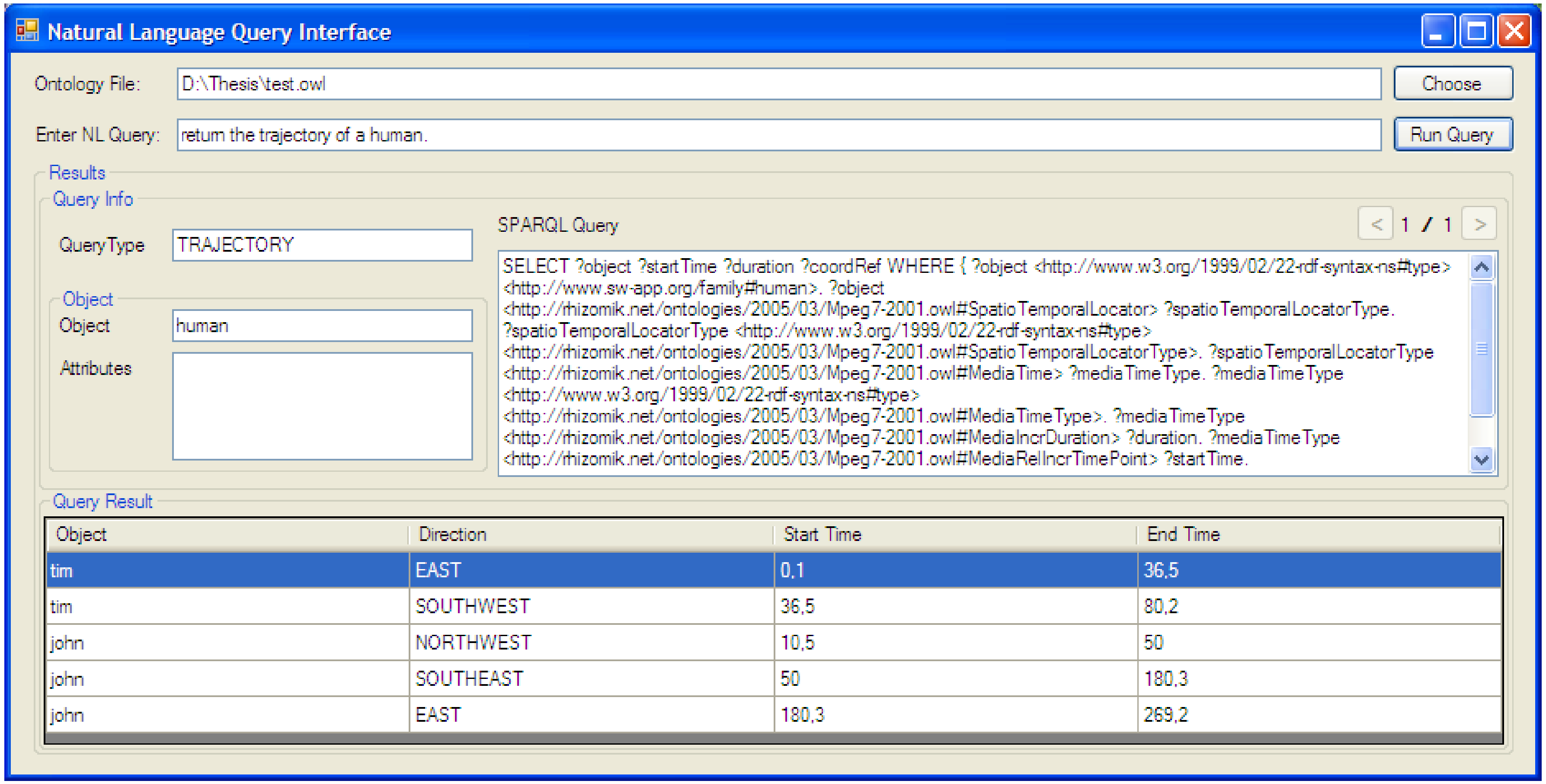
Calculating Trajectory Directions:

When the query results are retrieved from the ontology, an object frame list is formed. The list consists of *<object name, spatial location list>* pairs. If the user queries an ontological class, then more than one object can be returned belonging to this class. The list of spatial location holds the time intervals and the regions that the object is seen. For each object in the object frame list, its route is calculated by using the spatial location list. In order to be able to calculate the trajectories of objects, the following conditions must be satisfied:

1. The successive time intervals should be adjacent. Let *I* be an interval with values (t_s, t_f), where t_s stands for start time and t_f stands for end time. Let $I_i(t_{is}, t_{if})$ and $I_j(t_{js}, t_{jf})$ be two successive time intervals. Then the neighborhood of the intervals can be defined as [33]:

   $t_{js} = t_{if} + 1$

2. The successive regions should be neighbor. Let R1 and R2 are two rectangles in time intervals which are defined above. R1 has coordinates *(r1x1, r1y1, r1x2, r1y2)* and R2



Ontology File: D:\Thesis\test.owl

Enter NL Query: return the trajectory of a human.

QueryType: TRAJECTORY

Object: human

Attributes:

SPARQL Query

```
SELECT ?object ?startTime ?duration ?coordRef WHERE { ?object <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.sw-app.org/family#human>. ?object
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocator> ?spatioTemporalLocatorType.
?spatioTemporalLocatorType <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType>. ?spatioTemporalLocatorType
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTime> ?mediaTimeType. ?mediaTimeType
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType>. ?mediaTimeType
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaIncrDuration> ?duration. ?mediaTimeType
<http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaRelIncrTimePoint> ?startTime.
```

Object	Direction	Start Time	End Time
tim	EAST	0,1	36,5
tim	SOUTHWEST	36,5	80,2
john	NORTHWEST	10,5	50
john	SOUTHEAST	50	180,3
john	EAST	180,3	269,2

Figure 6.6: Natural language query interface showing the results for “return the trajectory of a human.”

has coordinates *(r2x1, r2y1, r2x2, r2y2)*. R2 is a neighbor of R1, if R1 intersects with R2 or R1 touches R2 as given in the Figure 6.7.

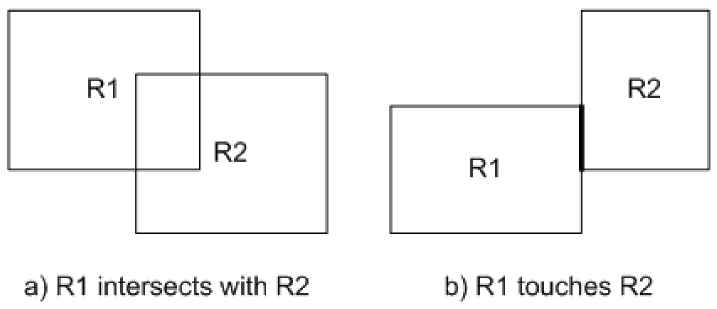


Figure 6.7: Intersect and touch conditions of two rectangles.

If the two conditions are satisfied, then the trajectory direction is calculated by finding the angle between the x-axis and the line that goes through the center points of the two rectangles. This angle can have values from $-\pi$ to π. The directions depending on the angle values can be seen in Figure 6.8.

6.3 Evaluation

In this thesis, a natural language query interface is developed for querying ontologies based on multimedia data. While developing such an interface, we aim to provide a simple and easy to use query interface for end users and save the users from the burden of complex form-based interfaces.

There are five types of query supported: Concept, complex concept, spatial, temporal and trajectory. Rules are defined to handle each query type. They are designed to give the user flexibility of stating his query by different sentence structures.

To handle concept query, seven rules are defined. We intend to cover the wh-questions, subordinate clauses and relative clauses in the rules. The user can query an ontological class, an instance of a class, or a class with an attribute.

There is one rule to capture the negative meaning in the query. In this rule, the “N” link which connects the word “not” to the preceding auxiliaries and modals is being searched. Hence,

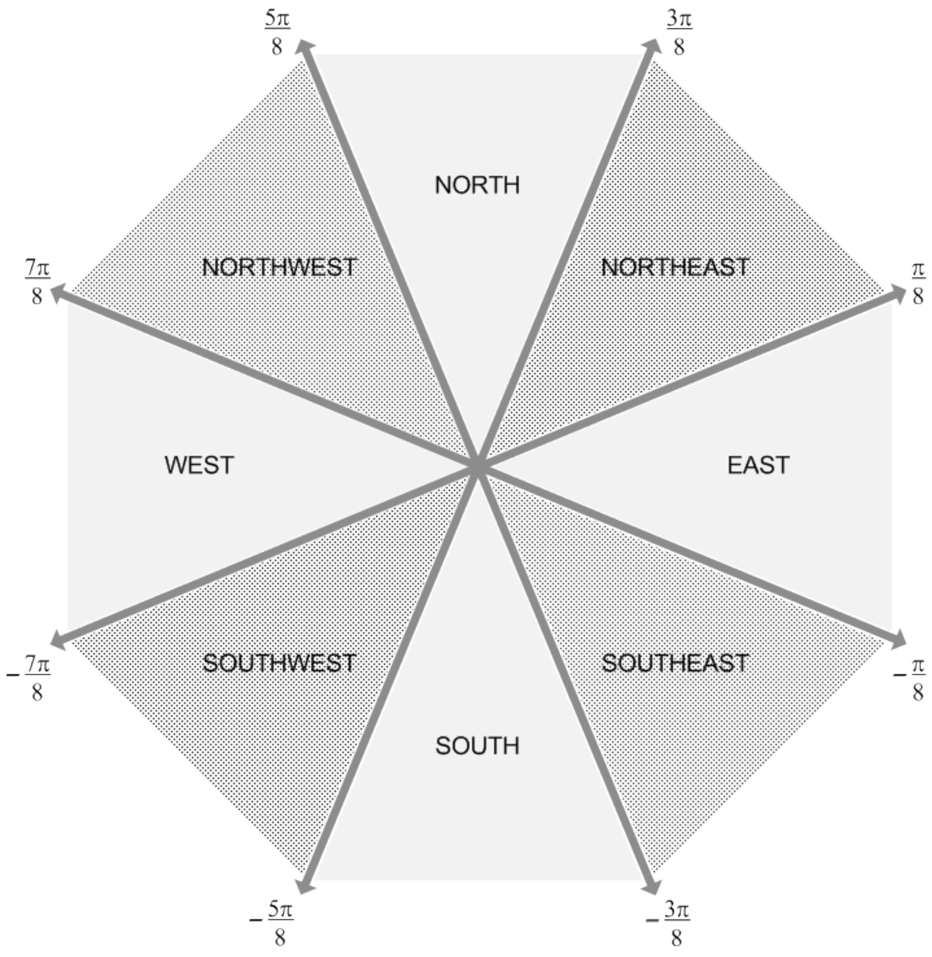


Figure 6.8: The directions according to angle intervals.

we handle the sentences having the word “not” in its word sequence. In order to broaden the scope of the query, new rules must be defined. For example, if we want to handle the inputs including negative meaning such as “Show all frames where no man is seen”, new rules must be defined depending on the link names and orders.

In complex concept query, the rules that are defined for handling concept query are used. In addition to these rules, one rule is used to find the type of the query. There is no limit on the count of the concepts in the query. Each concept can be an ontological class, an instance of a class, or a class with an attribute.

In order to handle spatial query type, six rules are defined. The count of rules is affected by the number of spatial relations. Also, supporting both subordinate clauses and relative clauses affects the rule count.

We define eight rules to answer temporal queries. Three of the rules are used to find the objects and the temporal relation. The remaining five rules are used to extract the time filter information.

Six rules are used to extract trajectory query information. One rule is used for object trajectory and the remaining five rules are used for directional trajectory.

The user can restrict the query within a given time interval. Two rules are used for extracting the time information.

We generate three rules to find the attributes of objects. One rule is used for the object names that are composed of two or more words. Additionally, one rule is used for proper nouns which are composed of two or more words.

In this system, a total of 40 rules are defined to understand the user input. These rules seem to be sufficient to make semantic, spatio-temporal and trajectory queries. However, if more rules are added, the querying capability of the system will be enhanced.

The link parser is used to parse the user queries. The reason why we chose the link parser is that it can tolerate the errors in the NL input to some extent. The link parser does not require to fully understand the given sentence; rather it assigns the syntactic structure to the portion it understands. This feature of the link parser makes our system more flexible, i.e. the system can accept ill-formed NL queries too.

Our system is domain-independent. By using the NL interface, users can query any videos such as TV series, news videos, documentary videos, or personal videos; as long as they are annotated beforehand. The vocabulary is not limited; however the expressiveness of the system is restricted by the rule set.

For now, the videos are manually annotated. As a result, the system is not scalable for a realistic application at this point. However, if a detailed video annotation is done automatically, the proposed framework can be used effectively. This thesis is conducted as a part of a research project partially supported by TUBITAK with grant number EEEAG 107E234. In the scope of the project, an annotation module is currently being developed for automatic face detection and recognition. When our system is integrated with the annotation module, faces will be automatically annotated and semantically queried in natural language.

CHAPTER 7

CONCLUSION

In this thesis we propose a natural language interface for semantic and spatio-temporal querying of multimedia based ontologies. The system offers a user-friendly solution to the semantic content retrieval from ontologies. Users do not need to know the data schema of the ontology nor deal with the complex syntax of formal query languages. They can pose queries to any domain ontologies as long as the ontologies are MPEG-7 based. Furthermore, the use of MPEG-7 based domain ontologies enables the reasoning mechanism.

By using the NL interface, the user can pose concept, complex concept (objects connected with an “AND” or “OR” connector), spatial (left, right . . .), temporal (before, after, at least 10 minutes before, 5 minutes after . . .), object trajectory and directional trajectory (east, west, southeast . . . , left, right, upwards . . .) queries. Furthermore, the system handles the negative meaning in the user input. Hence, the querying capabilities of the system is extended. When the user enters an NL input, it is parsed with the link parser. Link parser is preferred in the parsing step, because it is a robust parser which means that it can assign the syntactic structure to the portion it understands. Specific portions in the sentence, such as objects, attributes, spatial relation, temporal relation, trajectory relation, and time information, are being investigated according to the query type. They are extracted from the parser output by using predefined rules. After the information extraction, SPARQL queries are generated.

To be able to generate a SPARQL query, an entity with the given object name must exist in the ontology. The queries are constructed differently according to the type of the entity (class or instance) in the ontology. The object attributes given in the query text make queries more precise, so we give an importance to attribute handling. For this purpose, three different approaches are developed to map the attributes to SPARQL. The generated queries are executed against the ontology by using SemWeb, an RDF API. Afterwards, the query results are used to calculate spatial, temporal, and trajectory relationships according to the query type.

As a future extension, event queries can be added to the system. Thus, the user can query not only objects but also events such as running, reading a book, and driving a car. Moreover, we are planning to handle compound queries, sentences connected with an “AND” or “OR” connector, since they can be more expressive and beneficial in semantic and spatio-temporal querying of multimedia data.

The spatial and temporal relations that are provided in this thesis are limited. If more spatial relations such as near, far, inside, touch and disjoint and temporal relations such as during, overlaps, meets, and between are added to the system, the usability of NL query interface will be increased.

In this study, the spatial, temporal, and trajectory relations between the objects are calculated after the results are retrieved from the ontology. In the future, it can also be done with ontology rules [13] or extension functions [11]. For example, the spatial relation “left” or the temporal relation “before” can be defined with rules or extension functions, and the objects satisfying the relations will be retrieved from the ontology. As a result, performance of two approaches can be compared.

REFERENCES

[1] Allen's Interval Algebra. http://en.wikipedia.org/wiki/Allen's_Interval_Algebra, last visited on 14 Jan. 2010.

[2] Chapter 6. The SeRQL query language (revision 1.2). http://www.openrdf.org/doc/sesame/users/ch06.html, last visited on 10 May 2010.

[3] Graphics, Multimedia, and Printing Recipes-Recipe 8 12. http://en.csharp-online.net/Graphics,_Multimedia,_and_Printing_Recipes%E2%80%94 Recipe_8_12, last visited on 15 May. 2010.

[4] Link Grammar. http://en.wikipedia.org/wiki/Link_grammar, last visited on 4 Jan. 2010.

[5] Link Grammar. http://www.link.cs.cmu.edu/link/, last visited on 3 Jan. 2010.

[6] Link Grammar .NET Lib. http://www.proai.net/linkgrammar.htm, last visited on 3 Jan. 2010.

[7] Resource Description Framework. http://en.wikipedia.org/wiki/Resource_Description_Framework#RDF_Topics, last visited on 2 Jan. 2010.

[8] Semantic Web/RDF Library for C#/.NET. http://razor.occams.info/code/semweb/, last visited on 3 Jan. 2010.

[9] Sesame: RDF Schema Querying and Storage. http://www.openrdf.org/, last visited on 15 Jan. 2010.

[10] Shallow Parsing. http://en.wikipedia.org/wiki/Shallow_parsing, last visited on 5 Jan. 2010.

[11] SPARQL. http://www.w3.org/TR/rdf-sparql-query/, last visited on 2 Jan. 2010.

[12] SPARQL Tutorial. http://openjena.org/ARQ/Tutorial/index.html, last visited on 2 Jan. 2010.

[13] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/, last visited on 20 Jan. 2010.

[14] The Porter Stemming Algorithm. http://tartarus.org/˜martin/PorterStemmer/, last visited on 14 Jan. 2010.

[15] The Stanford Parser: A statistical parser. http://nlp.stanford.edu/software/lex-parser.shtml, last visited on 15 Jan. 2010.

[16] WordNet - About WordNet. http://wordnet.princeton.edu/, last visited on 17 Jan. 2010.

[17] J.F. Allen. Maintaining knowledge about temporal intervals. 1983.

[18] N. Athanasis, V. Christophides, and D. Kotzinos. Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL). *Lecture notes in computer science*, pages 486–501, 2004.

[19] A. Bernstein, E. Kaufmann, and C. Kaiser. Ginseng: A guided input natural language search engine for querying ontologies. In *5h European Semantic Web Conference (ESWC 2008)*, 2008.

[20] N. Bhatia, P. Gaharwar, P. Patnaik, and S. Sanyal. GuNQ–A Semantic Web Engine with a Keyword based Query Approach.

[21] J. Borsje and H. Embregts. Graphical Query Composition and Natural Language Processing in an RDF Visualization Interface. *Erasmus School of Economics and Business Economics, Vol. Bachelor. Erasmus University, Rotterdam*, 2006.

[22] T. Catarci, P. Dongilli, T. Di Mascio, E. Franconi, G. Santucci, and S. Tessaris. An ontology based visual tool for query formulation support. In *Proceedings/ECAI 2004, 16th Eureopean Conference on Artificial Intelligence, August 22-27, 2004, Valencia, Spain: including Prestigious Applicants of Intelligent Systems,(PAIS 2004)*, page 308. IOS Press, 2004.

[23] M.E. Dönderler, E. Saykol, Ö. Ulusoy, and U. Güdükbay. BilVideo: A video database management system. *IEEE MultiMedia*, 10(1):66–70, 2003.

[24] G. Erözel, N.K. Çiçekli, and I. Çiçekli. Natural language querying for video databases. *Inf. Sci.*, 178(12):2534–2552, 2008.

[25] A. Fadhil and V. Haarslev. OntoVQL: a graphical query language for OWL ontologies. *Proc, of DL*, 7, 2007.

[26] R. Garcia and O. Celma. Semantic integration and retrieval of multimedia metadata. In *5th Knowledge Markup and Semantic Annotation Workshop*, 2005.

[27] A. Şimşek. Ontology-based spatio-temporal video management system. Master's thesis, METU, 2009.

[28] M. Jarrar and M.D. Dikaiakos. MashQL: a query-by-diagram topping SPARQL. 2008.

[29] S. Kara, O. Sabuncu, S. Akpınar, Ö. Alan, N.K. Çiçekli, and F.N. Alpaslan. An Ontology-Based Retrieval System Using Semantic Indexing. In *DesWeb (ICDE) 2010*, 2010.

[30] E. Kaufmann and A. Bernstein. How Useful Are Natural Language Interfaces to the Semantic Web for Casual End-Users? *Lecture Notes in Computer Science*, 4825:281, 2007.

[31] E. Kaufmann, A. Bernstein, and L. Fischer. NLP-Reduce: A "naïve" but domain independent natural language interface for querying ontologies. In *4th European conference on The Semantic Web (ESWC 2007)*, 2007.

[32] E. Kaufmann, A. Bernstein, and R. Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *5th International Semantic Web Conference (ISWC 2006)*, pages 980–981, 2006.

[33] M. Koprulu, N.K. Cicekli, and A. Yazici. Spatio-temporal querying in video databases. *Information Sciences*, 160(1-4):131–152, 2004.

[34] Y. Lei, V. Uren, and E. Motta. Semsearch: A search engine for the semantic web. *Lecture Notes in Computer Science*, 4248:238, 2006.

[35] P. Liu, A. Chankraborty, and L.H. Hsu. A predicate logic approach for MPEG-7 XML document queries. *Markup Lang.*, 3(3):365–381, 2001.

[36] W. Ren, S. Singh, M. Singh, and YS Zhu. State-of-the-art on spatio-temporal information-based video retrieval. *Pattern Recognition*, 42(2):267–282, 2009.

[37] A. Russell, P.R. Smart, D. Braines, and N.R. Shadbolt. NITELIGHT: A Graphical Tool for Semantic Query Construction. 2008.

[38] D. Sleator and D. Temperley. Parsing English with a link grammar. In *3rd International Workshop on Parsing Technologies*, 1993.

[39] V. Tablan, D. Damljanovic, and K. Bontcheva. A natural language query interface to structured information. In *5h European Semantic Web Conference (ESWC 2008)*, 2008.

[40] H. Tarakçı. An ontology-based multimedia information management system. Master's thesis, METU, 2008.

[41] R. Troncy, Celma O., Little S., Garcia R., and Tsinaraki C. MPEG-7 based multimedia ontologies: Interoperability support or interoperability issue. In *Proceedings of the 1st International Workshop on Multimedia Annotation and Retrieval enabled by Shared Ontologies (MAReSO)*, 2007.

[42] H. Wang, K. Zhang, Q. Liu, T. Tran, and Y. Yu. Q2semantic: A lightweight keyword interface to semantic search. *Lecture Notes in Computer Science*, 5021:584, 2008.

[43] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics Morristown, NJ, USA, 1994.

[44] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. Spark: Adapting keyword query to semantic search. *Lecture Notes in Computer Science*, 4825:694, 2007.

APPENDIX A

LINK TYPES USED FOR PARSING

The link names which are used in this thesis and their descriptions are taken from [5].

Table A.1: The link names and their descriptions.

A	A connects pre-noun ("attributive") adjectives to following nouns.
Am	Am is used with comparatives.
AN	AN connects noun-modifiers to following nouns.
C	C connects subordinating conjunctions and certain verbs and adjectives with subjects of clauses.
Cs	Cs is used in several kinds of subordinate clauses. It is used with certain conjunctions, like "when" and "after".
D	D connects determiners to nouns.
D*uw Dmcn Dmcw Ds*w	The first two subscript places on D connectors relate to number agreement. there are three categories of noun and determiner: singular, mass, and plural. The first subscript place distinguishes between singular ("s") and everything else ("m"); the second place distinguishes between plural ("c") and mass ("u") (for "countable" and "uncountable"). The third subscript place on D connectors relates only to postprocessing. D##w connectors are used for question-determiners: "which", "what" and "whose".
DD	DD connects definite determiners ("the", "his") to certain things like number expressions and adjectives acting as nouns.
G	G connects proper noun words together in series.
IDBBA	ID is a special class of link-types that is generated by the parser for idiomatic strings listed in the dictionary.

J	J connects prepositions to their objects.
Js	Js connects prepositions to singular objects.
Jp	Jp connects prepositions to plural objects.
L	L connects certain determiners to superlative adjectives.
M	M connects nouns to various kinds of post-noun modifiers: prepositional phrases, participle modifiers, prepositional relatives, and other kinds.
Mg	Mg connects nouns with present participles.
Mv	Mv connects nouns with passive participles.
Mp	Mp is used for prepositional phrases modifying nouns.
Mr	Mr is used for relative clauses involving “whose”.
MV	MV connects verbs and adjectives to modifying phrases that follow, like adverbs, prepositional phrases, subordinating conjunctions, comparatives, participle phrases with commas, and other things.
MVp	MVp connects prepositions to verbs.
N	N connects the word “not” to preceding auxiliaries and modals.
ND	ND connects numbers with certain expressions which require numerical determiners.
NIa	NI is used in a few special idiomatic number phrases.
NS	NS serves a similar function to ND, only for singular expressions.
NSn	The words “one” and “1” carry NSn.
O	O connects transitive verbs to their objects, direct or indirect.
Mv	Mv connects nouns with passive participles.
Os	Os connector marks nouns as being singular.
Op	Op connector marks nouns as being plural.
P	P connects forms of the verb “be” to various words that can be its complements: prepositions, adjectives, and passive and progressive participles.
Pa	Pa connects certain verbs to predicative adjectives.
Pp	Pp is used to attach forms of “be” to prepositions.
Pv	Pv is used to connect forms of “be” to passive participles.

Table A.1 – continued from previous page

RS	RS is used in subject-type relative clauses to connect the relative pronoun to the verb.
S	S connects subject nouns to finite verbs.
Ss	Ss connects singular nouns to singular verb forms.
Sp	Sp connects plural nouns to plural verb forms.
SI	SI connects subject nouns to finite verbs in cases of subject-verb inversion.
SIs	SIs is used for singular noun and verb forms.
SIpx	SIp is used for plural noun and verb forms.
W	W connects the subjects of main clauses to the wall, in ordinary declaratives, imperatives, and most questions (except yes-no questions).
Wi	Wi is used to connect imperatives to the wall.
Wj	Wj is used to connect prepositional questions to the wall.
Wq	Wq is used to connect object questions, where/when/why questions and adjectival questions to the wall.
Ws	Ws is used to connect subject questions to the wall.

APPENDIX B

RULES TO EXTRACT INFORMATION

B.1 Rules to Extract Return Type

Rule 1:

1. If linkage information starts with *Wi* link, then the question is an imperative question.

   Return type: Right word of the *Op*, *Os*, or *Opn* links.

Example: Return the man that appears in the video.

```
    +---------------------------Xp---------------------------+
    |        +----Os-----+------Bs------+     +---Jp----+    |
    +----Wi--+      +-Ds-+---R--+---RS---+-MVp-+   +-D*u-+    |
    |        |      |    |     |        |     |   |     |    |
LEFT-WALL return.v the man.n that.r appears.v in the video.n .
```

Rule 2:

1. If linkage information starts with *Wj* link, then the question is a prepositional question.

   Return type: Right word of the *Jp*, or *Js* links.

Example: In which frames is the goalkeeper seen?

```
    +-------------------------Xp-------------------------+
    |     +--------Qd--------+----------Pv----------+    |
    |     +----Jp-----+      +----SIs-----+         |    |
    +--Wj-+-JQ-+--Dmc-+      |    +---Ds--+         |    |
    |     |    |      |      |    |       |         |    |
LEFT-WALL in which frames.n is.v the goalkeeper.n seen.v ?
```

Rule 3:

1. If linkage information starts with *Ws* link, then the question is a subject question.

   Return type: Right word of the *Dmcw*, or *Ds*w* links.

Example: Which animals are seen before Ali?

```
    +--------------------Xp---------------------+
    +---Ws--+--Dmcw-+--Spx--+--Pv--+--MVp-+-Js-+  |
    |       |       |       |      |      |    |  |
LEFT-WALL which animals.n are.v seen.v before Ali ?
```

Rule 4:

1. If linkage information starts with *Wq* link, then the question is an object, or where / when / why, or adjectival question.
2. If it is a where / when question, then:
   Return type: Frames.
3. Else
   Return type: Right word of the *Jp*, *Js*, *Dmcw*, or *Ds*w* links.

Example: When is John displayed?

```
    +---------------Xp---------------+
    |             +-----Pv-----+     |
    +---Wq---+--Q-+-SIs+       |     |
    |        |    |    |       |     |
LEFT-WALL when is.v John displayed.v ?
```

B.2 Rules to Extract Concept Query Information

Rule 1:

1. Search for *Cs* link.
2. If one of *Ss*, *Sp*, *Spx* links follows the *Cs* link, and the right word of the *Ss*, *Sp*, or *Spx* links is “appear” or “appears”, then:
   Object: Right word of the *Cs* link.

Example: Show all frames where the ball appears.

```
    +-----------------------Xp-------------------------+
    |        +--------MVp-------+                      |
    |        +-----Op----+      +----Cs----+           |
    +---Wi---+    +--Dmc-+      |    +--Ds-+---Ss--+   |
    |        |    |      |      |    |     |       |   |
LEFT-WALL show.v all frames.n where the ball.n appears.v .
```

Rule 2:

1. Search for *Cs* link.

2. If one of *Ss*, *Sp*, *Spx* links follows the *Cs* link, and a *Pv* link follows one of these links, then:

   Object: Right word of the *Cs* link.

Example: Return the intervals where Mary is seen.

```
    +--------------------------Xp--------------------------+
    |         +----------MVp----------+                    |
    |         +------Op------+        |                    |
    +----Wi---+     +--Dmc---+        +-Cs--+-Ss-+-Pv--+   |
    |         |     |        |        |     |    |     |   |
LEFT-WALL return.v the intervals.n where Mary is.v seen.v .
```

Rule 3:

1. Search for *SIs* or *SIpx* links.

2. If one of *Pv* or *Mv* links follows the link found in step 1, then:

   Object: Right word of the *SIs* or *SIpx* links.

Example: In which frames is the goalkeeper seen?

```
    +-------------------------Xp-------------------------+
    |      +--------Qd--------+----------Pv----------+   |
    |      +----Jp-----+      +-----SIs----+         |   |
    +--Wj--+-JQ+--Dmc--+      |   +---Ds---+         |   |
    |      |   |       |      |   |        |         |   |
LEFT-WALL in which frames.n is.v the goalkeeper.n seen.v ?
```

Rule 4:

1. Search for *Ss* or *Spx* links.

2. If one of *Pv* or *Pvf* links follows the link found in step 1, then:

   Object: Return type.

Example: Which objects are seen in the video?

```
    +-----------------------Xp------------------------+
    |                                   +---Jp---+    |
    +---Ws--+--Dmcw-+--Spx--+--Pv--+-MVp+  +-D*u-+    |
    |       |       |       |      |    |  |     |    |
LEFT-WALL which objects.n are.v seen.v in the video.n ?
```

Rule 5:

1. Search for *RS* link.

2. If the right word of the *RS* link is "appear" or "appears", then:

   Object: Return type.

Example: Show all humans that appear in the video.

```
    +---------------------------Xp---------------------------+
    |        +-----Op----+--------Bp-----+     +---Jp---+    |
    +---Wi---+    +--Dmc-+---R---+---RS--+-MVp-+  +-D*u-+    |
    |        |    |      |       |       |     |  |     |    |
LEFT-WALL show.v all humans.n that.r appear.v in the video.n .
```

Rule 6:

1. Search for *RS* link.

2. If one of *Pv* or *Pvf* links follows the *RS* link, then:

   Object: Return type.

Example: Find the males that are displayed in the scene.

```
    +-------------------------------Xp-------------------------------+
    |        +----Op----+------Bp-----+                +---Js---+    |
    +---Wi---+    +-Dmc-+---R---+--RS-+---Pv---+--MVp--+  +--Ds-+    |
    |        |    |     |       |     |        |       |  |     |    |
LEFT-WALL find.v the males.n that.r are.v displayed.v in the scene.n .
```

Rule 7:

1. Search for *Mg* link.

2. If the right word of the *Mg* link is “appearing”, then:

   Object: Return type.

Example: Return an animal appearing in the last 100 seconds.

```
    +--------------------------------Xp--------------------------------+
    |                                       +---------Jp---------+     |
    |         +-----Os----+                 |  +----DD----+      |     |
    +----Wi---+     +--Ds-+----Mg---+--MVp--+  +--L--+    +-Dmcn-+     |
    |         |     |     |         |       |  |     |    |      |     |
LEFT-WALL return.v an animal.n appearing.v in the last.a 100 seconds.n .
```

B.3 Rules to Find Negative Meaning in the Query

Rule 1:

1. Search for *N* link. If it is found, then the query is a negation query.

Example: Show all frames where Mary is not seen.

```
    +-------------------------Xp-------------------------+
    |        +--------MVp-------+                        |
    |        +-----Op----+      |          +----Pv---+   |
    +---Wi---+    +--Dmc-+      +--Cs-+-Ss-+-N-+     |   |
    |        |    |      |      |     |    |   |     |   |
LEFT-WALL show.v all frames.n where Mary is.v not seen.v .
```

Example: Find all intervals where a male and a female do not appear in the video.

```
    +---------------------------------------------Xp--------------------------------------------+
    |        +---------MVp---------+----------Cs----------+                                     |
    |        +-----Op-----+        +---Cs---+---------Sp---------+----Ifd---+     +---Jp---+    |
    +---Wi---+    +--Dmc--+        |   +-Ds-+       +--Ds-+--Sp--+-N-+      +-MVp-+  +-D*u-+    |
    |        |    |       |        |   |    |       |     |      |   |      |     |  |     |    |
LEFT-WALL find.v all intervals.n where a male.n and a female.n do.v not appear.v in the video.n .
```

B.4 Rules to Extract Complex Concept Query Information

Rule 1:

1. If the linkage has more than one *Conjunction*, then the query is a complex concept query.
2. For each conjunction in the linkage, rules in Appendix B.2 are applied to find the concept query information corresponding to that conjunction.
3. Connector is found by examining the words in the NL input.

Example: Return frames where Jack, Kate and a black car appear.

```
    +----------------------------------Xp----------------------------------+
    |                         +---------------Cs--------------+            |
    |                         |     +----------------Sp---------------+    |
    |                         |     |      +------------Sp------------+    |
    |         +------MVp------+-----Cs-----+      +-----Ds----+       |    |
    +----Wi---+---Op---+      +--Cs-+      |      |    +---A--+---Sp--+    |
    |         |        |      |     |      |      |    |      |       |    |
LEFT-WALL return.v frames.n where Jack , Kate and a black.a car.n appear.v .
```

Conjunction 1:

```
    +----------------------------------Xp----------------------------------+
    |         +------MVp------+                                            |
    +----Wi---+---Op---+      +--Cs-+----------------Sp---------------+    |
    |         |        |      |     |                                 |    |
LEFT-WALL return.v frames.n where Jack , Kate and a black.a car.n appear.v .
```

Conjunction 2:

```
    +----------------------------------Xp----------------------------------+
    |         +------MVp------+                                            |
    +----Wi---+---Op---+      +-----Cs-----+------------Sp------------+    |
    |         |        |      |            |                          |    |
LEFT-WALL return.v frames.n where Jack , Kate and a black.a car.n appear.v .
```

Conjunction 3:

```
    +----------------------------------Xp----------------------------------+
    |                         +--------------Cs---------------+            |
    |         +------MVp------+                   +-----Ds----+            |
    +----Wi---+---Op---+      |                   |    +---A--+---Sp--+    |
    |         |        |      |                   |    |      |       |    |
LEFT-WALL return.v frames.n where Jack , Kate and a black.a car.n appear.v .
```

B.5 Rules to Extract Spatial Query Information

Rule 1:

1. Search for *Ss* or *Spx* links.

2. If *Pp + Js + Mp + (Js | Jp)* or *MVp + Js + Mp + (Js | Jp)* link sequence follows the link found in step 1, then:
   First object: Left word of *Ss* or *Spx*.
   Relation: Right word of first *Js*.
   Second object: Right word of second *Js* or *Jp*.

Example: Show all frames where Ali is seen on the left of the car.

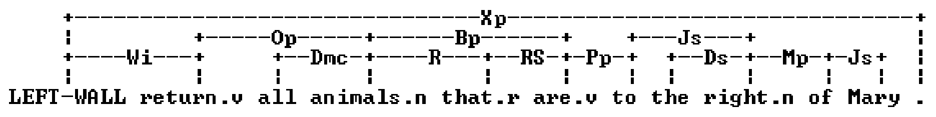
```
    +------------------------------------Xp------------------------------------+
    |        +--------MVp-------+                                              |
    |        +-----Op----+      |                    +---Js---+    +---Js--+   |
    +---Wi---+    +--Dmc-+      +-Cs-+-Ss-+--Pv-+-MVp+  +--Ds-+-Mp-+  +-Ds-+   |
    |        |    |      |      |    |    |     |    |  |     |    |  |    |   |
LEFT-WALL show.v all frames.n where Ali is.v seen.v on the left.n of the car.n .
```

Rule 2:

1. Search for *RS* link.

2. If *Pp + Js + Mp + (Js | Jp)* or *MVp + Js + Mp + (Js | Jp)* link sequence follows the link found in step 1, then:
   First object: Return type.
   Relation: Right word of first *Js*.
   Second object: Right word of second *Js* or *Jp*.

Example: Return all animals that are to the right of Mary.

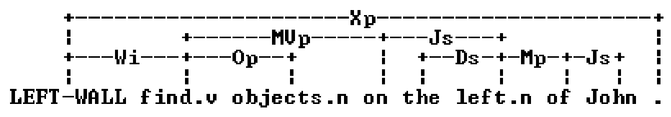


```
    +-------------------------------Xp-------------------------------+
    |         +-----Op-----+------Bp------+    +---Js---+            |
    +----Wi---+     +--Dmc-+----R---+--RS-+-Pp-+  +--Ds-+--Mp-+-Js+  |
    |         |     |      |        |     |    |  |     |     |   |  |
LEFT-WALL return.v all animals.n that.r are.v to the right.n of Mary .
```

Rule 3:

1. If *Mp + Js + Mp + (Js | Jp)* or *MVp + Js + Mp + (Js | Jp)* link sequence is found in the link parser output, then:
   First object: Return type.
   Relation: Right word of first *Js*.
   Second object: Right word of second *Js* or *Jp*.

Example: Find objects on the left of John.



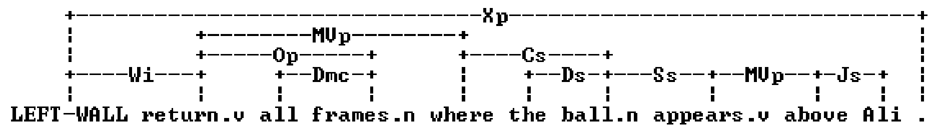
```
    +---------------------Xp---------------------+
    |        +------MVp-----+---Js---+           |
    +---Wi---+---Op--+      |  +--Ds-+-Mp-+-Js+  |
    |        |       |      |  |     |    |   |  |
LEFT-WALL find.v objects.n on the left.n of John .
```

Rule 4:

1. Search for *Ss* or *Spx* links.

2. If *Pp + (Js | Jp)* or *MVp + (Js | Jp)* link sequence follows the link found in step 1 and the right word of the *Pp* or *MVp* link is one of "above" or "below", then:
   First object: Left word of *Ss* or *Spx*.
   Relation: Right word of *Pp* or *MVp*.
   Second object: Right word of second *Js* or *Jp*.

Example: Return all frames where the ball appears above Ali.



```
    +-------------------------------Xp-------------------------------+
    |         +--------MVp--------+                                  |
    |         +-----Op-----+      +----Cs----+                       |
    +----Wi---+     +--Dmc-+      |    +--Ds-+---Ss--+--MVp--+-Js-+  |
    |         |     |      |      |    |     |       |       |    |  |
LEFT-WALL return.v all frames.n where the ball.n appears.v above Ali .
```

Rule 5:

1. Search for *RS* link.

2. If *Pp + (Js | Jp)* or *MVp + (Js | Jp)* link sequence follows the link found in step 1 and the right word of the *Pp* or *MVp* link is one of “above” or “below”, then:

   First object: Return type.

   Relation: Right word of *Pp* or *MVp*.

   Second object: Right word of second *Js* or *Jp*.

Example: Return the cats which are displayed below the dogs.

```
    +-------------------------------Xp---------------------------------+
    |         +----Op----+-----Bp-----+                 +---Jp----+    |
    +----Wi---+     +-Dmc+--R---+-RS--+---Pv---+--MVp---+    +Dmc-+    |
    |         |     |    |      |     |        |        |    |    |    |
LEFT-WALL return.v the cats.n which are.v displayed.v below the dogs.n .
```

Rule 6:

1. *Pp + (Js | Jp)*, *MVp + (Js | Jp)* or *Mp + (Js | Jp)* link sequence is found in the link parser output, and the right word of the *Pp*, *MVp* or *Mp* link is one of “above” or “below”, then:

   First object: Return type.

   Relation: Right word of *Pp*, *MVp* or *Mp*.

   Second object: Right word of second *Js* or *Jp*.

Example: Return the objects below the fireman.

```
    +-----------------------Xp-----------------------+
    |         +--------MVp---------+                 |
    |         +-----Op-----+       +----Js-----+     |
    +----Wi---+     +-Dmc--+       |    +-Ds---+     |
    |         |     |      |       |    |      |     |
LEFT-WALL return.v the objects.n below the fireman.n .
```

B.6 Rules to Extract Temporal Query Information

Rule 1:

1. Search for *Ss* or *Spx* links.

2. If *Pp + (Js | Jp)* or *MVp + (Js | Jp)* link sequence follows the link found in step 1 and the right word of the *Pp* or *MVp* link is one of “after” or “before”, then:

   First object: Left word of *Ss* or *Spx*.

   Relation: Right word of *Pp* or *MVp*.

   Second object: Right word of second *Js* or *Jp*.

Example: Retrieve the frames where Mary appears before Tim.

```
    +-----------------------------Xp------------------------------+
    |          +---------MVp--------+                             |
    |          +------Op-----+      |                             |
    +----Wi----+      +--Dmc-+      +--Cs-+--Ss--+---MVp--+-Js-+  |
    |          |      |      |      |     |      |        |    |  |
LEFT-WALL retrieve.v the frames.n where Mary appears.v before Tim .
```

Rule 2:

1. Search for *RS* link.
2. If *MVp + (Js | Jp)* link sequence follows the link found in step 1 and the right word of the *MVp* link is one of “after” or “before”, then:
   First object: Return type.
   Relation: Right word of *MVp*.
   Second object: Right word of second *Js* or *Jp*.

Example: Return all objects that appear after the dog.

```
    +----------------------------Xp------------------------------+
    |         +-----Op-----+-------Bp-------+      +----Js---+   |
    +----Wi---+     +--Dmc-+----R---+---RS--+--MVp-+    +-Ds-+   |
    |         |     |      |        |       |      |    |    |   |
LEFT-WALL return.v all objects.n that.r appear.v after the dog.n .
```

Rule 3:

1. If *MVp + (Js | Jp)* link sequence is found in the link parser output and the right word of the *MVp* link is one of “after” or “before”, then:
   First object: Return type.
   Relation: Right word of *MVp*.
   Second object: Right word of *Js* or *Jp*.

Example: Return the objects before the policeman.

```
    +-------------------------Xp------------------------+
    |         +---------MVp---------+                   |
    |         +-----Op-----+        +-----Js-----+      |
    +----Wi---+     +--Dmc-+        |    +---Ds--+      |
    |         |     |      |        |    |       |      |
LEFT-WALL return.v the objects.n before the policeman.n .
```

B.7 Rules to Extract Temporal Time Filter Information

Rule 1:

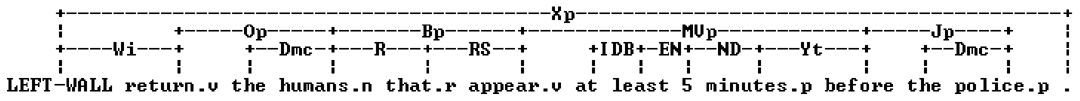
1. The query type should be found as temporal.
2. Search for *IDBBA* link.
3. If one of *ND* or *NSn* link follows the link found in step 1, then:

   Filter Type: LEAST

   Filter Amount: Left word of *ND* or *NSn*

   MinOrSec: Right word of *ND* or *NSn*

Example: Return the humans that appear at least 5 minutes before the police.



```
+-----------------------------------------Xp-----------------------------------------+
|          +-----Op-----+-------Bp------+-------------MVp------------+-----Jp----+     |
+----Wi---+     +--Dmc-+---R---+---RS--+      +IDB+-EN+--ND-+----Yt---+     +--Dmc-+     |
|          |     |      |       |       |      |   |   |     |         |     |      |     |
LEFT-WALL return.v the humans.n that.r appear.v at least 5 minutes.p before the police.p .
```

Rule 2:

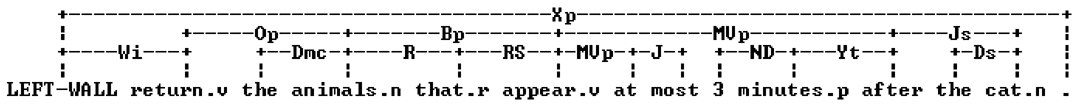
1. The query type should be found as temporal.
2. Search for *J* link and if the right word of the *J* link is “most” and,
3. If one of *ND* or *NSn* link follows the *J* link , then:

   Filter Type: MOST

   Filter Amount: Left word of *ND* or *NSn*

   MinOrSec: Right word of *ND* or *NSn*

Example: Return the animals that appear at most 3 minutes after the cat.



```
+-----------------------------------------Xp-----------------------------------------+
|          +-----Op-----+-------Bp-------+------------MVp-----------+----Js---+      |
+----Wi---+      +--Dmc-+----R---+---RS--+-MVp-+-J-+  +--ND-+---Yt--+     +-Ds-+      |
|          |      |      |        |       |     |   |  |     |       |     |    |      |
LEFT-WALL return.v the animals.n that.r appear.v at most 3 minutes.p after the cat.n .
```

Rule 3:

1. The query type should be found as temporal.
2. Search for *MVp* or *Mp* links.
3. If one of *Jp* or *J* links follows the link found in step 1, and the left word of the *Jp* or *J* link is “from”, then:

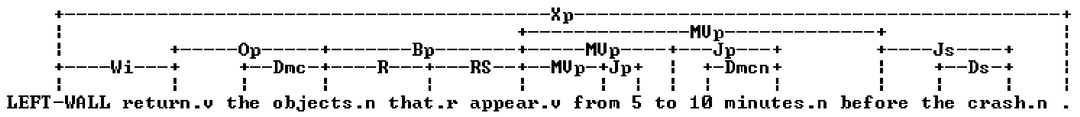
   Filter Type: INTERVAL

   Start: Right word of first *Jp* or right word of *J*

End: Left word of *Dmcn*

MinOrSec: Right word of second *Jp*

Example: Return the objects that appear from 5 to 10 minutes before the crash.



```
    +------------------------------------------Xp------------------------------------------+
    |                                       +--------------MVp-------------+               |
    |         +-----Op-----+-------Bp-------+-----MVp----+---Jp---+        +----Js----+    |
    +----Wi---+     +--Dmc-+----R---+---RS--+--MVp-+Jp+  |  +-Dmcn+        |    +--Ds-+    |
    |         |     |      |        |       |      |  |  |  |     |        |    |     |    |
LEFT-WALL return.v the objects.n that.r appear.v from 5 to 10 minutes.n before the crash.n .
```

Rule 4:

1. The query type should be found as temporal.

2. Search for *ND* or *NSn* link.

3. If *IDBBA* link follows the *ND* or *NSn* link, then:

   Filter Type: LEAST

   Filter Amount: Left word of *ND* or *NSn*

   MinOrSec: Right word of *ND* or *NSn*

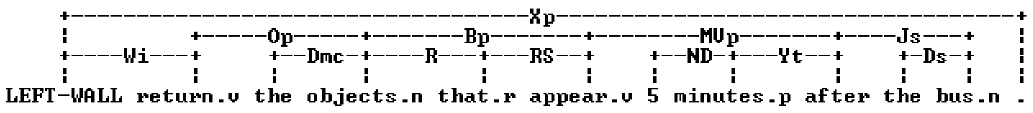
4. Else:

   Filter Type: EXACT

   Filter Amount: Left word of *ND* or *NSn*

   MinOrSec: Right word of *ND* or *NSn*

Example: Return the objects that appear 5 minutes after the bus.



```
    +-----------------------------------Xp-----------------------------------+
    |         +-----Op-----+-------Bp-------+-------MVp--------+----Js---+   |
    +----Wi---+     +--Dmc-+----R---+---RS--+    +--ND-+---Yt--+    +-Ds-+   |
    |         |     |      |        |       |    |     |       |    |    |   |
LEFT-WALL return.v the objects.n that.r appear.v 5 minutes.p after the bus.n .
```

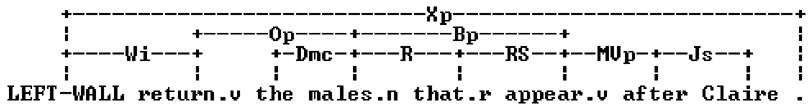
Rule 5:

1. The query type should be found as temporal.

2. If none of the above 4 rules applied, then:

   Filter Type: STANDARD

Example: Return the males that appear after Claire.



```
    +---------------------------Xp--------------------------+
    |         +-----Op----+-------Bp------+                 |
    +----Wi---+     +-Dmc-+---R---+---RS--+--MVp-+--Js--+   |
    |         |     |     |       |       |      |      |   |
LEFT-WALL return.v the males.n that.r appear.v after Claire .
```

B.8 Rules to Extract Trajectory Query Information

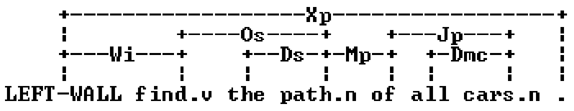
Rule 1:

1. Search for *Wi* + *Os* or *Wi* + *Op* links.
2. If the right word of *Os* or *Op* is one of “trajectory”, “trajectories”, “path”, and “paths”, then:

   Trajectory type: Object.

   Object: Right word of *Js* or *Jp*.

Example: Find the path of all cars.



```
    +-----------------Xp------------------+
    |        +----Os----+    +---Jp----+  |
    +---Wi---+    +--Ds-+-Mp-+  +-Dmc--+  |
    |        |    |     |    |  |      |  |
LEFT-WALL find.v the path.n of all cars.n .
```

Rule 2:

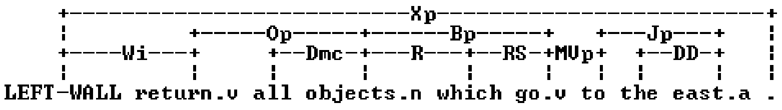
1. Search for *RS* link.
2. If *MVp* + *Jp* link sequence follows the link found in step 1 and the right word of *Jp* is one of “east”, “west”, “north”, “south”, “southeast”, “southwest”, “northeast”, “northwest”, “right” and “left” then:

   Trajectory type: Directional.

   Object: Return type.

   Direction: Right word of *Jp*.

Example: Return all objects which go to the east.



```
    +-------------------------Xp--------------------------+
    |         +-----Op-----+------Bp-----+   +---Jp---+   |
    +----Wi---+     +--Dmc-+---R---+--RS-+MVp+  +--DD-+   |
    |         |     |      |       |     |   |  |     |   |
LEFT-WALL return.v all objects.n which go.v to the east.a .
```

Rule 3:

1. Search for *Ss, Sp* or *Spx* link.
2. If *MVp* + *Jp* link sequence follows the link found in step 1 and the right word of *Jp* is one of “east”, “west”, “north”, “south”, “southeast”, “southwest”, “northeast”, “northwest”, “right” and “left” then:

Trajectory type: Directional.

Object: Left word of *Ss, Sp,* or *Spx*.

Direction: Right word of *Jp*.

Example: Show all frames where a player goes to the east.

```
    +------------------------------Xp-------------------------------+
    |       +--------MVp--------+                                   |
    |       +----Op-----+       +---Cs---+            +---Jp---+    |
    +--Wi---+     +-Dmc-+       |   +-Ds-+--Ss---+-MVp+   +-DD-+    |
    |       |     |     |       |   |    |       |    |   |    |    |
LEFT-WALL show.v all frames.n where a player.n goes.v to the east.a .
```

Rule 4:

1. Search for *MVa* link.

2. If the right word of MVa link is one of “upward”, “upwards”, “downward” and “downwards”, then:

   Trajectory type: Directional.

   Object: Return type or left word of *Ss, Sp,* or *Spx*.

   Direction: Right word of *MVa*.

Example: Return objects that go downwards.

```
    +----------------------Xp----------------------+
    |        +--------------MVa--------------+     |
    |        |         +-----Bp------+       |     |
    +---Wi---+---Op----+---R---+-RS--+       |     |
    |        |         |       |     |       |     |
LEFT-WALL return.v objects.n that.r go.v downwards .
```

Rule 5:

1. Search for *MVp* + *Js* or *Mp* + *Js* links.

2. If the link sequence in step 1 is found, then search for *Am* link.

3. If *Am* link is found, then:

   Trajectory type: Directional.

   Object: Return type or left word of *Ss, Sp,* or *Spx*.

   Direction: Left word of *Am* + Right word of *Js*.

Example: Return the objects that go to the lower right.

```
    +------------------------------Xp-------------------------------+
    |                                        +-------Js--------+    |
    |        +-----Op------+-----Bp------+   |   +-----Ds------+    |
    +---Wi---+      +-Dmc--+---R---+-RS--+MVp+   |     +--Am---+    |
    |        |      |      |       |     |   |   |     |       |    |
LEFT-WALL return.v the objects.n that.r go.v to the lower.a right.n .
```

Rule 6:

1. Search for *MVp* + *Js* or *Mp* + *Js* links.

2. If the link sequence in step 1 is found, then search for *AN* or *A* link.

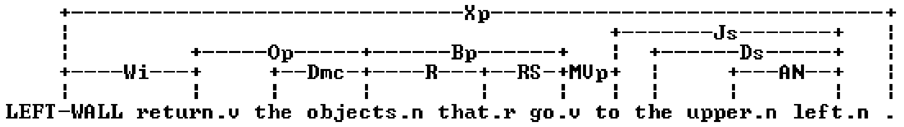
3. If *AN* or *A* link is found, then:

   Trajectory type: Directional.

   Object: Return type or left word of *Ss, Sp,* or *Spx*.

   Direction: Left word of *AN* or *A* + Right word of *Js*.

Example: Return the objects that go to the upper left.



```
    +------------------------------Xp------------------------------+
    |                                         +-------Js-------+   |
    |         +-----Op-----+------Bp------+   |  +------Ds-----+   |
    +----Wi---+     +--Dmc-+----R---+--RS-+MVp+  |     +---AN--+   |
    |         |     |      |        |     |   |  |     |       |   |
LEFT-WALL return.v the objects.n that.r go.v to the upper.n left.n .
```

B.9 Rules to Extract Time Information

Rule 1:

1. Search for *MVp* or *Mp* links.

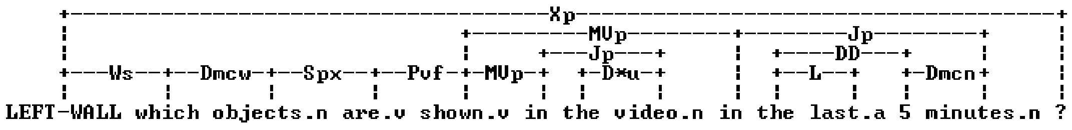
2. If *Jp* link follows the link found in step 1, and the right word of the *Jp* link is “seconds” or “minutes”, then:

   First / Last: Right word of *L*.

   Time: Right word of *DD*.

   Seconds / Minutes: Right word of *Jp*.

Example: Which objects are shown in the video in the last 5 minutes?



```
    +-------------------------------------Xp-------------------------------------+
    |                             +---------MVp--------+--------Jp--------+     |
    |                              |     +---Jp---+     |  +----DD---+     |     |
    +---Ws--+--Dmcw-+--Spx--+--Pvf-+-MVp-+  +-D*u-+     |  +--L--+   +-Dmcn+     |
    |       |       |       |      |     |  |     |     |  |     |   |     |     |
LEFT-WALL which objects.n are.v shown.v in the video.n in the last.a 5 minutes.n ?
```

Rule 2:

1. Search for *MVp* or *Mp* links.

2. If one of *Jp* or *J* links follows the link found in step 1, and the left word of the *Jp* or *J* link is “from”, then:

   Start Time: Right word of first *Jp* or right word of *J*.

   End Time: Left word of *Dmcn*.

   Seconds / Minutes: Right word of second *Jp*.

Example: Return all humans appearing from 5 to 50 seconds.

```
    +------------------------------Xp------------------------------+
    |         +-----Op-----+         +------MVp-----+---Jp---+     |
    +----Wi---+     +--Dmc-+----Mg---+---MVp--+Jp+  |  +-Dmcn+     |
    |         |     |      |         |        |  |  |  |     |     |
LEFT-WALL return.v all humans.n appearing.v from 5 to 50 seconds.n .
```

B.10 Rules to Extract Attributes

Rule 1:

1. After finding objects, search for *A* link which has the object as right connector. Then:

   Attribute: Left word of *A*.

Example: Return all frames where the big red car is seen.

Attributes in example: big, red.

```
    +--------------------------------Xp---------------------------------+
    |                             +----------Cs---------+               |
    |         +--------MVp--------+    +-------Ds-------+               |
    |         +-----Op-----+      |    |    +-----A-----+               |
    +----Wi---+     +--Dmc-+      |    |    |     +--A--+--Ss-+--Pv-+   |
    |         |     |      |      |    |    |     |     |     |     |   |
LEFT-WALL return.v all frames.n where the big.a red.a car.n is.v seen.v .
```

Rule 2:

1. Search for *Mp* link.

2. If the right word of *Mp* link is “with”, and a *Jp* link follows the *Mp* link, then:

   Attribute Name: Right word of *Jp*, Attribute Value: Left word of *A*.

Example: Find the intervals where a car with red color is seen.

Attribute Name: color, Attribute Value: red.

```
    +------------------------------------Xp-----------------------------------+
    |        +---------MVp---------+       +-----------Ss-----------+         |
    |        +-----Op-----+        +---Cs--+     +-----Jp----+      |         |
    +---Wi---+    +--Dmc--+        |   +-Ds+--Mp-+    +---A--+      +--Pv-+   |
    |        |    |       |        |   |   |     |    |      |      |     |   |
LEFT-WALL find.v the intervals.n where a car.n with red.a color.n is.v seen.v .
```

Rule 3:

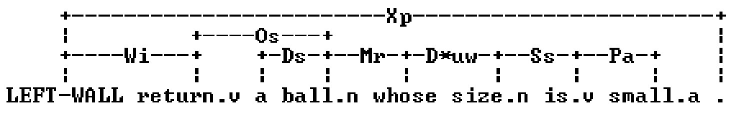
1. Search for *Mr* link.

2. If one of *Ss* + *Pa* or *Ss* + *Opt* link sequence follows the *Mr* link, then:

   Attribute Name: Right word of *D*uw*, Attribute Value: Right word of *Pa* or *Opt*.

Example: Return a ball whose size is small.

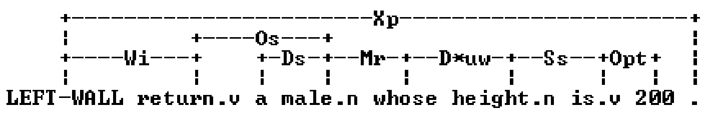
Attribute Name: size, Attribute Value: small.



```
    +------------------------Xp-----------------------+
    |         +----Os---+                             |
    +----Wi---+    +-Ds-+--Mr-+-D*uw-+--Ss-+--Pa-+    |
    |         |    |    |     |      |     |     |    |
LEFT-WALL return.v a ball.n whose size.n is.v small.a .
```

Example: Return a male whose height is 200.

Attribute Name: height; Attribute Value: 200.



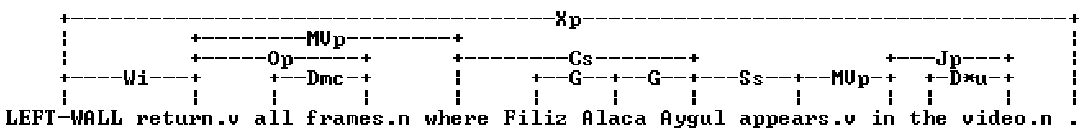
```
    +-----------------------Xp----------------------+
    |         +----Os---+                           |
    +----Wi---+    +-Ds-+--Mr-+--D*uw-+--Ss--+Opt+  |
    |         |    |    |     |       |      |   |  |
LEFT-WALL return.v a male.n whose height.n is.v 200 .
```

B.11 Special Cases

Case 1:

- If there's a *G* link, then the object is a proper noun which is composed of two or more words.

Example: Return all frames where Filiz Alaca Aygul appears in the video.



```
    +-------------------------------------Xp-------------------------------------+
    |         +--------MVp--------+                                              |
    |         +-----Op-----+      +--------Cs-------+              +---Jp---+    |
    +----Wi---+     +--Dmc-+      |     +--G--+--G--+---Ss--+--MVp-+  +-D*u-+    |
    |         |     |      |      |     |     |     |       |      |  |     |    |
LEFT-WALL return.v all frames.n where Filiz Alaca Aygul appears.v in the video.n .
```

Case 2:

- If there's an *AN* link, then the object name is composed of two or more words.

Example: In which frames is the prime minister seen?

```
    +----------------------------Xp----------------------------+
    |      +--------Qd--------+--------BIt--------+            |
    |      +-----Jp----+      +---SIs---+         |            |
    +--Wj--+-JQ+--Dmc--+      |   +-D*u-+         +---Mv---+   |
    |      |   |       |      |   |     |         |        |   |
LEFT-WALL in which frames.n is.v the prime.n minister.i seen.v ?
```

APPENDIX C

AN EXAMPLE ONTOLOGY

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns="http://www.sw-app.org/family#"
    xmlns:j.0="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
  <owl:Ontology rdf:about="http://www.sw-app.org/family"/>
  <rdfs:Class rdf:about="http://www.sw-app.org/family#female">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
    <owl:disjointWith>
      <rdfs:Class rdf:about="http://www.sw-app.org/family#male"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <rdfs:Class rdf:about="http://www.moass.com/ontology#MOASSSemanticBaseType"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <rdfs:Class rdf:about="http://www.sw-app.org/family#human"/>
    </rdfs:subClassOf>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://www.sw-app.org/family#human">
    <rdfs:subClassOf rdf:resource="http://www.moass.com/ontology#MOASSSemanticBaseType"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
    <owl:equivalentClass>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <rdfs:Class rdf:about="http://www.sw-app.org/family#female"/>
          <rdfs:Class rdf:about="http://www.sw-app.org/family#male"/>
        </owl:unionOf>
      </owl:Class>
    </owl:equivalentClass>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://www.sw-app.org/family#male">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.moass.com/ontology#MOASSSemanticBaseType"/>
    <rdfs:subClassOf rdf:resource="http://www.sw-app.org/family#human"/>
    <owl:disjointWith rdf:resource="http://www.sw-app.org/family#female"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#MediaTimeType"/>
  <rdfs:Class rdf:about="http://www.sw-app.org/family#vehicle">
    <rdfs:subClassOf rdf:resource="http://www.moass.com/ontology#MOASSSemanticBaseType"/>
```

```
</rdfs:Class>
<rdfs:Class rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#SpatioTemporalLocatorType"/>
<rdfs:Class rdf:about="http://www.sw-app.org/family#car">
  <rdfs:subClassOf rdf:resource="http://www.sw-app.org/family#vehicle"/>
</rdfs:Class>
<owl:DatatypeProperty rdf:about="http://www.sw-app.org/family#model">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ford Fiesta</rdf:first>
          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
        </rdf:rest>
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ford Focus</rdf:first>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
  <rdfs:domain rdf:resource="http://www.sw-app.org/family#vehicle"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.sw-app.org/family#color">
  <rdfs:domain rdf:resource="http://www.sw-app.org/family#car"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<female rdf:about="http://www.sw-app.org/family#Kate">
  <j.0:MediaUri>C:\family.avi</j.0:MediaUri>
  <j.0:SpatioTemporalLocator>
    <j.0:SpatioTemporalLocatorType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#sTLocType_Sun_51.8-53.0">
      <j.0:CoordRef>403 376 569 449</j.0:CoordRef>
      <j.0:MediaTime>
        <j.0:MediaTimeType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#mTType_51.8-53.0">
          <j.0:MediaIncrDuration>1.2</j.0:MediaIncrDuration>
          <j.0:MediaRelIncrTimePoint>51.8</j.0:MediaRelIncrTimePoint>
        </j.0:MediaTimeType>
      </j.0:MediaTime>
    </j.0:SpatioTemporalLocatorType>
  </j.0:SpatioTemporalLocator>
  <j.0:SpatioTemporalLocator>
    <j.0:SpatioTemporalLocatorType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#sTLocType_Sun_57.4-62.5">
      <j.0:CoordRef>175 163 641 459</j.0:CoordRef>
      <j.0:MediaTime>
        <j.0:MediaTimeType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#mTType_57.4-62.5">
          <j.0:MediaIncrDuration>5.2</j.0:MediaIncrDuration>
          <j.0:MediaRelIncrTimePoint>57.4</j.0:MediaRelIncrTimePoint>
        </j.0:MediaTimeType>
      </j.0:MediaTime>
    </j.0:SpatioTemporalLocatorType>
  </j.0:SpatioTemporalLocator>
</female>
<male rdf:about="http://www.sw-app.org/family#Sawyer">
  <j.0:MediaUri>C:\family.avi</j.0:MediaUri>
  <j.0:SpatioTemporalLocator>
    <j.0:SpatioTemporalLocatorType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#sTLocType_Sawyer_87.8-90.8">
      <j.0:CoordRef>243 166 575 470</j.0:CoordRef>
      <j.0:MediaTime>
        <j.0:MediaTimeType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#mTType_87.8-90.8">
          <j.0:MediaIncrDuration>3.0</j.0:MediaIncrDuration>
          <j.0:MediaRelIncrTimePoint>87.8</j.0:MediaRelIncrTimePoint>
        </j.0:MediaTimeType>
      </j.0:MediaTime>
    </j.0:SpatioTemporalLocatorType>
  </j.0:SpatioTemporalLocator>
  <j.0:SpatioTemporalLocator>
    <j.0:SpatioTemporalLocatorType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#sTLocType_Sawyer_94.0-99.5">
```

```
        <j.0:CoordRef>337 117 768 492</j.0:CoordRef>
        <j.0:MediaTime>
          <j.0:MediaTimeType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#mTType_94.0-99.5">
            <j.0:MediaIncrDuration>5.6</j.0:MediaIncrDuration>
            <j.0:MediaRelIncrTimePoint>94.0</j.0:MediaRelIncrTimePoint>
          </j.0:MediaTimeType>
        </j.0:MediaTime>
      </j.0:SpatioTemporalLocatorType>
    </j.0:SpatioTemporalLocator>
  </male>
  <car rdf:about="http://www.sw-app.org/family#RedCar">
    <color>red</color>
    <j.0:SpatioTemporalLocator>
      <j.0:SpatioTemporalLocatorType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#sTLocType_RedCar_60-70">
        <j.0:CoordRef>200 249 400 337</j.0:CoordRef>
        <j.0:MediaTime>
          <j.0:MediaTimeType rdf:about="http://rhizomik.net/ontologies/2005/03/Mpeg7-2001.owl#mTType_60-70">
            <j.0:MediaRelIncrTimePoint>60.0</j.0:MediaRelIncrTimePoint>
            <j.0:MediaIncrDuration>10.0</j.0:MediaIncrDuration>
          </j.0:MediaTimeType>
        </j.0:MediaTime>
      </j.0:SpatioTemporalLocatorType>
    </j.0:SpatioTemporalLocator>
    <model>Ford Fiesta</model>
    <j.0:MediaUri>C:\family.avi</j.0:MediaUri>
  </car>
</rdf:RDF>
```