Natural Language Query Processing in Multimedia Ontologies

Filiz Alaca Aygul[1], Nihan Kesim Cicekli[2] and Ilyas Cicekli[3]
[1]*Turkish Scientific and Technological Research Council (TUBITAK), Ankara, Turkey*
[2]*Department of Computer Engineering, METU, Ankara, Turkey*
[3]*Department of Computer Engineering, Hacettepe University, Ankara, Turkey*

Keywords: Natural Language Querying, Spatio-temporal Querying, MPEG-7 Ontology, SPARQL.

Abstract: In this paper 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, spatial, temporal, object trajectory and directional trajectory queries. Furthermore, the system handles the negative meaning in the user query. When the user enters a natural language query, 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 information extraction rules. After the information extraction, SPARQL queries are generated, and executed against the ontology by using an RDF API. The results are used to calculate spatial, temporal, and trajectory relations between objects. The results satisfying the required relations are displayed in a tabular format and the user can navigate through the multimedia content.

1 INTRODUCTION

Due to the increasing diversity and volume of multimedia data, the problem of storing and querying the semantic content of such data has attracted the attention of a considerable amount of research work (Donderler et al., 2003); (Koprulu et al., 2004); (Ren et al., 2009). MPEG-7 standard is proposed in order to standardize the description of semantic contents of multimedia data. Since it is implemented by XML-Schemas, it does not have a reasoning capability. Also, its XML based architecture prevents interoperability between multimedia management systems. In order to overcome these difficulties of MPEG-7 standard, four MPEG-7 based multimedia ontologies, represented in OWL, have been developed (Troncy et al., 2007).

In order to query the contents of ontologies or knowledge bases, there are some formal query languages such as SPARQL (Prud'hommeaux and Seaborne, 2008) and SeRQL (Broeskstra and Kampman, 2003). Since these languages have a quite complex syntax and require a good understanding of both the underlying schema and the language syntax, they are not preferred by the casual end users. Five different methods have been proposed to ease the query construction over ontologies: keyword-based, graph-based, form-based, view-based, and natural language based.

Users are mostly familiar with keyword-based querying due to the popularity of search engines like Google. SPARK (Zhou et al., 2007), Q2Semantic (Wang et al., 2008), SemSearch (Lei et al., 2006), and GuNQ (Bhatia et al., 2006) 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 (Russell et al., 2008); (Borsje and Embregts, 2006), the user constructs 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 and 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 a form should be created for each query type. In view-based interfaces (Athanasis et al.,

Alaca Aygul F., Kesim Cicekli N. and Cicekli I..
Natural Language Query Processing in Multimedia Ontologies.
DOI: 10.5220/0004134900660075
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2012), pages 66-75
ISBN: 978-989-8565-30-3
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

2004); (Catarci et al., 2004); (Jarrar and Dikaiakos, 2008), 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 for non-experienced users. The most sophisticated method for querying semantic data is natural language based querying (Erozel et al., 2008); (Kara et al., 2010); (Kaufmann et al., 2007); (Tablan et al., 2008).

In this paper, a natural language interface is presented for querying ontologies which contain multimedia data. The underlying ontology is created by attaching domain ontologies to the core Rhizomik MPEG-7 ontology (García and Celma, 2005). The ontology is used to represent 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 objects are annotated. By using the natural language interface, the user can pose concept, complex concept (objects connected to each other with an "AND" or "OR" connector), spatial, temporal, object trajectory and directional trajectory queries. The contributions of this paper can be summarized as follows:

- The main contribution of this paper is a natural language (English) query interface for concept, spatio-temporal and trajectory queries over the MPEG-7 based domain ontologies.
- The proposed solution is a generic solution in which 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, which provides some level of abstraction.
- The semantic objects can be queried with their attributes.
- Complex concept querying is also provided.

The rest of the paper is organized as follows. Section 2 introduces the system architecture and Section 3 presents the supported query types together with their internal representations. Section 4 discusses the mapping of queries to SPARQL queries. Section 5 presents query processing by discussing the execution of each query type. Section 6 concludes the paper with some remarks.

2 SYSTEM ARCHITECTURE

In this paper, a natural language querying interface is presented for querying multimedia domain ontologies which are attached to the core Rhizomik MPEG-7 Ontology. The system accepts 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 query results are processed to calculate spatial, temporal and trajectory relationships according to the type of the query. The system architecture is presented in Figure 1.

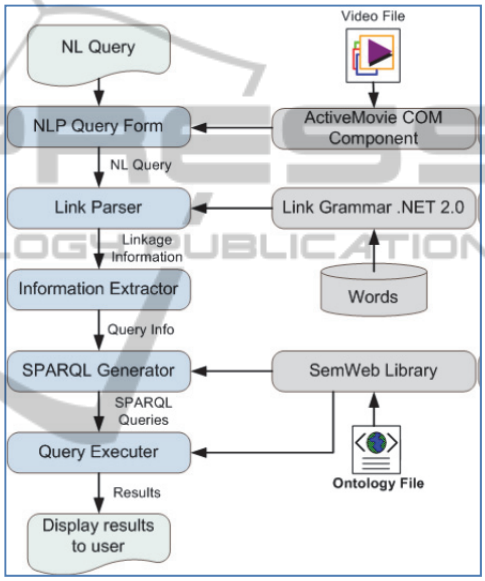


Figure 1: System architecture for querying multimedia domain ontologies.

NLP Query Form is the main user interface of the application. It allows the user to select ontology and a video. The user enters the natural language query by using this form. The results of the executed query are displayed in a tabular format. By double clicking on a result, the user can view the video fragment that corresponds to the selected result.

Link Parser module is used to parse the natural language input in order to produce the linkage information of the input query.

Information Extractor module uses the output of the link parser to construct an internal representation of the query from its linkage information. According to the query type, it extracts objects, attributes, spatial relation, temporal relation, trajectory relation, and time information from the query.

SPARQL Generator module is responsible for creating the SPARQL queries from the internal

representation of the query by using SemWeb library (Tauberer, 2010). 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. While generating the SPARQL queries, it checks if the query information exists in the ontology by using the extracted information.

Query Executer module is used to execute the generated SPARQL queries. It uses SemWeb library (Tauberer, 2010) 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.

3 SUPPORTED QUERY TYPES

In the presented system, the user can query videos which are annotated using domain ontologies attached to the MPEG-7 ontology. The system supports five types of queries: *Concept*, *Complex Concept*, *Spatial*, *Temporal*, and *Trajectory*. 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 an internal representation.

Concept Query: This query type is used in order to retrieve frames containing objects of interest. Moreover, objects can be queried by specifying one or more attributes. In addition, 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. Some example concept queries are given below:

Show all frames where a female is seen.
Return all frames where John Locke is seen.
Return scenes where a male is not seen.

The internal representation of a concept query is as follows:

ConceptQuery:
Concept (ObjectName, AttributeList);
QueryStartTime; QueryEndTime;
IsNegationQuery; VideoDuration

Concept is composed of *ObjectName* and *AttributeList*. *ObjectName* is the name of the object in the query such as “dog”, “car” and “John Locke”. *AttributeList* contains the attributes of the object if stated in the query sentence. The fields *QueryStartTime* and *QueryEndTime* are used to retrieve the concepts in the specified time interval. If *QueryStartTime* and *QueryEndTime* are not shown in the representation, then the query is not restricted to a specific time interval, rather the whole video is considered as the scope of the query. *IsNegationQuery* is a boolean flag, which is set to true if negative meaning exists in the NL input. *VideoDuration* is the total duration of the video. It is used to calculate the time intervals when the query is a negation concept query.

Complex Concept Query: Users can select more than one object that are connected with an “AND” or “OR” connector by using the complex concept query. Thus, a complex concept query can be thought as a collection of concept queries, along with a connector. When handling the “AND” connector in the query, the intersection of the results for each concept query is found. If “OR” connector is given in the NL query, the result is the union of the results for each concept query. The negative meaning is also captured in the complex concept query. Examples of the complex concept query are given below:

Retrieve all frames where Jack and Kate appear.
Show all frames where a male or a female is seen.
Return frames where Jack and Kate are not seen

The internal representation of a complex concept query is as follows:

ComplexConceptQuery:
ListofConcepts; Connector;
QueryStartTime; QueryEndTime
IsNegationQuery; VideoDuration

ListofConcepts contains more than one concept and *Connector* can be “AND” or “OR”.

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 interval. There are four positional relations supported in the system: *Left, Right, Above* and *Below*. An example of a spatial query is as follows:

Return all objects that are on the left of the red car.

The internal representation of a spatial query is:

SpatialQuery:
FirstConcept(ObjectName,AttributeList);
SecondConcept(ObjectName,AttributeList);
SpatialRelation;
QueryStartTime; QueryEndTime

In spatial queries, the spatial relation between two objects is queried. Therefore, there are two *Concepts* and one *SpatialRelation* in the representation. Each concept has an object name and can have any number of attributes. *SpatialRelation* can be *Left*, *Right*, *Above* or *Below*.

Temporal Query: Temporal queries are used to query the relative appearances of objects in a time sequence. There are two temporal relations supported in the system: *Before* and *After*. Allen's interval algebra (Allen, 1983) is used to define the semantics of these two temporal relations. 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. Some examples of temporal queries are as follows:

Show all frames where John appears after the car.
Return frames where a female appears at least 1 minute before a male.

The internal representation of this type of query is:

TemporalQuery:
FirstConcept(ObjectName,AttributeList);
SecondConcept(ObjectName,AttributeList);
TemporalRelation;
QueryStartTime; QueryEndTime;
TFilter(FType,FAmount,Start,End,FInMin)

In temporal queries, the temporal relation between two objects is queried. Therefore, there are two *Concepts* and a *TemporalRelation* that can be *Before* or *After*. *TFilter* is a time filter to restrict the *TemporalRelation* and five types of the time filter are supported: EXACT, LEAST, MOST, INTERVAL and STANDARD. When *FType* is one of EXACT, LEAST and MOST, then *FAmount* is used to define the time restriction amount for the time filter, and *Start* and *End* fields are not used. When the *FType* is INTERVAL, then *Start* and *End* are used to define the time interval without using *FAmount*. When *FType* is STANDARD, *FAmount, Start* and *End* fields are not used. *FInMin* specifies whether the filter is in minutes or in seconds.

Trajectory Query: Trajectory queries are used to query the paths of 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. In directional trajectory, the objects that go to the specified direction are searched. The following are some examples of trajectory queries:

Return the path of the dog.
Return all players who go to the east.

The internal representation of a trajectory query is:

TrajectoryQuery:
TrajectoryType;
Concept(ObjectName,AttributeList);
Direction;
QueryStartTime; QueryEndTime

TrajectoryType defines the type of the trajectory and *Direction* can be one of *East, West, North, South, Northeast, Northwest, Southeast,* and *Southwest*.

4 MAPPING QUERY TO SPARQL QUERY

The backbone of the system is to correctly map a query given in natural language to a SPARQL query. The given query is parsed using LinkParser and the linkage information for the words of the query is returned as the parser output. In order to create the internal representation of the query, predefined information extraction rules are applied to the parser output. In order to get data from the ontology, SPARQL queries are generated from the extracted representation of the query. This section explains how to parse the natural language query and how to extract internal representations from parser outputs. The mapping of the extracted representations of queries to SPARQL queries is also presented.

4.1 Parsing Natural Language Queries and Extracting Query Representations

LinkParser is a light syntactic parser for English, which builds relations between pairs of words in a sentence (Sleator and Temperly, 1993). It constructs the syntactic structure of a given sentence by assigning links to word pairs. LinkParser uses a link grammar which has a dictionary which consists of a set of words and linking requirements for each word. 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. A linkage is the set of links for a valid sentence in the language.

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.

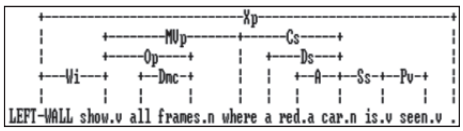
The aim of parsing the natural language query is to extract the information being queried. In order to achieve this, hand-crafted extraction rules are

defined according to the link information. Link types and link orders are used to define a rule. An example rule which is used to find the query type and its object is given below:

i. Search for Cs link.

ii. 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, the type of the query is found first by using rules. One or more rules are defined for each query type. 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 us explain the extraction process with an example. The link parser output of the natural language query "*Show all frames where a red car is seen"* is:



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. The internal representation of this query is.

ConceptQuery:
Concept: *ObjectName* = car
AttributeList = {red}

4.2 Mapping Query Representation 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. After the internal representation is extracted from the given sentence, the internal representation is converted to SPARQL query. The SPARQLGenerator module is responsible for constructing SPARQL queries from internal representations. Each concept in internal representations is retrieved from the ontology. For example, in *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 type of the entity is a 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 type of the entity is an 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.

Mapping Attributes to SPARQL: The attributes are mapped to SPARQL queries by using one of the following three approaches.

First Approach: If the attribute name is specified in the query text as in the sentence *"Find the intervals where a car with red color is seen"*, the first approach is used. The attribute name "color" and its attribute value "red" are mentioned 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 following 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, additional SPARQL text is generated for the attribute. To illustrate, in the following code segment, the "car" class is a subclass of the "vehicle" class and the domain of the "color" attribute is the "vehicle" class. Thus, the SPARQL text below 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. For instance, assume that the given query is "*Show all frames where a red car appears*". Here, only the attribute value "red" is given in the sentence. In other words, we do not know the attribute name whose value is specified to be "red". Therefore, the ontology must be searched to see if there is an attribute which includes the value "red" in its data range values. The data range values of an attribute are specified in the ontology under the <owl:DataRange> property. If such an attribute is found, then the domain of the attribute is extracted. If the domain matches with the class "car" (or one of its super-classes), a SPARQL text is generated for the attribute.

Third Approach: The third approach is used when the objects are described as triples in the ontology. A triple is composed of a subject, a predicate, and an object. An example triple is given below:

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

If neither the attribute name nor data range values are specified in the ontology, but the objects are described with triples, then the triples are searched to construct the SPARQL query. We explain the third approach with the same example query "*Find all intervals where a red car is seen*". Again, the attribute name is not given in the query but an attribute value is extracted from the sentence. This attribute value is searched in the triples in the ontology. 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 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 super-classes of the "car" class, then the predicate value is used as the attribute name and the SPARQL text is constructed for the attribute.

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. 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 QUERY PROCESSING

In the proposed system, a user friendly GUI is designed so that the user can enter NL input and can 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.

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.

5.1 Execution of Concept Query

The generated SPARQL query for a concept query is executed against the ontology by using SemWeb library. If no negative meaning is found in the query, the results of the query execution are directly displayed to the user. For instance, in Table 1, the execution results of the query sentence "*Show all frames where a male is seen*" are shown. The table illustrates the males that appear in the video and the time intervals in which they appear.

Table 1: The execution results of the concept query "*Show all frames where a male is seen*".

Jack	6.5	8.6
Jack	8.6	16.7
Hurley	14.0	39.5
Jin	39.5	48.7
Jin	48.2	57.3
Boone	72.5	74.7
Sawyer	78.0	86.4

Table 2: The inverse of the execution results of the concept query in Table 1.

Male	0.0	6.5
Male	57.3	72.5
Male	74.7	78.0
Male	86.4	128.6

If negative meaning is captured in the NL input, 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 1. However, since negative meaning exists in the input, the inverse of Table 1 is calculated as illustrated in Table 2.

5.2 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. For complex concept query type, 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. 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 a time interval overlaps with another time interval, then the overlapping interval is found and added to the intersection result. The overlapping interval of two intervals is the common interval region of those intervals.

If the connector is an "OR" connector, the result is the union of the execution results of concept queries. 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 a time interval overlaps with another time interval, then the union of two intervals is found and added to the union result.

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

5.3 Execution of Spatial Query

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 the result couple that must have an overlapping time interval. If this condition holds, then the spatial relation between two objects is calculated as explained later in this section. The result set contains the objects satisfying the spatial relation, the spatial relation itself, start and end times of overlapping intervals satisfying the spatial relation.

In order to find the spatial relation between two objects, the center points of the rectangles covering the objects are used. For instance, let A and B are two objects, and they are represented with their center points A($x1$,$y1$) and B($x2$,$y2$).

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

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

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

The calculations of spatial relationships are

simplified for the purposes of this paper. Actually we need to deal with spatial relationships that change over time.

5.4 Execution of Temporal Query

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 first object's start and end times, and the second object's start and end times.

Temporal relations between two objects are calculated according to the appearance order of objects. During calculation, Allen's interval algebra (Allen, 1983) 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*.

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. 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.

5.5 Execution of Trajectory Query

A SPARQL query is generated for the queried concept and it is executed against the ontology. 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 the user.

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:

i. The successive time intervals should be adjacent. If Ii and Ij be two successive time intervals, these intervals are adjacent if the end time of Ij is one more than the start time of Ii.

ii. The successive object regions should be neighbor. Let R1 and R2 are two rectangles of the regions in time intervals. R2 is a neighbor of R1, if R1 intersects with R2 or R1 touches R2.

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.

5.6 Evaluation

While developing the natural language query 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. The supported query types are designed to give the user flexibility of stating queries by different sentence structures.

To handle concept queries, the extraction rules are defined to cover the wh-questions, subordinate clauses and relative clauses. The user can query an ontological class, an instance of a class, or a class with an attribute. The rules that are designed to capture the negative meanings in queries search links which connects the word "not" to the preceding auxiliaries and modals. Since we only handle the sentences having the word "not" in its word sequence to capture meanings, in order to broaden the scope, new rules should 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 related with the word "no".

In order to handle complex concept query, the rules that are defined for handling concept query are used. In addition to these rules, the rules are used to find the type of the query and links related with connectors. In order to handle spatial query type, a set of rules are defined and the number of rules is affected by the number of spatial relations. We also defined rules to handle temporal queries so that the objects and the temporal relation can be found. There are also a set of rules to extract the time filter information. The rules are also designed to extract trajectory query information. As a result, we have approximately 50 rules in the system to understand

the user queries. 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 reason why we chose the link parser to parse the user queries is that it can tolerate the errors in the NL input to some extent. The link parser does not require full understanding of 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. Using the NL interface, users can query any videos such as TV series, documentary videos, or personal videos as long as they are annotated beforehand. Although the vocabulary is not limited, 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 the video annotation is done automatically, the proposed framework can be used effectively. This study is conducted as a part of a research project. 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.

6 CONCLUSIONS

In this paper 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, spatial, temporal, object trajectory and directional trajectory queries. Furthermore, the system handles the negative meaning in the user input. When the user enters an NL input, it is parsed with the link parser. 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.

In order 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 have focused 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 library which is 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 study are limited. If more spatial relations such as near, far, inside, touch and disjoint and temporal relations such as during, overlap, meets, and between are added to the system, the usability of NL query interface will be increased. In this paper, 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 or extension functions. 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.

ACKNOWLEDGEMENTS

This work is partially supported by The Scientific and Technical Council of Turkey Grant ‘‘TUBITAK EEEAG-107E234”.

REFERENCES

Allen, J. F., (1983). Maintaining knowledge about

temporal intervals. *Communications of the ACM*. 21(11): 832-843.

Athanasis, N., V. Christophides, and D., Kotzinos, (2004). Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL), *The Semantic Web – ISWC 2004*, Lecture Notes in Computer Science, Volume 3298/2004, pp. 486-501.

Bhatia, N., Gaharwar, P., Patnaik, P., and Sanyal, S., (2006). GuNQ–A Semantic web engine with a keyword based query approach. In *Proceedings of AAAI, Fall Symposium on Semantic Web for Collaborative Knowledge Acquisition*, Arlington, VA, USA.

Borsje J., and Embregts, H., (2006). 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.

Broeskstra, J., and Kampman, A., (2003). SeRQL: A second generation RDF query language. In *Proceedings of SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pp. 13–14.

Catarci, T., Dongilli, P., Di Mascio, T., Franconi, E., Santucci, G., and Tessaris, S., (2004) An ontology based visual tool for query formulation support. In *Proceedings of 16th European Conference on Artificial Intelligence (ECAI)*, Valencia, Spain, pp. 308.

Donderler, M. E., Saykol, E., Ulusoy, O., and Gudukbay, U., (2003). BilVideo: A video database management system. *IEEE MultiMedia*, 10(1), 66–70.

Erozel, G., Cicekli, N. K., and Cicekli, I., (2008). Natural language querying for video databases. *Information Sciences*, 178(12):2534–2552.

García, R. and Celma, O., (2005). Semantic Integration and Retrieval of Multimedia Metadata. In *Proceedings of the Knowledge Markup and Semantic Annotation Workshop, Semannot'05*.

Jarrar, M., and Dikaiakos, M. D., (2008). MashQL: a query-by-diagram topping SPARQL. In *Proceedings of ONISW*, Napa Valley, California, USA.

Kara, S., Sabuncu, O., Akpınar, S., Alan, O., Cicekli, N. K., and Alpaslan, F. N., (2010). An ontology-based retrieval system using semantic indexing. In *Proceedings of the 1st Intl. Workshop on Data Engineering meets the Semantic Web, in conjunction with ICDE 2010*, Long Beach CA, USA.

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

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

Lei, Y., Uren, V. and Motta, E., (2006). Semsearch: A search engine for the semantic web. In *Proceedings of the 5th International Conference on Knowledge Engineering and Managing Knowledge in a World of Networks*, Lecture Notes in Computer Science, 4248:238.

Prud'hommeaux, E., and Seaborne, A., (2008). SPARQL query language for RDF. *W3C recommendation*, http://www.w3.org/TR/rdf-sparql-query/.

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

Russell, A., Smart, P. R., Braines, D., and Shadbolt, N. R., (2008). NITELIGHT: A graphical tool for semantic query construction. In *Proceedings of Semantic Web User Interaction Workshop (SWUI 2008)*, Florence, Italy, 2008.

Sleator, D., and Temperly, D., (1993). Parsing English with a Link Grammar. In *Proceedings of the Third International Workshop on Parsing Technologies*.

Tablan, V., Damljanovic, D., and Bontcheva, K., (2008). A natural language query interface to structured information. In *Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*.

Tauberer, J., (2010). *Semantic Web/RDF Library for C#/.NET*. Retrieved February 18, 2011, from http://razor.occams.info/code/semweb/

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

Wang, H., Zhang, K., Liu, Q., Tran, T., and Yu, Y., (2008). Q2semantic: A lightweight keyword interface to semantic search. In *Proceedings of the 5th International Semantic Web Conference (ESWC'08)*, Lecture Notes in Computer Science, 5021:584.

Zhou, Q., Wang, C., Xiong, M., Wang, H., and Yu, Y., (2007). Spark, Adapting keyword query to semantic search. In *Proceedings of the 6th International and 2nd Asian Semantic Web Conference (ISWC2007 + ASWC2007)*, Lecture Notes in Computer Science, 4825:694.