Management of Large Spatial Ontology Bases

Evangelos Dellis and Georgios Paliouras

Institute of Informatics & Telecommunications,
National Centre for Scientific Research "Demokritos",
15310 Ag. Paraskevi, Athens, Greece
{dellis, paliourg}@iit.demokritos.gr

Abstract. In this paper we propose a method for efficient management of large spatial ontologies. Current spatial ontologies are usually represented using an ontology language, such as OWL and stored as OWL files. However, we have observed some shortcomings using this approach especially in the efficiency of spatial query processing. This fact motivated the development of a hybrid approach that uses an R-tree as a spatial index structure. In this way we are able to support efficient query processing over large spatial ontologies, maintaining the benefits of ontological reasoning. We present a case study for emergency teams during Search and Rescue (SaR) operations showing how an Ontology Data Service (SHARE-ODS) can benefit from a spatial index. Performance evaluation shows the superiority of our proposed technique compared to the original approach. To the best of our knowledge, this is the first attempt to address the problem of efficient management of large spatial ontology bases.

Keywords: Spatial Databases, Ontologies, Knowledge Bases.

1 Introduction

The SHARE project[1] develops a Push-To-Share (PTS) advanced mobile service that provides communication support for emergency teams during Search and Rescue (SaR) operations. SaR operations are conducted by fire-brigade, rescue and medical units, operating under a complex unified command-and-communications structure. The SHARE Ontology Data Service (SHARE-ODS) [5, 6], which supports the PTS service, combines multimedia semantic modeling and spatio-temporal modeling in a unified ontology. A model for the semantic indexing of multimedia objects in the context of SaR processes and activities is also proposed in [5, 6]. This model unifies the various aspects of a SaR operation, while allowing the semantic cross-checking of possibly-unreliable information automatically extracted from multimedia objects.

Ontologies represent concepts, relations among them and instances. Commonly, an ontology consists of a relatively small conceptual part (TBox in Description Logic terminology) and a much larger instance base (ABox in Description Logic terminology), or a dense combination of concepts, relations and instances [9]. Like most ontologies, the SHARE ontology (TBox and ABox) is stored as an external xml

[1] http://www.ist-share.org/

file (OWL-file) to disk. This approach, however, is inefficient for very large spatial ontologies that require loading the entire OWL file from disk into main memory.

In the context of non-spatial ontologies there have been approaches [9, 7] to distinguish the ontological model from the actual instances, which make use of a relational database system (RDBMS). In this way individual instances are stored as tuples in the database, and as a consequence only the required instances are read from the database (disk) and processed in memory. Despite the obvious gains from this approach, in SHARE ODS, we have observed significance performance problems in the spatial sub-ontology. This is due to the lack of efficient spatial query processing.

The most important reason is that traditional (relational) storage modelling is inefficient when dealing with space and for that reason during the last few decades scientists have been trying to model the spatial dimension of the real world in order to produce information systems which are aware of space. This quality becomes a necessity when it comes to Geographic Information Systems and Spatio(-temporal) Information Systems in general.

Early attempts in this area focused on creating spatial database models and query languages, as well as on devising logic representations and algebras for reasoning about space. The emergence of the Semantic Web vision and related technologies, such as ontologies and semantic web services, has put Spatial Knowledge Representation under a whole new perspective. Ontologies in particular have introduced a new means for describing, representing and sharing knowledge, by combining syntactic and semantic aspects under the formal umbrella of logics.

In this paper, we focus on supporting the querying and management of large spatial ontologies. We propose to store the instances of the spatial ontology using a spatial index structure (e.g. R-tree). In this way we are able to support efficient query processing over large spatial ontologies. In addition, we are able to combine spatial query processing with reasoning mechanisms to speed up the process of inferring new knowledge. By separating the ontological model (TBox) from the actual instances (ABox) we are moving towards efficient management of very large spatial knowledge bases. Our contributions can be summarized as follows:

- We address the problem of management of large spatial ontologies and we propose a hybrid approach that uses a spatial index structure (R-tree) as the underlying storage model.
- A case-study using the SHARE system is presented showing how a spatial ontology can benefit from a spatial index structure. In addition, we present a series of experiments showing the performance gains.

The rest of the paper is organized as follows. Section 2 discusses related work on spatial ontologies and management of spatial information. Section 3 presents the SHARE Ontology Data Service and motivates the need for spatial indexing. In Section 4 we show how we can index large spatial ontologies supporting efficient spatial query processing and present a case study using the SHARE system. The experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

2 Background

The basis for our work is the SHARE Ontology Data Service (SHARE-ODS) [5, 6], which supports the Push-to–Share service of the SHARE project. SHARE-ODS combines multimedia semantic modeling and spatio-temporal modeling in a unified ontology. Besides SHARE-ODS, various general-purpose space, time, and spatio-temporal ontologies have been proposed, which can be directly plugged in an existing ontology to allow it to refer to spatial concepts. All these attempts are, generally speaking, part of a more complete concept and interface specification, which aims to enhance interoperability between databases and applications that make geographic and temporal references. In the remaining of this section we provide some background information related to ontology management and spatial query processing.

2.1 Ontology Management

In the context of non-spatial ontologies, an early attempt in [9] describes an environment for supporting very large ontologies. The system was created to manage ontologies of essentially unlimited size. The architecture of the system uses a relational database system as the storage model and describes different approaches to ontology management. In [7], the authors discuss DL reasoning over large ontologies (ABoxes) and present the KAON2. The system can decide knowledge base and concept satisfiability, compute the subsumption hierarchy, and answer conjunctive queries in which all variables are distinguished. The architecture of KAON2 is presented where the ontology API provides ontology manipulation services, such as adding and retrieving ontology axioms. The API fully supports OWL and the Semantic Web Rule Language (SWRL) at the syntactic level. ABox assertions are stored in a relational database (RDBMS). By mapping ontology entities to database tables, KAON2 is able to query the database on the fly during reasoning.

In the spatial domain, the majority of the work so far was related to the evolution of spatial databases, whose primary objective was to store spatial information and evolving geometries. The area of spatial databases was firstly very much attached to Geographic Information Systems (GIS) and their research advances were evolving in parallel. Nevertheless, as information technology has evolved, other application areas have emerged such as Intelligent Transport Systems (ITS) and Field Analysis Systems and they have presented the spatiotemporal research community with innovative challenges. The International Organisation for Standardisation (ISO) has published a set of standards, aiming to enhance interoperability in expressing geo-semantics. They specify the conceptual and logical data model and the exchange format for geographic databases. The Open Geospatial Consortium[2] developed OpenGIS, a specification for geographic data representation and interchange that renders complex spatial information and services accessible by all kinds of applications.

[2] http://www.opengeospatial.org/

2.2 Spatial Query Processing

The problem of spatial indexing has motivated several research efforts. In this regard, the R-tree [3] is one of the most popular spatial index structures. Each spatial data object in the R-tree is represented by a Minimum Bounding Rectangle (MBR). Leaf nodes in the R-tree contain entries of the form *(oid, rect)* where *oid* is a pointer to the object in the database and *rect* is the MBR of the object. Non-leaf nodes contain entries of the form *(ptr, rect)* where *ptr* is a pointer to a child node in the R-tree and *rect* is the MBR that covers *all* the MBRs in the child node.

For the following examples we use the R-tree of Figure 1, which indexes a set of points {*a*,*b*,…,*k*}, assuming a capacity of three entries per node. Points that are close in space (e.g., *a* and *b*) are clustered in the same leaf node (*N3*), represented as a minimum bounding rectangle (MBR). Nodes are then recursively grouped together following the same principle until the top level, which consists of a single root. See [3] for more details on the R-tree construction.

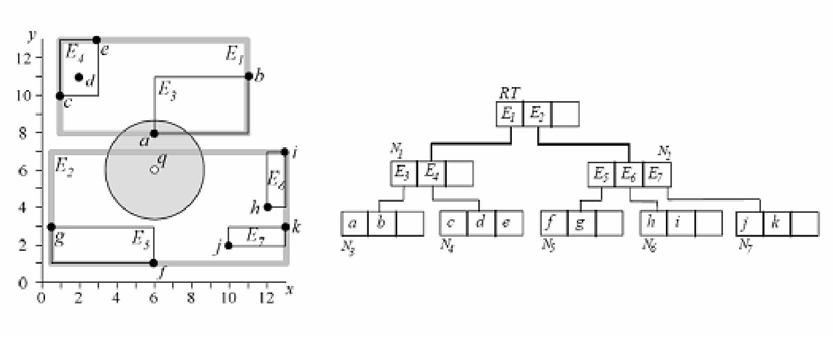

Figure 1: An R-tree example

R-trees (like most spatial access methods) were motivated by the need to efficiently process *range queries*, where the range usually corresponds to a rectangular window or a circular area around a query point. The R-tree answers the range query *q* (shaded area) in Figure 1 as follows. The root is first retrieved and the entries (e.g., *E1*, *E2*) that intersect the range are recursively searched because they may contain qualifying points. Nonintersecting entries (e.g., *E4*) are skipped. Notice that for non-point data (e.g., lines, polygons), the R-tree provides just a *filter* step to prune non-qualifying objects. The output of this phase has to pass through a *refinement* step that examines the object representation to determine the actual result. The concept of filter and refinement steps applies to all spatial queries on non-point objects.

Besides range queries the *nearest neighbor* (NN) *query* retrieves the *k* ($k \geq 1$) data point(s) closest to a query point *q*. The R-tree NN algorithm proposed in [4] keeps a *heap* with the entries of the nodes visited so far. Initially, the heap contains the entries of the root sorted according to their minimum distance (*mindist*) from *q*. The entry with the minimum *mindist* in the heap (*E2* in Figure 1) is expanded, i.e., it is removed from the heap and its children (*E5*, *E6*, *E7*) are added together with their *mindist*. The next entry visited is *E1* (its *mindist* is currently the minimum in the heap), followed by *E3*, where the actual 1NN result (*a*) is found. The algorithm terminates, because the *mindist* of all entries in the heap is greater than the distance of *a*. The algorithm can be easily extended for the retrieval of *k* nearest neighbors (*k*NN). Furthermore, it is optimal (it visits only the nodes necessary for obtaining the nearest neighbors) and *incremental*, i.e., it reports neighbors in ascending order of their distance to the query point, and can be applied when the number of nearest neighbors to be retrieved is unknown in advance.

3 The SHARE Ontology Data Service

The main objective of the project SHARE is to develop a new type of advanced mobile service, called *Push-To-Share*, to support "mobile content sharing" by the participants of field operational teams, such as fire rescue forces. Push-To-Share is an innovative extension of the Push-To-Talk mobile technology and provides a new concept for simple ways of complex communication, combining an easy-to-use interface with a comfortable delivery of multimedia content. SHARE incorporates innovations in the area of multimodal interaction, robust speech interfaces, interactive digital maps, in conjunction with location-based services and intelligent information processing of multimedia data.

3.1 Ontology Data Service

This Section discusses the Ontology Data Service of the SHARE system. More particularly, in subsection 3.1.1 we give the architectural framework of SHARE-ODS, in subsection 3.1.2 the spatial sub-ontology is presented and in subsection 3.1.3 we discuss several operations of SHARE-ODS such as population, querying and reasoning.

3.1.1 Architecture

The Ontology Data Service (ODS) is responsible for the semantic indexing and retrieval of the data in the Knowledge Base. More specifically, the various data (geographical, temporal, multimedia, operational) required for the operation of the overall architecture, are represented as ontology instances that are interconnected with semantic relations, providing an integrated view of the information needed to support the operation. The Ontology Data Service allows the querying of the Knowledge Base, so that the other services are able to retrieve not only the explicitly stated knowledge but also the implicit knowledge inferred by the reasoning engine.

The ODS operates as the back-end of the Data Server, supporting the functions of front-end data services like, for example, multimedia data retrieval and geographical annotation (Figure 2). Its main functionality is to allow the other Services to access populate and query the Knowledge Base. The functions of the ODS are available through a Web Service interface, so that they can be utilized by different software modules, which are implemented in different programming languages and situated in different network locations. In addition, the Ontology Data Service includes internal operations (consistency check, classification), which are responsible for detecting and correcting semantic discrepancies in the Knowledge Base. From the above discussion it is clear that efficient management of large ontologies is of major importance for the SHARE system.

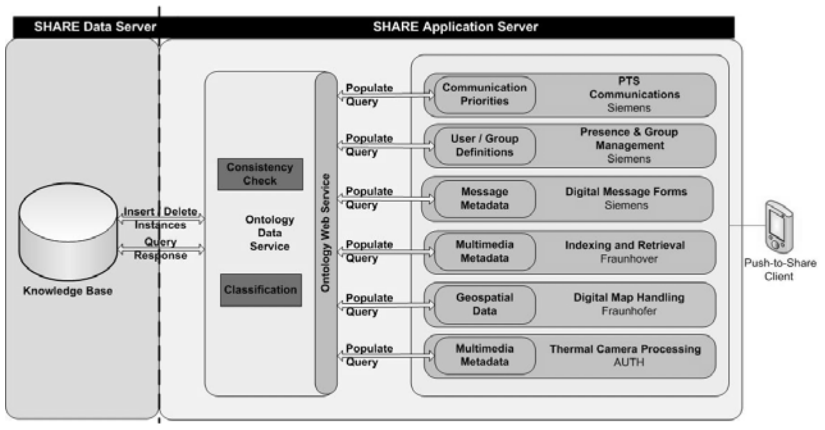

Figure 2: General architecture of the Ontology Data Service

3.1.2 The Spatial sub-ontology

The space conceptual model was designed taking into account the relationships that the map format provides, and its aim is to effectively model all the concepts related with geographical objects. The space conceptual model is represented by the spatial sub-ontology. The sub-ontology is broken down into two component sub-ontologies. The first one comprises abstract geographical concepts, their geo-reference, and the relationships between them. The second one includes actual features (buildings, streets, etc). Both SAR instances and geographical features represent their georeferences as relationships to the abstract geographical instances. Geographical meta-data describe the spatial properties of each entity, as well as the spatial relations among different entities. This information is crucial for constructing a semantic spatial context. The client application implementing the geographical data visualization is responsible for populating and updating the Knowledge Base with the appropriate spatial entities by utilizing specially predefined functions of the Web Service interface.

3.1.3 Operations

The Ontology Data Service allows the *population* of the ontology with instances, which are related to SaR operations, space or multimedia objects. The population process can be divided into two phases: the population that occurs during the initialization of the system and the population that occurs during the progress of a SaR operation. During initialization, the ontology is populated with static data (e.g. the geographical objects of the digital maps, the roster of the fire department) and setup data (e.g. the communication groups, the available communication devices). On the other hand, during operation time the ontology is populated with dynamic data (e.g. event logging, formation of new sections, moving objects).

Furthermore, the Ontology Data Service allows the *querying* of the stored knowledge. Queries are expressed in RDQL, in order to take advantage of the graph-like structure of the ontology. More specifically, the queries can impose constraints on the property values, as well as on the relations of instances. This way a query graph is created which has to be matched with the actual ontology graph.

Besides population and querying, the Ontology Data Service supports two *reasoning* services: Classification and Consistency Checking. More specifically, classification is responsible for classifying an instance to the appropriate class taking into account the knowledge incorporated in the ontology. Consistency checking is responsible for detecting inconsistencies in the ontology knowledge. Inconsistencies can occur when the ontology is populated with inaccurate data or when an unacceptable situation (fact) has occurred.

4 Incorporation of R-trees into Spatial Ontologies

In this Section we show how to incorporate an R-tree inside SHARE-ODS. More specifically section 4.1 motivates the need for spatial query processing, section 4.2 presents the process of the ontology population and finally in section 4.3 we discuss query processing using the SHARE Ontology Data Service. Moreover, in the remaining of this section we show how to distinguish the ontological model from the actual instances.

4.1 Why We Need a Spatial Index Structure?

In this Section we define two cases, which require efficient support of spatial query processing inside SHARE-ODS. The first case concerns spatial query processing in a dynamically changing environment, i.e. during a SaR operation, whereas the second case deals with static spatial entities, such as entities populated in the initialization stage.

Spatial Query Processing in Dynamic Environments: Consider as an example that we store in the ontology the sections and sub-sections defined by the Officer in Charge (see [5, 6] for more details). Each section (sub-section) is assigned to a B-Level Officer and is represented as a spatial region (area).

Assuming a querying operation that request the B-Level Officers, which are responsible for sections:

- *Find all B-Level Officers and corresponding sections in a particular area.* (Range Query)

- *Find the B-Level Officer who is closest to a (new) accident (e.g. fire).* (Nearest Neighbor Query)

The first query retrieves all B-Level Officers inside a particular range. The second query retrieves the nearest B-Level Officer to a specific point.

Spatial Query Processing for Static Spatial Entities: In this case we are interested in buildings appropriate for landing a helicopter:

- *Find which buildings in a particular area (range) are appropriate for landing a helicopter.*
- *Find the nearest building, according to my current location, which is appropriate for landing a helicopter.*

Unfortunately, the SHARE system is unable to answer effectively these types of spatial queries, resulting in a high processing time. These queries can be either made directly from an ODS client or they can be part of a longer chain of reasoning steps, involving various concepts and relations in the SHARE ontology.

4.2 Population

In this sub-section we propose to support SHARE-ODS using a disk-resident R-tree to store spatially related instances and at the same time we propose the use of the OWL file for non-spatial instances. In this way, individual non-spatial instances are stored in the OWL file, while spatial instances are stored in an R-tree and as a consequence only the required instances are read from the disk and processed in memory. Note that for large non-spatial sub-ontologies a back-end RDBMS may be used to store non-spatial instances as tuples in the database. However, management of non-spatial ontologies is beyond the scope of this paper.

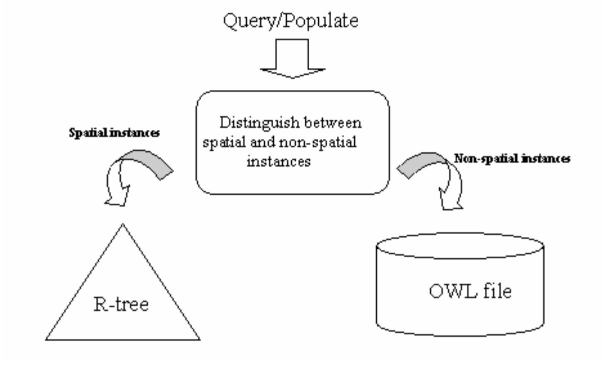

Figure 3: Storing the instances

In Figure 3, the process of distinguishing between spatial and non-spatial instances is shown. Note that, this process happens inside the Ontology Data Service. The ontological model is kept in main memory (due to its relatively small size) as it is requested very often.

4.3 Spatial Query Processing using SHARE-ODS

Our framework can execute queries that combine both the ontology and the R-tree in longer chains of reasoning steps. In such cases, the R-tree queries are used to reduce the complexity of the RDQL statements. We show in more detail this process using our hybrid approach. For this reason, we use the two query types of subsection 4.2.

Range Query: Consider as an example that we are interested in buildings appropriate for landing a helicopter inside a specified region (consider section 3.2). The R-tree is used as a filter step to prune non-qualifying objects, i.e. buildings outside the specified region. The output of this phase has to pass through the non-spatial ontology to determine the objects with the appropriate roof type. Finally, the actual result is presented to the user (i.e. highlight the buildings on the map).

Nearest Neighbor Query: A nearest neighbor query is executed using our framework in the same way as range queries. More specifically, we execute an incremental k nearest neighbor query (with unknown k) to obtain a ranking of the nearest buildings according to the user's current location. This means that the first nearest neighbor is retrieved and checked in the ontology (non-spatial part) if this building has an appropriate roof type. If this is not the case, the second nearest neighbor is retrieved and we continue until we have found a building with appropriate roof type.

Note that, for the above queries the R-tree is used to speed-up the query processing. Therefore we are able to retrieve only a small fraction of the actual dataset (ABox) and organize the spatial entities in a more appropriate way.

5 Experimental Evaluation

For the purpose of the experiments, we used SHARE-ODS [5, 6] for generating the dataset. As a scenario for the experiments we consider the buildings described in the spatial sub-ontology. In our experiments, the dataset contains buildings with varied cardinalities, ranging from 100 points to 10,000 points. In the first set of experiments we study the scalability of the original SHARE-ODS and show that this approach has serious performance shortcomings. In the second set of experiments, we examine the query performance with the dataset cardinality for spatial range queries and nearest neighbor queries (pre-computed and dynamic). More specifically, we examine the performance degradation in the case of increasing dataset cardinality and we report the running time as a function of the number of instances. In our third experiment we examine the more complex case of combining spatial query processing with the rest ontology. We study the influence of the R-tree on the performance and examine the scalability with respect to the query.

All these experiments use an R-tree indexing 2-dimensional points corresponding to static landmarks (e.g. buildings), which is built during the initialization process. In the last experiment we examine the performance of the R-tree in case of dynamically changing spatial information corresponding to the operational spatial sub-ontology. In this case regions are inserted in the R-tree that corresponds to the areas that are assigned to a B-Level Officer during an operation. For the R-tree implementation we use the XXL library [1].

5.1 Shortcomings of the original ontology

In this subsection we are particularly interested in the impact of the current SHARE-ODS implementation. We examine the performance by varying the number of instances (in our case 2-dimensional points) from 100 to 1,000. Note that, a building on a 2-dimensional map is covering a particular area. This area is represented in SHARE-ODS as a closed line. On the other hand, a very common approach used by almost all maps is to represent a building as a 2-dimensional point. To support efficient query processing, a building in SHARE-ODS is additional represented as a point characterized by the latitude and the longitude (data properties). In the original ontology to support nearest neighbor query each building has eight object properties corresponding to the eight closest buildings in each direction (e.g. south, west, etc.), which in turn increases the size of the OWL file. We report the CPU time (in msec) required for the ODS population.

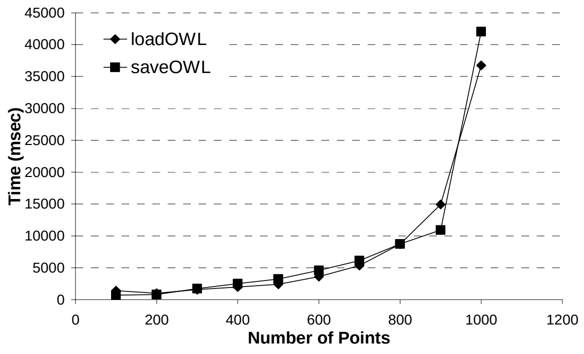

Figure 4: Performance shortcomings

Figure 4 shows the CPU time needed to load the dataset (OWL file) from the disc. In addition we report the time for saving the OWL file to disc. For both cases we observed a high increase in the CPU time for large datasets (above 900 instances).

5.2 Query Processing for Spatial Ontologies

In the second set of experiments we study the performance of two spatial query types, namely range queries and nearest neighbor queries. Additionally, we distinguish between the pre-computed nearest neighbor and the dynamic case. The former, is used for comparison using the R-tree.

In our running example, the points stored in the R-tree correspond to the buildings represented in the spatial sub-ontology. Actually, the latitude and the longitude that are data properties of the building are stored in the R-tree. In the first experiment, we vary the number of buildings and we are interested to find the buildings that lay in a particular area.

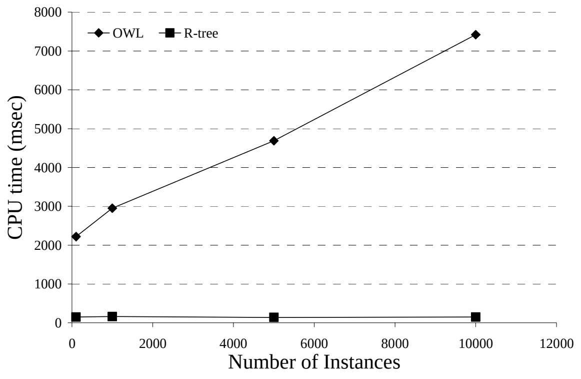

Figure 5: Spatial Range Query

In Figure 5, we report the CPU time needed for answering a range query with constant cardinality for different number of instances. We vary the number of instances between 100 and 10,000. For the OWL file we observed a high increase in the CPU time, while for the R-tree the CPU time remains almost constant.

In the following experiment we are interested to find the nearest building based on a particular direction. As already mentioned, in the original approach the nearest neighbors are pre-calculated and stored in the OWL file as object properties of a building. In contrast to the original case, the R-tree supports such nearest neighbor queries dynamically and the nearest neighbor is calculated real time with respect to the preferred direction.

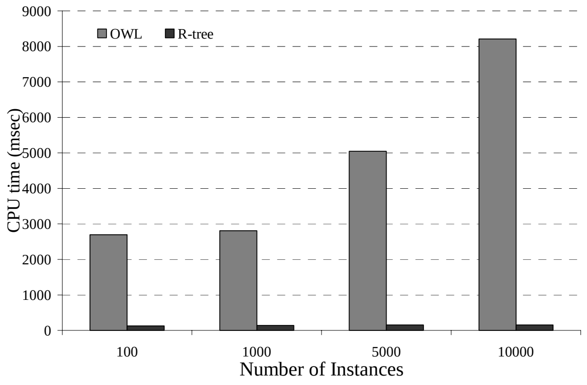

Figure 6: Nearest Neighbor (pre-computed in OWL)

Figure 6 shows the CPU time needed to answer a nearest neighbor query with respect to the number of points inserted in the ontology. Again, for the OWL file we observed a high increase in the CPU time, and more specifically when the dataset cardinality exceeds a certain threshold. For the R-tree the CPU time remains almost constant.

In our final experiment for this subsection, we examine the dynamic case of nearest neighbor query. We are interested to find the nearest building to a random point on the map. This means that the query point is not necessary a building. The original approach does not support efficiently this query since the nearest neighbor of a random point cannot be pre-calculated and stored in the OWL file.

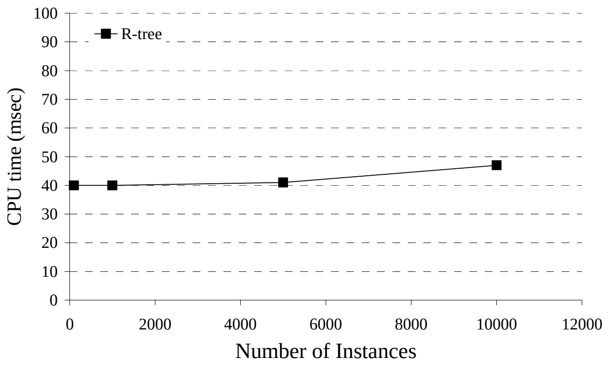

Figure 7: Dynamic Nearest Neighbor Query

Figure 7 depicts the CPU time needed for retrieving the nearest neighbor from an arbitrary query point. As already mentioned, this query cannot be pre-computed and cannot be computed efficiently when using the OWL file. The number of building is varied between 100 and 10,000 instances and we measure the CPU time. We observe that the required CPU time increases very slowly with increasing number of instaces.

5.3 Combining Spatial Query Processing with the rest of the Ontology

In this subsection we investigate the performance of our approach under the assumption that the query is answered using both the OWL file and the R-tree. Such an example is the query „Find which buildings in a particular area (range) are appropriate for landing a helicopter". The location of the building is a spatial attribute and kept in the R-tree. The data property of a building that indicates if a helicopter is capable to land on a building is a non-spatial attribute and is kept in the OWL file.

In this example the R-tree is used as a filter to simplify the RDQL statement. In a first step, all buildings that lie in the particular area are retrieved. The building identifier is also stored in the R-tree as a label of each point. In a second step, the buildings are retrieved from the OWL file according to the identifier and buildings that are not suitable for a helicopter are discarded from the result set.

In the following experiment we compare the combined approach in contrast to the performance of the RDQL statement of the original approach. Figure 8 illustrates the effect of the number of instances on a query that can be answered by the OWL file and additionally by the combination of the OWL file and the R-tree. As expected, using the combined approach is superior to the original approach, which uses only the OWL file. Especially when the number of instances increases, the gain of our approach increases rapidly. This is due to the fact that the RDQL query is simplified through a filtering step using the R-tree, which in turn decreases the time, needed for the query processing. Moreover, the combined approach avoids the retrieving of points based on the data properties of an object, i.e. the latitude of the point corresponding to the building, but accesses the building directly by the building's identifier. The identifier is stored in the R-tree and therefore is able to reduce significantly the CPU time. In order to study the effect of scalability for our point dataset, we vary the dataset cardinality between 100 and 5,000 instances.

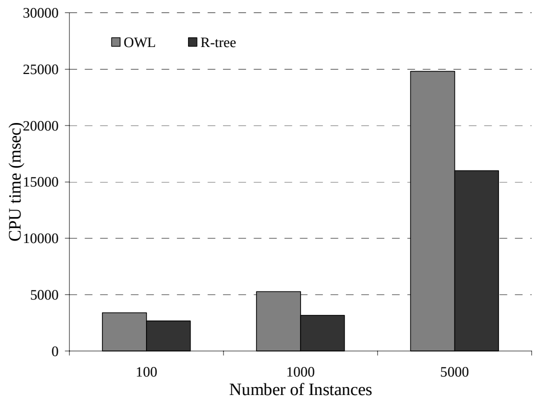

Figure 8: Combined Query

5.4 Querying Dynamically-changing Spatial Information

In this experiment we simulate a rescue operation by inserting and deleting region that are assigned to a B-Level Officer (consider section 3.2). The B-Level Officer's identifier is stored in the R-tree and therefore is able to reduce significantly the CPU time.

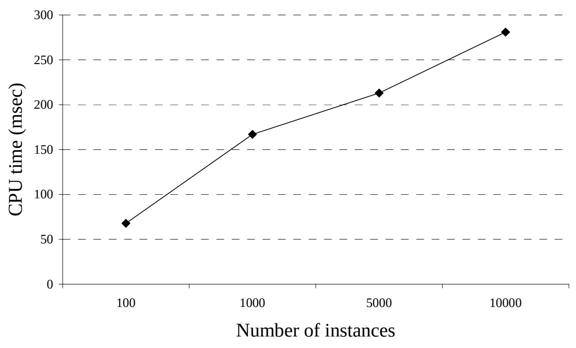

Figure 9: Dynamic R-tree

Figure 9 depicts the CPU time needed for retrieving the B-Level Officer of the nearest region to an arbitrary query point. The total number of inserted region instances is varied between 100 and 10,000. To simulate the process of a Search and Rescue operation we randomly delete some regions and reinsert other regions. The general performance of an R-tree is influenced by deletions and we encounter this fact in our experiments. We observe that the required CPU time increases very slowly when the number of instances increases rapidly. Despite the deletions required for dynamically changing spatial information the R-tree performs well in terms of CPU time for answering spatial queries.

6 Conclusion

We have proposed a method for efficient management of large spatial ontologies, which combines a typical OWL ontology with an R-tree for indexing spatial entities. The performance evaluation shows the superiority of our proposed technique compared to the original approach, using only the OWL file. Using the proposed approach, we are able to support efficient query processing over large spatial ontologies and integrate it in a larger chain of reasoning steps. In addition we present a case study for emergency teams during Search and Rescue (SaR) operations showing how the SHARE Ontology Data Service can benefit from a spatial index, which is integrated inside the SHARE system.

Future work may include the investigation of more complex spatial query operators as well as the integration of spatio-temporal indexing into ontologies, which represent both concepts of space and time.

Acknowledgments

The authors wish to acknowledge the support of the EC-funded project SHARE (contract no. FP6-IST-004218) to the work reported in this paper.

References

1. Bercken, J., Blohsfeld, B., Dittrich, J.-P., Krämer, J., Schäfer, T., Schneider, M., Seeger, B.: *XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries*. In Proceedings of VLDB 2001, September 11-14, Roma, Italy, 2001.

2. Fonseca, F. and Egenhofer, M.: *Ontology-Driven Geographic Information Systems.* In Proceedings of ACM GIS, Kansas City, MO, USA. (1999)

3. Guttman, A. *R-Trees: A Dynamic Index Structure for Spatial Searching.* ACM Conference on the Management of Data (SIGMOD), 47-57, Boston, MA, June 18-21, 1984.

4. Hjaltason, G., Samet, H.: *Distance Browsing in Spatial Databases.* ACM Trans. Database Syst. 24(2): 265-318 (1999)

5. Konstantopoulos, S., Paliouras, G., and Chatzinotas, S.: 2006a, *SHARE-ODS: An Ontology Data Service for Search and Rescue Operations.* Technical Report DEMO-2006-1, NCSR 'Demokritos', Athens.

6. Konstantopoulos, S., Paliouras, G., and Chatzinotas, S.: 2006b, *SHARE-ODS: An Ontology Data Service for Search and Rescue Operations.* In 4th Hel. Conf. on AI, SETN 2006,

7. Motik, B. and Sattler, U.: *Practical DL Reasoning over Large ABoxes with KAON2.* Submitted for publication. { http://www.fzi.de/ipe/eng/publikationen.php}

8. Necib, C-B. and Freytag, J-C.: *Ontology based Query Processing in Database Management Systems.* In Proceedings of ODBASE 2003.

9. Stoffel, K., Taylor M., and Hendler, J.: *Efficient Management of Very Large Ontologies.* In Proceedings of AAAI 1997.