Exploiting Geographic References of Documents in a Geographical Information Retrieval System Using an Ontology-based Index

Nieves R. Brisaboa · Miguel R. Luaces · Ángeles S. Places · Diego Seco

Received: 31 January 2009 / Revised: 4 August 2009 / Accepted 11 January 2010 / Published online: 30 January 2010

Abstract Both *Geographic Information Systems* and *Information Retrieval* have been very active research fields in the last decades. Lately, a new research field called *Geographic Information Retrieval* has appeared from the intersection of these two fields. The main goal of this field is to define index structures and techniques to efficiently store and retrieve documents using both the text and the geographic references contained within the text. We present in this paper two contributions to this research field. First, we propose a new index structure that combines an inverted index and a spatial index based on an ontology of geographic space. This structure improves the query capabilities of other proposals. Then, we describe the architecture of a system for geographic information retrieval that defines a workflow for the extraction of the geographic references in documents. The architecture also uses the index structure that we propose to solve pure spatial and textual queries as well as hybrid queries that combine both a textual and a spatial component. Furthermore, query expansion can be performed on geographic references because the index structure is based in an ontology.

Keywords geographic information retrieval · spatial index · textual index · ontology · system architecture

1 Introduction

Although the research field of Information Retrieval [1] has been active for the last decades, the growing importance of Internet and the World Wide Web have made it

This is an Author's Original Manuscript of an article whose final and definitive form has been published in GeoInformatica. The final publication is available at http://link.springer.com/article/10.1007/s10707-010-0106-3

This work has been partially supported by Ministerio de Educación y Ciencia (PGE and FEDER), grant TIN2009-14560-C03-02, and by "Xunta de Galicia" ref. 08SIN009CT.

Database Laboratory, University of A Coruña
Campus de Elviña, 15071 A Coruña, Spain
E-mail: {brisaboa, luaces, asplaces, dseco}@udc.es

one of the most important research fields nowadays. Many different index structures, compression techniques, and retrieval algorithms have been proposed in the last few years. More importantly, these proposals have been widely used in the implementation of document databases, digital libraries, and web search engines.

Another field that has received much attention during the last years is the field of Geographic Information Systems [2]. Recent improvements in hardware have made the implementation of this type of systems affordable for many organizations. Furthermore, a cooperative effort has been undertaken by two international organizations (ISO [3] and the Open Geospatial Consortium [4]) to define standards and specifications for interoperable systems. This effort is making possible that many public organizations are working on the construction of spatial data infrastructures [5] that will enable them to share their geographic information.

During the last decades these two research fields have advanced independently. However, many of the documents stored in digital libraries and document databases include geographic references within their texts. For example, news documents reference the place where the event happened and often the place where the document has been written. Furthermore, the information in a spatial data infrastructure often includes documents with geographic information such as construction licences or urban planning information. Finally, geographic references can also be attached to web pages by using information from the text, the location of the web server, and many other information elements.

Even though it is very common that textual and geographic information occur together in information systems, the geographic references of documents are rarely used in information retrieval systems. Few index structures or retrieval algorithms take into account the spatial nature of geographic references embedded within documents. Pure textual techniques focus only on the language aspects of the documents and pure spatial techniques focus only on the geographic aspects of the documents. None of them are suitable for a combined approach to information retrieval because they completely neglect the other type of information. As a result, there is a lack of system architectures, index structures, and query languages that combine both types of information.

Some proposals have appeared recently [6–8] that define new index structures that take into account both the textual and the geographic aspects of a document. However, there are some specific particularities of geographic space that are not taken into account by these approaches. Particularly, concepts such as the hierarchical nature of geographic space and the topological relationships between the geographic objects must be considered in order to fully represent the relationships between the documents and to allow new and interesting types of queries to be posed to the system.

In this paper, we present an index structure that takes these issues into account. We first describe some basic concepts and related work in Section 2. Section 3 presents some conceptual ideas about our index structure. Then, in that same section, we describe the types of queries that can be answered with this system and we sketch the algorithms that we use to solve these queries. In Section 4, we present the general architecture of the system and describe how the index structure has been integrated in it. Some implementation details about the index structure are sketched in this section as well. Furthermore, Section 5 presents some experiments that we made to compare our structure with other ones that use a pure spatial index. Finally, Section 6 presents some conclusions and future lines of work.

2 Related Work

Inverted indexes are considered the classical text indexing technique. An inverted index associates to each word in the text (organized as a *vocabulary*) a list of pointers to the positions where the word appears in the document. The set of all those lists is called the *occurrences* [1]. The main drawback of these indexes is that geographic references are mostly ignored because place names are considered words just like the others. If the user poses a query such as *hotels in Spain*, the place name *Spain* is considered a word, and only those documents that contain that word are retrieved. A document containing only names of cities of Spain but not the exact word *Spain* is not retrieved by the system because it does not fulfil the textual query.

Regarding indexing geographic information, many different spatial index structures have been proposed along the years. A good survey of these structures can be found in [9]. The main goal of spatial index structures is improving access time to collections of geographic data objects. One of the most popular spatial index structures, and a paradigmatic example, is the R-Tree [10]. The R-Tree is a balanced tree derived from the B-tree which splits space in hierarchically nested, possibly overlapping, minimum bounding rectangles. The number of children of each internal node varies between a minimum and a maximum. The tree is kept in balance by splitting overflowing nodes and merging underflowing nodes. Rectangles are associated with the leaf nodes, and each internal node stores the bounding box of all the rectangles in its subtree. The decomposition of space provided by an R-Tree is adaptive (dependent on the rectangles stored) and overlapping (nodes in the tree may represent overlapping regions).

A drawback of spatial index structures is that they do not take into consideration the hierarchy of space. Internal nodes in the structure are meaningless in the real world, they are just meaningful for the index structure. For example, imagine that we want to build an index for a collection of countries, provinces, and cities. These objects are structured in a topological relationship of containment, that is, a city is contained within a province that is itself contained within a country. If we build an R-Tree with these geographic objects, the containment hierarchy will not be maintained. The internal nodes of the R-Tree do not represent provinces nor countries, and therefore, the hierarchy of space is not maintained in the index. It is not possible to associate some information to the node of a province and have the cities belonging to this province inherit this information because there is no relation at all between a province and its cities in the R-Tree index structure.

Some work has been done to combine both types of indexes. Finding geographical references in text is a very difficult problem and there have been many papers that deal with different aspects of this problem and describe complete systems such as Web-a-where [11], MetaCarta [12], and STEWARD [6]. The papers about the SPIRIT (Spatially-Aware Information Retrieval on the Internet) project [13–17] are a very good starting point. In [16], the authors conclude that keeping separate text and spatial indexes, instead of combining both in one, results in less storage costs but it could lead to higher response times. More recent works can be broadly classified into two categories depending on how they combine textual and spatial indexes. On the one hand, some proposal have appeared that combine textual and spatial aspects in an hybrid index [18, 19]. On the other hand, some proposals define structures that keep separate indexes for spatial and text attributes [6–8]. Our index structure is part of this second group because this division has many advantages [8]. First of all, all textual queries can be efficiently processed by the text index, and all spatial queries can be efficiently

processed by the spatial index. Moreover, queries combining textual and spatial aspects are supported. Updates in each index are handled independently, which makes easier the addition and removal of data. Finally, specific optimizations can be applied to each individual indexing structure. After that research, in [7,8], the authors survey the work in the SPIRIT project and propose improvements to the system and the algorithms defined. In their work they propose two naive algorithms: *Text-First* and *Geo-First*. Both algorithms use the same strategy: one index is first used to filter the documents (textual index in Text-First and spatial index in Geo-First), the resulting documents are sorted by their identifiers and then filtered using the other index (spatial index in Text-First and textual index in Geo-First). Nevertheless, none of these approaches take into account the relationships between the geographic objects that they are indexing.

A structure that can properly describe the specific characteristics of geographic space is an *ontology*, which is a formal explicit specification of a shared conceptualization [20]. An ontology provides a vocabulary of classes and relations to describe a given scope. In [21], a method is proposed for the efficient management of large spatial ontologies using a spatial index to improve the efficiency of the spatial queries. Furthermore, in [14,17] the authors describe how ontologies are used in query term expansion, relevance ranking, and web resource annotation in the SPIRIT project. However, as far as we know, nobody has ever tried to combine ontologies with other types of indexes to have a hybrid structure that captures both the topological and the spatial relationships between the geographic objects indexed.

3 Index Structure

There are two main requirements that must be fulfilled by the design of the index structure:

1. It must provide a description of the hierarchical nature of the geographic space in order to be able to associate information to the elements in the hierarchy.
2. It must be possible to use the index to solve efficiently pure textual and spatial queries as well as queries that combine both types of information.

In this section, we describe how the design of our index structure supports these requirements.

In order to support the first requirement, we base the design of the index structure on an ontology [20,22] of the geographic space that describes the concepts in our domain and the relationships that hold between them. There are different ontology languages that provide different formal and reasoning facilities. OWL [23] is a W3C standard language to describe ontologies and can be categorised into three species or sub-languages: OWL-Lite, OWL-DL, and OWL-Full. Our spatial ontology is described in OWL-DL and it can be downloaded from the following URL: `http://lbd.udc.es/ontologies/spatialrelations`. OWL classes can be interpreted as sets that contain individuals (also known as instances). Individuals can be considered *instances of classes*. Our ontology describes eight classes of interest: *SpatialThing*, *GeographicalThing*, *GeographicalRegion*, *GeopoliticalEntity*, *PopulatedPlace*, *Region*, *Country*, and *Continent*. In our ontology there are hierarchical relations among *SpatialThing*, *GeographicalThing*, *GeographicalRegion*, *GeopoliticalEntity* because:

- *GeopoliticalEntity* is subclass of *GeographicalRegion*

- *GeographicalRegion* is subclass of *GeographicalThing* and
- *GeographicalThing* is subclass of *SpatialThing*.

That is, these four classes are organised into a superclass-subclass hierarchy, which is also known as *taxonomy*. Subclasses specialise (are subsumed by) their superclasses. *GeopoliticalEntity* has four subclasses: *PopulatedPlace*, *Country*, *Continent*, and *Region*. All the individuals are members of these subclasses. These four subclasses have an additional necessarily asserted condition regarding their relations with each other. They are connected by the property *spatiallyContainedBy* that describes the existence of an spatial relationship among them. For instance, all the individuals of class *PopulatedPlace* are *spatiallyContainedBy* individuals of class *Region* (described in OWL as *PopulatedPlace spatiallyContainedBy only (AllValuesFrom) Region*). Fig. 1 shows an example of these relationships. Ontology classes are represented as circles, individuals as rectangles, and the relationships as labelled lines.

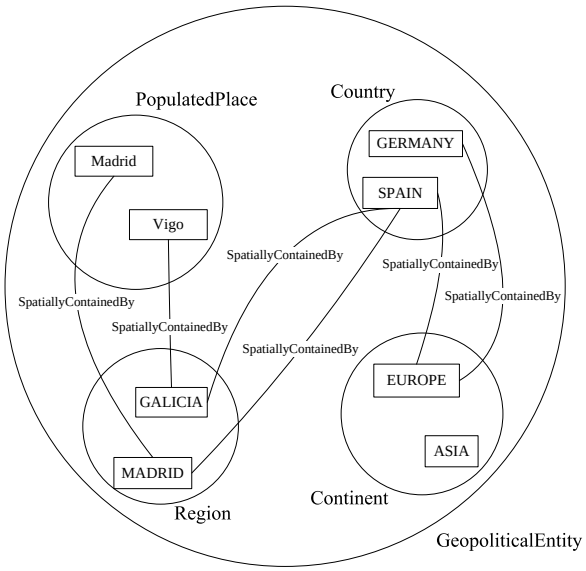

Fig. 1 Ontology instances

After having defined this ontology, we can define an spatial index structure based on it. This structure is a tree with four levels, one for each of the subclasses of *GeopoliticalEntity*. The top-most level contains a node for each of the instances of the class *Continent*. Each node in this level references the instances of the class *Country* that are connected by the *spatiallyContainedBy* relationship. The levels of *Region* and *PopulatedPlace* are built using the same strategy. That is, the structure of the tree follows the taxonomy of the ontology. Fig. 2 shows the spatial index structure built from the instances shown in Fig. 1.

Each node contains, in addition to the list of child nodes that are *spatiallyContainedBy* the node, the following information: (i) the keyword (a location name), (ii) the bounding box of the geometry representing this location, and (iii) a list with the

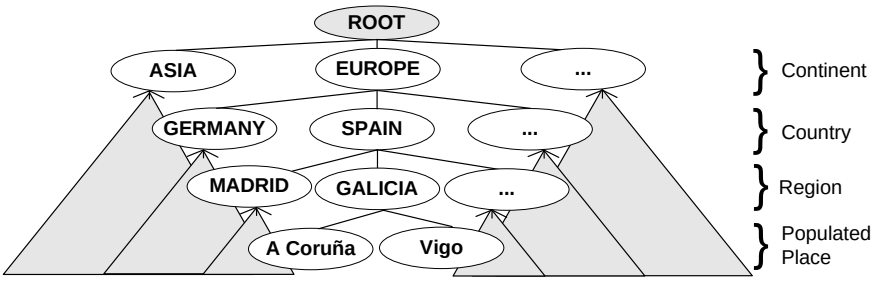

Fig. 2 Ontology tree

document identifiers of the documents that include geographic references to this location. Therefore, this tree structure can be used as a spatial index structure because, given a query point, a top-down traversal of the structure can be performed discarding those branches of the tree that do not contain the query point. The nodes in the tree that remain after the traversal are those that contain the query point.

The main advantage of this spatial index structure over other alternatives is that intermediate nodes in the structure have a meaning in the geographic space and they can have additional information associated. For instance, we can associate a list of documents that reference a given *Country* and use this list of documents to solve combined textual and spatial queries. Moreover, given that there is a superclass-subclass relationship between the levels, the bottom levels can inherit the properties of the top levels. Particularly, the documents associated to a node in the structure also refer to all nodes in its subtree. Furthermore, the index structure is general in the sense that the ontology of geograhic space can be adapted to each particular application. For example, if a particular application uses a restricted area of the geographic space where the classes *Continent* and *Country* are not necessary and, on the other hand, the classes *Province*, *Municipality*, *City*, and *Suburb* are needed, we could define a different ontology of space and base the index structure on it as long as the relationship *spatiallyContainedBy* still holds between the classes. Finally, we could define additional spatial relationships in the ontology such as *spatiallyAdjacent* and mantain these relationships in the index structure to improve the query capabilities of the system.

The second requirement is met through the definition of two additional components in the index structure: a textual index and a place name hash table. The *textual index* is an inverted index that associates to each word a list of documents that contain it. Finally, the *place name hash table* stores for each location name its position in the spatial index structure and it is used to optimize the resolution of a particular type of very common queries.

Fig. 3 shows an example of the complete index structure with the inverted index, the place name hash table, and the spatial index structure. We use this example to explain how our index is built. In order to index documents in our structure, each document is analyzed to obtain and geo-reference place names cited in it (a detailed description of this process can be read in Section 4.3). In the example, the document *D1* cites the place name *Germany*, the document *D2* cites the place name *Madrid*, etc. Our structure is dynamically built using the information obtained in the geo-reference process of those place names. This information contains data necessary to include each place in the structure. For example, when the first document (e.g. *D1*) is indexed

the structure is empty, and the nodes *EUROPE* and *GERMANY* are created. Then, when the document *D2* is indexed, the node *EUROPE* is not created again, but nodes *SPAIN*, *MADRID*, and *Madrid* are created. No nodes are created when the document *D3* is indexed, but its document identifier is added to the lists associated with the nodes *GERMANY* and *MADRID*. Each time that a new node is created, a new element is added to the place name hash table with the name associated with the new node and a pointer to it. In parallel, documents are indexed in the inverted index, which stores for each word the document identifiers where it is contained. For example, the document *D2* contains the words *hotel* and *sunny*.

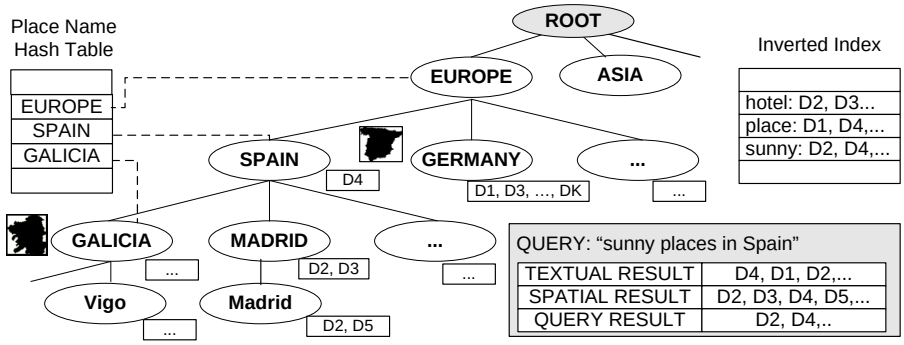

Fig. 3 Example of the index structure

The most important characteristic of an index structure is the type of queries that can be solved with it. The following types of queries are relevant in a geographic information retrieval system:

- *Pure textual queries.* These are queries such as "*retrieve all documents where the words* hotel *and* sea *appear*".
- *Pure spatial queries.* An example of this type of queries is "*retrieve all documents that refer to the following geographic area*". The geographic area in the query can be a point, a query window, or even a complex object such as a polygon.
- *Textual queries with place names.* In this type of queries, some of the words are place names. For instance, "*retrieve all documents with the word* hotel *that refer to* Spain".
- *Textual queries over a geographic area.* In this case, a geographic area of interest is given in addition to the set of words. An example is "*retrieve all documents with the word* hotel *that refer to the following geographic area*".

Pure textual queries can be solved by our system because a textual index is part of the index structure. Similarly, pure spatial queries can also be solved because the index structure is built like a spatial index. Each node in the tree is associated with the bounding box of the geographic objects in its subtree. Therefore, the same algorithm that is used with spatial indexes can be used with our structure.

Furthermore, the index structure that we propose can be used to solve queries that involve a textual and a spatial component. In this case, the textual index is used to retrieve the list of documents that contain the words and the spatial index structure is used to compute the list of documents that reference the geographic area. The result

to the query is computed as the intersection of both lists. In the case of queries such as "*sunny places in Spain*" (see Fig. 3), our system uses the *place name hash table* to retrieve the index node that represents *Spain*. Thus, we save some time by avoiding a tree traversal.

Another improvement over text and spatial indexes is that our index structure can easily perform query expansion on geographic references because the index structure is built from an ontology of the geographic space. Consider the following query "*retrieve all documents that refer to Spain*". The query evaluation service will discover that Spain is a geographic reference and the place name index will be used to quickly locate the internal node that represents the geographic object *Spain*. Then, all the documents associated to this node are part of the query result. Moreover, all the children of this node are geographic objects that are contained within Spain (for instance, the city of Madrid). Therefore, all the documents referenced by the subtree are also part of the result of the query. The consequence is that the index structure has been used to expand the query because the result contains not only those documents that include the term *Spain*, but also all the documents that contain the name of a geographic object included in Spain (e.g., all the cities and regions of Spain).

4 System Architecture

Fig. 4 shows our proposal for the system architecture of a geographic information retrieval system. The architecture can be divided into three independent layers: the index construction workflow, the processing services, and the user interfaces. The bottom part of the figure shows the index construction workflow, which, in turn, consists of three modules: the document abstraction module (described in Section 4.1), the index construction module (the textual part of this process is described in Section 4.2 and the spatial part of this process is described in Section 4.3), and the index structure itself (described also in Section 4.3).

The processing services are shown in the middle of the figure. On the left side, the *Geographic Space Ontology Service* used in the spatial index construction is shown. This service is used extensively in the index construction module, and therefore it is described in Section 4.3. On the right side, one can see the two services that are used to solve queries. The rightmost one is the *query evaluation service*, which receives queries and uses the index structure to solve them. The types of queries that can be solved by this service, as well as the algorithms that are used to solve these queries have been presented in Section 3. In Section 4.4 we present how the *query evaluation service* uses these algorithms to create the relevance ranking of the results. The other service is a *Web Map Service* following the OGC specification [24] that is used to create cartographic representations of the query results. This service is not described in this paper. On top of these services a *Geographic Information Retrieval Module* is in charge of coordinating the task performed by each service to respond to the user requests.

The topmost layer of the architecture shows the two user interfaces that exist in the architecture: the *Administration User Interface* and the *Query User Interface*. These user interfaces are described in Section 4.5.

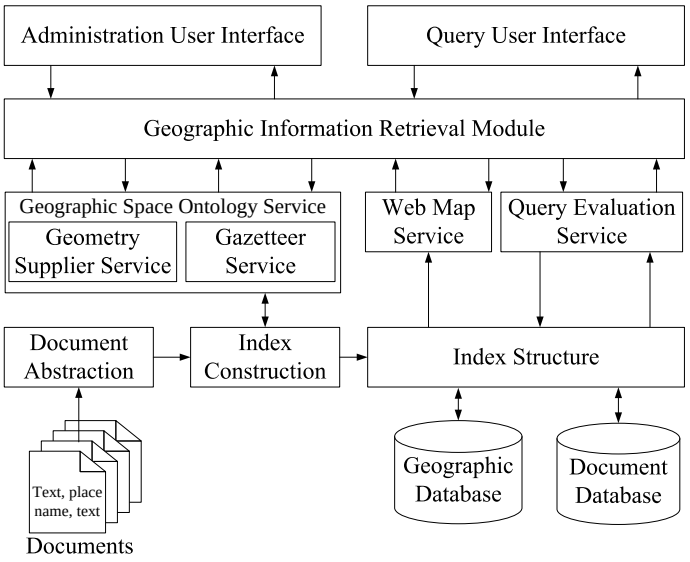

Fig. 4 System Architecture

4.1 Document Abstraction

Given that the system must be generic, it must support indexing several kinds of documents. These documents will be different not only because they may be stored using different file formats (plain text, XML, etc.), but also because their content schema may be different. A document collection may have a set of attributes that have to be stored in the index (such as *document id*, *author*, and *document text*), whereas other document collection may have a different set of attributes (such as *document id*, *summary*, *text*, *author*, and *source*).

To solve this problem, we have defined an abstraction that represents a *document* as a set of *fields*, each one with a value that is extracted from the document text. Each field can either be *stored*, *indexed*, or both. If a field is stored, its contents are stored in the index structure and they can be retrieved by a query. If a field is indexed, then this field is used to build the index structure. Furthermore, a field can be indexed textually, spatially, or in both indexes. The definition of a document as a set of fields is similar to the one used in the Lucene text search engine [25]. We have extended this idea adding the spatial indexing possibility.

In order to support different types of documents and different file formats, the document abstraction is exposed by the system as a programming interface that can be extended with particular implementations for different configurations of file formats and document schemas. In order to support a new configuration, a developer only has to implement the interface *DocumentFactory* that defines the operations that must be implemented in order to create *Documents*.

As an example for the validation of the system, we have indexed documents from the Financial Times collection [26]. The document collection is marked up in SGML (*Standard Generalized Markup Language*). Each document has a <DOCNO> tag including the TREC document identifier string and a <TEXT> tag including the main

content of the document. Fig. 5 shows a partial example of a document in this collection.

```
<DOC>
     <DOCNO>FT941-6371</DOCNO>
     <TEXT>
          Senior European company executives are being invited to 'vote'
          for Europe's Most Respected Companies . . .
     </TEXT>
</DOC>
```

Fig. 5 Financial Times (TREC) example

To support this document collection, we defined a *TRECFTDocumentFactory* that builds documents with two fields. The first field contains the tag DOCNO content and it is stored but not indexed. The second field contains the tag TEXT content and it is not stored but indexed in both indexes.

4.2 Textual Indexing

As we said before, the index structure at the core of the system architecture contains both a textual index and a spatial index. We use Lucene [25] to implement the textual index. Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is an open source project part of the Apache project. Lucene uses an object representation of the indexable documents. A *Document* in Lucene contains several *Fields*. A *Field* in Lucene is a pair (*name*, *value*) and information about whether it is stored and/or indexed. Field values are set using *Analyzers*. These analyzers implement several classical information retrieval techniques to reduce the number of indexed words and to improve the index performance such as *removing stopwords*, *stemmers*, etc. *StandardAnalyzer* is the most sophisticated analyzer built into Lucene's core. It is a parser with rules for email addresses, acronyms, hostnames, floating point numbers, as well as converting the value to lowercase and removing stop words.

In this stage of the workflow process, the system builds a Lucene index. Each of the documents built in the previous stage is inserted into the textual index. The document identifier is stored but not indexed in the textual index, and each field marked to be indexed in the textual index or in both indexes is indexed tokenized in the Lucene index but not stored.

4.3 Spatial Indexing

After building the textual index, the spatial index must be built. The spatial indexing is the most complex stage, and it comprises three steps. First, the system analyses the document fields that are marked as spatially indexable and extracts candidate location names from the text. In a second step, these candidate locations are processed in order to determine whether the candidates are real location names, and, in this case, to compute their geographic locations. There are some problems that can happen at this

point. First, a location name can be ambiguous (*polysemy*). For instance, "*London*" is the capital of the United Kingdom and it is a city in Ontario, Canada too. Second, there can be multiple names for the same geographic location, such as "*Los Angeles*" and "*LA*". Finding geographical references in text is a very difficult problem and there have been some papers that deal with different aspects of this problem [6, 11, 12]. Web-a-where [11] uses "spatial containers" in order to identify locations in documents, MetaCarta (the commercial system described in [12]) uses a natural language processing method, and STEWARD [6] uses an hybrid approach. It is not the aim of this paper to deal with this problem but we describe how we obtain geographic references in order to complete the architecture description. Finally, the third step consists in building the spatial index with the geo-referenced locations computed in the previous step together with references to the documents containing them. We describe these three steps and the spatial index structure below.

4.3.1 Discovery of Location Names

Unlike geographic information systems, information in GIR systems is not structured. It is not possible to know *a priori* where geographic references are stored, nor their categories (e.g. city, state, country, etc.). In this kind of systems, geographic references are contained in the text of the indexed documents. Therefore, these texts have to be analyzed in order to discover the geographic references.

In this analysis, all the spatially indexable fields are processed in order to discover the place names contained within. There are two *Linguistic Analysis* techniques that are widely used for this: *Part-Of-Speech* tagging and *Named-Entity Recognition*. On the one hand, Part-Of-Speech tagging is a process whereby tokens are sequentially labelled with syntactic labels, such as "verb" or "gerund". On the other hand, Named-Entity Recognition is the process of finding mentions of predefined categories such as the names of persons, organizations, locations, etc. Combine both techniques is a good solution to discover possible place names contained in the text of documents.

Our *Location Names Discovery* module uses the *Natural Language Tool LingPipe* [27] to find locations. It is a suite of Java libraries for the linguistic analysis of human language free for research purposes that provides both Part-Of-Speech tagging and Named-Entity Recognition. LingPipe involves the supervised training of a statistical model to recognize entities. The training data must be labelled with all of the entities of interest and their types. In the system validation with the Financial Times collection, we use LingPipe trained with the MUC6 corpus (http://www.ldc.upenn.edu) labelled with locations, people, and organizations. After the LingPipe processing, the module filters the resultant named entities selecting only the locations, and discarding people and organization names.

For each spatially indexable field of each document, the result of this analysis is a set of possible place names cited in its text. In the next step, all these candidates must be analyzed again to discard false positives because the performance of the used linguistic techniques is not complete.

4.3.2 Geo-referenciation of Location Names

After discovering a collection of candidate location names, the system must distinguish false candidates and geo-reference the real ones. In this context, geo-referencing a location name implies not only to obtain its coordinates in a particular coordinate

system, but also to obtain all the data needed to include the location name in the spatial index. We have developed a system based on an ontology of the geographic space that is built using a *Gazetteer* and a *Geometry Supplier*.

A Gazetteer is a geographical dictionary that contains, in addition to location names, alternative names, populations, location of places, and other information related to the location. In our test implementation we use *Geonames* [28] that provides a geographical database available under a creative commons attribution license. This database contains more than two million populated places over the world with their latitude/longitude coordinates in WGS84 (*World Geodetic System 1984*). All the populated places are categorized so that it is possible to classify them into different administrative division levels (continents, countries, regions, etc.).

However, Geonames (and Gazetteers in general) does not provide geometries for the location names other than a single representative point. But for our spatial index we need the real geometry of the location name (for example, the boundary of countries). Hence, we defined a *Geometry Supplier* service to obtain the geometries of those location names. As a base for this service we used the *Vector Map* (VMap) cartography [29]. VMap is an updated and improved version of the National Imagery and Mapping Agency's Digital Chart of the World. It supplies geometries and information for the first and the second level of administrative divisions of the World. Even though the information is in a proprietary format, there are free tools that can create *shapefiles* from that format, such as FWTools [30]. We have created a PostGIS [31] spatial database with these shapefiles and we have done several corrections and improvements over this database. Although our test implementation uses Geonames and VMAP, it has been designed so that these components are easily exchangeable. All accesses to these components are performed through generic interfaces that can be easily implemented for other components.

Both services are used to obtain the information necessary to include each location name in the spatial index structure. That is, for each location name, the services must return the path in the tree from the root node to the leaf node that represents the location name. As we noted at the beginning of the section, several problems, such as *polysemy* or *ambiguity*, can cause that more than one node represents a location name in the index structure. Thus, the services must return all the possible paths.

For this task, a hierarchical structure following the design pattern *Chain of Responsibility* [32] was designed and implemented. Fig. 6 shows a brief class diagram of this component. The structure is composed of four levels (continent, country, region, and populated place); one for each level of the ontology of the geographic space used in the system. Each level of the hierarchy has a connection with the gazetteer and with the geometry supplier in order to retrieve the data needed by the process. Since this hierarchy depends on the ontology of the geographic space, it must be easily extensible to support the definition of new levels in the ontology.

Furthermore, an algorithm in two steps was developed to obtain all possible georeferences of a location name. In the first step, each level of the hierarchy obtains from the gazetteer all the locations with the requested name. If the gazetteer does not return any location for a given candidate location name, the candidate is discarded. In the second step, the system builds the complete path from bottom to top using the gazetteer to find the parent element and the geometry supplier to find the geometry of the object. For instance, if the requested location name was London, in the first step the system obtains two locations with this name. After that, it returns the paths *Europe*, *United Kingdom*, *England*, *London* and *North America*, *Canada*, *Ontario*, *London*.

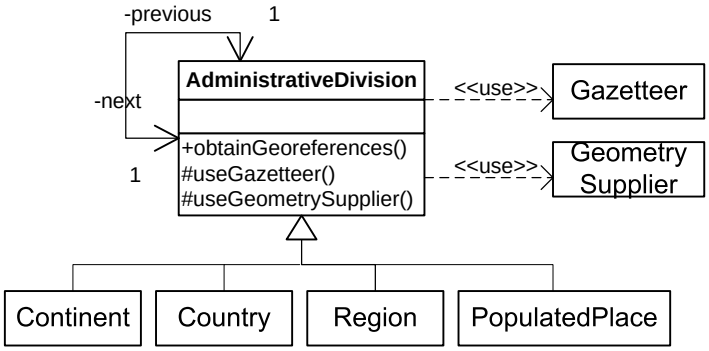

Fig. 6 Geo-references module

4.3.3 Spatial Index Construction

Fig. 7 shows a class diagram of the index structure. The design and implementation of this structure are based on the conceptual ideas presented in Section 3. Therefore, the main component of the index structure is a tree composed by nodes (*IndexStructureNode*) that represent location names. These nodes are connected by means of *spatiallyContainedBy* relationships and in each one we store the location name, the bounding box of the geometry representing this location, and the list with the identifiers of the documents that include geographic references to this location. Finally, an R-Tree is used to improve the access performance to the child nodes. Many child nodes can be accessed from each node (for example, there are hundreds of regions in each country and thousands of populated places in each region) and compare sequentially the *bounding box* of each child node with a query is a very expensive process. Therefore, we use an R-tree to reduce the number of comparisons performed to access the next level from each node.

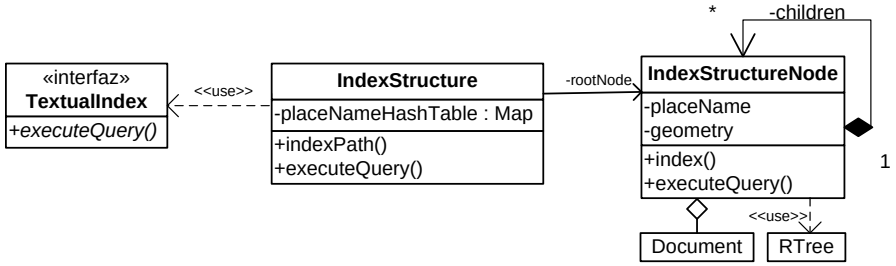

Fig. 7 Class diagram of the index structure

Two auxiliary structures are used in the index. First, a *place name hash table* that stores for each location name its position in the index structure. This provides direct access to a single node by means of a keyword that is returned by the Gazetteer Service

if the word processed is a location name. The second auxiliary structure is the textual index with all the words in the documents that is used to solve textual queries (this index is described in section 4.2).

Keeping separate indexes for text and geographical scopes has many advantages, which were described in Section 2. As a summary, both pure textual and spatial queries can be efficiently processed by each index, queries combining textual and spatial aspects are supported, updates in each index are handled independently, and specific optimizations can be applied to each individual indexing structure. However, this structure has two main drawbacks. First, the tree that supports the structure is possibly unbalanced penalizing the efficiency of the system. We present some experiments in Section 5. Our intention is to prove that it is not a very important problem. Second, ontological systems have a fixed structure and thus our structure is static and it must be constructed *ad-hoc*.

Access to the main databases of the system is allowed to the index structure. There are two main databases in our system: a document database where documents are stored to be consulted by the users and a geographic database. The geographic database stores geographic data used in the system. Although the access to these data is centralized in the index structure, all the modules of the architecture can access this information by means of the geographic information retrieval module. Share the information between all modules allows us to easily improve the performance of the whole system improving the base cartography. For example, when the VMap cartography is improved both the geo-reference process and the web map service are improved. The same happens when gazetteer data are corrected.

Finally, the index construction is a time-expensive process but it is similar to other index structures proposed for GIR systems. Although insert a new node in our structure is an easy task, obtain all the data necessary to perform this task is much more complex. As we noted before, all the documents are analyzed to obtain cited place names and all these place names are geo-referenced. The main difference between our index structure and other proposals is that we have to obtain for each node all the information of the nodes in the path from the root to it. However, this process is optimized by means of a cache that stores the nodes already obtained. Therefore, for example when the geo-reference process of the place name *Madrid* is performed after the geo-reference process of the place name *Barcelona*, the data about *Spain* and *Europe* is contained in the cache and it have not to be retrieved from the gazetteer and from the geometry supplier.

4.4 Query Evaluation Service

We presented in Section 3 the different types of queries that can be solved using this index structure, and we sketched briefly the algorithms used to solve the queries. The *Query Evaluation Service* is the component in charge of using the index structure to answer the queries posed by the users. Moreover, in order to return a useful result, this service must also provide a relevance ranking of the results. In this section, we describe the equations used to compute the relevance of the result for each type of queries: *pure textual queries*, *pure spatial queries*, and *hybrid queries*.

4.4.1 Pure textual queries

Pure textual queries are solved by the textual index that is part of the index structure. We use Lucene to implement this textual index, and thus the relevance ranking depends on its scoring. Lucene scoring uses a combination of the vector space model and the boolean model of information retrieval [1]. All scores are guaranteed to be 1.0 or less. More information about the Lucene scoring can be found in [33, 34].

4.4.2 Pure spatial queries

Pure spatial queries are solved using the spatial index structure. Given that a document in the result set of a query can include geographic references to one or more location names relevant to the query, we have to compute the relevance of the document d with respect to the query q due to each location name l. We denote this relevance as $toponymRelevance_{q,d,l}$. We guarantee that this value is 1.0 or less in order to make the integration of both spatial and textual relevances easier. In [35] both spatial and textual relevances are also normalized to values between 0 and 1. Finally, we compute the relevance of the document d with respect to the query q as the maximum relevance due to any location name (Equation 1).

$$spatialRelevance_{q,d} = max\{toponymRelevance_{q,d,l}\} \tag{1}$$

Some papers [36, 18] present quantitative measures for the spatial relevance using the area of the *bounding boxes*, overlap areas, distances, and other spatial-based measures. Furthermore, authors of [8, 37] define equations to calculate the spatial relevance when an ontology is used in the system. In [38], a proposal to combine spatial-based measures and ontology-based measures is presented. In our system, the computation of $toponymRelevance_{q,d,l}$ depends on whether the user specifies the spatial context for the query *using a location name*, *selecting a node in the spatial index*, or *using a query window*. We study all these cases and define equations based on previous works.

In the case of queries specified using a location name, the query result contains all the documents in the nodes that have the location name as the node keyword plus all the documents in the respective subtrees. Considering that there may be more than one node in the index structure for a given location name (due to *polysemy*), we associate to each node in the spatial index structure an attribute *importance* that represents the importance of a node relative to the other nodes that represent locations with the same name. Population, size, level in the ontology of geographic space, and other intrinsic values are used to calculate the importance. For each location name, this attribute always has the value 1.0 for the most important nodes and values higher than 1.0 for the rest. Equation 2 combines this attribute with the tree *distance* from the root of the subtree to the node where the document is associated to compute the spatial relevance of each document d due to the location name l.

$$toponymRelevance_{q,d,l} = \frac{0.5^{distance}}{importance} \tag{2}$$

The computation of $toponymRelevance_{q,d,l}$ for queries specified selecting a node in the spatial index is a simplification of the previous one because in this case we have the certainty that the query refers to a specific node in the tree. Therefore, the documents associated to this node have relevance 1.0. The relevance of a document associated to

the nodes in the subtree is computed using the previous equation. This is reflected in Equation 3.

$$toponymRelevance_{q,d,l} = \begin{cases} 1 & \text{if l is specified in the query} \\ \frac{0.5^{distance}}{importance} & \text{otherwise} \end{cases} \quad (3)$$

The sketch of the index structure shown in Fig. 8 is useful to understand the difference between both types of queries. Each node in the figure is annotated with its importance between parenthesis. On the one hand, when the user specifies a query using the location name *England*, the relevance of a document due to *England* (an important city of Arkansas) will be higher than the relevance due to *England* (a small city of Oppland Fylke), and lower than the relevance due to *England* (a part of the United Kingdom). Concrete values of relevance are 0.5 for England in Arkansas, 0.33 for England in Oppland, and 1.0 for England in the United Kingdom. Moreover, the relevance of the document due to important cities of England (UK) like London or Liverpool is 0.5. This value is high enough to be taken into consideration. On the other hand, when the query is specified selecting the node for England in Arkansas the relevance of a document due to this node is 1.0 because the user explicitly indicates the interest about documents with geographic references to that location.

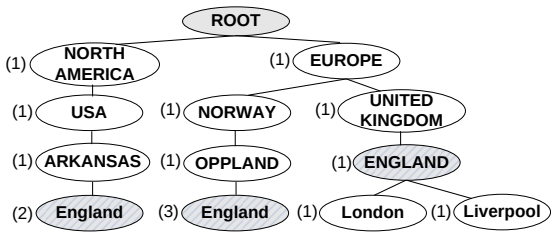

Fig. 8 Queries specified using a location name vs queries specified selecting a node

Finally, in the case of queries specified using a query window the nodes are selected using the classical algorithm of spatial indexes. Therefore, the computation of $toponymRelevance_{q,d,l}$ must be performed using the distance ($dc_{q,l}$) and the overlap area ($oa_{q,l}$) between the query window and the location name. Equation 4 defines this computation. We use parameters w_{dc} and w_{oa} to weight the relevance of each factor and we use the importance of the location name to assign more relevance to the most important nodes that reference the location name.

$$toponymRelevance_{q,d,l} = \frac{w_{dc} * dc_{q,l} + w_{oa} * oa_{q,l}}{importance} \quad (4)$$

Equation 5 defines how to calculate the relevance due to the distance to the query window. $centerDistance_{q,l}$ represents the Euclidean distance between the location name l and the query window q. Similarly, $cornerDistance_q$ is a weight factor that represents the maximum distance to the center of the window.

$$dc_{q,l} = 1 - \frac{centerDistance_{q,l}}{cornerDistance_q} \quad (5)$$

The relevance due to the overlap area with the query window is calculated according to Equation 6. When the geometry stored in the node is a point (leaf node), the overlap area is not significant. Thus, we use $\frac{1}{[area(q)+1]^{0.15}}$. This value depends only on the query window and is inversely proportional to its area. The concrete equation has been constructed based on the average area of the nodes in each level of the ontology of geographic space.

$$oa_{q,l} = \begin{cases} \frac{1}{[area(q)+1]^{0.15}} & \text{if l is a point} \\ max\{0, \frac{area(l \cap q)}{area(q)} - \frac{area(l \otimes q)}{area(l)}\} & \text{otherwise} \end{cases} \quad (6)$$

Fig. 9 uses an example query window in central Italy to clarify the aforementioned equations. The bounding boxes of two regions, Umbria and Abruzzi, and a populated place, Rome, are shown in this figure. These bounding boxes as well as the query window, q, are used to compute the area of their respective entities (i.e. area(Umbria), area(Abruzzi), and area(q)). The region of Umbria is used to illustrate the relevance due to the overlap area (Equation 6). This relevance is computed using the area of the intersection of the region with the query (i.e. $area(Umbria \cap q)$) and the area in the part of the region that does not intersect with the query (i.e. $area(Umbria \otimes q)$. Moreover, three distances used to compute the relevance due the distance to the query window (Equation 5) are shown. The weight factor *corner distance* is depicted as a solid line, and the distances from Rome and Abruzzi to the center of the query window are depicted as dotted lines.

Fig. 9 Queries specified using a query window

4.4.3 Hybrid queries

Hybrid queries that involve a textual and a spatial component are solved using the textual and spatial indexes. Hence, we use the previous equations to compute spatial and textual relevance rankings. Equation 7 defines how we combine both relevance

rankings. The weighted sum of the spatial and textual ranking values is one of the simplest methods and is commonly used [8, 18, 37]. Furthermore, it is the base of more complex ranking methods [39]. We assume $w_t = 1 - w_s$ and calculate w_t to normalize the differences between textual and spatial rankings.

$$relevance_{q,d} = w_t * textualRelevance_{q,d} + w_s * spatialRelevance_{q,d} \quad (7)$$

4.5 User Interfaces

The system has two different user interfaces: an administration user interface and a query user interface. The administration user interface was developed as a stand-alone application and it can be used to manage the document collection. The main functionalities are: creation of indexes, addition of documents to indexes, loading and storing indexes, etc. The main screen of this interface shows useful information about the loaded index such as the number of documents indexed, the fields of each of these documents, the number of location names in the index, etc.

Fig. 10 shows a screenshot of the query user interface. This interface was developed as a web application using the *Open Layers API* [40]. This API provides a number of utilities for manipulating maps and adding content to the map.

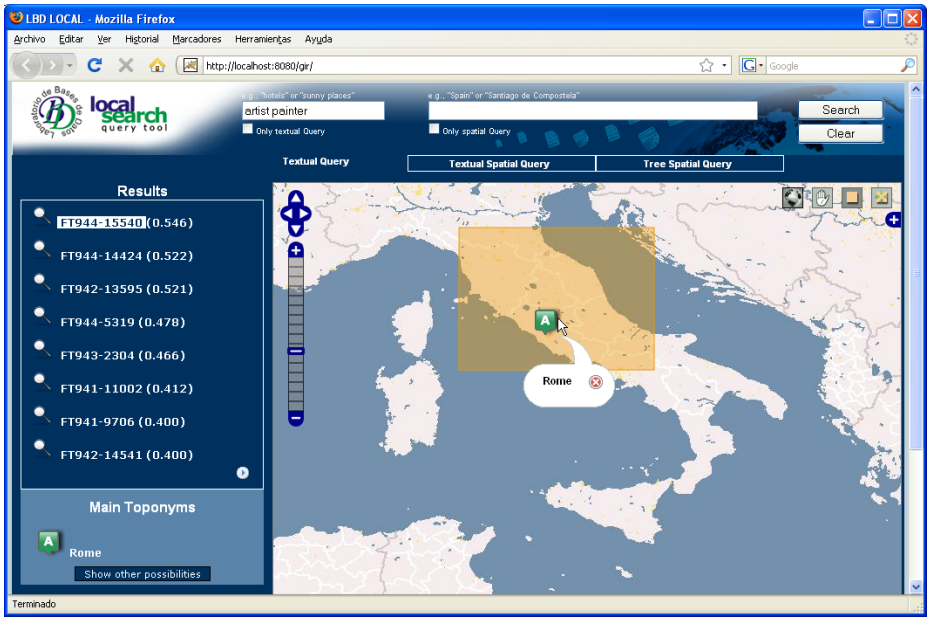

Fig. 10 Query User Interface

In Section 3, we have sketched the types of queries that can be solved with this system. These queries have two different aspects: a textual aspect and a spatial aspect. The query user interface allows the user to indicate both aspects. The spatial context can be introduced in three ways that are mutually exclusive:

- *Typing the location name.* In this case, the user types the location name in a text box. This is the most inefficient way because the system has to obtain all the geographic references associated with the place name typed by the user, which is a time-expensive process.
- *Selecting the location name in a tree.* In this case, the user sequentially selects a continent, a country within this continent, a region within the country, and a populated place within the region. If the user wants to specify a location name of a higher level than a populated place, it is not necessary to fill in all the levels. The operation is very easy and intuitive because the interface is implemented with a custom-developed component using the AJAX technology that retrieves in the background the location names for the next level. When the user selects a place in the component, the map on the right zooms in automatically to the selected place.
- *Selecting the spatial context of interest in the map.* The user can navigate using the map on the right to visualize the spatial context of interest. After that, a rectangle can be drawn over it. The system will use this rectangle as the query window if the user did not type a place name or did not select a location name.

5 Experiments

We showed in the previous section that our structure has a qualitative advantage over systems that combine a textual index with a pure spatial index because query expansion can be performed directly with our index structure. Hence, our index structure supports a new type of query that cannot be easily implemented with a pure spatial index. However, unlike pure spatial index structures, our index structure is not balanced and therefore, the query performance can be worse. In this section we describe the experiments that we performed to compare our structure with other ones based on pure spatial indexes in order to evaluate whether having an unbalanced structure would penalize efficiency.

We used the TREC FT-91 (Financial Times, year 1991) and the TREC FT-94 (Financial Times, year 1994) document collections. Table 1 summarizes the most relevant characteristics of these collections. The first row of the table shows the number of documents in each collection. The FT-94 collection has thirteen times more documents than the FT-91. We use these collections to proof the scalability of the different index structures with respect to the number of documents. The rest of the rows in this table are closely linked with the steps of the workflow presented in Section 4. First, *documents with candidates* are those that contain at least one candidate location name discovered at the *discovery of location names* step. Sometimes, these candidates can be false location names and the system will discard them at the *geo-referenciation of location names* step. Therefore, *documents geo-referenced* are those that contain at least one location name correctly geo-referenced in the system. Finally, *textual entries* and *spatial entries* are the number of entries in the textual index and in the spatial index respectively.

Three indexes were built over these collections. The first one uses our index structure as described in this paper. The second one uses a textual index and an R-Tree where each pair *{document identifier, location}* is stored as a single element (e.g., {d2, Madrid (40.26, 3.41)} means that the document *d2* mentions the city of Madrid). The last one, named *Improved R-Tree*, uses a textual index and an R-Tree as well, but locations are stored only once. Therefore, each location in the index stores a list of

Table 1 TREC document collections

	FT-91	**FT-94**
Documents	5,368	71,489
Documents with candidates	4,652	60,823
Documents geo-referenced	4,182	54,899
Textual entries	64,711	251,057
Spatial entries	21,282	64,843

Table 2 Ontology-based index versus R-Tree

	FT-91				**FT-94**			
Query area (%)	0.001	0.01	0.1	1	0.001	0.01	0.1	1
Our index	0.013	0.017	0.052	0.360	0.016	0.035	0.228	2.938
R-Tree	0.010	0.016	0.057	0.370	0.041	0.113	0.583	4.704
Improved R-Tree	0.008	0.007	0.023	0.154	0.008	0.016	0.095	1.069

documents that mentions it. This improvement reduces the size of the index structure and improves its performance.

Furthermore, we developed an algorithm to generate random spatial query windows based on the performance comparisons of the R*-Tree in [41]. We compared the structures with respect to four different query window areas, namely 0.001%, 0.01%, 0.1%, and 1% of the world. We generated 100,000 random query windows for each area, and we averaged the computing time of each query execution. Table 2 shows the results of this experiment. The first row of the table shows the results obtained with our structure (in milliseconds), the second one shows the results obtained with the structure using an R-Tree, and the third one shows the results obtained with the structure using an R-Tree improved.

In the light of the results, the index structure using an R-Tree adapted to take into consideration the specific characteristics of the problem (i.e. the Improved R-Tree) improves the query performance of our solution. As we noted before, this was an expected result because our index structure is not balanced. However, we believe the query performance of our solution is acceptable because differences are not too significant. Furthermore, our solution and the improved R-Tree present similar results in terms of scalability. Therefore, our solution can be considered an alternative to the use of pure spatial indexes.

Another important conclusion is about the query performance and scalability of the solution using a pure R-Tree. This alternative presents an acceptable performance in the FT-91 collection, but its scalability is quite worse than the scalability of the other alternatives. The explanation resides in the number of nodes in this structure. Both our solution and the improved R-Tree employ a list to store the documents geo-referenced in a place. However, the solution using a pure R-Tree requires one node for each pair {document identifier, location}. Thus, the number of nodes is quite higher. An unexpected result is that the performance of the pure R-Tree improves the performance of our structure when the collection and the query window are small. In order to discover the reason of this behaviour we performed additional experiments considering

Table 3 Low document density vs high document density

	High density				Low density			
Query area (%)	0.001	0.01	0.1	1	0.001	0.01	0.1	1
Our index	0.03	0.11	1.05	9.84	0.02	0.03	0.09	0.4
R-Tree	0.07	0.22	1.64	12.85	0.02	0.03	0.07	0.2

the distribution of the documents. Table 3 breaks down the general results performed with the FT-91 collection in two zones with different density of documents (zones with a high density of documents and zones with a low density). When the document density is low, the number of nodes in both structures is similar, but when the document density is high the difference in the number of nodes is quite significant. Thus, when the query window is small the probability of that query window being in a high document density zone is small and, therefore, the R-Tree performance is better. However, when the query window is bigger that probability is higher and, therefore, the R-Tree performance is lower.

6 Conclusions and Future Work

We have presented in this paper a system architecture for an information retrieval system that takes into account not only the text in the documents but also the geographic references included in the documents and the ontology of the geographic space. This is achieved by a new index structure that combines a textual index, a spatial index, and an ontology-based structure. We have also presented how traditional queries can be solved using the index structure, and new types of queries that can be solved with the index structure are described and the algorithms that solve these queries are sketched. Finally, we performed some experiments that show that the performance of our structure is acceptable in comparison with index structures using pure spatial indexes.

Future improvements of this index structure are possible. We are currently working on the evaluation of the performance of the index structure, particularly we are performing experiments to determine the precision and recall. Moreover, *Toponym Resolution* techniques must be implemented to solve ambiguity problems when we geo-reference the documents. Another line of future work involves exploring the use of different ontologies and determining how each ontology affects the resulting index. Furthermore, we plan on including other types of spatial relationships in the index structure in addition to inclusion (e.g., adjacency). These relationships can be easily represented in the ontology-based structure, and the index structure can be extended to support them.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley (1999)
2. Worboys, M.F.: GIS: A Computing Perspective. CRC (2004). ISBN: 0415283752
3. ISO/IEC: Geographic Information – Reference Model. International Standard 19101, ISO/IEC (2002)
4. Open GIS Consortium, Inc.: OpenGIS Reference Model. OpenGIS Project Document 03-040, Open GIS Consortium, Inc. (2003)

5. Global Spatial Data Infrastructure Association: Online documentation. Retrieved March 2008 from http://www.gsdi.org/
6. Lieberman, M.D., Samet, H., Sankaranarayanan, J., Sperling, J.: STEWARD: Architecture of a Spatio-Textual Search Engine. In: Proceedings of the 15th ACM Int. Symp. on Advances in GIS (ACMGIS'07), pp. 186 – 193. ACM Press (2007)
7. Chen, Y.Y., Suel, T., Markowetz, A.: Efficient query processing in geographic web search engines. In: SIGMOD Conference, pp. 277–288 (2006)
8. Martins, B., Silva, M.J., Andrade, L.: Indexing and ranking in Geo-IR systems. In: GIR '05: Proceedings of the 2005 workshop on Geogr. Inform. Retrieval, pp. 31–34. ACM Press, New York,USA (2005). DOI http://doi.acm.org/10.1145/1096985.1096993
9. Gaede, V., Günther, O.: Multidimensional access methods. ACM Comput. Surv. **30**(2), 170–231 (1998). DOI http://doi.acm.org/10.1145/280277.280279
10. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: B. Yormark (ed.) SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984, pp. 47–57. ACM Press (1984)
11. Amitay, E., Har'El, N., Sivan, R., Soffer, A.: Web-a-where: geotagging web content. In: SIGIR '04: Proceedings of the 27th ACM SIGIR, pp. 273–280. ACM, New York, USA (2004). DOI http://doi.acm.org/10.1145/1008992.1009040
12. Rauch, E., Bukatin, M., Baker, K.: A confidence-based framework for disambiguating geographic terms. In: Proceedings of the HLT-NAACL 2003 workshop on Analysis of Geogr. references, pp. 50–54. Association for Computational Linguistics, Morristown, USA (2003). DOI http://dx.doi.org/10.3115/1119394.1119402
13. Jones, C.B., Purves, R., Ruas, A., Sanderson, M., Sester, M., van Kreveld, M., Weibel, R.: Spatial information retrieval and geographical ontologies an overview of the SPIRIT project. In: Proceedings of the 25th ACM SIGIR Conference, pp. 387 – 388 (2002)
14. Jones, C.B., Abdelmoty, A.I., Fu, G.: Maintaining ontologies for geographical information retrieval on the web. In: Proceedings of On The Move to Meaningful Internet Systems 2003: ODBASE 03, *LNCS*, vol. 2888 (2003)
15. Jones, C.B., Abdelmoty, A.I., Fu, G., Vaid, S.: The SPIRIT Spatial Search Engine: Architecture, Ontologies and Spatial Indexing. In: Proceedings of the 3rd Int. Conf. on Geogr. Inform. Science, *LNCS*, vol. 3234, pp. 125 – 139 (2004)
16. Vaid, S., Jones, C.B., Joho, H., Sanderson, M.: Spatio-Textual Indexing for Geographical Search on the Web. In: Proceedings of the 9th Int. Symp. on Spatial and Temporal Databases (SSTD), *LNCS*, vol. 3633, pp. 218 – 235 (2005)
17. Fu, G., Jones, C.B., Abdelmoty, A.I.: Ontology-Based Spatial Query Expansion in Information Retrieval. In: Proceedings of In On the Move to Meaningful Internet Systems 2005: ODBASE 2005, *LNCS*, vol. 3761, pp. 1466 – 1482 (2005)
18. Zhou, Y., Xie, X., Wang, C., Gong, Y., Ma, W.Y.: Hybrid index structures for location-based web search. In: Proceedings of CIKM 05, pp. 155–162. ACM, New York, USA (2005). DOI http://doi.acm.org/10.1145/1099554.1099584
19. Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems. In: Proceedings of the 19th Int. Conf. on Scientific and Statistical Database Management (SSDBM07). IEEE Computer Society (2007). DOI http://doi.ieeecomputersociety.org/10.1109/SSDBM.2007.22
20. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition **5**(2), 199 – 220 (1993)
21. Dellis, E., Paliouras, G.: Management of Large Spatial Ontology Bases. In: Proceedings of the Workshop on Ontologies-based techniques for DataBases and Information Systems (ODBIS) of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006) (2006)
22. Gruber, T.R.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In: N. Guarino, R. Poli (eds.) Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers, Deventer, The Netherlands (1993)
23. World Wide Consortium: Owl web ontology language reference. Retrieved March 2008 from http://www.w3.org/TR/owl-ref/
24. Open GIS Consortium, Inc.: OpenGIS Web Map Service Implementation Specification. OpenGIS Project Document 01-068r3, Open GIS Consortium, Inc. (2002)
25. Apache: Lucene. Retrieved March 2008 from http://lucene.apache.org
26. National Institute of Standards and Technology (NIST): TREC Special Database 22, TREC Document Database: Disk 4. Retrieved March 2008 from http://www.nist.gov/srd/nistsd22.htm

27. Alias-i: LingPipe, Natural Language Tool. Retrieved March 2008 from http://www.alias-i.com/lingpipe/
28. Geonames: Gazetteer. Retrieved March 2008 from http://www.geonames.org
29. National Imagery and Mapping Agency (NIMA): Vector Map Level 0. Retrieved March 2008 from http://www.mapability.com
30. FWTools: Open Source GIS Binary Kit for Windows and Linux. Retrieved March 2008 from http://fwtools.maptools.org
31. Refractions Research: PostGIS. Retrieved March 2008 from http://postgis.refractions.net
32. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley (1996)
33. Gospodnetić, O., Hatcher, E.: Lucene IN ACTION. Manning (2005). ISBN: 1932394281
34. Lucene, A.: Scoring. Retrieved June 2008 from http://lucene.apache.org/java/2_2_0/scoring.html
35. Van Kreveld, M., Reinbacher, I., Arampatzis, A., Van Zwol, R.: Multi-dimensional scattered ranking methods for geographic information retrieval. Geoinformatica **9**(1), 61–84 (2005). DOI http://dx.doi.org/10.1007/s10707-004-5622-6
36. Godoy, F., Rodríguez, A.: Defining and comparing content measures of topological relations. GeoInformatica pp. 347–371 (2004)
37. Andrade, L., Silva, M.J.: Relevance Ranking for Geographic IR. In: GIR '06: Proceedings of the 2006 workshop on Geogr. Inform. Retrieval, SIGIR. ACM Press (2006)
38. Jones, C., Alani, H., Tudhope, D.: Geographical information retrieval with ontologies of place. In: COSIT'01: 3rd International Conference on Spatial Information Theory, pp. 322–335 (2001)
39. Yu, B., Cai, G.: A Query-Aware document Ranking Method for Geographic Information Retrieval. In: GIR '07: Proceedings of the 2007 workshop on Geogr. Inform. Retrieval. ACM Press (2007)
40. OSGeo: Open Layers API. Retrieved May 2008 from http://openlayers.org
41. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec. **19**(2), 322–331 (1990). DOI http://doi.acm.org/10.1145/93605.98741