On Querying Geospatial and Georeferenced Metadata Resources in G-Portal

Zehua Liu, Ee-Peng Lim, Wee-Keong Ng
Centre for Advanced Information Systems
School of Computer Engineering
Nanyang Technological University
Singapore, 639798
{aseplim, awkng, aszhliu}@ntu.edu.sg

Dion H. Goh
Division of Information Studies
School of Communication and Information
Nanyang Technological University
Singapore, 639798
ashlgoh@ntu.edu.sg

Abstract

G-Portal is a web portal system providing a range of digital library services to access geospatial and georeferenced resources on the Web. Among them are the storage and query subsystems that provide a central repository of metadata resources organized under different projects. In G-Portal, all metadata resources are represented in XML (Extensible Markup Language) and they are compliant to some resource schemas defined by their creators. The resource schemas are extended versions of a basic resource schema making it easy to accommodate all kinds of metadata resources while maintaining the portability of resource data. To support queries over the geospatial and georeferenced metadata resources, a XQuery-like query language known as $\mathcal{RQL}$ (Resource Query Language) has been designed. In this paper, we present the $\mathcal{RQL}$ language features and provide some experimental findings about the storage design and query evaluation strategies for $\mathcal{RQL}$ queries.

1. Introduction

1.1. Overview of G-Portal

Integrated access to geospatial and georeferenced resources on the Web and making them available for both classroom learning and extramural research are important challenges in digital libraries. G-Portal, a research project at the Nanyang Technological University, attempts to address these challenges by providing a web portal system that offers a flexible approach to manage and access collections of metadata resources constructed for geospatial and georeferenced web resources [7].

There are several digital library projects focusing on geospatial and georeferenced information available on the Web. Among them are Alexandria Digital Library (ADL), Alexandria Digital Earth ProtoType (ADEPT) [11, 12], Digital Library for Earth System Education (DLESE) [13], Geospatial Knowledge Representation System (GKRS) [17] and GEOREP [10]. G-Portal is quite distinct compared to these systems due to the following novel features:

- It supports a storage subsystem for metadata resources represented in XML. The resources are organized into projects and they are compliant with some resource schemas. The adoption of XML and schemas ensures that G-Portal can support a wide variety of metadata resources.
- G-Portal provides both map-based and classification-based user interfaces to access the metadata resources. The former are suitable for geospatial resources with location information while the latter caters for all kinds of resources with or without location information. The two interfaces coexist and are synchronized. That is, resources selected using the map-based interface will also be highlighted in the classification-based interface, and vice versa.
- G-Portal allows different classification hierarchies to be defined for metadata resources according to the needs of different digital library user communities. This is a stark contrast to traditional digital libraries adopting only a single classification hierarchy for cataloging data.
- G-Portal provides user annotation facilities on the metadata resources to support better knowledge sharing. By treating annotations as a type of metadata resource, the annotations can be accessed in the same way as other metadata resources [8].

As a digital library system, G-Portal has to provide efficient query processing services to its users. Compared to ADL [11] and ADEPT [12] which assume a distributed digital library architecture consisting of multiple collection

servers, G-Portal currently has a centralised storage subsystem for its metadata resources represented in XML. To query these resources, an XML-based query language is required. Furthermore, there is a need to design a suitable query subsystem to support such queries. In GEOREP [10] and DLESE [16], resources are also maintained in centralized storage systems. The two systems however use standard structured database systems (relational or object-oriented) to manage their resources. In other words, the kind of resources that can be handled by GEOREP and DLESE are highly structured and they must comply to some pre-determined database schema. Lastly, the GKRS project focuses on knowledge representation of text and multimedia information instead of metadata resources [17].

XML has been well accepted as a more flexible way to represent semistructured information. Due to its self-describing nature, XML is ideal for representing metadata resources. In G-Portal, all metadata resources are represented in XML and each resource must comply to some resource schema defined using the XML Schema language [15]. To ensure that all resources contain some essential elements required for identification and classification purposes, all resource schemas must share some common elements defined in a basic resource schema [7].

Several query languages have been proposed for XML. Among them, XQuery offers very powerful query capabilities and has been selected by the W3 Consortium XML Query Working Group for further design and standardization [2]. XQuery, nevertheless, does not provide much support for querying spatial elements in resources. It simply treats spatial elements as numerical values and not all spatial search criteria can be expressed as query predicates on these individual spatial elements. Furthermore, the existing XML database systems are not able to process spatial queries efficiently as will be shown in our experiments (see Section 6).

1.2. Objectives and Contributions

In this paper, we will examine the query facilities provided by G-Portal. To this end, we examine the unique way G-Portal represents and organizes its metadata resources, derive the user query requirements, and design a XQuery-based query language to query G-Portal resources. The new query language known as $\mathcal{RQL}$ is small and easy-to-use. While $\mathcal{RQL}$ has a strong XQuery flavor, it also introduces features that cater to integrated queries on XML and spatial information of metadata resources, project-centric organization of resources, and resource schema-compliant resources.

We will also describe some experiments conducted on integrated XML and spatial query processing on a real dataset. In the experiments, we evaluate different XML-spatial query processing strategies and determine their strengths and shortcomings. The experience in these experiments will guide us in the final design of the G-Portal storage and query processing subsystems.

In the following, we summarize the specific contributions of this paper:

- A new query language, $\mathcal{RQL}$, has been developed for G-Portal resources. This new query language incorporates the concepts of projects and schemas into the query syntax. It also supports geometry object representation and spatial query predicates required for querying geospatial information.

- We have designed the system architecture for G-Portal's storage and query subsystems. We chose the Tamino XML database system to store resources, and the Informix database system to store the spatial information of these resources. Several query evaluation strategies have been developed, some appropriate for queries involving non-spatial predicates only and others are for queries involving both spatial and non-spatial predicates.

- Experiments have been conducted to compare the storage alternatives for G-Portal resources, and their query processing strategies. The experiments show that for queries that involve spatial predicates returning small number of resources, it is better to explore the spatial indexing method provided by Informix to first obtain the resources qualifying the spatial predicates before evaluating the non-spatial predicates on the resources in the Tamino database. Otherwise, we should query both Tamino and Informix separately and obtain the query results by finding the common resources returned by the two queries.

1.3. Outline of Paper

The remaining sections of this paper are structured as follows. Section 2 gives an overview of the related research. Section 3 briefly describes the definition of resources in G-Portal. Our proposed query language $\mathcal{RQL}$ is described in Section 4. Section 5 presents some considerations of processing $\mathcal{RQL}$ queries and the architecture of the G-Portal storage and query subsystems. The experiments and results are given in Section 6. Finally, we give our conclusion in Section 7.

2. Related Work

Both XML and spatial information co-exist in G-Portal resources. We therefore require an integrated approach to

formulate queries on resources. We first examine the extensions of the relational query language, SQL, to handle spatial queries. As traditional relational databases only support simple data types, Open GIS Consortium has defined an extension of SQL data model to represent and query geometry data [9]. This extension includes a set of spatial data types and spatial predicate functions to simplify queries on spatial columns. We borrow this idea in our query language design.

The evaluation of spatial queries on relational databases has been a very active area of research. Most research focus on designing specialized indexing methods such as R-tree and its variants to support efficient spatial searches [5, 3]. R-tree and other indexing methods are now supported by several relational database products including the Informix database system used in our G-Portal implementation. However, due to the very flexible structure in XML data, spatial indexing methods are generally not available in XML database systems. Other than spatial indexing, there have been also much research work on spatial join optimization [4]. Since G-Portal only provides direct queries on resources however, spatial joins are not involved.

Our XML and spatial query processing research is also related to the SDSC and UCSD's research on representing remote spatial data sources in XML and employing XML-based query languages to integrate them [1]. In the above project, a query language XMAS was defined and the mapping from XMAS to relational queries is performed to retrieve XML-formatted results from the remote data sources. XMAS is based on XML-Query, not XQuery. Unlike G-Portal, there is no physical repository of metadata resources in XML.

3. Definition of Resources

In G-Portal, metadata resources (or simply resources) are descriptors about geospatial and georeference information on the Web. Represented in XML, they serve as searchable and browsable items in G-Portal as well as pointers to the original sources. To give the flexibility to accommodate different kinds of metadata resources, G-Portal requires every metadata resource to be an instance of some *resource schema*. Resource schemas define the internal structure of resources and are used by G-Portal to interpret the content of resources.

Consider the sample resource schema in Figure 1. It defines the resource schema for County Census resources extracted from the U.S. Census Bureau [14]. Written in the XML Schema language [15], the resource schema derives its resource elements from the *basic resource schema*, *Resource.xsd*, and defines additional elements and attributes relevant to county census resources.

The basic resource schema includes 6 basic resource elements common to all resources in G-Portal. They are *resource id*, *name*, *location*, *creator*, *source*, and *content*. Briefly, the *resource id* is used to uniquely identify a resource in G-Portal. Each resource is also given a *name*, *location* (indicating the spatial information of the resource), and *creator* information. The *location* information is represented by a 2-D geometry object. The *source* element refers to the external link to the original web resource. The *content* element refers to the remaining bulk of the resource information and can be customized for different types of metadata resources. In this example, the content element has been extended to capture the county census information. The extended content element consists of the general, people, business and geography components identified by the *GeneralQuickFacts*, *PeopleQuickFacts*, *BusinessQuickFacts*, and *GeographyQuickFacts* respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
  <xsd:redefine schemaLocation="Resource.xsd"/>
  <xsd:complexType name="ContentType">
    <xsd:sequence>
      <xsd:element name="GeneralQuickFacts" type="GeneralQuickFactsType"/>
      <xsd:element name="PeopleQuickFacts" type="PeopleQuickFactsType"/>
      <xsd:element name="BusinessQuickFacts" type="BusinessQuickFactsType"/>
      <xsd:element name="GeographyQuickFacts" type="GeographyQuickFactsType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="GeneralQuickFactsType">
    <xsd:sequence>
      <xsd:element name="State" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="PeopleQuickFactsType">
    <xsd:sequence>
      <xsd:element name="Population" type="PopulationType"/>
      <xsd:element name="HouseHold" type="HouseHoldType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="PopulationType">
    <xsd:sequence>
      <xsd:element name="Year2001Estimate" type="xsd:int"/>
      <xsd:element name="Year2000" type="xsd:int"/>
      ... ...
      <xsd:element name="Distribution" type="DistributionType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="DistributionType">
    <xsd:sequence>
      ... ...
      <xsd:element name="ByRace" type="ByRaceType"/>
    </xsd:sequence>
    <xsd:attribute name="Year" type="xsd:short" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="ByRaceType">
    <xsd:sequence>
      <xsd:element name="Category" type="CategoryType" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  ... ...
</xsd:schema>
```

Figure 1. QuickFacts County Census Resource Schema

Based on the above sample resource schema, resources can be created accordingly. Figure 2 depicts a sample census resource about the Harris County in Georgia.

In G-Portal, resources are further organized into different projects which define a logical set of resources pertinent to a specific task and digital library application. While resources with different schemas can be grouped into a single project, the grouping is logical since the same resource may be shared among different projects. This sharing mecha-

```
<?xml version="1.0" encoding="UTF-8"?>
<Resource xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="CountyQuickFacts.xsd"
         ShapeID="1">
  <ID>CountyQuickFacts_1</ID>
  <ResourceName><Name>Harris County</Name></ResourceName>
  <Location Type="Geometry">
    <Geometry>
      <NumberOfParts>1</NumberOfParts>
      <Part Type="Point">
        <NumberOfPoints>1</NumberOfPoints>
        <Point><X>-84.9060</X><Y>32.7407</Y></Point>
      </Part>
    </Geometry>
  </Location>
  <Creator> ... ... </Creator>
  <Source><Link Type="External">
     http://quickfacts.census.gov/qfd/states/13/13145.html
  </Link></Source>
  <Content>
    <GeneralQuickFacts><State>Georgia</State></GeneralQuickFacts>
    <PeopleQuickFacts>
      <Population>
        <Year2001Estimate>24548</Year2001Estimate>
        <Year2000>23695</Year2000>
        ... ...
        <Distribution Year="2000">
          ... ...
          <ByRace>
            <Category Desc="White">78.4</Category>
            <Category Desc="Black or African American"> 19.5 </Category>
            <Category Desc="American Indian and Alaska Native"> 0.4 </Category>
            <Category Desc="Asian">0.5</Category>
            <Category Desc="Native Hawaiian and Other Pacific Islander">
              0</Category>
            <Category Desc="Others">0.3</Category>
          </ByRace>
        </Distribution>
      </Population>
      ... ...
    </PeopleQuickFacts>
    ... ...
    <GeographyQuickFacts>
      ... ...
      <MetropolitanArea>Columbus, GA-AL MSA</MetropolitanArea>
    </GeographyQuickFacts>
  </Content>
</Resource>
```

Figure 2. QuickFacts County Census Resource

nism is provided to prevent replication of resources in G-Portal and to promote reusability among different projects.

4. ($\mathcal{RQL}$) - A Resource Query Language

As XML quickly becomes the choice language for representing and exchanging information on the Web, there is ongoing work to standardize the languages for querying, transforming, exchanging, and publishing XML data. Among the query languages for XML, XQuery has gained acceptance by the W3 Consortium as the XML query language standard [2]. While still evolving, XQuery has been implemented in several database systems, e.g., Tamino, Oracle, etc.. As XQuery is a powerful query language designed to query and transform XML data which are semistructured and nested, the optimized evaluation of queries in XQuery remains a research topic. Moreover, there is little support for spatial access and indexing in XQuery.

G-Portal essentially adopts an XQuery approach to querying its metadata resources represented in XML format. Although XQuery is ideal for generic XML queries, it has to be further customized due to the following salient features of the query model used in G-Portal:

- *Project and schema groupings of resources*: As G-Portal resources are grouped by both projects and resource schemas, it is important to be able to query resources under some specified project and resource schema(s).

- *Selection-based retrieval queries*: XQuery is an highly expressive language that supports selection of XML elements in an XML document, and transforming the XML document into another representation. The transformation aspect of XQuery primarily caters for information exchange. Since information exchange is not the target usage of G-Portal's resources, the XQuery syntax can be further simplified for use.

- *Designated spatial attributes in resource schemas*: Spatial location is a core element in the basic resource schema. This schema element allows us to display resources on the map-based interface and to specify queries with *spatial predicates* describing the selection criteria based on the spatial location of resources. Instead of a verbose specification of such spatial query predicates using XQuery syntax, we have chosen to augment XQuery with spatial object representation and a set of spatial predicates for specifying different kinds of spatial conditions easily.

4.1. $\mathcal{RQL}$ Query Syntax

The $\mathcal{RQL}$ query model follows the XQuery syntax closely. A query can be expressed as follows:

```
select <project>
for $<res_var> in schema((<res_sch>)+)/*/Resource
(<for_expression>|<let_expression>)*
where <filter_expression>
return $<res_var>
```

In the above query expression, the *select* clause indicates the G-Portal project to be queried. The $\langle res\_sch \rangle$ token refers to a resource schema of the project. By specifying the project and resource schema, the resources to be queried can thus be identified. The expression $\$\langle res\_var \rangle$ *schema*($(\langle res\_sch \rangle)+)/*/Resource$ iterates the resources with the specified schema under the project, and binding the resources to $\$\langle res\_var \rangle$ one at a time. The zero or more $\langle for\_expression \rangle$ and $\langle let\_expression \rangle$ allows further iteration over the child (or descendant) elements of $\$\langle res\_var \rangle$ and binding some specific child element (or descendant) of $\$\langle res\_var \rangle$ with other variables. In both the *for* and *let* expressions, additional variables could be defined to bind different portions of the resource elements. Once this is done, the *where* clause filters the unwanted resources by specifying the filtering expression on the binding variables. Finally, the *return* clause generates the resources for each of these bindings fulfilling the filtering expression.

Compared to the original XQuery syntax, the above query expression is restricted to the FLWR (pronounced as "FLO-WER") expression using the *for* expression to iterate the resources in the XML resource file. Unlike XQuery, we introduce the *select* clause to identify the project used in the query. We also do not assume that all resources are stored within a XML file (an assumption made in XQuery's syntax). Instead, we use the project and schema(s) to identify the collection of resources to be queried. Only resources associated with the project and schema will be returned by the query. We further simplify the return clause to include the different bindings of resource variables satisfying the filter expression.

4.2. Queries on Non-Spatial Elements Only

Using the example Census resource schema and assuming that CountyProj is a project that includes some county census resources, we formulate the following query examples using the modified XQuery language.

Q1: Find the census resources that are from the state of Georgia.

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where $i/Content/GeneralQuickfacts/State = "Georgia"
return $i
```

The expression *$i/Content/GeneralQuickfacts/State* in the above query refers to the path leading to the state element of each resource in the XML file using the slash ('/') as the separator between a parent element and its child. Since there is only one state element within a resource, the above query can be simplified further using the double-slash symbol ('//') which indicates stepping through multiple levels of a hierarchy in each resource:

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where $i//State = "Georgia"
return $i
```

Both existential and universal quantifiers are supported by our query language. In the following query Q2, we use an universal quantifier that is specified using the "every" keyword.

Q2: Assuming that all non-white race categories are considered minorities, find the census resources from Georgia that has all minority race categories exceeding 0.1%.

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where every $i//ByRace/Category[@Desc!="White"]>0.1
      and $i//State = "Georgia"
return $i
```

In a census resource, the race distribution is found in the *ByRace* element and the race categories are captured by the *Desc* attribute of the *Category* elements nested within the *ByRace* elements. The predicate expression *[@Desc!="White"]* evaluates to a true value if the "White" is not found in the *Desc* attribute of the *Category* element.

Q3: Find the census resources from Georgia that contain at least one minority race category equal 0%.

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where some $i//ByRace/Category[@Desc!="White"] = 0
      and $i//State = "Georgia"
return $i
```

4.3. Queries on Resources with Spatial Information

Although not every G-Portal resource carries spatial location information, the resource schema includes a *location* element should the information be available. The location element can be as simple as a point to as complex as a set of polygons. G-Portal includes a map-based user interface to both display resources on a map, and to use the map to define spatial search criteria for querying the resources.

To facilitate queries involving the *location* element, the $\mathcal{RQL}$ query syntax includes the following spatial predicates to be used within the where clause.

- equal(): The predicate returns true when the resource has a location identical to a given geometry object.
- cover(): The predicate returns true when the resource has a location containing a given geometry object, false otherwise.
- coveredby(): The predicate returns true when the resource has a location completely enclosed by a given geometry object, false otherwise.
- overlap(): The predicate returns true when the resource has a location overlapping a given geometry object, false otherwise.
- meet(): The predicate returns true when the resource has a location adjacent to a given geometry object, false otherwise.
- disjoint(): The predicate returns true when the resource has a location disjoint with a given geometry object, false otherwise.

Similar to the location element, each geometry object used in the spatial predicates can range from a point to a set of polygons. Due to space constraints, we will not elaborate how all types of location elements (or geometry objects) can be represented in our XML-based resources. Instead, we will use the point and rectangle data throughout our examples.

Q4: Find the census resources that are within a given bounding rectangle with (-95, 30) and (-70, 50) representing the latitude and longitude values of the left-bottom and right-upper corners of the rectangle respectively.

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where coveredby($i,
   Polygon((-95,30),(-95,50),(-70,50),(-70,30)))
return $i
```

Q4 uses the *Polygon* keyword to define a rectangle object using the four corner points given (in latitude and longitude). Other keywords including *Point*, *Linestring*, *Multipoint*, *Multilinestring*, *Multipolygon*, etc., are used to construct other types of geometry objects. To ease the specification of complex geometry objects in a query, it is necessary to allow end users to draw the objects using a user-friendly query client. Q4 can be further extended with other non-spatial query conditions as shown in Q5 (using the overlap spatial predicate instead).

Q5: Find the census resources that overlap the above given bounding rectangle and are from the state of "Georgia".

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where overlap($i,
   Polygon((-95,30),(-95,50),(-70,50),(-70,30)))
   and $i//State = "Georgia"
return $i
```

4.4. Mixed Schema Queries

So far, our query examples have only involved a single resource schema. As a project may include resources using different resource schemas, it is possible to formulate queries on resources that are associated with different resource schemas. However, due to the schema discrepancies, it is only possible for such queries to involve the common elements in these resources.

Q6: Assume that the project CountyProj also consists of another set of business pattern resources with another resource schema known as "BusPattern". Find the census and business pattern resources that are from the state of "Georgia".

```
select CountyProj
for $i in schema(censusDB,busPatternDB)/*/Resource
where $i//State = "Georgia"
return $i
```

The above query can be treated as two separate queries each involving a single resource schema, and the final result is the union of the two subquery results. In other words, each resource must fulfill the where condition in order to be included in the query result. This is illustrated by the following equivalent query expression.

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where $i//State = "Georgia"
return $i
union
select CountyProj
for $i in schema(busPatternDB)/*/Resource
where $i//State = "Georgia"
return $i
```

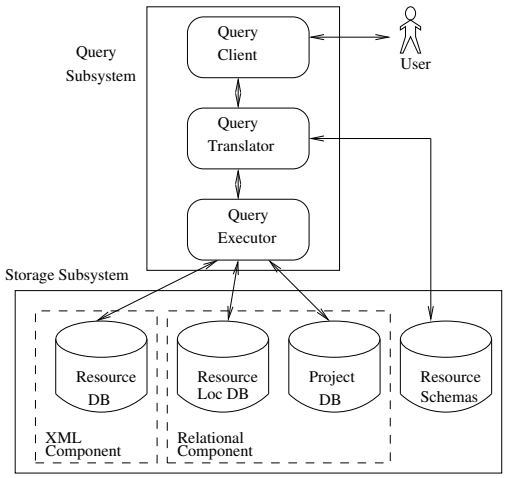

Figure 3. Architecture of the Data Storage and Query Processing Subsystems

5. $\mathcal{RQL}$ Query Processing

5.1. Storage and Query Subsystems

$\mathcal{RQL}$ allows queries on both XML and spatial elements in G-Portal resources. It therefore cannot be directly implemented on existing XML database systems that usually do not support spatial objects and spatial query predicates. Instead of implementing a new XML database system from scratch, we use the Tamino and Informix database systems as our database backends to support both XML and spatial data storage and query processing as shown in Figure 3.

As shown in the figure, G-Portal's storage subsystem consists of a relational component, an XML component, and a collection of *resource schema files*. The relational and XML components are implemented using Informix and Tamino database systems respectively. The relational storage component includes a *resource location database* and a *project database*. The resource location database stores the spatial elements of all resources together with the resource ids. The spatial elements are indexed using R-tree to support efficient evaluation of spatial query predicates. The project database stores the ids of resources belonging to each project. As mentioned in Section 3, projects are logical groupings of resources and the same resource can be shared among different projects. By maintaining the project database, any updates on the project membership of resources can be easily done. The XML storage component of G-Portal consists of a large *resource database* storing all resources (including their spatial elements). Although the spatial elements are also stored in the resource location database, keeping the spatial elements in the resource

database makes it much simpler to retrieve resources. Otherwise, we will have to obtain the spatial elements from the resource location database and merge them into the resources from the resource database.

G-Portal's query subsystem consists of a *query client* for users to easily formulate their $\mathcal{RQL}$ queries. Each $\mathcal{RQL}$ query is first translated by the *query translator* into sub-queries on the various databases. In the query translation process, a query evaluation strategy is adopted to ensure that the sub-queries can be efficiently performed on the Tamino and Informix database systems. The actual evaluation of sub-queries is handled by the *query executor* which is also responsible for combining the results from sub-queries (if necessary).

5.2. Evaluation of $\mathcal{RQL}$ Queries

In this section, we explore some query evaluation strategies that can be implemented for $\mathcal{RQL}$ queries. For simplicity, we ignore the project grouping of resources specified by the $\mathcal{RQL}$ queries and focus on the query predicates found in the *where* clause. We first broadly divide $\mathcal{RQL}$ queries into the following 3 categories:

- *Type 1- Non-spatial queries*: They involve only the non-spatial elements of resources.
- *Type 2- Spatial only queries*: They involve only the spatial elements of resources.
- *Type 3- XML and spatial queries*: They involve both the non-spatial and spatial elements of resources.

For Type-1 queries, it is quite clear that the resource location database is not required since the query predicates only apply to the elements found in the resource database. The spatial index provided by the former is also irrelevant to such queries. In this case, the **Tamino-Only** query evaluation strategy is most appropriate where the non-spatial queries are evaluated only on the resource database.

For Type-2 queries, we can apply the **Tamino-Only** strategy or the **Informix-First** strategy depending on the kind of spatial query predicates and the location data of resources involved. The former is applicable only when the spatial query predicates and location information involved are simple and can be translated into simple comparison predicates on individual parts of the location information , e.g. latitude and longitude values of points, connected by AND's and OR's. This is however not always possible for complex spatial query predicates and location which involve complex geometry objects, for example, checking if a polygon overlaps with another polygon. The key feature of the **Tamino-Only** strategy is that it uses only the XML storage component. The **Informix-First** strategy uses the spatial access methods provided by the Informix database system to first retrieve the ids of resources meeting the spatial query predicates. The ids are later used to construct a disjunction of id comparison predicates in the another query against the resource database.

For Type-3 queries, we can use the **Tamino-First** and **Naive** strategies other than the **Tamino-Only** (only for queries using simple spatial predicates) and the **Informix-First** strategies. The Tamino-First strategy allows the non-spatial query predicates to be first evaluated against the resource database. The ids of resources retrieved are later incorporated into the spatial sub-query sent to the resource location database.

The Naive strategy simply divides a Type 3 query into two sub-queries, one for the resource database and another for the resource location database. The two sub-queries are evaluated by both the Tamino and Informix database systems. Their resultant resource ids are later intersected before the final query result is obtained.

While the Tamino-Only strategy can be used in some situations, we note that this strategy is quite restrictive for Types 2 and 3 queries. In general cases, we need to employ the Informix-First or Naive strategies to handle Types 2 and 3 queries.

6. Experimental Results

We have conducted experiments in order to evaluate the storage alternatives and query evaluation strategies. There are two distinct objectives for the experiments:

- To determine which of the two resource storage alternatives is better; and
- To determine the most suitable query evaluation strategies.

The two resource storage alternatives are:

- **DB1**: storing resources only in an XML database
- **DB2**: storing resources in an XML database and resource location information in a database which supports efficient spatial access method

As described in previous sections, G-Portal adopts the Tamino database as the repository of XML resources and the Informix database for storing spatial attributes of these resources. For both **DB1** and **DB2**, the Tamino database stores the complete resource information, including the location element. The Informix database is used in **DB2** to store the resources' spatial attributes (indexed by R-Tree) together with the resource ids.

The four query evaluation strategies are: **Tamino-Only** (TO), **Informix-First** (IF), **Tamino-First** (TF), and **Naive** (NA). They have been elaborated in detail in Section 5.2. We, nevertheless, want to re-emphasize that the **Tamino-Only** strategy is only suitable for queries with simple spatial

predicates. For spatial queries involving complex polygons, the evaluation of the spatial predicates have to be performed at the application level, which is very inefficient.

6.1. Dataset

The dataset in our experiments is constructed using actual data from the U.S. Census Bureau [14]. The original Website contains figures of population, business and geometry information of all counties in the United States. The latitude and longitude of the center of each county is also provided. To convert the data from HTML Webpages into an XML format, we first derived a schema for these resources, which is partially shown in Figure 1. The Webpages containing the data were then downloaded from the Website and the relevant figures were extracted from the pages and stored in XML according to the schema. The latitude and longitude of each county were also extracted and stored in both the location element of the resources in the resource (Tamino) database and in the resource location (Informix) database.

After removing some counties with incomplete information, there were a total of 3138 resources in the dataset. Each resource contained 90 XML elements and 50 XML attributes. The maximum nesting level was 7. The values of most of the elements were numbers (integer or double); whereas attribute values were mostly short text strings. Since we only had the latitude and longitude of the center of each county, the spatial attribute of each resource was of the type Point. To allow numerical predicates (e.g., greater than, less than, etc..) to be evaluated efficiently on the XML location element, indexes on the elements of latitude and longitude were created using Tamino's internal index.

6.2. Experiments

Although we have two different objectives in our experiments, they can actually be achieved with a single set of experiments. This is because the **DB1** storage alternative can only be queried with the **Tamino-Only** strategy. Comparing the efficiency of the **Tamino-Only** strategy against the other three allows us to compare the two storage alternatives.

As discussed in Section 5.2, there are three categories of queries. For Type-1 (non-spatial) queries, all four query evaluation strategies are reduced to **Tamino-Only**. It is thus not necessary to compare strategies for Type-1 queries. In addition, for Type-2 (spatial only) queries, both **Tamino-First** and **Naive** strategies will return all resources in the XML database, due to the absence of XML constraints. Obviously, the **Tamino-Only** approach will be faster than these two. Therefore, in our experiments, we test Type-2 queries only with the **Tamino-Only** and **Informix-First** strategies and test Type-3 queries with all four strategies.

To ensure a fair comparison, we used only rectangles in our spatial predicates to allow these predicates be easily transformed into XML queries on the location elements so that no application level processing is required for the **Tamino-Only** strategy.

Five queries were used in the experiments. The first (query spatial) is a Type-2 spatial only query with the coveredby() predicate. The other four are Type-3 queries with coveredby() as the spatial predicate. coveredby() is the only meaningful spatial predicate in this case because we only have points in the resources and we had decided to use only rectangles in the queries. The first two Type-3 queries are **Q2** and **Q3** in Section 4.2 coupled with a coveredby() spatial predicate, called query every and query some respectively. These two are more restrictive and return fewer than 100 resources even if the query rectangle covers all counties. The last two queries are less restrictive and can return a much larger amount of resources. The third query, query business, has five conjunctions about the figures concerning the business quickfacts of the counties in the where clause and will return 785 resources without considering the spatial constraints. The last query, query population, also contains two conjunctions of XML predicates but can return up to 1458 resources.

Q7:(Query business)

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where coveredby($i, Polygon((x1, y1),(x1,y2),
      (x2, y2),(x2,y1)))
  and $i//RetailSales > 100000
  and $i//RetailSalesPerCapita > 4000
  and $i//FederalFundsAndGrants > 60000
  and $i//WomenOwnedFirms > 5
  and $i//MetropolitanArea ~= "*A*"
return $i
```

Q8:(Query population)

```
select CountyProj
for $i in schema(censusDB)/*/Resource
where coveredby($i, Polygon((x1, y1),(x1,y2),
     (x2, y2),(x2,y1)))
  and $i//Population/Year2000 > 10000
  and $i/ByAge/Category[@Desc="65 and Above"] > 10
return $i
```

For each query, four types of spatial query rectangles were generated: large area, median area, small area, and random area (random side length). The first three were of around 25%, 5% and 0.5% of the size of minimum bounding box of all counties respectively. The positions of the rectangles were randomly generated. For random area, both the size and the position of the rectangle were randomized. For each area size, we generated 120 rectangles at different positions and the resulting queries were evaluated based on the four strategies. The average time for performing each query and the average number of resources returned by the 120 queries were recorded. In order to obtain a statistical support for our comparisons, the time taken for different

strategies were compared using t-test statistics for testing matched pairs' mean difference [6] at 5% significance level.

The results are shown in Tables 1 to 4. Due to space constraints, only the details of the spatial query, query every and query population are shown and only the comparison of **Informix-First** against the other three strategies are presented. The column headers L, M, S, and R represent large area, median area, small area, and random area respectively. In Tables 1 to 3, the columns with (t) is the time in seconds and those with (n) are the number of resources returned. In Table 4, for each X *vs* Y entry, there is a null hypothesis of "X is as good as Y", and two alternate hypotheses of "X is better than Y" (positive) and "X is worse than Y" (negative). A Z indicates that we cannot reject the null hypothesis, while an A means the null hypothesis is rejected and the positive alternative hypothesis is accepted, whereas R means the negative one is accepted.

Table 1. Results of Spatial Only Queries

	L (t)/(n)	M (t)/(n)	S (t)/(n)	R (t)/(n)
TO	10.78/891.35	3.26/174.14	0.78/14.95	3.44/431.65
IF	8.76/891.35	1.26/174.14	0.15/14.95	2.16/431.65

Table 2. Results of query every

	L (t)/(n)	M (t)/(n)	S (t)/(n)	R (t)/(n)
TO	4.42/1.47	1.85/0.38	0.73/0.00	2.18/0.61
IF	3.55/1.47	0.56/0.38	0.11/0.00	2.00/0.61
NA	5.20/1.47	2.97/0.38	1.27/0.00	3.95/0.61
TF	5.16/1.47	5.00/0.38	4.99/0.00	5.80/0.61

Table 3. Results of query population

	L (t)/(n)	M (t)/(n)	S (t)/(n)	R (t)/(n)
TO	11.66/385.43	2.25/85.30	0.71/8.36	2.64/198.12
IF	15.98/385.43	1.50/85.30	0.11/8.36	5.50/198.12
NA	10.45/385.43	4.50/85.30	1.65/8.36	5.37/198.12
TF	15.80/385.43	13.69/85.30	12.73/8.36	13.70/198.12

6.3. Discussion

The results in Table 1 indicate that for spatial only queries, the average query processing time for **Informix-First** is consistently shorter than that of **Tamino-Only**. The advantage of **Informix-First** is reaffirmed by the statistical test results in Table 4 (third row), where the positive alternate hypothesis is always accepted. The reason for the better performance of **Informix-First** is largely due to the use of the efficient spatial access method (R-Tree) provided by Informix database. Since we can only use numerical comparisons to evaluate spatial predicates on the Tamino database, **Tamino-Only** is much slower than the dedicated spatial access method like R-Tree. From Table 1, we can see that evaluating a disjunction of around 900 resource ids in an XML query is still faster than the numerical comparisons for the coveredby() predicate.

Table 4. Comparison of Storage Alternatives and Evaluation Strategies

	L (t)	M (t)	S (t)	R (t)
IF vs TO				
spatial	A	A	A	A
every	A	A	A	Z
some	Z	A	A	Z
business	R	A	A	R
population	R	A	A	R
IF vs NA				
every	A	A	A	A
some	Z	A	A	A
business	R	A	A	Z
population	R	A	A	Z
IF vs TF				
every	A	A	A	A
some	Z	A	A	A
business	Z	A	A	A
population	Z	A	A	A

In some extreme cases where the sub-query to Informix returns 0 records, there is not even a need to query the Tamino database. This happened quite frequently when the queries involved small areas, making **Informix-First** an order of magnitude faster than **Tamino-Only** when querying with small areas. However, we did observe that when the number of resources returned exceeds 1500, **Tamino-Only** started to outperform **Informix-First**. Nevertheless, with 3138 resources in the entire dataset, it is unlikely that there will be many queries returning more than half of the dataset.

Table 2 shows the results of different strategies for query every. Due to the very restrictive XML query conditions, the number of resources returned by this query is very small (at most 7 for large areas). This means that when a small number of resources are returned, ids disjunctions (averaged 891 for large areas) plus the non-spatial conditions will be evaluated faster than the evaluation of numerical comparisons (in place of spatial predicates) and the non-spatial predicates on the Tamino database.

For query population (Table 3), we can see that in the case of large and random area spatial queries, **Tamino-Only** and **Naive** are faster than **Informix-First**. The statistical test results in Table 4 also ascertains that **Informix-First** performance worse in the two cases. However, for smaller size query rectangles, **Informix-First** is still better than the other three. The performance of **Informix-First** for query some and query business (not shown here) are somewhere between that of query every and query population. We noted that the number of results returned increased from query every to query population. This suggests that the smaller the size of the result, the faster **Informix-First** is compared with the rest.

It can be seen from the Table 4 that most of the time we are confident that the **Informix-First** strategy performs better than the **Tamino-Only** strategry. In other words, we are quite confident that the **DB2** alternative is more efficient than **DB1**, if the **Informix-First** strategy is used to evaluate queries with **DB2**. Since points are the simplest forms of geometry objects, it is obvious that when lines and polygons are involved in the spatial predicates, **DB1** will be slower or even not applicable. Therefore, we conclude that **DB2** is the most suitable choice for implementing the storage subsystem.

When comparing **Informix-First** with **Tamino-First**, Table 4 shows that in most cases **Informix-First** performs statistically better, with only two draws. Therefore, we can also conclude that we should always prefer **Informix-First** to **Tamino-First**.

However, when comparing **Informix-First** with **Tamino-Only** and **Naive**, such conclusions cannot be drawn so easily. We notice that **Tamino-Only** and **Naive** outperformed **Informix-First** in some queries involving large areas. This happened when the number of resources returned by the non-spatial sub-query is large. In particular, we observed that **Informix-First** is more sensitive to this number than the other strategies. When the non-spatial sub-query to Tamino returns more than 700 resources, **Informix-First** starts to slow down. This occurred for queries involving large and random areas which return more than 25% of the records. The reason remains to be investigated mainly due to insufficient knowledge about the internal query processing methods used by Tamino.

Another observation of slower performance of **Informix-First** occurs when the spatial sub-query returns large numbers of records. This causes the sub-query to Tamino to contain a large number of disjunctions of ids to be tested. The reason is also not clear since it is not known to us how disjunctions are evaluated in Tamino. One possible explanation is that such long disjunctions forces the internal query processing engine of Tamino to perform a large number of random database accesses, which is relatively slower than accessing by range index (**Tamino-Only**) or simple non-spatial query (**Naive**).

Nevertheless, we anticipate that most queries will unlikely cover large areas, thus only returning resources fewer than those returned by queries with median areas. The queries would also be relatively restrictive as compared with the ones in query population and query business. For queries with small to median areas, **Informix-First** consistently performs better than the other three strategies. Therefore, we consider it reasonable to adopt the **DB2** alternative and the **Informix-First** strategy.

7. Conclusion

In this paper, we describe the storage design and query features of G-Portal, a digital library system. As G-Portal maintains metadata resources about geospatial and georeferenced information on the Web, it uses different resource schemas for interpreting different types of metadata resources, and supports queries on resources carrying both spatial and non-spatial elements. To handle the geometry object representation of location elements of resources, we have proposed a new resource query language $\mathcal{RQL}$, a variant of XQuery language adapted to G-Portal's resource model and spatial query requirements.

This paper also describes the design of G-Portal's storage and query subsystems and presents our experiments on several proposed query evaluation strategies for $\mathcal{RQL}$ queries. The experiment results suggest that other than spatial-only queries, the Informix-First strategy works well for queries that consist of predicates that are more restrictive. Note that the results were obtained for relatively simple types of spatial predicates. For queries involving complex spatial predicates, the Informix-First and Naive strategies will be more appropriate.

At this point in time, we have implemented the G-Portal storage subsystem. A preliminary version of the query subsystem including the query client has also been implemented in Java. Users are given a query window to formulate the spatial and non-spatial query predicates. So far, only bounding rectangle objects can be specified using the map interface as part of the spatial predicates. The query results are displayed both on the map-based and classification-based interfaces.

As part of our future work, we plan to conduct further experiments on the storage and query subsystems for larger datasets and more complex queries. A hybrid query evaluation strategy could be explored to evaluate query predicates of different selectivities. For sophisticated users, the current $\mathcal{RQL}$ syntax may be too restrictive. For example, it is currently not possible to relate resources from different schemas in a $\mathcal{RQL}$ query. Such queries however may be useful when detailed analysis is to be conducted on the resources within a project. We will look into extending $\mathcal{RQL}$ to support such queries and develop new query evaluation strategies for them.

8. Acknowledgments

This work is funded by the SingAREN Project M48020004.

We would also like to thank Linda Hill for her advice during this research.

References

[1] C. Baru, V. Chu, A. Gupta, B. Ludaescher, R. Marciano, Y. Papakonstantinou, and P. Velikhov. Xml-based information mediation for digital libraries. In *Proceedings of the Fourth ACM conference on Digital Libraries (ACMDL 1999)*, pages 214–215, Berkeley, CA, USA, August 1999.

[2] D. Chamberlin. XQuery: An XML Query Language. *IBM Systems Journal*, 41(4):597–615, 2002.

[3] V. Gaede and O. Gunther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[4] O. Gunther. Efficient computation of spatial joins. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 50–59, Vienna, Austria, April 1993.

[5] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference*, pages 47–57, Boston, MA, June 1984.

[6] G. Keller, B. Warrack, and H. Bartel. *Statistics for Management and Economics*, pages 353–358. Duxbury Press, 1994.

[7] E.-P. Lim, D. H.-L. Goh, Z. Liu, W.-K. Ng, C. S.-G. Khoo, and S. E. Higgins. G-portal: A map-based digital library for distributed geospatial and georeferenced resources. In *Proceedings of the Second ACM+IEEE Joint Conference on Digital Libraries (JCDL 2002)*, Portland, Oregon, USA, July 14-18 2002.

[8] Z. Liu, E.-P. Lim, and D. H.-L. Goh. Resource annotation framework in a georeferenced and geospatial digital library. In *Proceedings of the 5th International Conference On Asian Digital Libraries (ICADL 2002)*, Singapore, December 11-14 2002.

[9] Open GIS Consortium. Open GIS Simple Features Specification for SQL. Revision 1.1. http://www.opengis.org/techno/specs/99-049.pdf.

[10] M.-J. Proulx, Y. Bédard, F. Létourneau, and C. Martel. GEOREP: A WWW customizable georeferenced digital library for spatial data. D-Lib Magazine, December 1996.

[11] T. Smith. A digital library for geographically referenced materials. *IEEE Computer*, 29(5):54–60, 1996.

[12] T. Smith, G. Janee, J. Frew, and A. Coleman. The Alexandria Digital Earth ProtoType system. In *Proceedings of the First ACM+IEEE Joint Conference on Digital Libraries (JCDL 2001)*, pages 118–119, Roanoke, VA, USA, June 2001.

[13] T. Sumner and M. Dawe. Looking at digital library usability from a reuse perspective. In *Proceedings of the First ACM+IEEE Joint Conference on Digital Libraries (JCDL 2001)*, pages 416–425, Roanoke, VA, USA, June 2001.

[14] U.S. Census Bureau. State and County QuickFacts. http://quickfacts.census.gov/qfd/index.html.

[15] W3 Consortium. XML Schema. http://www.w3.org/xml/schema.

[16] J. Weatherley and T. Weingart. Designing a simple resource search user interface for DLESE. http://www.dlese.org/documents/bibliographies/ discovery_sys_final.pdf, March 25 2002.

[17] B. Zhu, M. Ramsey, H. Chen, R. Hauck, T. Ng, and B. Schatz. Create a large-scale digital library for georeferenced information. In *Proceedings of the Fourth ACM Conference on Digital Libraries (DL 1999)*, Berkeley, California, USA, Auguest 1999.