QUERYING GEOGRAPHIC DATA IN XML VIA EXTENSIBLE XML QUERY LANGUAGE X^2QL

Norihide Shinagawa[†], Takaaki Nagai[‡], Hiroyuki Kitagawa[§], and Yoshiharu Ishikawa[§]

[†] Center for Environmental Remote Sensing, Chiba University
1-33 Yayoi-cho, Inage-ku, Chiba 263-8522, Japan
Tel/Fax: +81-43-290-3847
mailto:siena@cr.chiba-u.ac.jp

[‡] Graduate School in System and Information Engineering, University of Tsukuba
[§] Institute of Information Sciences and Electronics, University of Tsukuba
1-1-1 Tennohdai, Tsukuba, Ibaraki 305-8573, Japan
Tel/Fax: +81-298-53-5051
mailto:{tnagai,kitagawa,ishikawa}@kde.is.tsukuba.ac.jp

KEYWORDS : G-XML, XML, Query Language, X^2QL, Spatial Operation

ABSTRACT

XML has attracted a great deal of attention as standard data exchange format, and XML representing geographic information such as G-XML has been developed. In near future, geographic data written in XML will be exchanged through the Internet. Therefore, it will become a very important issue to efficiently query geographic data in XML. To query geographic data in XML, spatial operations such as distance calculation and spatial containment test need be provided in query languages. However, in general, XML query languages do not support such spatial operations. This paper illustrates how G-XML data can be queried via eXtensible XML Query Language X^2QL, which has been developed by our research group. X^2QL features inclusion of user-defined foreign functions to introduce application-oriented processing capability. Thus, we can realize various spatial operations in X^2QL as appropriate foreign functions. This paper also describes the development of a prototype X^2QL query processing system.

1. INTRODUCTION

In these days, geographic information has been used widely for a variety of applications, and many kinds of GIS (Geographic Information System) have been developed. XML has attracted a great deal of attention as standard data exchange format, and it has been investigated to use XML for representation of geographic data; G-XML is one of them [DPC]. In near future, geographic data written in XML will be exchanged through the Internet for the purpose of data integration and application development. Therefore, it will become a very important issue to efficiently query geographic data in XML which may be stored temporarily or persistently. To declaratively specify queries on XML data, several query languages have been proposed. For querying geographic data in XML, the query language has to support spatial operations such as distance calculation and spatial containment test. Although several query languages have been proposed so far, they do not support spatial operations.

In this paper, we study querying G-XML data via eXtensible XML Query Language X^2QL, which has been developed by our research group [Shinagawa, 2000] [Shinagawa, 2001]. X^2QL features inclusion of user-defined foreign functions to extend contents processing facilities. This extensibility enables us to introduce various spatial operations into the query facility. This paper explains foreign functions designed for G-XML geographic data processing and shows how queries for G-XML data are realized.

The rest of this paper is organized as follows. Section 2 explains G-XML and gives a sample G-XML instance. Section 3 shows some query examples for the G-XML instance. Section 4 explains X^2QL and specifies the example queries in X^2QL. Section 5 describes the development of a prototype X^2QL query processing system. Section 6 mentions related work. Finally, Section 7 summarizes this paper.

2. G-XML

G-XML is an XML application for GIS data representation. G-XML instances consist of substructures such as *geometric spaces*, *topological geospaces*, *points of interest*, *movers*, *routes*, *pictures*, and *rendering styles*. A geometric space substructure and a topological geospace substructure model objects within some boundary in Euclidean metric space, and in topological space based on associations of nodes and edges, respectively. A point of interest substructure and a mover substructure represent a point and a moving object in geographical space, respectively. A route substructure is given as an ordered list of route nodes and nested sub-routes. A picture substructure represents geometric space as an image. A rendering style substructure specifies how applications should render the G-XML data. A G-XML instance can also have its *meta data*.

This paper illustrates how X^2QL queries are specified for G-XML data taking a geometric space substructure as an example. Of course, the essential idea is applicable other G-XML substructures. A geometric space substructure consists of *geographic features*, *points of interest*, *movers*, *routes*, and *pictures*. A geographic feature represents an object in the real space which can be associated with the geometric space. Each object is described as a shape such as a point, a list of line segments, a polygon, an abstracted symbol, a comment string, or a picture.

Figure 1 shows sample G-XML data used in this paper. Its rectangle boundary is given by the `Boundary` element. It includes the `GeographicFeature` elements representing geographic features such as University of Tsukuba, Tsukuba Public Library, and Amakubo area in Tsukuba city. Their shapes are given as `Polygon` elements, whose `Coordinates` elements give sequences of coordinates. `GeographicFeature` elements can have `Property` elements to describe properties associated with them.

3. QUERY EXAMPLES

This section shows example queries. Queries for XML data generally involve both the data structure manipulation and contents processing. The data structure manipulation includes data extraction, selection, grouping, and more complex restructuring. Contents processing usually depends on application domains and requirements. In our context, operations for lengths, volumes, boundaries and other spatial features of geometric shapes are required. The followings are example queries for the G-XML data in Figure 1.

Query 1 Select the geometric feature which represents University of Tsukuba. This requirement is simply accomplished by selecting elements by their attribute and content values.

Query 2 Generate a new XML instance by extracting elements from the G-XML data and restructuring them; the generated instance should contain descriptions of buildings, and they should be grouped by the building type. Each building description should contain a pair of the building name and the associated URL.

Query 3 Calculate the distance between University of Tsukuba and Tsukuba Public Library. Geographic features corresponding to University of Tsukuba and Tsukuba Public Library can be extracted as in Queries 1 and 2, and the distance can be calculated from the coordinates of their shapes. This calculation can be realized by a foreign function to calculate the distance between given `Polygon` elements.

Query 4 Find all the shops in Amakubo area in Tsukuba city that are located within one kilometer from University of Tsukuba. This query requires the selection based on the distance calculaton and the spatial containment relationship.

```
<!--Sample data (G-XML 2.0)-->
<?xml version="1.0"?>
<!DOCTYPE G-XML SYSTEM "G-XML20.dtd">

<G-XML>

  <-- Explanation of this G-XML data -->
  <Metadata>
    <Metadata.gxml>
      <Name>Tsukuba City House Map</Name>
      <Purpose/>
      <Author>N. Shinagawa</Author>
      <At time="1997" />
      <Description>
        1997 Tsukuba City House Map
      </Description>
    </Metadata.gxml>
  </Metadata>

  <MetricGeospace id ="" category="house map"
      spatialreferencesystem="30169"
      unit.location="mm" unit.length="mm">

    <-- Specification of the boudary ractangle
        of this metric geospace -->
    <Boundary>
      <Rectangle>
        <Coordinates>
          16800000,14800000
          16200000,15600000 ...
        </Coordinates>
      </Rectangle>
    </Boundary>

    <!-- Description of University of Tsukuba
         including its name, type, URL, and
         geometrical shape -->
    <GeometricFeature
        id="" category="building">
      <Property propertyname="name">
          University of Tsukuba</Property>
      <Property propertyname="type">
          school</Property>
      <Property propertyname="url">
          http://www.tsukuba.ac.jp</Property>
      <Geometry>
        <Polygon>
           <OuterBoundary>
            <LinearRing>
              <Coordinates>
                15250000,16450000
                15340000,16430000 ...,...
              </Coordinates>
            </LinearRing>
          </OuterBoundary>
        </Polygon>
      </Geometry>
    </GeometricFeature>

    <!-- Desctiptions of Tsukuba Public
         Library and Amakubo area are given
         as like as University of Tsukuba -->

    ...

  </MetricGeospace>
</G-XML>
```

Figure 1: Example of G-XML data.

Query 5 For each school, find the three nearest shops from the school, and sort them by the distance. X^{2}QL supports ranking and sorting as advanced features. Foreign functions can be used to decide the ranking and sorting key values.

Query 6 Count the number of shops that are located within one kilometer from University of Tsukuba. X^{2}QL features user-defined aggregation functions, which are advanced types of foreign functions, so that this query requirement is also realized.

Queries 1 and 2 are examples of data structure manipulation, and they are specified in Subsection 4.1. Queries 3 and 4 are examples of contents processing via foreign functions, and they are specified in Subsection 4.2. Queries 5 and 6 are advanced examples, and they are specified in Subsection 4.3.

4. QUERYING G-XML DATA VIA X^{2}QL

4.1 Basic Queries

X^{2}QL [Shinagawa, 2000] is an extensible XML query language developed by our research group. Its query syntax is based on XML-QL, which is the most well-known XML query language, as shown below.

```
where      patterns in source
          [,patterns in source]* [,predicate]*
[group-by grouping-keys]
[rank-by   ranking-keys   top number]
[order-by ordering-keys [descending]]
construct each-output-structure
```

`Where` clause binds variables according to specified conditions. Each variable is bound to either an element, the content of an element, an element name, an attribute name, or an attribute value. The expression `element_as $e` binds the variable `$e` to the preceding element, and the expression `content_as $c` binds `$c` to the content of the preceding element. For convenience, we call each set of variable bindings a *tuple*. `Construct` clause specifies the structure of the output constructed from each tuple. Optional `group-by`, `rank-by` and `order-by` clauses are explained in Subsection 4.3.

For instance, Query 1 can be specified as shown in Figure 2. This query specification binds the variable `$univ` to each `GeometricFeature` element whose `category` attribute value is `building`. In addition, the `GeometricFeature` element must have a `Property` element whose `propertyname` value is `name` and content is `University of Tsukuba`. Its result is the `GeometricFeature` element.

More complex restructuring queries can be expressed. When `ID` is specified in a tag in a `construct` clause, output elements having the same `ID` values are merged into one element. `ID` values are usually given by Skolem functions. Note that the outermost `where` clause can be omitted, and `construct` clauses can have nested sub-queries.

Query 2 can be specified as shown in Figure 3. A `result` root element containing the result of the sub-query is constructed. The sub-query binds `$name`, `$type` and `$url` to the contents of `Property` elements whose `propertyname` values are `name`, `type` and `url`, respectively. Then, `building` elements are generated for individual tuples, and they are grouped into `typeGroup` elements by `$type` values.

```
where
    <G-XML>
      <MetricGeospace>
        <GeometricFeature
            category="building">
          <Property propertyname="name">
              "University of Tsukuba" </>
        </> element_as $univ
      </>
    </> in "buildings.gxml"
construct
    $univ
```

Figure 2: Specification of Query 1.

```
construct
    <result>
  where
      <G-XML>
        <MetricGeospace>
          <GeometricFeature
              category="building">
            <Property propertyname="name">
                $name </>
            <Property propertyname="type">
                $type </>
            <Property propertyname="url">
                $url  </>
          </>
        </>
      </> in "buildings.gxml"
  construct
      <typeGroup ID=TID($type) type=$type>
        <building>
          <name> $name </> <url> $url </>
        </>
      </>
    </>
```

Figure 3: Specification of Query 2.

4.2 Queries using Foreign Functions

Foreign functions in X^{2}QL are classified into *general functions*, which are normal-style functions, and *element methods*, which are associated with specific element types. To use foreign functions, their definitions and implementations need to be given. Foreign function implementations are given as external programs. In the current version of X^{2}QL, we assume that they are written in Java. Foreign function definitions describe the signatures of foreign functions and the URLs of their implementations. Definitions are written in the following syntax.

```
function    type-name
            general-function-name ( argument-list )
defined-by "URL of the implementation"
```

Table 1: Examples of element methods of `Polygon`.

elemeht method	retuen type	function
`toLineString()`	`LineString`	decompose to a list of line segments
`length()`	`number`	calculate the length of the circumference
`center()`	`Point`	calculate the center
`area()`	`number`	calculate the area
`intersect(Polygon)`	`Polygon`	create a new `Polygon` which is the intersection with the given `Polygon`
`distance(`*object*`)`	`number`	calculate the distance from the given *object*
`overlapArea(`*object*`)`	`number`	calculate the area of the intersection with the given *object*
`contain(`*object*`)`	`boolean`	judge containment relationship with the given *object*

(*object* can be a geometic shape of any type.)

```
function     type-name   element-type-name
             .element-method-name( argument-list )
defined-by   "URL of the implementation"

argument-list  ::=  type-name   argument-name
                    [, type-name   argument-name ]*
```

Foreign functions can be overloaded and are invoked according to their argument types. In particular, element methods are selected from foreign functions associated with their target elements. Therefore, foreign function definitions specify the data types of arguments and return values. The data types `string`, `number`, `boolean`, `content`, `element` and any element types are allowed. Each element type is treated as a subtype of `element`.

Our query examples need to calculate the distance between geographic features, and to judge wheather an area contains a specified geographic feature. In the G-XML data, the geographic shapes of University of Tsukuba, Tsukuba Public Library and Amakkubo area are expressed as polygons, so that we can define such functions as element methods associated with `Polygon` element type as shown below. Other examples of element methods of `Polygon` are shown in Table 1.

```
function    number
            Polygon.distance(Polygon other)
defined-by  "http://host/path/
                    gxml.Polygon#distance"

function    boolean
            Polygon.contain(Polygon other)
defined-by  "http://host/path/
                    gxml.Polygon#contain"
```

By using these foreign functions, Queries 3 and 4 can be specified as shown in Figures 4 and 5, respectively. Query 3 calls `distance()` in the `construct` clause. Query 4 uses `distance()` and `contain()` in the inner `where` clause to select `GeometricFeature` elements representing shops.

In Java binding, types are mapped into Java interfaces extending `XObject` interface as shown in Table 2. In the implementation of an element method, the element is passed as the first argument `self` in the argument list. Figure 6 shows the example of the implementation of `distance()`. This implementation is invoked with `self` and `other` which are `Polygon` elements. First, the coordinates are extracted from the content of the `Coordinates` element of `self`. The coordinates of `other` are also extracted in the same way. Finally, the distance between `self` and `other` is calculated based on the extracted data.

Table 2: Type mapping in Java binding.

X^2QL	Java interface	DOM super interface
`boolean`	`XBoolean`	
`number`	`XNumber`	
`string`	`XString`	
`content`	`XContent`	`DocumentFragment`
`element`	`XElement`	`Element`
each element type	`XElement`	`Element`

4.3 Queries Using Advanced Features

This subsection illustrates advanced features of X^2QL. X^2QL have optional clauses: `rank-by` and `order-by` clauses. When an `order-by` clause is specified, all output elements are sorted by the given *ordering-keys*. A `rank-by` clause is used to specify the top *number* tuples. Selection is based on the value of the *ranking-keys*. This selection is performed before the `construct` clause is processed. The order of selected tuples is not affected by this clause; the order of output elements is based on the order in the input.

Using these clauses, Query 5 is specified as shown in Figure 7. In this query specification, the three nearest shops are selected for each school by the `rank-by` clause, then output `shop` elements are sorted by the `order-by` clause. This specification is redandant. Note that X^2QL has the tuple filtering mechanism via user-defined filtering functions, which is the generalized mechanism of ranking and sorting. By using a filtering function having both the ranking and sorting capability, such redandancy can be removed. Futher explanation on the mechanism is omitted from this paper.

```
where     <G-XML>
            <MetricGeospace>
              <GeometricFeature category="building">
                <Property propertyname="name"> "University of Tsukuba" </>
                <Geometry> <Polygon/> element_as $univ </>
              </>
              <GeometricFeature category="building">
                <Property propertyname="name"> "Tsukuba Public Library" </>
                <Geometry> <Polygon/> element_as $lib </>
              </>
            </>
          </> in "buildings.gxml"
construct <result> $univ.distance( $lib ) </>
```

Figure 4: Specification of Query 3.

```
where     <G-XML>
            <MetricGeospace>
              <GeometricFeature category="building">
                <Property propertyname="name"> "University of Tsukuba" </>
                <Geometry> <Polygon/> element_as $univ </>
              </>
              <GeometricFeature category="area">
                <Property propertyname="name"> "Amakubo" </>
                <Geometry> <Polygon/> element_as $area </>
              </>
            </> content_as $space
          </> in "buildings.gxml"
construct <result>
  where     <GeometricFeature category="building">
              <Property propertyname="type"> "shop" </>
              <Geometry> <Polygon/> element_as $shop </>
            </> element_as $feature in $space,
            $area.contain( $shop ),
            $univ.distance( $shop ) < 1000000
  construct $feature
          </>
```

Figure 5: Specification of Query 4.

```
package gxml;
import jp.ac.tsukuba.is.kde.x2ql.object.*;
import org.w3c.dom.*;

public class Polygon {
  public XNumber distance( XElement self, XElement other ) {
    String text = self.getFirstChild().getFirstChild().getFirstChild().getFirstChild();
           // Corresponding XPath: OuterBoundary[0]/LinearRing[0]/Coordinates[0]/text()
    StringTokenizer list = new StringTokenizer( text.getNodeValue() );
    Array points = new Array();
    while( list.hasMoreToken() ) {
      String pos = coords.nextToken();
      double[] point = ... // Split pos by ',' into double[ 2 ].;
      points.add( point );
      }
    ... // Extract the coordinate data from other in the same way.
    XNumber result = ... // Calculate the distance using points.;
    return result;
    }
  }
```

Figure 6: Implementation of `distance()`.

User-defined aggregation functions are also supported. Tuples can be grouped by the `group-by` clause, and aggregation functions calculate a value from each group of tuples. This paper explains only their definitions and an example. Their definitions are given by the following syntax, which is similar to the syntax for general functions.

```
function    type-name
            aggregation-name ( argument-list )
defined-by  "URL of the implementation"
```

Query 6 uses an aggregation fuction to count shops in each group. The definition is geven as follows, and the query can be specified as shown in Figure 8.

```
construct <result>
  where     <G-XML>
              <MetricGeospace>
                <GeometricFeature category="building">
                  <Property propertyname="type"> "school" </>
                  <Property propertyname="name"> $schlname </>
                  <Geometry> <Polygon/> element_as $school </>
                </>
              </> content_as $space
            </> in "buildings.gxml"
  construct <shoplist name=$schlname>
    where     <GeometricFeature category="building">
                <Property propertyname="type"> "shop" </>
                <Property propertyname="name"> $shopname </>
                <Geometry> <Polygon/> element_as $shop </>
              </> in $space
    rank-by   $school.distance($shop)  top 3
    order-by  $school.distance($shop)
    construct <shop> $shopname </>
            </>
          </>
```

Figure 7: Specification of Query 5.

```
where     <G-XML>
            <MetricGeospace>
              <GeometricFeature category="building">
                <Property propertyname="name"> "University of Tsukuba" </>
                <Geometry> <Polygon/> element_as $univ </>
              </>
            </> content_as $space
          </> in "buildings.gxml"
construct <result>
  where     <GeometricFeature category="building">
              <Property propertyname="type"> "shop" </>
              <Geometry> <Polygon/> element_as $poly </>
            </> element_as $shop in $space,
            $univ.distance( $poly ) < 1000000
  construct <number> count( $shop ) </>
          </>
```

Figure 8: Specification of Query 6.

```
aggregation number count(Element item)
defined-by  "http://host/path/
                      gxml.Counter#count"
```

When group-by clause is omitted, the aggregation function is applied to all tuples. The sub-query generates therefore one number element containing the number of shops within one kilometer from University of Tsukuba.

5. QUERY PROCESSING SYSTEM

We developed a query processor that translates X^2QL queries into XSLT template rules and processes them using an XSLT processor [Shinagawa, 2000]. It provides a basic query processing facility. We have been developing a new version of an X^2QL query processing system which can work with XML storages and process queries more efficiently [Shinagwa, 2001]. This section explains its architecture.

5.1 System Architecture

Figure 9 shows the system architecture. This system consists of *Processing Manager*, *Query Parser*, *Query Planner*, *Storage Manager*, *XML Storages*, *Query Evaluator* and *Foreign Function Invoker*. We can query G-XML data managed by XML servers, stored simply in file systems, and provided by WWW-based GIS services. The current prototype can work with Tamino [Tamino], which is an XML server that supports XQL.

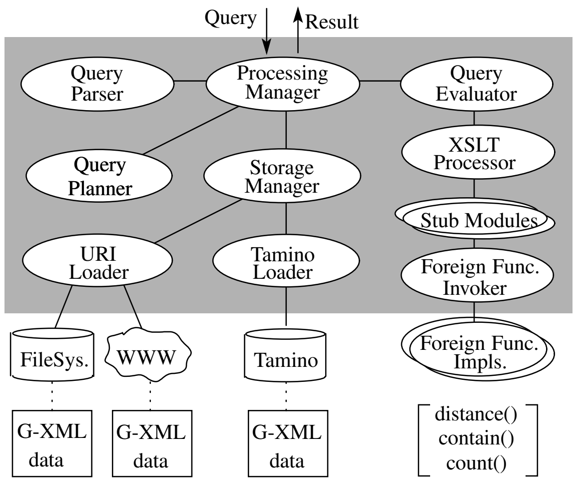

Figure 9: System archtecture.

The Processing Manager interacts with outer environments to get queries and return the results. It provides a simple UI for users and APIs for application programs. This module processes given queries via other modules.

The Query Parser parses a given query and translates it into its *Query Information*, which is internal representation.

The Query Planner makes a *Processing Plan* for a given item of Query Information. It also extracts subconditions of the given query which can be processed by individual XML strages, then generates *filtering queries* to access XML instances in the XML storages. For example, filtering queries for Tamino are expressed in XQL.

The Storage Manager fetches XML instances from XML storages with the corresponding filtering queries, file paths or URLs generated by the Query Planner.

The Query Evaluator processes the core part of the query using an XSLT processor. It is based on the previous version of the X^{2}QL processor [Shinagawa, 2000]. Foreign function calls are processed by the Foreign Function Invoker. X^{2}QL foreign function implementations are wrapped via Stub Modules, which makes them look like XSLT extension function implementations. Stub Modules are automatically generated by the Query Translator submodule.

The Foreign Function Invoker checks the types of arguments and return values, and invokes foreign functions.

5.2 Query Processing Flow

The Processing Manager processes a given query as follows.

1. Gets the Query Information of the query from the Query Parser.
2. Gets the Query Plan for the Query Information from the Query Planner.
3. Gets XML instances from the Storage Manager based on the Query Plan.
4. Processes the remaining core part of the query for the XML instances with the Query Evaluator.
5. Returns the query result.

5.3 Query Execution Snapshot

Figure 10 shows the querying result of Query 4 in the prototype system. The result is displayed with a G-XML viewer. Note that the result of Query 4 is not a valid G-XML instance. We have therefore actually executed a slightly modified query based on Query 4 to out a valid G-XML instance.

6. RELATED WORK

GML [Cox, 2001] developed by Open GIS Consortium is an XML application to describe geographic data. Database Promotion Center developing G-XML and Open GIS Consortium are building the harmonization / interoperability between G-XML and GML. This paper shows queries for G-XML data, but our approach is also applicable to GML

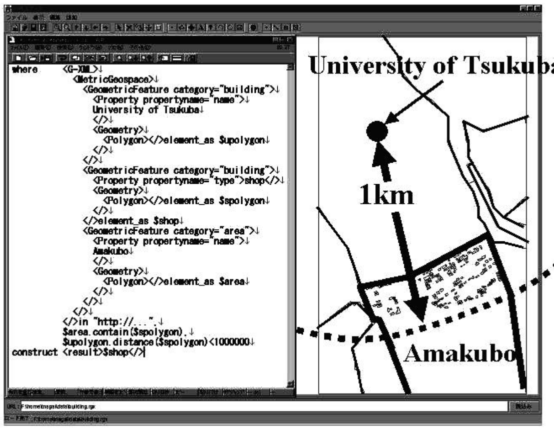

Figure 10: Execution of Query 4 in the prototype.

beacuse the idea of foreign functions does not depend on specific XML applications.

Several query languages for XML instances have been proposed. Most of them have no enough extensibility to include processing facilities depending on application domains such as geographic data. To specify queries for geographic XML data, generality as XML query languages and speciality for the application domain are both important.

Quilt [Chamberlin, 2000] is also an extensible XML query language featuring user-defined functions, so that it may appropriately specify queries for geographic data. However, details of the extensibility are not yet clear, and X^{2}QL provides more flexible extensibility such as element methods and filtering functions. Note that Quilt was proposed after we started to develop X^{2}QL, and that XQuery [DeRose, 2001], based on Quilt, has been proposed to W3C.

XML-QL [Deutsch, 1999] is a well-known XML query language used as a basis of X^{2}QL. It has powerful data structure manipulation capability. XPath [Clark, 1999a] and XQL [Robie, 1998] are simple XML path languages, so that they are widely used to express paths in more complex XML query languages, and are used as basic query languages in some XML servers. All of them have no extensibility. Thus, they can not process geographic data satisfactorily.

XSLT [Clark, 1999b] is an XML transformation language featuring user-defined extension functions, but the codes tend to be low-level and procedural. The declarativeness is desirable to specify easily manipulations for geographic XML data. The X^{2}QL query processor was developed using an XSLT processor.

YATL [Cluet, 1998], StruQL [Fernandez, 1998], Lorel [Abiteboul,1997] and UnQL [Buneman, 1996] are query languages for semistructured data, and XML data are their potential targets. Although only YATL features extensibility, its user-defined functions are not enough to process geographic data.

There are some efforts of extension of RDBMS. INGRES [Stonebraker, 1986] is known as the first RDBMS supporting abstract data types (ADTs), and an ADT for geometric data with spatial operations can be defined. Gral [Güeting] is also an extended RDBMS for geometric applications, and it has the general extensibility based on many-sorted algebra. These approaches make the relational data model extensible, and geographic data can be expressed by user-defined types. However, their databases are too rigid for geographic XML data. Moreover, user-defined types should be well-defined for geometric data types, and their internal structures should be hidden. These characteristics do not necessarily coincide with the handiness of user-defined element types and the clarity of XML data structures.

7. CONCLUSION

In this paper, we have shown how to query G-XML data via X^2QL. To realize the spatial operations need to query data with geographic properties, we have employed foreign functions supported in X^2QL. We have presented how G-XML data can be queried in X^2QL, and shown implementation of foreign functions. Development of a prototype X^2QL query processing system has also been outlined. Future research issues include the evaluation of usages of foreign functions in processing G-XML data and efficient query processing schemes.

ACKNOWLEDGMENT

This research was supported in part by the Grant-in-Aid for Scientific Research from the Ministry of Education, Sports, Culture, Science and Technology, Japan.

REFERENCES

[Abiteboul,1997] Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J., 1997. The Lorel Query Language for Semistructured Data. International Journal on Digital Libraries, 1 (1), pp. 68–88.

[Buneman, 1996] Buneman, P., Davidson, S. B., Hillebrand, G. G., and Suciu, D, 1996. A Query Language and Optimization Techniques for Unstructured Data. Proc. ACM-SIGMOD International Conference on Management of Data '96, pp. 506–516.

[Chamberlin, 2000] Chamberlin, D. D., Robie, J., and Florescu, D., 2000. Quilt: An XML Query Language for Heterogeneous Data Sources. Proc. Third International Workshop on the Web and Databases (WebDB 2000). pp. 53–62.

[Clark, 1999a] Clark, J. and DeRose, S. (eds.)., 1999. XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath, W3C.

[Clark, 1999b] Clark, J. (ed.), 1999. XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt, W3C.

[Clark, 2001] Clark, J. (ed.), 2001. XSL Transformations (XSLT) Version 1.1. http://www.w3.org/TR/xslt11, W3C.

[Cluet, 1998] Cluet, S., Delobel, C., Simeon, J., and Smaga, K., 1998. Your Mediators Need Data Convention!, Proc. ACM-SIGMOD International Conference on Management of Data '98, pp. 177–188.

[Cox, 2001] Cox, S., Cuthbert, A., Lake, R., and Martell, R. (eds.), 2001. Geography Markup Language (GML) 2.0. http://www.opengis.net/gml/01-029/GML2.html, Open GIS Consortium.

[Deutsch, 1999] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu D., 1999. A Query Language for XML. Proc. the Eighth International World Wide Web Conference (WWW8). Computer Networks 31 (11–16), pp. 1155–1169.

[DPC] Database Promotion Center. G-XML. http://gisclh.dpc.or.jp/gxml/.

[DeRose, 2001] DeRose, S. J., and Scientist, C., 2001. XQuery: A unified syntax for linking and querying general XML documents. The Query Languages Workshop (QL'98). http://www12.w3.org/TandS/QL/QL98/pp/xquery.html.

[Fernandez, 1998] Fernandez, M. F., Florescu, D., Kang, J., Levy, A. Y., and Suciu, D., 1998. Catching the Boat with Strudel: Experiences with a Website Management System. Proc. ACM-SIGMOD International Conference on Management of Data '98, pp. 414–425.

[Güeting] Gueting, R. H. , 1989. Gral : an Extensible Relational Database for Geometric Applications. Proc. of the VLDB International Conference. pp. 33–44.

[Robie, 1998] Robie, J., Lapp, J., and Schach., D., 1998. XML Query Language (XQL). The Query Languages Workshop (QL'98). http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[Shinagawa, 2000] Shinagawa, N., Kitagawa, H., and Ishikawa, Y., 2000. X^2QL: An eXtensible XML Query Language Supporting User-defined Foreign Functions. Proc. 2000 ADBIS-DASFAA Symposium on Advances in Databases and Information Systems. LNCS 1884, pp. 251–264.

[Shinagawa, 2001] Shinagawa, N., Kuragaki, K., and Kitagawa, H. Integration of WWW Applications Based on Extensible XML Query and Processing Languages. Proc. 12th International Conference on Database and Expert Systems Applications (DEXA 2001). (to appear).

[Stonebraker, 1986] Stonebraker, M., 1986. The INGRES Papers: Anatomy of a Relational Database System. Addison-Wesley.

[Tamino] Software AG. Tamino. http://www.softwareag.com/tamino.