Web geoprocessing services on GML with a fast XML database

Clarisse KAGOYIRE
CGIS-NUR, PO BOX 212, Rwanda
Cell phone n°: +250788837322
ckagoyire@nur.ac.rw, clarisse.kagoyire@cgisnur.org

Abstract

Nowadays there exist quite a lot of Spatial Database Infrastructures (SDI) that facilitate the Geographic Information Systems (GIS) user community in getting access to distributed spatial data through web technology. However, sometimes the users first have to process available spatial data to obtain the needed information. The geoprocessing services can be provided over the web using the conventional databases (such as the relational databases or object-oriented databases) as back-end, though this causes a concrete problem of overhead of data conversion. In this research, we proposed an approach to provide the web geoprocessing services, using an XML (eXtensible Markup Language) database system as back-end and the Geography Markup Language (GML) as data encoding standard. Currently there is not yet a formal standard query language for GML, we demonstrate that the XML Query (XQuery) language can be extended with spatial semantics to carry out spatial computation upon GML data.

A scenario was chosen, namely the assessment of soil erosion caused by rainfall, to apply the proposed approaches. After describing and analysing the requirements of an assessment of soil erosion caused by rainfall, we proposed a suitable system prototype design combining the Model View Controller (MVC) architectural pattern with Service-Oriented Architecture (SOA) principles. To add robustness and flexibility to the system, the implemented web geoprocesses were provided through Web service orchestration. Such system prototype can support the decision-making activities, such as planning the land use for environmental conservation purposes.

Keywords: *Geoprocessing, XQuery, XML database, spatial queries, GML, Web service orchestration.*

1. Introduction

The exponential growth of the use of web technology has led people to hail it as one of the innovating technological events in information sharing. In the domain of Geographic Information Systems (GIS), this growth has emphasized the need for sharing spatial information as well as the need for interoperability among heterogeneous spatial information systems over the web. In fact, nowadays there exist quite a lot of SDIs (Spatial Database Infrastructures) that facilitate the GIS user community in getting access to

distributed spatial data through web technology. However, sometimes the users are unable to get the precise required information, and are forced to first process the available spatial data to determine the needed information (Kiehle et al., 2007). In such cases, users may need to make use of distributed geoprocessing services available through the web. The basis of distributing geoprocesses over the web is the spatial data on which they are applied. They must not only have access to the spatial data sources (spatial databases or image repositories), but also carry out geographic computation tasks on those data and return response messages and/or data outputs. To achieve an efficient distribution of geoprocessing services among various spatial information systems (based on interoperability); there should be a standard for encoding the spatial data to support the data exchange. Moreover, there should also be a flexible orchestration of geoprocessing services with the aim of facilitating the implementation of an expeditious application system that can handle spatio-analytic functions on the web.

An SDI can use GML (Geographic Mark-up Language) as data encoding to support their transportation from one application system to another (from one node to another) over the Internet. Considering the state-of-art of data management technologies (such as RDBMS,[3] ORDBMS,[4] OODBMS[5]) (Pardede et al. 2006; Shimura et al., 1999), each node that receives the data in GML format, has to parse and store them into an internal or proprietary format (depending on the data management system in use), apply a number of functions on them and then convert back the resulting information into GML before transmitting them to the next node. This brings out a concrete problem of overhead of data conversion. However, given that GML encodes spatial and non-spatial data using XML (eXtensible Markup Language) encodings (Lake et al., 2004), GML documents can be stored and manipulated in an XML DBMS[6] (such as MonetDB/XQuery[7]) (A. Boncz et al., 2006) without any need of data conversion. Therefore, XML databases are more favourable to manage GML data for providing geoprocessing services in an SDI. However, we have to bear in mind that there is a need of optimizing their manageability as the characteristics of spatial data and their data volume may drastically affect the effectiveness of application systems that handle GML documents.

[3] Relational Database Management System.
[4] Object-Relational Database Management System.
[5] Object-Oriented Database Management System.
[6] XML Database Management System.
[7] A fast XML database that fully supports the World Wide Web Consortium XQuery language.

A number of studies has been carried out and recommended several approaches to tackle the problem of providing geoprocesses on the web and the problem of spatial data exchange to ensure interoperability among various spatial information systems. However, the integration of different spatial data sources with different schemas and the performance of SDIs (Kiehle et al., 2007) within a distributed computing environment remain questionable due to the complexity of spatial data and their large data volumes. For instance, Chia-Hsin, et al. (2006) implemented a GML query processor to support and speed-up geospatial functionality but concluded that the technique used of *XML/GML prefiltering* was not good enough to reduce the cost of retrieving the index, thus recommended more studies to improve the way of processing GML documents. Kiehle (2006, pp.1749-1750) described the development of a business-logic component for the geoprocessing of distributed geodata, showing the main web technologies available to utilize distributed and heterogeneous spatial data.

This research has the aim of optimizing the provision of distributed web geoprocessing service applied on GML data sources, using an XML database as a back-end. After describing MonetDB/XQuery database system and enquiring into what XQuery offers to retrieve and manipulate GML data stored in an XML database; a number of spatial functions will be realized in MonetDB/XQuery and provided over Internet through Web service orchestration. The Service-Oriented Architecture (SOA) approach will be used to support the organisation and utilization of distributed functions and distributed spatial data. Each computation node in our distributed geoprocessing system, will be equipped with MonetDB/XQuery database system and some spatial functions and/or spatial data that are accessible over the Web. The rest of the paper is structured as follows: Section 2 provides the basic concepts of XQuery and describes its implementation, MonetDB/XQuery. Section 3 describes the implementation of an extension of XQuery for spatial functionality in MoneDB/XQuery. Section 4 and Section 5 discuss the design of our system prototype and its implementation, respectively. Finally, the last Section concludes the paper, outlining the achievements in regards with the innovation aimed at and recommendations for further research.

2. The XML Query Language

This section contains the basic concepts of XQuery. The XML Query Language (XQuery) is a language for processing XML data. It is built on XPath (XML Path Language) (W3C, 2007b) expressions. XML was

designed for storing and transferring data. It reduces the efforts spent on data transformation for transport and facilitates the integration of data from different platforms and different formats, owing to their self-descriptive characteristics. XML allows us to define and use new elements according to the data to be represented without any restriction. In other words, XML allows us to define our own tags which are like attribute in table-based databases. For instance Figure 1 shows an XML document (buildings.xml) with data about city buildings; note that the names of the tags give an idea of the data they enclose. This flexibility entails irregularity and heterogeneity in the structure of XML data, in contrast with relational data whose structure is regular and homogeneous. To take advantage of XML flexibility and extensibility, there is a need for a query language that can provide possibilities to retrieve and manage easily information represented in the XML format given their irregular structure. XQuery was designed to retrieve information from XML databases including relational databases that store XML data or that present non-XML data sources as XML views. XQuery was developed by the W3C XML Query Working Group to be a language in which queries are short and simple (W3C, 2007f).

```
<?xml version="1.0" ?>
<cityBuildings>
    <city name="Leicester">
       <building year="1959">
           <title>Leicester University Engineering Building</title>
           <style>Modern</style>
           <!--this is a research university based in England-->
           <architect>
                <last>Stirling</last>
                <first>James</first>
           </architect>
       </building>
    </city>
    <city name="Rome">
       <building year="1913">
           <title>American Academy in Rome</title>
           <style>Neoclassical</style>
           <architect>
                <last>Charles Follen</last>
                <first>McKim</first>
           </architect>
           <architect>
                <last>Stanford</last>
                <first>White</first>
            </architect>
       </building>
    </city>
</cityBuildings>
```

Figure 1: Example of an XML document.

XQuery is based on the XPath language, which is a W3C standard that facilitates to navigate through the hierarchical structure of an XML

document using path expressions (W3C, 2007b). As the name reveals, the path expressions describe a path to select a particular node within an XML document. An XML document has a tree-based structure of which the root is represented by a document node. A node can be one of seven kinds of node: document, element, attribute, text, namespace, processing instruction and comment. The structure of the XML document represented in Figure 1 is illustrated in Figure 2.

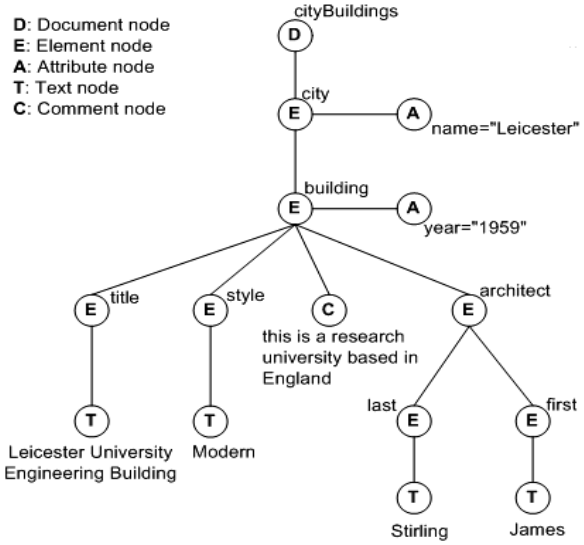

Figure 2: Representation of a document part in the XQuery Data Model (adapted from (Howard et al., 2003)

A document node or an element node is a parent of its children. A child node can have only one parent and the nodes that have the same parent are siblings. The ancestors of a node are the parent of that node, its parent's parent, and so on. The descendants of a node consist of the children of that node, children of the children, and so on. To navigate through XML documents, path expressions are used and they commonly start at the root node. XQuery extends XPath, providing more flexibility to handle complex tasks such as record-selection that may require recursion.

2.1. XQuery Formal semantics

The XQuery Formal Semantics provides a mathematical meaning to each of the XQuery expressions through the XQuery Data Model. For more details about the XQuery Data Model, refer to (W3C, 2007c). XQuery is a

functional language as it is based on expressions that can be nested given that the result of an expression can be used as an argument in another expression. XQuery is a strongly typed language because it prevents any compilation of incorrectly typed expressions; it supports compulsory dynamic typing and optionally static typing. The dynamic type checking is carried out at run-time, and detects type errors on the values of expressions such as using zero as a divisor within an arithmetic operation expression. Static typing detects type errors in expressions at compile-time, and permits to detect the error before the expression is evaluated. This can be used as a basis of certain classes of optimisation as well (Howard et al., 2003; W3C, 2007d).

In XQuery language, each query consists of either a simple expression or a composed expression; and various kinds of expressions can be combined by operators to constitute new expressions. XQuery has two input functions that can be used within expressions to access input data from an XML document, meaning from an XML database in which the document is stored. The *doc ()* function returns the document or more precisely the document node associated with the specified URI. The *collection ()* function returns a sequence of document nodes, from a given URI. The most common XQuery expressions are literals, path expressions, FLWOR (For-Let-Where-Order by-Return) expressions, elements constructors, and conditional expressions (W3C, 2007d).

- **Literals**

The literals are the simplest XQuery expressions. A literal consists of an atomic value that can be numeric or string. There are three kinds of numeric literals; integers, decimals and doubles.

- **Path expressions**

The path expressions are used to navigate through the tree structure of XML documents to locate nodes of interest. The expression: *doc ("buildings.xml")/cityBuildings/city/building/title* will return all the title elements along such paths. The function *doc ()* is used to access and open, the operator / is used to navigate through the document *buildings.xml* (Figure 1), and find the element nodes *title*.

- **FLWOR expressions**

The FLWOR expression (pronounced ”flower expression”) format is the most powerful XQuery expression, and is syntactically similar to the SQL statement SELECT-FROM-WHERE. It allows us to express iteration in XQuery. Here is a simple example of an expression that returns the titles of the buildings that have been constructed in 1913 and the number of architects associated with each of the returned buildings, using FLWOR.

```
for $b in doc ("buildings.xml")//building
let $a := $b/architect
where $b/@year="1913"
return
   <building>
      {$b/title,<count>{count($a)}t</count>}
   </building>
```

In our case, there is only one building constructed in 1913. The result of the query above is:

```
<building>
   <title>American Academy in Rome</title> <count>2</count>
</building>
```

The *for clause* binds the variable *$b* to the list of the buildings evaluated from the expression *doc("buildings.xml")/city/building*, the *let clause* binds the variable to all returned architects of each building, the *where clause* filters the returned buildings to retain only the buildings constructed in 1913 and the *return clause* returns the title of each of those buildings, and its number of architects. Observe that both *for* and *let* are a means to state iteration over a list of elements and the evaluation of where and return are nested inside those iterations.

- **Element constructors**

In the above example, the element constructor is used to create the element node "building" in the return clause.

- **Conditional expressions**

Queries in XQuery can also use conditional expressions just as they are used in other languages. For example, to return only the buildings that have been designed by more than one architect, one uses the following query:

```
for $b in doc ("buildings.xml")/city/building
let $a := $b/architect
return
   if (count ($a)>1)  then $b/title
   else  ()
```

Note that else () indicates that in case the expression if (count ($a)>1) is evaluated to false, nothing will be returned.

XQuery provides a number of functions and operators, some of which are commonly used in other languages. It has arithmetic operators, comparison operators and sequence operators. In addition to operators, XQuery has built-in functions and supports user-defined functions. The full list of these

functions and their descriptions can be found in (W3C, 2007e). It is always better to make use of user-defined functions instead of a long complex query as these are reusable and easier to understand. The user defined functions can be put in a module, which in return can be imported by a query to access its functions (Howard et al., 2003). Here is the syntax of a function definition with its module declaration:

```
module namespace prefix="nameModule" ;
declare functionprefix:functionName($parameter AS datatype)
AS returnDatatype { (: function code :) };
```

Note that the character "*$*" indicates a variable, that means, the *$parameter* is a variable passed as argument to the function *functionName*, whose module is *nameModule*. Whatever appears inside "(: :)" is considered as comment and is not evaluated. To call the function within a query expression, the module should first be imported before calling the function, as follows:

```
import module namespace prefix="nameModule"
at "URI_locationModule/filename.xq";
prefix:functionName ($argument )
```

2.2. XQuery implementation: MonetDB/XQuery

Relational database technology is well-known and has been proven to be very effective in structured data management. Its mature infrastructure can be used to build a fast and scalable XML database management system such as MonetDB/XQuery. MonetDB/XQuery is an XML database system that supports the W3C XQuery language (Teubner, 2007; A. Boncz et al., 2006).

MonetDB/XQuery system architecture

MonetDB/XQuery consists of the Pathfinder XQuery compiler on top of the MonetDB RDBMS. It resulted from the effort of using a process model of an existing relational database system (MonetDB RDBMS) to construct a purely relational XQuery processor (Pathfinder) (Teubner, 2007; Peter Boncz, 2005b). Pathfinder was used to link the relational processing model with the XQuery Data Model following the XQuery processing stack shown in Figure 3.

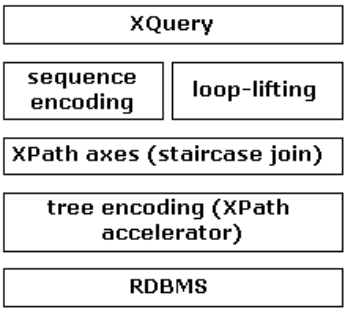

Figure 3: XQuery processing stack (adapted from (Peter Boncz, 2005b))

MonetDB/XQuery allows to store XML documents and manipulate them using XQuery. Its architecture (shown in Figure 4) consists of the Pathfinder XQuery compiler that turns the input XQuery expression into a relational algebra query for execution in the MonetDB kernel. The relational algebra query is sent out to the MonetDB kernel expressed in terms of MIL (MonetDB Interpreter Language), after the query processing has finished, the result is serialized into XML and sent back to the user.

Pathfinder XQuery compiler

The Pathfinder XQuery compiler supports almost the full XQuery standard, closely following the W3C XQuery Formal Semantics as stated in (Peter Boncz, 2005a). To implement the Pathfinder, the following techniques were used as described in (Grust, 2002): XPath accelerator, the staircase join and the loop-lifting compilation procedure.

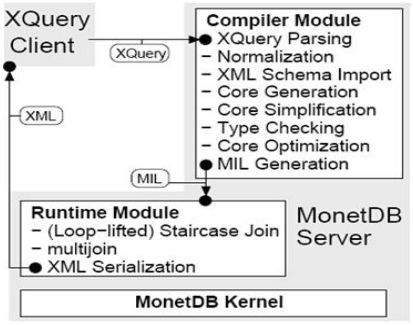

Figure 4. Architecture of the MonetDB/XQuery system (from (Peter Boncz, 2005b))

3. Extending MonetDB/XQuery with spatial functionality

This section details the implementation of an extension of XQuery for spatial functionality in MonetDB/XQuery. The functions currently implemented in Monet/XQuery do not provide any spatial functionality, hence it should be improved with an extension of functions that can handle GML data and data in other spatial formats (such as KML, KMZ, GPX and GeoRSS) to provide spatial computations. GML (the Geography Mark-Up Language) is an OGC (Open Geospatial Consortium) standard specification that expresses geographic information, supports their exchange among disparate applications and is an XML encoding. XQuery can provide the ability to navigate through a GML document given its XML navigation mechanism; however, it lacks the spatial semantics required to carry out spatial computation upon GML features. For the moment, there is not yet a formal standard query language for GML, though several previous researches have been carried out proposing different ways to query GML data. Córcoles and González (2001) proposed a specification of a query language over GML without taking into account XQuery, as the latter was still under development, but this leaves out a number of features supported in XQuery. Moreover it is based on predefined geometry elements, which constrains the flexibility of GML.

3.1. Basic concepts of Geographic Mark-Up Language (GML)

GML objects

GML can be used to integrate spatial and non-spatial data for representing real-world objects, i.e the geographic features; though not all GML objects are features. GML objects include the features and geometries.

- **Features:** In GML, features represent real-world objects, which can be physical or abstract objects. The concrete objects are physical objects such as buildings, roads, bridges, . . . And the abstract objects may consist of political boundaries, health regions, ... GML features are described by their properties which may be geometric or non-geometric. A GML feature is defined by creating its feature instance and each feature instance must have its own unique identifier (unique within the GML document) (Lake et al., 2004). For example, to create a Building instance, one may use the following syntax: <Building gml:id = "id01"> ... </Building>
Note the use of XML Namespaces in GML with the prefix *gml* (Refer to (W3C, 2006) for more details on XML namespaces).

- **Geometries:** GML geometries are objects that can be used to represent the geometric characteristics of a feature such as position, centre, extent,

location . . . For instance, to represent the position of the Building feature; position can be used as the geometry-valued property of the Building instance.

```
<Building gml:id = "id01">
   <name>Four Times Square</name>
   <type>Commercial office</type>
   <owner>The Durst Organization</owner>
   <gml:position>
      <gml:Point srsName="ReferenceSystemNumb">
         <gml:pos>...</gml:pos>
      </gml:Point>
   </gml:position>
</Building>
```

Other GML objects: Besides the features and geometries, GML has other objects introduced in its third specification version (GML 3.0), namely: topologies, coordinate systems, coverages, and units of measure . . .

3.2. Syntax of XQuery extension for GML

XQuery is a more appropriate query language to query XML data than traditional query languages like SQL, given its ability to navigate through the XML document. Even though GML data is based on XML data model and can be queried using XQuery, XQuery does not have the spatial semantics required to handle spatial computation over GML data. As a solution to this, we propose a syntax for spatial functions in XQuery based on the GML data model. The proposed syntax is based on the object-property model and not on a predefined GML application schema, so that the spatial functions can be applied to any GML data regardless its application, and this is in accordance with GML extensibility. Moreover, the proposed syntax takes into consideration only the GML objects defined in GML Simple features profile that consists of points, lines, polygons geometries and their respective geometry collections. The spatial functions in XQuery will have the same expression structure as other non-spatial XQuery functions:

prefix:functionName ($argumentsGeom as argumentType) as return Type

For example, to extract the geometric properties of a given feature from a GML document, the following function is proposed:

geoxml:geometry ($nodeGeom as node*) as string*[=wkt*]

Here, *geoxml* is the prefix of the concerned spatial functions module (or library), *geometry* is the name of the function, *nodeGeom* is the argument of the function, *node** is the type of the arguments (as defined in XQuery data model, but it should also be a valid GML object) and *string** is the type of the returned result. The *geoxml:geometry* function should receive a node as argument and return a string, namely a geometry in well-known text (WKT) representation. Thus, any node containing geometric properties which is valid in GML can be passed to this function regardless of its application schema. The spatial queries resulting from this syntax can be compared with SQL spatial queries as shown in Table 1.

Table 1 Comparison of spatial queries in XQuery and SQL spatial queries

XML databases	**Relational databases**
XQuery	SQL
GML document	Table containing geometries
Node element with geometric properties	Geometry column

Spatial Queries using XQuery

Generally, spatial queries involve spatial functions applied on geometries. These may concern spatial relationships between geometries, spatial processing (computing the area, length, ... of geometries), geometry constructors and accessors.

a. Geometry constructors and accessors

A function accesses the geometric properties of a given feature and return the represented geometry is important for spatial queries in XQuery. Besides the access to geometries already represented in GML, one would like to create a new geometry from a string representing a geometry in well-known text format. Below is a list of some geometry accessors and constructors:

Accessors

1. geoxml:geometry($nodeGeom as node*) as xs:string*[=wkt*]
2. geoxml:srid($geom as xs:string*[=wkt*]) as xs:integer
3. geoxml:geometryType($geom as xs:string*[=wkt*]) as xs:string

Note that the last three functions above receive an argument with same type as the return type of the function *geoxml:geometry ()*, namely geometry in WKT. Their returned results are respectively: the spatial reference system number and the geometry type of the given geometry (if the geometry is simple, NULL is returned). The list of accessors can be extended, with more

functions that return different property values of a geometry, such as *geoxml:X()*, *geoxml:Y()*, and *geoxml:Envelope()*, . . .

Constructors

1. geoxml:geometryFomText($geom as xs:string*[=wkt*], $srs as xs:integer) as node*
2. geoxml:polygonFomText($geom as xs:string*[=wkt*], $srs as xs:integer) as node*

The above function returns a geometry encoded in GML, given a WKT representing that geometry and its spatial reference system number. There are more geometry constructors that can be used to construct specific GML geometries namely *geoxml:pointFomText()*, *geoxml:linestringFomText()*, *geoxml:polygonFomText()*, and *geoxml:multipointFomText(),* . . . .

b. Functions for spatial relationships

To check the spatial relationship between two geometries, a number of functions is needed namely *geoxml:intersects()*, *geoxml:touches()*, *geoxml:within()*, *geoxml:overlaps()*, *geoxml:relate()*... These functions return boolean accordingly. For instance, for the *geoxml:intersects()*, *geoxml:within()* we have:

1. geoxml: intersects ($geom as xs: string* [= wkt*], $geom as xs: string* [= wkt*]) as xs: Boolean
2. geoxml: within ($geom as xs: string* [= wkt*], $geom as xs: string* [= wkt*]) as xs: Boolean

c. Functions for spatial processing

The spatial processing involve functions to compute the area, the length, the centroid, the boundary, the buffer of a geometry, the intersection, the difference, the union of two geometries. The first four functions are unary function whereas the remaining functions are binary. Note that there are two kinds of union function, the binary union function that takes two geometries as arguments and returns their union, and the aggregate union function that take one set of geometries and returns their union. Below, is the syntax of the two kinds of union function and the intersection function syntax:

1. geoxml:union ($geom1 as xs:string*[=wkb*], $geom2 as
2. xs:string*[=wkb*]) as xs:string*[=wkb*]
3. geoxml:union ($geomset as item*) as xs:string*[=wkt*]
4. geoxml:intersection ($geom1 as xs:string*[=wkb*], $geom2 as
5. xs:string*[=wkb*])
   as xs:string*[=wkb*]

3.3. GEOXML: an extension of XQuery for spatial functionality in MonetDB/XQuery

The proposed extension of XQuery was implemented in the MonetDB/XQuery database management system. Currently, it consists of only a few spatial functions that provide basic spatial computation functionality over GML data, namely two geometry accessors: *geoxml: geometry() and geoxml:wkb()*; two geometry processing functions: *geoxml: distance()* and *geoxml:intersection()* and one function for geometry spatial relationship *geoxml:relate()*. The *geoxml:relate()* function can be used to express other spatial relationships between geometries such as Disjoint, Overlaps, Touches, Within and Crosses by means of the Dimensionally Extended Nine-Intersection Matrix (DE-9IM) (Garnett and Owens). The implementation of the GEOXML library in MonetDB/XQuery is based on GEOS (Geometry Engine - Open Source) library. With a C++ API (Application Programming Interface), GEOS consist of all spatial functions and spatial operators for SQL Simple Features profile, included in JTS (JTS Topology Suite, a Java API consisting of spatial data operations).

4. Design of a distributed geoprocessing system prototype

In this section we discuss in details the design of our system prototype based on a chosen scenario of soil erosion assessment within the watershed of Sebeya river located in the Western Province of Rwanda. We designed a distributed environment for a geoprocessing prototype system that can compute the annual average of soil loss caused by precipitation. It consists of a number of autonomous nodes (computers) connected through a network, and where each node can provide processing services and those services should be accessible to any node within the system. All the nodes within our system are able to exchange information with each other, and appear to the user of the system as one single component. The term geoprocessing refers to information processing in which the involved processes apply operations to spatial data (Foerster and Schäffer, 2007). To assess the soil loss by water erosion, a number of spatial data sets is required, especially weather data, soil data, topographic data and land cover data. It is assumed that each data set is located on a node in our distributed geoprocessing system and all nodes are communicating over a network. Having each node equipped with MonetDB/XQuery, will allows us to use the Remote Procedure Call (RPC) mechanism to ensure the interprocess communication.

To compute the annual average soil loss, the Universal Soil Loss Equation (USLE) is mostly used (Wishmeier and Smith, 1998), it takes into account a number of physical and management parameters of the considered site,

which are expressed as numerical factors: rainfall erosivity factor R, soil erodibility factor K, slope length factor L, slope steepness factor S, cover management factor C and support practice factor P, as used in Equation 1.

A = R * K * L * S * C * P (**Equation 8**)

Since the USLE factors vary by location, their retrieval and management require spatial functions. That implies that the prototype should provide some spatial functions such as Intersection, Overlap, Within applied on GML data (stored in an XML database). To calculate the annual average of the soil loss of a selected (particular) area within Sebeya watershed, the following steps are followed (Figure 1):

a. A request of locating the site to be assessed is sent to Node 1 from user machine through a web browser.
b. Based on information stored in place names data set, the area of interest is delimited and the data (the geometry) representing that area is shipped to Node 2, where the annual average soil loss (A) is calculated.
c. Before computing A, all USLE factors should be retrieved, to do so the following tasks are executed:

- ✓ K factor is retrieved from the soil data set (Node 2). The soil data are clipped according to the site of interest, then K values are assigned to the clipped geometries according to their respective soil type.
- ✓ R factor is retrieved from the weather data set (Node 3). After the interpolation of the recorded rainfall data, the interpolated data are clipped according to the site of interest. Then, the annual and monthly (for the wettest month) rainfall values are retrieved and used to compute the value of R factor.
- ✓ LS factor is retrieved from the topographic data set (Node 4). After clipping the topographic data according to the selected site, the values of slope angle are retrieve to compute LS.
- ✓ C factor is retrieved from the land cover data set. The land cover data are clipped according to the site of interest, and then C values are assigned to their corresponding vegetation cover.

d. After computing all the needed factor, the annual average of the soil loss is computed (Node 2).
e. The computed annual average of the soil loss is sent back to Node 1.
f. Node 1 presents the computed annual average of the soil loss back to the user via a web browser.

To design our prototype, we used the SOA approach for which the principal technical concepts are services, interoperability and loose coupling. Services are self-contained functionality within a system that can be technically viewed as the interface for exchanged messages between the provider node

and consumer node of the services. The interoperability enables the distribution of services over different nodes within the system. The loose coupling concept reduces the dependencies. As stated in (Josuttis, 2007): "Web services might help provide the infrastructure, but you still have to construct the architecture." That being said, it should be pointed out that we adopted the Model-View-Controller (MVC) architectural pattern to separate the application functionality (business logic) from the presentation. The MVC architecture consists of the model, the view and the controller.

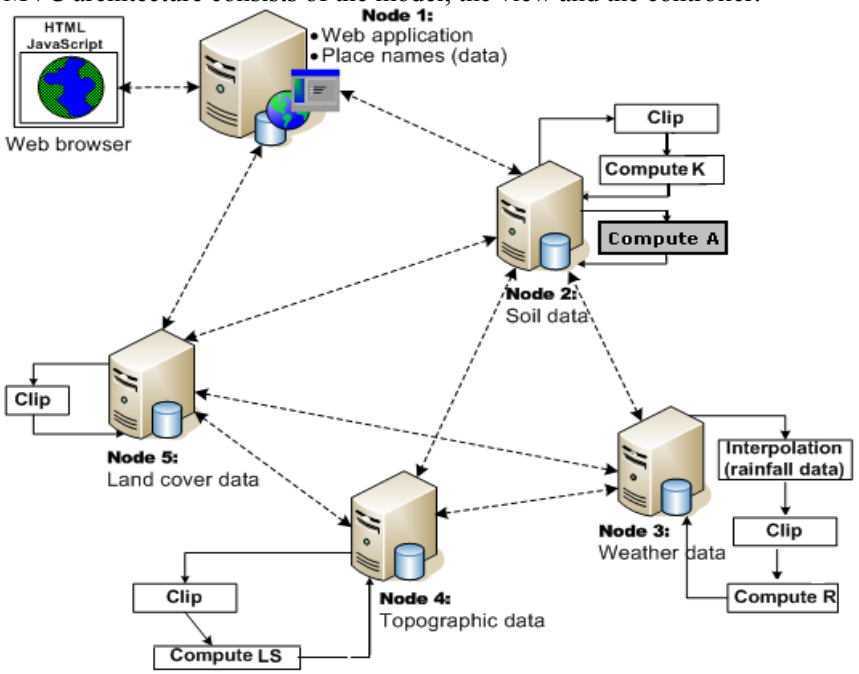

Figure 5: Processing steps for calculating the annual average of the soil loss

The model represents the data of the system, the view ensures the presentation of the data to the user of the system, and the controller deals with the application functionality. The controller ensures the interaction between the user and the system translating the user actions to the model or to the view (Reenskaug, 1979; McGovern et al., 2003). The controller consists of business logic or middleware, it comprises a number of functions used to access data, process them and send the results to the view. For our prototype, these functions inside the business logic comprise the spatial functions to handle the geoprocesses. The communication between the view

and the business logic will be handled by XRPC, an XQuery extension that uses a Remote Procedure Call paradigm.

XQuery Remote Procedure Call (XRPC)

XRPC is an extension of XQuery that enables distributed querying and processing among heterogeneous data sources. It has been fully implemented in MonetDB/XQuery. XRPC adds the RPC concept to XQuery, adding a destination URI[8] to the ordinary XQuery function call (or procedure call) (Zhang and Boncz, 2007). It supports the use of heterogeneous XQuery engines within a distributed system, but our application happens not be heterogeneous since all the nodes will be equipped with MonetDB/XQuery. As XRPC enables the cooperation of different XQuery nodes to handle a given processing task, this involves a network protocol to support their communication. XRPC uses Simple Object Access Protocol (SOAP) over HTTP, which allows integrating XQuery data sources with web services and SOA. XQuery remote functions can be executed using SOAP XRPC messaging as long as they are defined in a module stored on an accessible node. Below is an example of an XRPC Request message, sent to a remote node *clark* (Figure 6). This request executes the function *bounds()* defined in *geom.xq* module.

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope
xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:xrpc="http://monetdb.cwi.nl/XQuery"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://monetdb.cwi.nl/XQuery
http://monetdb.cwi.nl/XQuery/XRPC.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <env:Body>
    <xrpc:request xrpc:module="bb"
    xrpc:location="http://clark:50001/export/geom.x
    q" xrpc:method="bounds" xrpc:arity="1">

      <xrpc:call>
        <xrpc:sequence>
          <xrpc:atomic-value
          xsi:type="xs:string">Helmerhoek
          </xrpc:atomic-value>
        </xrpc:sequence>
      </xrpc:call>
    </xrpc:request>
  </env:Body>
</env:Envelope>
```

Figure 6: An example of an XRPC Request message

[8] Uniform Resource Identifier

5. Implementation a distributed geoprocessing system prototype

The implementation of our system prototype relies on the logical architecture designed in section 4. A SOA approach was adopted to make available the geoprocessing services in a distributed environment through service orchestration. The model of our system prototype was implemented using the MonetDB database system with XQuery front-end (MonetDB/XQuery), which allowed us to store and manage our data in GML format. The user interface was implemented using HTML, JavaScript a client-side scripting language, and a stylesheet language CSS (Cascading Style Sheets). It is implemented in such a way that it allows the user to provide input data to the system and get back the output. It consists of a form that allows the user to choose the watershed and sector name for which s/he would like to compute the annual soil loss, and submit the input to the system as illustrated in Figure 7.

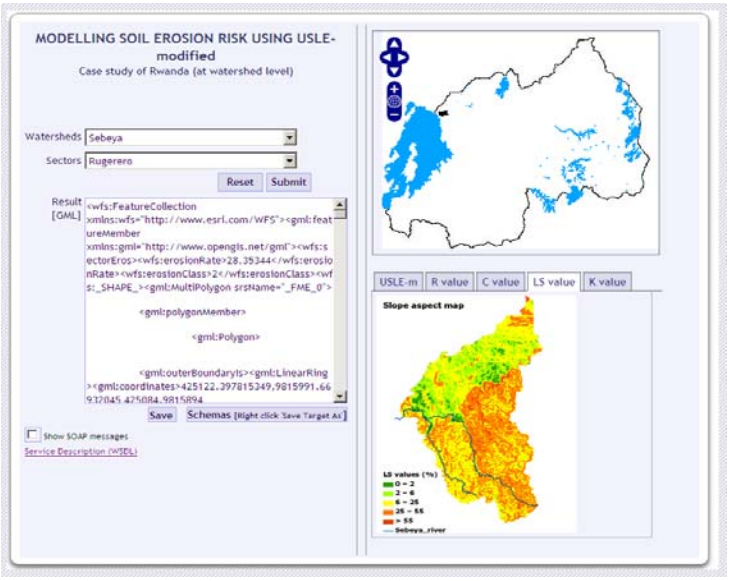

Figure 7: User interface

After submitting the form, the user retrieves the resulting data displayed in a text area in GML format (and this can be visualized in any WFS/GML viewer). The map on the top-right of the user interface as shown in Figure 7, consists of the output map and the bottom-right consists of maps of the different input parameters, namely rainfall, land cover, slope and soil. The main functionality of our web-based prototype system is to compute the annual average of soil loss caused by precipitation, in a distributed

environment. The provision of such a processing service involves the implementation of a number of processes, namely the site location, retrieval of USLE factors and the calculation of annual average soil loss (erosion rate). To model these processes and their workflow, eClarus Business Process Modeler for SOA Architects (eClarus BPMSOA) system was used with BPMN (Business Process Modeling Notation) standard. The resulting business process diagram (BPD) is shown in Figure 8.

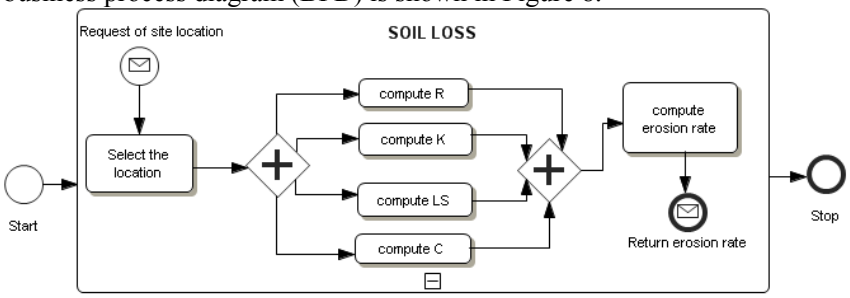

Figure 8. Process workflow for computing the annual average of soil loss caused by precipitation

Web service orchestration

A service consists of a function that performs a task (or many tasks); it is self-contained and independent. A service is called a Web service if it is accessible over the Internet, so it should be described using the WSDL standard (W3C, 2007a) to provide meta-information about how it can be accessed. Moreover, a Web service is published, to make it discoverable. Publishing Web services is optional and is done using the UDDI (Universal Description, Discovery and Integration) standard. As already mentioned, a Web service may consist of more than one activity (task) to fulfil a specific business goal, i.e., to carry out a business process. Orchestration is the act of defining the sequence flow of all activities involved to carry out a business process. After describing our Web service, we orchestrated it by means of eClarus BPMSOA system with BPEL[9] (Diane Jordan and John Evdemon, 2007) standard. Based on the BPD created to model the workflow of processes (Figure 8), we performed the Web service orchestration, mapping the semantics behind the graphical notation elements into BPEL language by means of eClarus BPMSOA system. To map the BPD into BPEL, the system uses the description of the Web service and the properties of the graphical

[9] Also known as WSBPEL (Web Services Business Process Execution Language) is an XML-based language used for composing and orchestrating services that result in Web service.

notation specified in the BPD. After setting up the BPEL document for computing the annual average of soil loss process, the process was then executed on the orchestration engine provided by eClarus BPMSOA system.

6. Conclusions

A distributed system prototype, that provides web geoprocessing service, was designed and implemented using an XML database system as back-end. The SOA approach was adopted to implement this system prototype, which allowed us to orchestrate our Web service and provide a more flexible geoprocessing service over the Web. Since GML was used as data format, we proposed a syntax for spatial functions in XQuery applied on GML data. A scenario was chosen, namely the assessment of soil erosion caused by precipitation, to apply the proposed approaches. After describing and analysing the requirements of an assessment of soil erosion caused by precipitation, we proposed a suitable system prototype design combining the MVC architectural pattern with SOA principles. Based on the experience acquired throughout the implementation phase, it can be concluded that the combination of MVC architectural pattern and SOA provides an opportunity to add robustness and flexibility to the implemented Web service. It allows to separate the business logic and the data model from the presentation and to control the sequence flow of the processes through Web service orchestration.

For further research, we propose a study on the implementation of OpenGis web services (such as Web Map Service, Web Feature Service, Catalog Service, . . .) using an XML database system as back-end. As an improvement to the current work, we recommend the following:

- The implementation of a complete library of spatial functions in MonetDB/XQuery database system.
- The implementation of spatial indexes in MonetDB/XQuery database system.
- Further work on map visualisation of GML data, using SVG and XSLT, in a web browser. This can be used to visualize the GML results returned by our system.

Bibliography

A. BONCZ, P., GRUST, T., VAN KEULEN, M., MANEGOLD, S., RITTINGER, J. & TEUBNER, J. Year. MonetDB/XQuery: a fast XQuery processor powered by a relational engine. *In:* Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, 2006 Chicago, IL, USA. 479-490.

CHIA-HSIN, H., TYNG-RUEY, C., DONG-PO, D. & HAHN-MING, L. 2006. Efficient GML-native processors for web-based GIS: Techniques and Tools. *Proceedings of the 14th annual ACM international symposium on Advances in Geographic Information Systems.* Arlington, Virginia, USA: ACM.

CORCOLES, J. E. & GONZALEZ, P. 2001. A specification of a spatial query language over GML. *GIS '01: Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems***,** 112--117.

DIANE JORDAN AND JOHN EVDEMON 2007. Web Services Business Process Execution Language Version 2.0. Organization for the Advancement of Structured Information Standards, Available at http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf [Accessed January 17 2009].

FOERSTER, T. & SCHÄFFER, B. Year. A client for distributed geo-processing on the web. *In:* TAYLOR, J. M. W. A. G. E., ed. Proceedings of web and wireless geographical information systems : 7th International Symposium, 28-29 November 2007 2007 United Kingdom. Springer-Verlag, 252--263.

GARNETT, J. & OWENS, B. Spatial Validation -- Academic References. Refractions Research, Inc.

GRUST, T. 2002. Accelerating XPath Location Steps. *Proceedings of the 21st ACM SIGMOD Int'l Conference on Management of Data (SIGMOD 2002)*. Madison, Wisconsin, USA, : ACM Press.

HOWARD, K., DON, C., MICHAEL, K., PHILIP, W. & DENISE, D. 2003. *XQuery from the Experts: A Guide to the W3C XML Query Language*, Addison-Wesley Longman Publishing Co., Inc.

JOSUTTIS, N. M. 2007. *SOA in practice : the Art of Distributed System Design,* Beijing etc., O'Reilly.

KIEHLE, C. 2006. Business logic for geoprocessing of distributed geodata. *Computers & Geosciences,* 32**,** 1746-1757.

KIEHLE, C., GREVE, K. & HEIER, C. 2007. Requirements for Next Generation Spatial Data Infrastructures-Standardized Web Based

Geoprocessing and Web Service Orchestration. *Transactions in GIS,* 11**,** 819-834.

LAKE, R., BURGGRAF, D. S., TRNINIC, M. & RAE, L. 2004. *GML : Geography mark - up language : foundation for the geo - web,* Chichester, Wiley & Sons.

MCGOVERN, J., AMBLER, S. W., STEVENS, M. E., LINN, J., SHARAN, V. & JO, E. K. 2003. *A Practical Guide to Enterprise Architecture*, Prentice Hall.

PARDEDE, E., RAHAYU, J. W. & TANIAR, D. Year. XML-Enabled Relational Database for XML Document Update. *In:* 20th International Conference on Advanced Information Networking and Applications, 2006. 205-212.

PETER BONCZ, S. M., AND JAN RITTINGER. 2005a. Updating the Pre/Post Plane in MonetDB/XQuery. *Proceedings of the ACM SIGMOD/PODS 2nd International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P 2005)*.

PETER BONCZ, T. G., MAURICE VAN KEULEN, STEFAN MANEGOLD, JAN RITTINGER, AND JENS TEUBNER. 2005b. Pathfinder: XQuery - The Relational Way. *Proceedings of the 31st Int'l Conference on Very Large Databases* Trondheim, Norway.

REENSKAUG, T. 1979. The original MVC reports. Department of Informatics, University of Oslo.

SHIMURA, T., YOSHIKAWA, M. & UEMURA, S. 1999. Storage and Retrieval of XML Documents using Object-Relational Databases. *Springer,* In Database and Expert Systems Applications**,** 206-217.

TEUBNER, J. 2007. Pathfinder: XQuery Compilation Techniques for Relational Database Targets. *Proceedings of the 12th GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW 2007)*.

W3C 2006. Namespaces in XML 1.0. Second Edition ed.: W3C, World Wide Web Consortium, Available at http://www.w3.org/TR/xml-names/ [Accessed September 19 2008].

W3C 2007a. Web Services Description Language (WSDL) Version 2.0. W3C recommendation, World Wide Web Consortium, Available at http://www.w3.org/TR/wsdl20.

W3C 2007b. XML Path Language (XPath) 2.0. W3C recommendation, World Wide Web Consurtium, Available at http://www.w3.org/ TR/xpath20/ [Accessed September 15 2008].

W3C 2007c. XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C recommendation, World Wide Web Consortium, Available at http://www.w3.org/TR/xpath-datamodel/ [Accessed September 11 2008].

W3C 2007d. XQuery 1.0 and XPath 2.0 Formal Semantics. W3C recommendation, World Wide Web Consortium, Available at http://www.w3.org/TR/xquery-semantics/ [Accessed September 16 2008].

W3C 2007e. XQuery 1.0 and XPath 2.0 Functions and Operators. W3C recommendation, World Wide Web Consortium, Available at http://www.w3.org/TR/xpath-functions/ [Accessed September 15 2008].

W3C 2007f. XQuery 1.0: An XML Query. W3C recommendation, World Wide Web Consortium, Available at http://www.w3.org/TR/xquery/ [Accessed September 11 2008].

WISHMEIER & SMITH 1998. *Predicting Rainfall Erosion Losses - A guide to conservation planning*.

ZHANG, Y. & BONCZ, P. 2007. XPRC: interoperable and efficient distributed XQuery. *Proceedings of the 33rd International Conference on Very Large Databases.*