

**Web geoprocessing services
on GML
with a fast XML database**

Clarisse Kagoyire

March, 2009

Web geoprocessing services
on GML
with a fast XML database

by

Clarisse Kagoyire

Thesis submitted to the International Institute for Geo-information Science and Earth Observation in partial fulfilment of the requirements for the degree in Master of Science in *Geoinformatics*.

Degree Assessment Board

Thesis advisor	Dr. Ir. R.A. de By Dr. Ir. R.L.G. Lemmens
Thesis examiners	Prof. Dr. M.J. Kraak Dr. Ir. M. van Keulen



INTERNATIONAL INSTITUTE FOR GEO-INFORMATION SCIENCE AND EARTH OBSERVATION
ENSCHEDA, THE NETHERLANDS

Disclaimer

This document describes work undertaken as part of a programme of study at the International Institute for Geo-information Science and Earth Observation (ITC). All views and opinions expressed therein remain the sole responsibility of the author, and do not necessarily represent those of the institute.

Abstract

The exponential growth of the use of web technology has emphasized the need for sharing spatial information over the Web. 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 first have to process available spatial data to determine the needed information. In this study, we propose an approach to provide the geoprocessing services over the Web, using an XML database system as back-end and GML as data encoding standard. Given that currently there is not yet a formal standard query language for GML. We demonstrate that the XQuery language can be extended with spatial semantics to carry out spatial computation upon GML data, proposing a syntax for spatial functions in XQuery based on the GML data model.

A scenario of assessing the soil erosion caused by rainfall, is used for developing a distributed system prototype that provides web geoprocessing services over the Web. To design and implement this system, we combined the MVC (Model-View-Controller) architectural pattern with SOA (Service-Oriented Architecture) principles. After describing the geoprocesses involved in soil erosion assessment, we implemented them using XQuery and provided them through Web service orchestration. The implemented distributed system prototype provides its users with a web geoprocessing service to assess the soil erosion caused by rainfall, given a site location. In this research, the soil erosion was assessed within the watershed of Sebeya river located in the Western Province of Rwanda.

Keywords

XQuery, XML database, GML, geoprocessing, Web services, Web service orchestration

Acknowledgements

This is a great opportunity to express my gratitude to my first supervisor Dr. Ir. R.A. de By for his supervision, constructive discussions and guidance throughout this work. I also thank my second supervisor Dr. Ir. R.L.G. Lemmens, for his support and suggestions.

I would like to thank Dr. Ir. M. van Keulen and Ir. Jan Flokstra for their support and collaboration. My sincere thanks to Ir. Adrie Mukashema (Msc), fellow worker at CGIS-NUR, for helping me to acquire information and data needed for this research.

I also gratefully acknowledge the financial support I received from the Government of Netherlands through the NUFFIC organisation.

To the National University of Rwanda and the Center for Geographic Information Systems and Remote Sensing, I extend my gratitude for giving me the opportunity to pursue my studies.

Many thanks to my classmates and all my friends, your friendship and encouragements were a real support throughout the research period.

Special thanks to my family for their love, support and prayers.

Contents

Abstract	i
Acknowledgements	iii
List of Figures	ix
List of Tables	xi
List of acronyms	xiii
1 Introduction	1
1.1 Motivation and problem statement	1
1.2 Research identification	2
1.2.1 Research objectives	2
1.2.2 Research questions	2
1.2.3 Innovation aimed at	3
1.2.4 Related work	3
1.3 Method adopted	3
1.4 Structure of the thesis	4
2 The XML Query Language	7
2.1 Introduction to XML	7
2.1.1 XML Namespaces	8
2.1.2 XPath	9
2.2 XQuery and other XML programming languages	12
2.3 XQuery Data Model	13
2.4 XQuery Formal Semantics	13
2.4.1 XQuery expressions	13
2.4.2 Processing model	17
2.5 XQuery Functions and Operators	19
2.5.1 XQuery Operators	19
2.5.2 XQuery Functions	21
2.6 XQuery implementation: MonetDB/XQuery	21
2.6.1 System architecture	21
2.6.2 Pathfinder XQuery compiler	22
2.6.3 XML Updates in MonetDB/XQuery	25
2.6.4 XQuery functions and extensions	26

3	Extending MonetDB/XQuery with spatial functionality	27
3.1	Geographic Mark-Up Language (GML)	28
3.1.1	Basic concepts of GML	28
3.1.2	GML and other XML languages	30
3.1.3	GML Model	32
3.1.4	GML core and application schemas	35
3.2	Syntax of XQuery extension for GML	36
3.2.1	Data description	38
3.2.2	Spatial queries using XQuery	39
3.3	GEOXML: an extension of XQuery for spatial functionality in MonetDB/XQuery	43
4	Design of a distributed geoprocessing system	45
4.1	Estimating soil erosion risk in Rwanda - The Sebeya watershed	45
4.1.1	Scenario description	45
4.1.2	Universal Soil Loss Equation(USLE)-modified	47
4.2	Requirements definition	52
4.3	System architecture	53
4.3.1	The model	54
4.3.2	The view	54
4.3.3	The controller	54
5	Implementation of a distributed geoprocessing system proto- type	61
5.1	Model implementation	61
5.2	User interface implementation	62
5.3	System prototype functionality	62
5.3.1	Tools and techniques used	62
5.3.2	The soil erosion processing service	64
5.3.3	Visualisation of the results	67
5.3.4	Web service orchestration	68
6	Discussions, conclusions and recommendations	71
6.1	Discussions	71
6.2	Conclusions	73
6.3	Recommendations	73
A		75
A.1	Schema for basic GML objects	75
B	Application schemas of the used GML data (Enschede data)	77
B.1	Application schema for neighbourhood data	77
B.2	Application schema for land use data	78
B.3	Application schema for buildings data	78
B.4	Application schema for roads data	78
B.5	Application schema for risk points data	78
B.6	Application schema for schools data	78

C	Application schemas of Sebeya watershed	83
C.1	Application schema for rainfall data	83
C.2	Application schema for soil data	85
C.3	Application schema for topographic data	85
C.4	Application schema for land cover data	85
C.5	Application schema for watershed boundary data	85
C.6	Application schema for sectors boundaries data	85
D	Web service orchestration	89
	Bibliography	91

List of Figures

2.1	Example of an XML document.	9
2.2	Fragment of an XML document with namespaces	10
2.3	Representation of a document part in the XQuery Data Model .	10
2.4	XML document, rating.xml	16
2.5	Processing Model Overview	18
2.6	XQuery processing stack	22
2.7	Architecture of the MonetDB/XQuery system	23
2.8	Representations tree of an XML document and its encoding . .	24
2.9	XPath semantics and XPath axes	25
3.1	Object-property model	33
4.1	Overview of the system prototype	46
4.2	Processing steps for calculating the annual average of the soil loss	53
4.3	Client HTTP request - Server HTTP Response	55
4.4	The structure of a SOAP message	56
4.5	Remote Procedure Call communication	57
4.6	An example of an XRPC Request message	58
4.7	XRPC Response message	59
5.1	User interface	63
5.2	Process workflow for computing the annual average of soil loss	66
5.3	Visualising the GML results in ArcMap	67
5.4	A fragment of the WSDL document	69
B.1	Application schema for neighbourhood data	77
B.2	Application schema for land use data	78
B.3	Application schema for buildings data	79
B.4	Application schema for roads data	80
B.5	Application schema for risk points data	81
B.6	Application schema for schools data	81
C.1	Application schema for rainfall data	83
C.2	Application schema for soil data	84
C.3	Application schema for topographic data	85
C.4	Application schema for land cover data	86
C.5	Application schema for watershed boundary data	87
C.6	Application schema for sectors boundaries data	88

D.1 BPEL document	89
-----------------------------	----

List of Tables

2.1	An example of a sequence encoding	24
2.2	Loop encoding	25
3.1	Comparison of spatial queries in XQuery and SQL spatial queries	38
4.1	Soil erodibility	50
4.2	C factor scores	51
5.1	Description of the BPD modeling elements	65

List of acronyms

BPD Business Process Diagram
BPM Business Process Management
BPMN Business Process Management Notation
CSS Cascading Style Sheets
DBMS Database Management System
DTD Document Type Definition
eClarus BPMSOA eClarus Business Process Modeler for SOA Architects
FLWOR For-Let-Where-Order by-Return
GeoRSS Geographically Encoded Objects for RSS feeds
GEOS Geometry Engine - Open Source
GIS Geographic Information Systems
GML Geography Markup Language
GML-SF GML Simple Features
GPX GPS Exchange Format
HTML HyperText Markup Language
HTTP Hypertext Transfer Protocol
IRI Internationalized Resource Identifier
ISO International Organization for Standardization
KML Keyhole Markup Language
KMZ Keyhole Markup Language, Zipped
MVC ModelViewController
OGC Open Geospatial Consortium, Inc.
OODBMS Object-Oriented Database Management System
ORDBMS Object-Relational Database Management System
RDBMS Relational Database Management System
RDF Resource Description Framework
RPC Remote procedure call
SDI Spatial Data Infrastructure
SF-SQL Simple Features for SQL
SOA Service-Oriented Architecture
SOAP Simple Object Access Protocol
SQL Structured Query Language
SVG Scalable Vector Graphics
UDDI Universal Description Discovery and Integration
URI Uniform Resource Identifier
USLE Universal Soil Loss Equation
W3C World Wide Web Consortium

WFS Web Feature Service
WSBPEL or BPEL Web Services Business Process Execution Language
WSDL Web Service Definition Language
WWW World Wide Web
XDM XQuery Data Model
XML Extensible Markup Language
XML DBMS XML Database Management System
XPATH XML Path Language
XPointer XML Pointer Language
XQuery XML Query Language
XRPC XQuery Remote procedure call
XSLT Extensible Stylesheet Language Transformations

Chapter 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 [KGH07]. In such cases, users may need to make use of distributed geoprocessing services available through the web.

1.1 Motivation and problem statement

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,¹ ORDBMS,² OODBMS³) [PRT06, SYU99], each node that receives the data in GML format, has to parse and

1. Relational Database Management System.
2. Object-Relational Database Management System.
3. Object-Oriented Database Management System.

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 [LBTR04], GML documents can be stored and manipulated in an XML DBMS⁴ (such as MonetDB/XQuery⁵) [BGvK⁺06] without any need of data conversion.

Furthermore, using those conventional databases to store XML documents requires restructuring them into either tables (for RDBMS, ORDBMS) or classes (for OODMS) which makes them lose the flexibility, heterogeneity and extensibility of XML [ABS00, CRZ03]. 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.

1.2 Research identification

The foremost issues addressed in this research project can be defined through the following research objectives and research questions.

1.2.1 Research objectives

1. To realize a number of geoprocesses (spatio-analytic functions) applied on GML data sources stored in an XML database.
2. To optimize the provision of distributed geoprocesses (over the web) through service orchestration.
3. To build a web application able to provide the implemented geoprocesses in a distributed environment.

1.2.2 Research questions

1. Can an XML DBMS be effectively used to manipulate the amount of spatial data involved in a typical geoprocessing workflow?
2. How to optimize the geoprocesses on GML data sources stored in an XML database?
3. How to design a geoprocessing service orchestration to implement faster web geoprocessing applications?

4. XML Database Management System.

5. A fast XML database that fully supports the World Wide Web Consortium XQuery language.

1.2.3 Innovation aimed at

This research has the aim of optimizing the efficiency of web geoprocessing of GML data sources, using an XML database as a back-end. We will implement a number of spatial functions to be applied on GML data and design a service orchestration to support the efficient provision of spatio-analytic functions as distributed geoprocessing services over the Internet.

1.2.4 Related work

A number of studies has been carried out and recommended several approaches to tackle the problem of geoprocessing service orchestration and the problem of 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 [KGH07] 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. [HCDL06] 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 [Kie06] 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 geodata. In [FCOLB07], an architecture for distributed geoprocessing was proposed based on OGC service specifications. Foerster and Schaffer [FS07] built a client for distributed geoprocessing on the web to demonstrate its applicability for a real time risk management scenario (on top of uDig⁶).

In this research, we will study and implement a number of spatial functions in MonetDB/XQuery to perform geoprocesses and provide those geoprocesses on the web as service. Considering that MonetDB/XQuery is a fast and scalable XML database, we will take advantage of its efficiency in managing XML documents to optimize and quicken the implemented geoprocessing services which have to be applied on GML data sources, as parsed and stored in MonetDB/XQuery [BGvK⁺06, BQK96].

1.3 Method adopted

To carry out our research project, we will follow the five core workflows of a software development process as described in [JBR99]: requirements definition, analysis, design, implementation and test. For the requirements definition, we will start by studying MonetDB/XQuery database system to understand its current functionality, and then specify a number of spatio-analytic functions required for a given sample scenario. And later, (during the implementation) these functions will be realized following OpenGIS standards and

6. User-Friendly Desktop Internet GIS, <http://udig.refrains.net/>

specifications to address the problems of interoperability in a distributed environment. During the analysis, design and implementation, we will make use of the Service Oriented Architecture (SOA) [Jos07] approach to support the organization and utilization of the distributed functions and the distributed data. Each computation node, in our distributed system, will be equipped with the MonetDB/XQuery system and some spatio-analytic functions and/or spatial data that are accessible over the web.

Depending on the scenario that is chosen and described, the first node may be equipped with spatial data on a specific theme such as land use and some spatio-analytic functions. The second node may be providing spatial data on risk points (for example, fireworks industries or gas stations). An end-user, who needs information about risky areas, may apply the functions provided by the first node on the data about the risk points data (stored on the second node), then send the resulting information to another node where they will be compared with the data on land use (stored in first node) to obtain more information on those risky areas. And finally, the user gets the resulted information returned to his/her node. Note that the comparison is also made by a function provided by one of the nodes in the distributed system. Thus, given a number of spatio-analytic functions and data available in a distributed environment, we will define a proper workflow of different geoprocesses that need to be carried out to ensure that the desired output information is sent back to the user, specifying where and when each process takes place.

SOA will allow us to use the Web Services Description Language (WSDL) to describe service interfaces and the Business Process Execution Language (BPEL) to orchestrate services “providing the ability to call services, process responses and handle process variables, control structures and error” as stated in [Jos07]. During the test, the performance of the realized spatio-analytic functions on GML data sources will be evaluated and a sample web application will be built to inquire into performance findings on orchestrating the service chain of the implemented processes.

1.4 Structure of the thesis

The thesis is structured in six chapters:

Chapter 1 introduces the research, describing the research problem, the motivation and the objectives of the research.

Chapter 2 provides the basic concepts of XQuery and describes an implementation of XQuery, MonetDB/XQuery.

Chapter 3 describes how to extend MonetDB/XQuery with spatial functionality. It provides the key concepts of GML, then describes the syntax proposed for spatial queries applied on GML data using XQuery. The implementation of GEOXML, an extension of XQuery for spatial functionality, in MonetDB/XQuery is also described in this chapter.

Chapter 4 discusses the design of our system prototype taking into account the chosen scenario, a model for soil erosion assessment. This soil erosion model is also described in this chapter, and all processes required to realize it are determined.

Chapter 5 describes the implementation of the system prototype. The processes, determined in Chapter 4, are modeled and implemented as Web services.

Chapter 6 concludes the report of the research, outlining the achievements in regards with the innovation aimed at and recommendations for further research.

Chapter 2

The XML Query Language

The XML Query Language (XQuery) is a language for processing XML data. It is built on XPath (XML Path Language) [W3C07b] expressions. XML is widely used to represent different kinds of data from different sources, because of its extensibility and flexibility. 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. 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 [W3C07c]. It is an extension of the specifications of XPath 2.0 and is based on the XQuery 1.0 and XPath 2.0 Data Model (XDM) [W3C07d], which defines the input information of an XQuery processor, and all values of expressions allowed in the XQuery language. This chapter contains the basic concepts of XQuery. An introduction to XML is given in Section 2.1. Section 2.3 and Section 2.4 recapitulate the data model and the formal semantics of XQuery, respectively. XQuery functions and operators are described in Section 2.5. Finally, an implementation of XQuery (MonetDB/XQuery) is presented in Section 2.6.

2.1 Introduction to XML

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. With XML, one can freely define tags to enclose the data within a document, and these tags give out a small description of the data they enclose. The tags within an XML document can be nested without any restraint,

hence the flexibility of XML. For instance Figure 2.1 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 and that they are nested up to the different levels. (Refer to our example in Figure 2.1; the last pair of building tags do not enclose any information about the architect, whereas all other pairs of building tags are nested up to the first name and last name tags).

The XML tags are used to define XML elements, which means that the opening and closing tags, including everything enclosed within these tags, characterize an XML element. Even though XML allows to create one's own tags, it has syntax rules to determine if an XML document is well-formed, and semantic rules to check if the document is valid. An XML document is well-formed if it has a single root element, all tags come in opening/closing pairs and are properly nested (they should not overlap), the names of tags are case sensitive (<City> is different from <city>), and the attribute values must be quoted. For instance, *year* is an attribute of the *city element*, so its value should be quoted as follows: *year*="1959". An XML element without any child element other than attribute can be represented by one closed tag. For instance, if the *city element* had no child elements, it could be written as follows: <city name="Leicester"/>. The root element, also called the document element, is the element that encloses all other elements within an XML document. An XML document is valid if it conforms to a specified *Document Type Definition* (DTD) or an *XML Schema* [W3C04b, W3C04c, W3C04d].

2.1.1 XML Namespaces

XML Namespaces [W3C06] are mainly used to avoid the conflicts of XML element (tags) and attribute naming. There are no predefined tags in XML, therefore each developer has to name her/his tags and attributes at will and this can cause conflicts of names. Considering our example represented in Figure 2.1, in this case the element *title* describes the title of a building but this name can be used to describe the title of a book also. So, XML namespaces should be used to enable an application to distinguish a title of building from a title of a book. An XML Namespace is declared by binding a namespace prefix to an Internationalized Resource Identifier (IRI) that uniquely identifies that namespace. To declare a namespace, the attribute *xmlns* should be used within the opening tag of an element following this syntax: *xmlns:prefix*="IRI". The IRI used in the namespace declaration is a Uniform Resource Identifier (URI) that does not point to any resource, it is just used as unique identifier of the namespace. As an example, we can declare a namespace for our example (Figure 2.2). Note that the Figure 2.2 shows a small fragment of an XML document where XML namespaces are used.

In this case, the namespace was declared within the root element name (cityBuildings), but this is not compulsory. A namespace can also be declared in any XML element but its child elements and attributes are the only ones that can be associated with the declared namespace. The names of XML elements and attributes that conform to the XML Namespaces specification are called *qualified names* [W3C06].

```
<?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>William Rutherford</last>
        <first>Mead</first>
      </architect>
      <architect>
        <last>Stanford</last>
        <first>White</first>
      </architect>
    </building>
  </city>
  <city name="Moscow">
    <building year="1554">
      <title>Cathedral St. Basil</title>
      <architect>
        <first>Barma</first>
      </architect>
      <architect>
        <first>Posnik</first>
      </architect>
    </building>
    <building year="1927">
      <title>Cathedral St. Basil</title>
      <architect>
        <last>Konstantin</last>
        <first>Melnikov</first>
      </architect>
    </building>
  </city>
</cityBuildings>
```

Figure 2.1: Example of an XML document.

2.1.2 XPath

XPath was developed to be embedded in other languages such as XSLT or XQuery or XPointer (XML Pointer) and it is based on the same data model as XQuery. It provides the path expressions to navigate through an XML documents as well as a number of built-in functions to manipulate string values,

```
<?xml version="1.0" ?>
<cityBuildings xmlns:bdg="http://www.itc.nl/buildingExample/">
  <bdg:city bdg:name="Leicester">
    <bdg:building bdg:year="1959">
      <bdg:title>Leicester University Engineering Building</bdg:title>
      <bdg:style>Modern </bdg:style>
      <!--this is a research university based in England-->
      <bdg:architect>
        <bdg:last>Stirling</bdg:last>
        ...
```

Figure 2.2: Fragment of an XML document with namespaces

numeric values, nodes, sequences, and more, as described in [W3C07b]. 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 2.1, is illustrated in Figure 2.3.

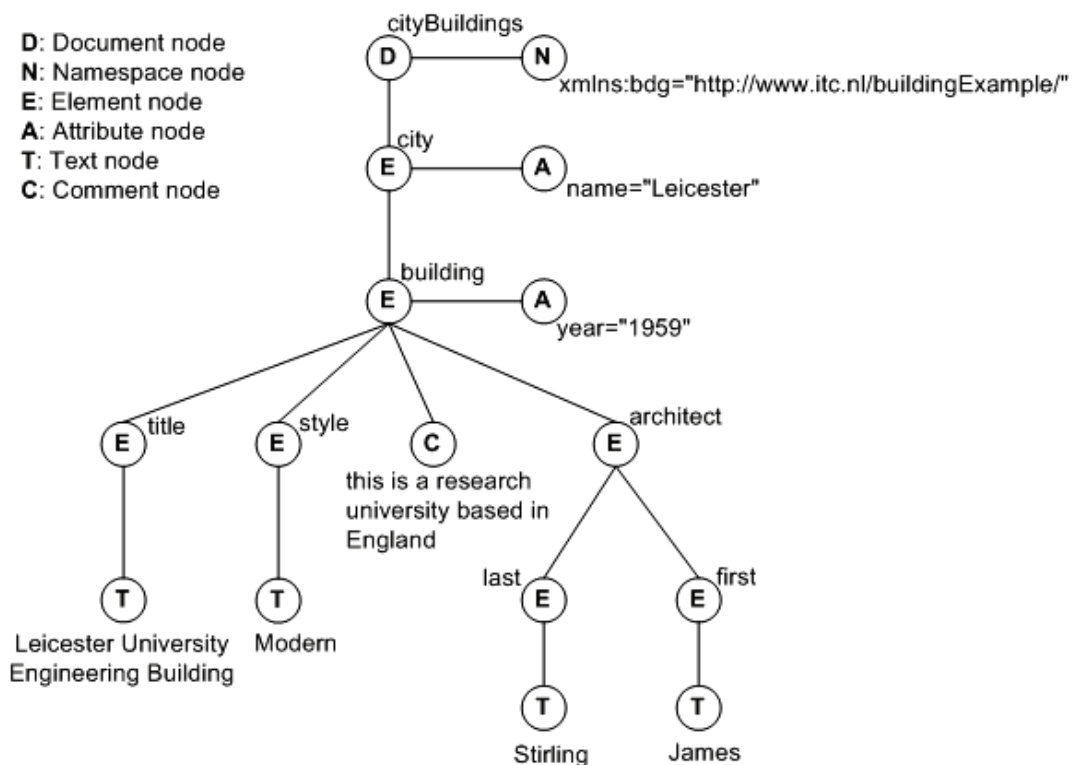


Figure 2.3: Representation of a document part in the XQuery Data Model (adapted from (KCK⁺03))

Document node: Considering that an XML document can be represented as a tree of nodes, the document node (also called root node) is the unique root

of that tree. The root node can only have the element, text, processing instruction and comment nodes as children and it should have at least one child. A well-formed XML document must have a document node that contains all other nodes defined within that document.

Element node: An element node can have child nodes of type element, processing instruction, text and/or comment. Note that attributes and namespaces are not considered as children of their parent elements even though the latter are the parent of the attributes or namespaces they own.

Attribute node: Each attribute node is owned by an element node called its *parent*. An attribute node can not have children *and is not considered as the child of its parent element*.

Text node: A text node characterizes the content of its parent node. The parent node of a text node can be either a document node or an element node.

Namespace Node: A namespace node represents the binding of a namespace prefix to a URI as described in Section 2.1.1.

Processing instruction node: A processing instruction node represents the XML processing instructions and is written using the following syntax: `<?XML processing instruction?>`. For instance, the processing instructions may consist of a reference to the stylesheet used within an XML document, as shown below:

```
<?xml-stylesheet type="text/xml" href="style.xsl" ?>
```

Comment node: The comment nodes recapitulate the XML comments and are written as follows: `< -- XML comments -- >`. They serve only to guide the human reader.

All nodes with no parent or child node are called *atomic values*, such as *Leicester* or *Cathedral St. Basil* from the example in Figure 2.1. An *item* is an atomic value or a node and a *sequence* is an ordered collection of items. All the nodes within a document relate with each other and their relationships depend on the document order. 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 parents parent, and so on. The descendants of a node consist of the children of that node, children of the children, and so on.

Path expressions are used to navigate through XML documents and they commonly start at the root node. Here is a list of expressions used in XPath :

- to select all the child nodes of a particular node with the name of *node-name*: `nodename`
- to select all nodes from the root node: `/`

- to select the nodes from the current node, regardless of their level in the document hierarchy: //
- to select the current node: .
- to select the parent of the current node: ..
- to select an attribute of an element node: @attributename
- to select all element nodes without specifying their names: *
- to select all attribute nodes without specifying their names: @*
- to select the child nodes of a particular node *nodename* in accordance with the predicate *pred*: *nodename*[*pred*]

XPath uses predicates to retrieve nodes depending on their values, and these are embedded in square brackets. Here are some sample path expressions applied on our XML document represented in Figure 2.1. The last three expressions use predicates:

- */cityBuildings/city/building* returns the building elements (that are child nodes of city elements).
- *//building* returns all building elements that are descendants of the root node, so also those reachable along other paths.
- *//building/@year* returns the attribute values of *year* of any the building element.
- *//city/** returns all child nodes of any city element.
- *//@** returns all the attribute values in the document.
- *//city[@name = 'Moscow']/building* returns all building elements, that are child nodes of the city elements whose name attribute has the value Moscow.
- *//city/building[1]* returns every first building element that is a child node of a city element.
- *(//building)[1]* returns the first building element in the whole document.

2.2 XQuery and other XML programming languages

As already mentioned, 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. XPath was designed to be used as part of other languages such as XQuery or XSLT (EXtensible Stylesheet Language Transformations) [W3C07g]. Although these two languages have XPath in common, they are different and each of them relates with XPath in its own

way. XSLT is a language that was mainly designed to describe XML document transformations; it uses XPath for navigating through an XML document to find information from the source document and transform that into a result document. XQuery extends XPath, providing more flexibility to handle complex tasks such as record-selection that may require recursion. Despite XSTL and XPath, the design of XQuery makes use of other XML standards such as XML Namespaces, as we discussed in Section 2.1.1, and XML Schema [KCK⁺03].

2.3 XQuery Data Model

XQuery operates on the XQuery data model. As a data model is an abstract model that specifies the portrayal of data and their accessibility, the XQuery Data Model (XDM) defines conceptually the structure of XML documents which are used as input and output to an XQuery processor. Therefore, XQuery is closed with respect to XDM as it also outlines valid expressions and operators in XQuery. XDM is based on previously existing XML Infoset specifications [W3C04a] which define the content of a well-formed XML document as an information set consisting of a number of information items [W3C07d, KCK⁺03]. XDM has a form of node hierarchy and represents every value, also called XDM instance, as an ordered sequence of items (nodes or atomic values). An atomic value may consist of one of the atomic data types defined by XML Schema like string, boolean, decimal, float, double, date, or time. Figure 2.3 illustrates the representation in the XQuery data model of a fragment of the document *buildings.xml*, shown in Figure 2.1.

2.4 XQuery Formal Semantics

The XQuery Formal Semantics provides a mathematical meaning to each of the XQuery expressions through the XDM. 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 [KCK⁺03, W3C07e].

2.4.1 XQuery expressions

XQuery is a programming language based on “expressions”, in which each query consists of either a simple expression or a composed expression. The language has various kinds of expressions that can be combined by operators to constitute new expressions. The compositionality of XQuery also allows us to use

the result of an expression as an argument of another expression, as already mentioned above. 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 [W3C07e]. We discuss all of these below.

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 that correspond to XML schemas simple types `xs:integer`, `xs:decimal` and `xs:double`, respectively. The string literals are of type `xs:string` and are delimited by quotation marks or apostrophes. A string literal delimited by quotation marks can contain apostrophes and if it is delimited by apostrophes, it can contain quotation marks [W3C07c, KCK⁺03]. Below are examples of literals:

- 3
- -3.5
- +3.2e5
- "string example"
- 'this is another string'
- "it's just another string"
- 'another "string" example'

Path expressions

The path expressions are used to navigate through the tree structure of XML documents to locate nodes of interest. For example, 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 2.1), and find the element nodes `title`. See Section 2.1.2 XPath for more examples; as XQuery's path expressions are derived from the path expressions of XPath 2.0 (as stated in [KCK⁺03]).

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 and sorting

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)}</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>3</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. The *order by clause* is used for sorting the results. Here is an example of a query that returns the titles of the buildings, sorted by the first name of the architect:

```
for $b in doc ("buildings.xml")//building
let $a := $b//first
order by $a
return $b/title
```

The *for clause* provides the possibility of a positional variable using the *at clause*. The positional integer-valued variable can be used to identify the position of an item in the sequence of the iteration. The following query returns the titles of the building whose positions are odd numbers:

```
for $b at $i in doc ("buildings.xml")//building
where $i mod 2 = 1
return $b/title
```

In addition to iteration and the use of a positional variable, the *for clause* allows to perform joins in XQuery, combining information from different XML sources (XML documents). Let us consider another XML document named *rating.xml* (in Figure 2.4) that contains the comments about and ratings of the buildings. Below is a query that returns the title of the building, the comments


```
<?xml version="1.0" ?>
<rating>
  <annotation>
    <reviewerNames>John Kyle</reviewerNames>
    <bTitle>American Academy in Rome</bTitle>
    <rate>4</rate>
    <comments>This is a pretty building</comments>
  </annotation>
</rating>
```

Figure 2.4: XML document, rating.xml

given to it and the names of the reviewer who rated it. Here, the *where clause* is used to filter the result, matching the building titles from *buildings.xml* with those from *rating.xml* and finally retain only the one that satisfies the condition.

```
for $b in doc ("buildings.xml")//title,
    $r in doc ("rating.xml")//annotation
where $b = $r/bTitle
return
  <rating>
    {$b, $r/comments, $r/reviewerNames}
  </rating>
```

The above example represents one way of nesting the FLWOR expressions and can also be written as follows:

```
for $b in doc ("buildings.xml")//title
for $r in doc ("rating.xml")//annotation
where $b = $r/bTitle
return
  <rating>
    {$b, $r/comments, $r/reviewerNames}
  </rating>
```

Or, alternatively as:

```
for $b in doc ("buildings.xml")//title
return
  for $r in doc ("rating.xml")//annotation
  where $b = $r/bTitle
  return
    <rating>
      {$b, $r/comments, $r/reviewerNames}
    </rating>
```

Element constructors

In the above example, the element constructor is used to create the element node “rating” in the return clause. The element constructor expressions have the following syntax:

```
<createdElementNodeName>child text node</createdElementNodeName>
```

or

```
element createdElementNodeName
{
  "child text node"
}
```

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.

It is of great value to formally describe the semantics of a language such as XQuery for its implementers as well as for its users, as it ensures the precision of its expressions, describing their meaning with mathematical rigor. XQuery Formal Semantics has three main components: dynamic semantics, static semantics and normalization rules, which are all discussed in the next section. The description of relationships between XQuery objects such as expressions, XML values and types reflect the meaning of the language if those objects are represented using formal notations. The dynamic semantics describes the relationships between an XQuery expression, its input data and the XML values returned as the output of that expression. The static semantics specifies the relationships between an XQuery expression, the type of input data, and type of the inferred output XML values.

2.4.2 Processing model

The XQuery formal semantics describes a query processing model composed of four phases: parsing, normalisation, static analysis and dynamic evaluation. Every phase takes the output of the previous phase as its input. This processing model provides an ideal representation of an XQuery implementation as depicted in Figure 2.5. In XQuery, a query consists of one or more modules and a module is made of a prolog and a query body. The query prolog contains the processing instruction nodes that provide information needed during the query processing. This information may comprise the declaration of XQuery version, the namespace declarations, the imports of modules, the imports of schemas,

the definitions of functions, ... After every declaration or definition or import there must be a semicolon, as follows:

```
declare namespace bdg = "http://www.itc.nl/buildingExample/";
```

The above is an example of a namespace declaration as it may occur in a query prolog; with *bdg* as the prefix and *http://www.itc.nl/buildingExample/* as the URI.

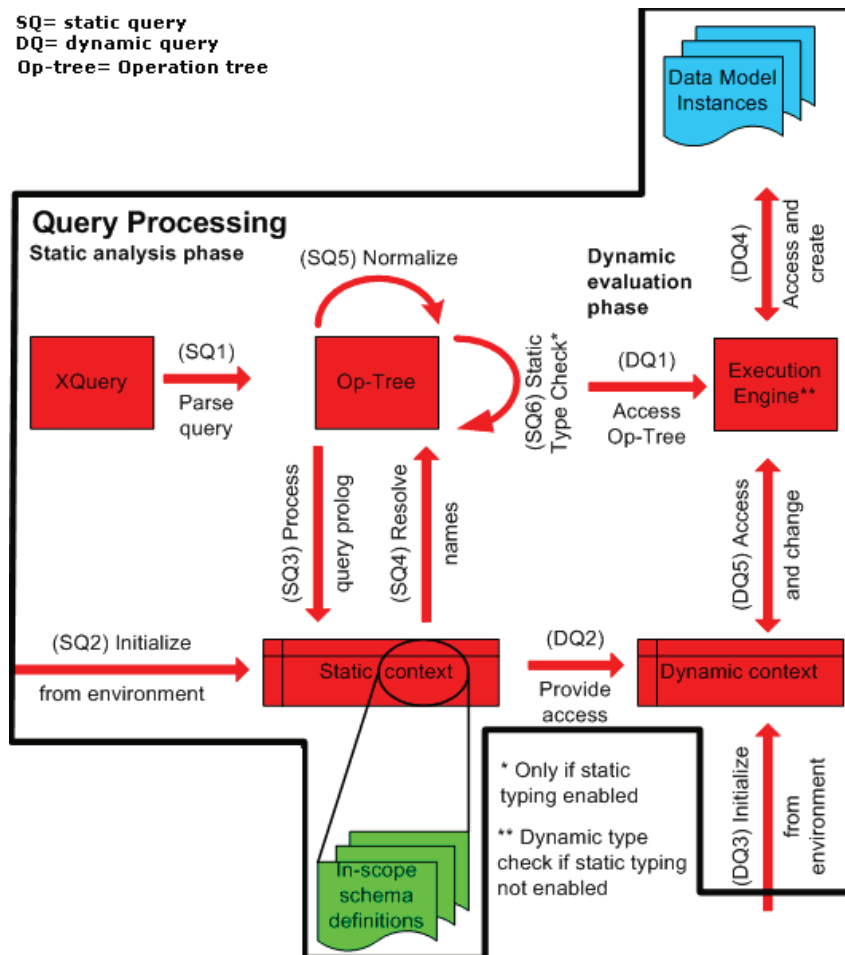


Figure 2.5: Processing Model Overview (adapted from (W3C07e))

Parsing: The parsing phase checks for syntax errors in an input expression (query) and, either parsing errors are raised if detected or the expression is parsed into an internal representation called an operation tree (SQ1 in Figure 2.5, where SQ stands for Static Query). The information to be used for the evaluation of the expression is initialized and then edited by the declarations drawn from the query prolog (SQ2, SQ3 and SQ4 in Figure 2.5).

Normalisation: Normalization simplifies the operation tree, by applying XQuery

normalization rules, and returning a corresponding operation tree in the Core XQuery language (SQ5 in Figure 2.5).

Static analysis: This phase is optional and may be skipped in an XQuery implementation. The static analysis phase only evaluates the type of the expressions (the operation-tree returned from the normalization phase) defined in the core language. If the expression is not well-typed, an error is raised; otherwise a type is assigned to the evaluated expression (SQ6 in Figure 2.5).

Dynamic evaluation: After the static analysis is completed, the value of the expression is computed and is returned as an XML value (DQ1, DQ2, DQ3, DQ4 and DQ5 in Figure 2.5, where DQ stands for Dynamic Query). Or, a dynamic error is returned if such is detected.

2.5 XQuery Functions and Operators

XQuery provides a number of functions and operators, some of which are commonly used in other languages. XQuery has arithmetic operators, comparison operators and sequence operators. In addition to operators, XQuery has built-in functions and also supports user-defined functions. While using operators and functions, sometimes XQuery expressions require to apply the concepts of atomization or typed value extraction for their processing. The typed value extraction may occur while comparing an element with an expression of another data type like string or integer. The query below illustrates the use of typed value extraction, that is applied in the expression “city[@name = ‘Moscow’]” where an attribute node “name” is compared with a string “Moscow”. This occurs because of the operator “=” that extract the typed value of the attribute (which is string) before comparing it with the string Moscow.

```
doc(buildings.xml)//city[@name = 'Moscow']/building
```

Atomization is another way of typed value extraction applied to sequences of items to convert them into sequences of atomic values. (See definition of *atomic value*, *item* and *sequence* in Section 2.1.2). For example, the following expression returns the integer 12:

```
sum ( (3, <x>4</x>, <y>5</y>) )
```

The atomization is applied on the sequence “(3, <x>4</x>, <y>5</y>)” to extract atomic values of type xs:integer 3, 4 and 5 before their addition. Note that XQuery has a formal function for extracting typed value, fn:data() [W3C07f], for instance fn:data(<x>3</x>) returns 3.

2.5.1 XQuery Operators

In addition to arithmetic and comparison operators, XQuery also provides operators handle sequences of nodes, considering that they are one of the basic XQuery data types.

Arithmetic operators: Arithmetic operators supported in XQuery are addition, subtraction, multiplication, division, integer division and modulo arithmetic represented by the symbols `+`, `-`, `*`, `div`, `idiv` and `mod`, respectively. Besides the `idiv` operator, all other arithmetic operator in XQuery operate as in other languages. The `idiv` operator takes operands of type `xs:integer` and returns a result of type `integer` as well, it applies rounding off whenever necessary.

Comparison operators: Comparison operators in XQuery can be classified into three categories: general comparison operators, value comparison operators and node comparison operators. They all take two operands and return a boolean result.

- The general operators comprise `=`, `!=`, `<`, `<=`, `>` and `>=` and take sequences of atomic values as operands. The general comparison operators compare two sequences regardless their length, thus to evaluate an expression, they check if there is at least one item in both sequences that satisfy the comparison.
- The value comparison operators comprise `eq`, `ne`, `lt`, `le`, `gt` and `ge` and accept operands only if they are single atomic values.
- The node comparison operators comprise `is`, `is not`, `<<` and `>>` and compare two (single) nodes. The operators `<<` and `>>` determine whether a node precedes or follows another in the document order. The operators `is`, `is not` determine if two nodes have the same identity or not. Note that each node has its unique identity assigned to it automatically by the system, therefore the expression (2.1) evaluates to *false* as the two constructed nodes have different identity. On the other hand, expression (2.2) evaluates to *true*.

`<x>4</x> is <x>4</x>` (2.1)

```
let $a:=doc("buildings.xml")//city[@name='Rome']//title
let $b:=doc("buildings.xml")//building[@year="1913"]//title
return $a is $b
```

(2.2)

Sequence operators: The sequence operators are *union*, *intersect* and *except* and are used to combine sequences of nodes. They all take two operands. The *union* operator returns a sequence of all nodes found in either operand (to apply this operator, one can use the keyword *union* or `|`). The *intersect* operator returns a sequence of nodes found in both operands. The *except* operator returns a sequence of nodes found in the first operand but not in the second operand.

2.5.2 XQuery Functions

Built-in Functions

XQuery has built-in functions that operate on XML values of types such as string, decimal, integer, date, time, node, boolean ... The full list of these functions and their descriptions can be found in XQuery 1.0 and XPath 2.0 Functions and Operators specifications [W3C07f]. XQuery built-in functions are bound to the prefix *fn*, the default prefix for function namespace. The prefix *fn* can be omitted, for instance the functions *fn:sum()*, *fn:avg()*, *fn:string-length()* can be written just as *sum()*, *avg()*, *string-length()*.

User-defined Functions

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 defined functions can be put in a module, which in return can be imported by a query to access its functions [KCK⁺03]. 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 )
```

2.6 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 [Teu00, BGvK⁺06].

2.6.1 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) [Teu00, BGvK⁺05]. Pathfinder was used to link the relational processing model with the XQuery Data Model following the XQuery processing stack shown in Figure 2.6.

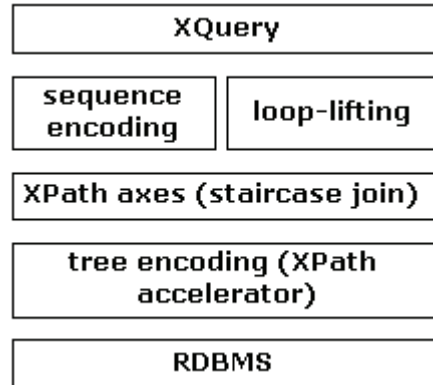


Figure 2.6: XQuery processing stack (adapted from (BGvK⁺05))

MonetDB/XQuery allows to store XML documents and manipulate them using XQuery. Its architecture (shown in Figure 2.7) 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.

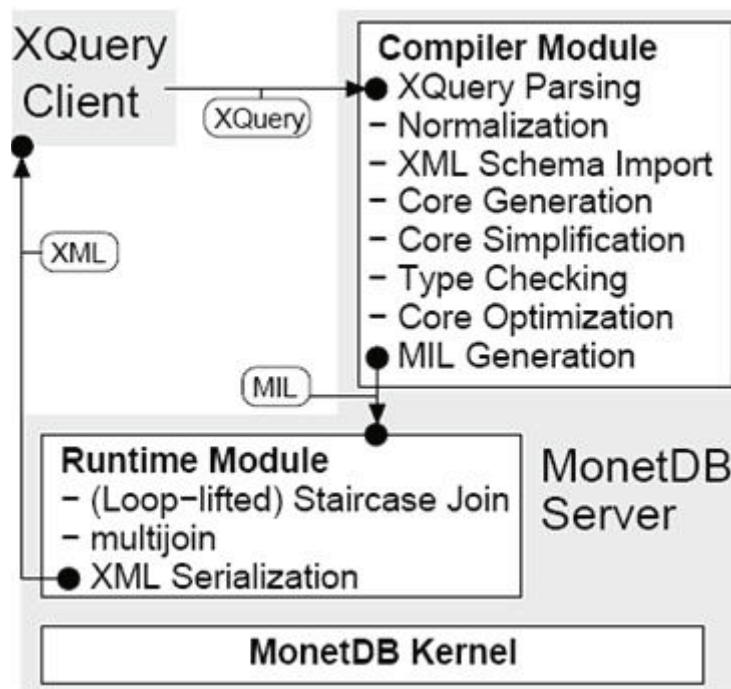
2.6.2 Pathfinder XQuery compiler

The Pathfinder XQuery compiler supports almost the full XQuery standard, closely following the W3C XQuery Formal Semantics as stated in [BMR05]. To implement the Pathfinder, the following techniques were used: XPath accelerator described in [Gru02], the staircase join and the loop-lifting compilation procedure.

XML document tree coding

Before processing an XML document by a relational engine, it should first be encoded relationally. Pathfinder uses the XPath accelerator approach to encode the XML tree document. This approach is based on preorder and postorder traversal ranks, which are used to encode the position of a given node in a document tree. The encoded position is saved in a table as a triple, for example, the node *f* of the document tree presented in Figure 2.8, is represented by the triple $\langle \text{pre}(f), \text{size}(f), \text{level}(f) \rangle$.

- $\text{pre}(f)$ denotes the number of nodes visited before the node *f* (five nodes: *a*, *b*, *c*, *d* and *e*).
- $\text{size}(f)$ denotes the number of nodes in the subtree below *f* (one node: *g*)

Figure 2.7: Architecture of the MonetDB/XQuery system (from (BGM⁺05))

- $level(f)$ denotes the distance from the root to the node f (two).

The post order rank denoted as $post(f)$ for the node f , depicts the number of nodes traversed, after all its child nodes have been traversed. And it can be computed by

$$post(f) = pre(f) + size(f) - level(f)$$

(as shown in Figure 2.8). So, $post(f)$ is 4 because there are four nodes traversed after all child nodes of f are traversed (d , e , c and g), f itself is not counted. This $pre|size|level$ table does not only contain the encoding of the nodes position but it also has additional columns describing other properties of each node such as node kind, text content [BGvK⁺06, BGvK⁺05].

Staircase join

Encoding the node position accelerates the XPath evaluation, but to gain additional performance, this encoding is associated with the XPath axes concept defined by XPath semantics. The four axes used in accelerating XPath evaluation [Gru02] are: descendant, ancestor, following and preceding (shown in Figure 2.9 with f as the context node). Figure 2.9 shows how the four axes correspond to four quadrants in the post/pre plane for a given context node f . This matching brings out more knowledge about the tree representation and may be used to express other XPath axes. The staircase join operator [Gru02] is used to exploit and encapsulate this tree knowledge present in the pre/post plane, and this speeds up the XPath evaluation.

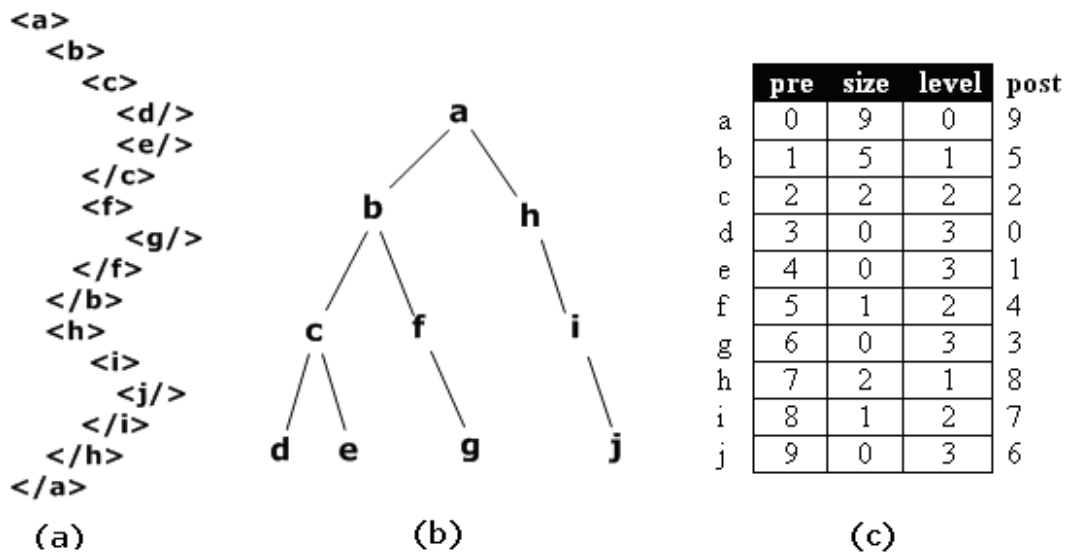


Figure 2.8: Representations tree of an XML document (a,b) and its encoding (c)

Sequence encoding

The XQuery Data Model is based on the XML document tree and sequence of items (ordered list of items). The sequence should also be encoded into a relational model; the position of each item is captured into the column `pos` and the item into the polymorphic column `item`. For example a sequence $(\text{"y"}, 3, \langle x / \rangle)$ is encoded as shown in Table 2.1. MonetDB used as the back-end, implements only monomorphic columns however it supports also this polymorphic data as detailed in [BGM⁺05].

Table 2.1: An example of a sequence encoding

pos	item
1	"y"
2	3
3	<x/>

Loop lifting

One of the most powerful expressions of XQuery is the FLWOR expression that allows evaluating iteration over a sequence of items. The technique of loop-lifting is used to evaluate the FLWOR expression by a relational processor, and the iteration is recorded in the column `iter`, added to the Table 2.1 of the sequence encodings. For instance, the expression: *for* $\$v$ *in* $(x_1, x_2, x_3, \dots, x_n)$ *return* e is encoded as shown in Table 2.2. The compilation of these FLWOR expressions for relational processing uses relational algebra. In addition to the optimization techniques mentioned above, Pathfinder XQuery Compiler also uses

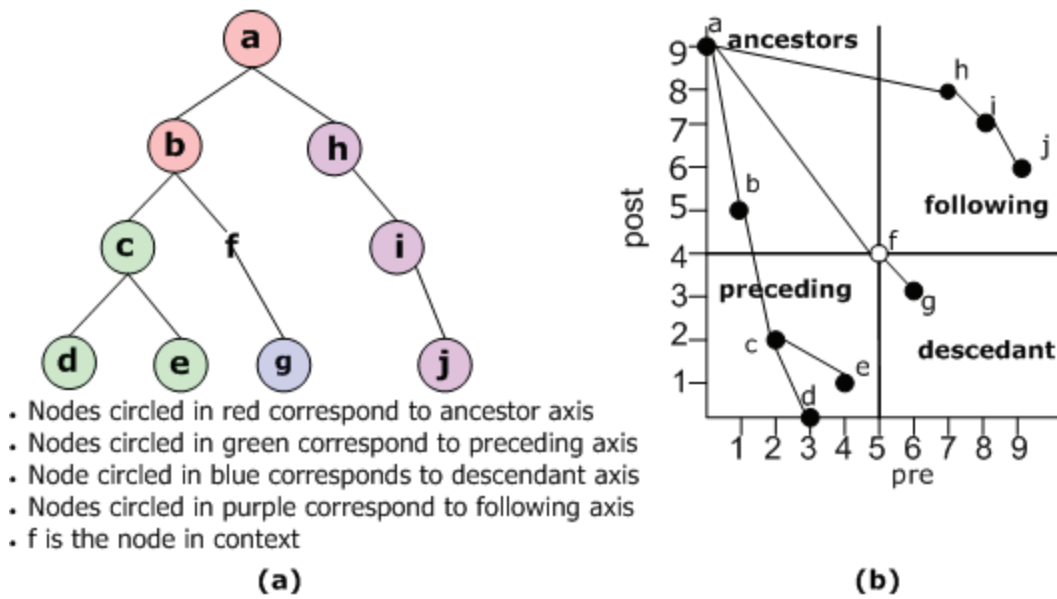


Figure 2.9: XPath semantics (a) and XPath axes in the pre/post plane (b) with f as the node in context

the XML schemas import and static type checking during query processing.

Table 2.2: Loop encoding

iter	item	item
1	1	x_1
2	1	x_2
3	1	x_3
⋮	⋮	
⋮	⋮	
⋮	⋮	
n	1	x_n

2.6.3 XML Updates in MonetDB/XQuery

XML updates can consist of value updates and structural updates [BGvK⁺06]. The value updates include the node value update (namely text, comment or processing instructions) and attribute value changes. The structural updates delete or insert nodes in a XML document. MonetDB/XQuery supports the W3C XQuery Update Facility (XQUF) specifications [W3C08b], but does not support yet the transform feature and uses the syntax *do rename ... into* instead of *do ... rename ... as* for renaming.

2.6.4 XQuery functions and extensions

MonetDB/XQuery uses XML documents as the basic unit of data storage and allows storing them in collections. A collection should contain at least one document, it is automatically deleted when its last document is deleted. MonetDB/XQuery provides a number of built-in functions recommended in the W3C specifications as well as various non-W3C recommended built-in functions (a list of all supported functions can be found in MonetDB/XQuery Reference Manual [Mon08]). The W3C XQUF is also supported in MonetDB/XQuery but with some restrictions mentioned in Section 2.6.3 above.

Among the functions that are supported in MonetDB/XQuery, are the documents management functions, PF/Tijah (Pathfinder/Tijah, pronounced as “Pee Ef Teeja”) [Pat08] functions that allow to create text indices on XML documents stored in MonetDB/XQuery, used for text search querying. MonetDB/XQuery has a built-in HTTP server that serves out the documents stored in the database, which allows building a website on top of MonetDB/XQuery and which can be used by XRPC requests. XRPC (see Section 4.3.3) is another XQuery extension that adds the concept of Remote Procedure Call (RPC) to XQuery functions. XRPC is fully implemented in MonetDB/XQuery and allows efficient and interoperable distributed queries (using modules or user-defined functions). To execute a remote function (user-defined), the following syntax is used:

```
execute at { Expr } { Fun(ParamList) }
```

with *Expr* denoting the XQuery expression that indicates the Uniform Resource Identifier (URI) of the server on which the function *Fun* is to be executed, and *ParamList* as the list of parameters passed to the function *Fun*.

Chapter 3

Extending MonetDB/XQuery with spatial functionality

MonetDB/XQuery is an extensible XML database that supports a number of XQuery features and a wide range of functions described in W3C specifications and non-W3C recommended functions (as described in Section 2.6). 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 [CG01] 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. Vatsavai [Vat02] and Jihong [Jih06] also proposed specifications of a spatial query language for GML documents, GML-QL and GQL, respectively.

In our research, we propose a syntax of XQuery for GML, based on its implementation in MonetDB/XQuery and on previous researches mentioned above. This is detailed in Section 3.2, after providing the key concepts of GML in Section 3.1. After that, GEOXML, an XQuery extension, is developed and implemented in MonetDB/XQuery as described in Section 3.3. For the time being, GEOXML consists of a few basic spatial functions that can be applied on GML features to provide limited spatial functionality. GEOXML is to be improved later in terms of completeness of functions and efficiency.

3.1 Geographic Mark-Up Language (GML)

GML is an XML-based language used to encode geographic objects for their transport and storage. The geographic objects representing the real-world objects are called features in GML, and their encodings contain their spatial and non-spatial information. GML was designed by OGC to support the interoperability, allowing geographic information exchange and sharing over the Web [Ope07]. As an XML-based language, GML is extensible, human-readable and separates data from its presentation. It describes a standard way of capturing geometry and non-geometry properties of geographic features between tags, providing a basic and restricted data model, the GML Simple features profile [Ope05]. The GML Simple features profile does not prevent GML users from using their own feature types but rather provides them a restricted framework (based on GML core schemas) to support them while defining their own feature types (GML application schemas). It should be pointed out that GML is an ISO standard (ISO 19136:2007). Though it is not the only way to represent geographic objects using XML technology, there is another language called Geospatial-eXtensible Markup Language (G-XML) developed by the Database Promotion Center (DPC) in Japan [Dat06].

3.1.1 Basic concepts of GML

GML objects

GML can be used to integrate spatial and non-spatial data for representing real-world objects, that means the geographic features, though not all GML objects are features. GML objects include the following:

- **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) [LBTR04]. For example, to create a *Building* instance, one may use the following syntax:

```
<Building gml:id = "id01">...</Building>
```

The feature properties are child elements of its instance node, which can be defined as follows:

```
<Building gml:id = "id01">
  <name>Four Times Square</name>
  <type>Commercial office</type>
  <owner>The Durst Organization</owner>
</Building>
```

Note the use of XML Namespaces in GML with the prefix *gml* (See Section 2.1.1 for more on namespaces). The example above shows how properties are used to describe a feature; namely *name*, *type* and *owner* (non-geometric properties).

- **Geometries**

GML geometries are objects that can be used to represent the geometric characteristics of a feature such as position, center, extent, location ... For instance, to represent the position of the Building feature (defined above), *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>
```

A GML geometry can also be defined as stand-alone object that describes a feature instance that refers to it. The feature and geometry properties are explained further in Section 3.1.3.

- **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, units of measure, ... These objects are encoded, in the same way as features are encoded (i.e. described by their properties).

GML instances and schemas

GML documents must be well-formed and valid XML documents. Therefore, they must conform to the XML syntax rules defined in XML specifications (XML 1.0, Fifth Edition [W3C08a]) and they must conform to semantic rules defined in their respective schemas. This means that to encode GML objects, there should be two documents: the *GML instance document*, which is a well-known XML document containing all information about the encoded objects, and the *GML application schema* document which is used to validate the GML instance documents. All elements used in a GML instance document must be defined in its corresponding application schema (More on GML schemas, in Section 3.1.4).

GML versions

GML3 is the latest version of the language (GML 3.2.1. Encoding Standard [Ope07]). GML1 was replaced by GML2 and is no longer in use, whereas GML2 is considered a subset of GML3 and is still in use. The main differences between GML1 and the two other versions is that it was based on XML DTD (XML Document Type Definition) for the validation of instance documents and it used RDF (Resource Description Framework) in its profile. GML2 and GML3 are both based on XML Schema as mentioned, but they also have some differences. With GML3, one can encode many complex kinds of object other than geometries and geographic features (such as temporality, dynamic features, coordinate reference systems, units of measure, coverage, three-dimensional and non-linear geometries, topology, styling and general mechanisms for metadata). GML2 and GML3 differ also in the way of encoding objects. This is illustrated by the examples below.

GML2:

```
<gml:LineString gml:id="L001" srsName="EPSG:28992">
  <gml:coordinates>
    248718.000000,470465.000000 248723.000000,470486.000000
    248734.000000,470498.000000 248831.000000,470604.000000
  </gml:coordinates>
</gml:LineString>
```

GML3:

```
<gml:LineString gml:id="L001" srsName="EPSG:28992">
  <gml:postList dimension="2">
    248718.000000 470465.000000 248723.000000 470486.000000
    248734.000000 470498.000000 248831.000000 470604.000000
  </gml:postList>
</gml:LineString>
```

The above example consists of GML encoding for a linestring in GML2 and GML3. Note the difference in how the coordinate values are expressed, in GML3 the list of coordinate values is represented as a string enclosed by the `<postList>...</postList>` tags whereas in GML2 it is enclosed by `<coordinates>...</coordinates>`. The use of *coordinates* is still supported in GML3 for compatibility purposes but the use of *postList* is recommended. When *coordinates* is used, the values of X and Y (and Z in case of a 3-dimensional geometry) are separated by commas and the coordinates are separated by spaces. Whereas if *postList* is used, there should be an attribute that indicates the dimension of the geometry to help GML users to identify the coordinates. For instance, when dimension equals two, each point has two elements (X and Y), which means that after two values we have one coordinate pair separated from the following pair by a space.

3.1.2 GML and other XML languages

The most important XML languages relevant to GML, are XML 1.0 DTD, XML Schema, RDF, RDF Schema and XML Namespaces. XML 1.0 DTD, XML Schema,

RDF and RDF Schema are used for GML documents validation, however XML 1.0 DTD, RDF and RDF Schema were only used in the first version of GML. In addition to these, while working with GML, one may need to make use of the following XML technologies: XLink and XPointer (XML Linking Language and XML Pointer language, respectively), XSLT (eXtensible Stylesheet Language for Transformations), SVG (Scalable Vector Graphic), WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol). We discuss these in more detail below.

1. XML Schema: XML Schema is used to describe the structure of XML documents just as DTD is used. But XML Schema has many advantages over DTD, such as its extensibility. DTD is not based on XML, thus not extensible. Furthermore, XML Schema supports namespaces and data types. In GML2 and GML3, XML Schema is used to create GML schemas for validating GML instance documents (as GML documents must be well-formed and valid XML documents). DTD was used in GML1 as metadata file just for describing the GML instance documents and not for their validation. More details on GML schema are given in Section 3.1.4.

2. XLink and XPointer: XLink is used to create hyperlinks in XML documents and XPointer enables hyperlinks to point to specific parts of an XML document. In GML, they are used to create associations among GML elements and allow using remote properties, and they are also used to represent relationships between geographic features. The example below illustrates how XLink and XPointer can be used in GML.

```
<Building gml:id="id01">
  <name>Four Times Square</name>
  <type>Commercial office</type>
  <owner>The Durst Organization</owner>
  <reachedBy xlink:href="#r1"/>
</Building>
<Road gml:id="r1">
  <gml:centerLineOf>
    <gml:LineString srsName="ReferenceSystemNumb">
      <gml:posList srsDimension="2">
        ...
      </gml:posList>
    </gml:LineString>
  </gml:centerLineOf>
  <reachTo xlink:href="#id01"/>
</Road>
```

In the above example, we have two distinct features: the *Building* and the *Road* that (mutually) refer to each other: a *Building* instance can be *reachedBy* a *Road* instance and a *Road* instance can *reachTo* a *Building* instance. Since both instances are defined within the same document, in this example, the hyperlink refers to the same document using relative path name and the XPointer point to the specific instances using “#” and

the identification number of the targeted instance. The relationship can be one way, it does not have to be necessarily represented in both features. Note that the value of the *reachedBy* and *reachTo* properties are remote and not in-line as is the case for other properties, where the values are enclosed within the property tags. Moreover, the related features can be in different documents. In such a case, the links should be formatted differently. For instance, if the Building feature was in the *buildings.gml* document and the Road instance was in the *roads.gml* document, we could have:

```
<reachedBy xlink:href="roads.gml#r1"/>
<reachTo xlink:href="buildings.gml#id01"/>
```

3. XSLT and SVG: XSLT is used to transform XML documents into other formats (the targeted format may be XML-based or not). SVG is an XML-based language used for representing two-dimensional graphics in XML. XSLT and SVG can be used to generate graphical representation of GML objects (map) [LBTR04]. For more information about how to visualize GML data using XSLT and SVG, the reader is referred to [Ten03].

4. WSDL and SOAP: As SOAP is a transport protocol based on XML that enables applications to exchange information over HTTP, it can be used to support the transport of requests and responses among geo-web applications which involve the transfer of GML data from one application to the other. And WSDL can be used to describe spatial services offered by such web applications.

3.1.3 GML Model

The GML model describes the structure of GML objects and their properties representing the objects' characteristics, hence the name of object-property model (Figure 3.1). A GML object is an XML element for which the values of its child elements constitute its properties. This implies that the properties of an object cannot be presented as its attributes, and that they are always the values of its child elements. For example, a GML object *Building*, with *name* and *type* as properties, is improperly represented as follows:

```
<Building gml:id="id01" name="4 Times Square" type="commercial"/>
```

The above example represents an invalid GML instance even though it is a valid XML element. Properties can be used to represent a relationship between two GML objects, but an object cannot be a direct child element of another object. However, a property can have a value that consists of an object. This can be explained by means of an example of two GML objects *Building* and *Road* (presented in Section 3.1.2). The *Road* object cannot be a direct child element of the *Building* object but it can be a child element of its *reachedBy* property as it can be the value of a property of the *Building* object. This may be encoded in GML as follows:

```
<Building gml:id="id01">
```

```

<name>Four Times Square</name>
<type>Commercial office</type>
<owner>The Durst Organization</owner>
<reachedBy>
  <Road gml:id="r1">
    <gml:centerLineOf>
      <gml:LineString srsName="ReferenceSystemNumb">
        <gml:posList srsDimension="2">
          ...
        </gml:posList>
      </gml:LineString>
    </gml:centerLineOf>
  </Road>
</reachedBy>
</Building>

```

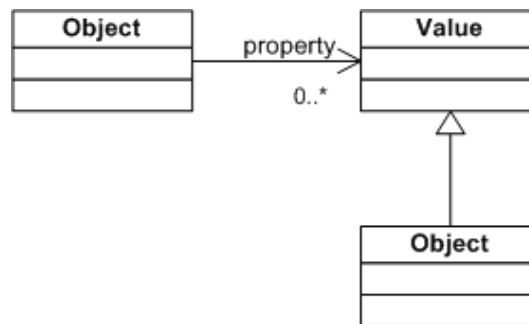


Figure 3.1: Object-property model (borrowed from (LBTR04))

Feature properties

In GML, features are GML objects that represent real-world objects and are described by their properties, which can be in-line or remote, as explained in Section 3.1.2. The feature properties consist of their geometric and non-geometric characteristics. By convention, to encode the geometric characteristic of a feature, geometry-valued properties can be used to describe the relation between a feature and its geometric property. GML has a number of predefined geometry-valued properties defined in its core schemas, namely:

- centerOf
- position
- extentOf
- edgeOf
- centerLineOf
- multiCenterOf
- multiPosition
- multiExtentOf
- multiEdgeOf
- multiCenterLineOf

- multiCoverage
- location

The use of the above geometry-valued properties is preferred to express the geometric characteristics of a feature, because it is more accurate to state that the Building feature has a geometry-valued property (extentOf for example) that expresses its geometric characteristics than stating that the Building feature has a geometry [LBTR04]. To encode the geometric characteristic of a feature, the following can be used to express a point, a line or a geometry, respectively:

```
<gml:position>
  <gml:Point srsName="ReferenceSystemNumb">
    <gml:pos>...</gml:pos>
  </gml:Point>
</gml:position>

<gml:centerLineOf>
  <gml:LineString srsName="ReferenceSystemNumb">
    <gml:posList srsDimension="2">
      ...
    </gml:posList>
  </gml:LineString>
</gml:centerLineOf>

<gml:extentOf>
  <gml:Polygon srsName="ReferenceSystemNumb">
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="2">
          ...
        </gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:Polygon>
</extentOf>
```

In the example above, *srsName* indicates the value of the Spatial Reference identification number and the value of *srsDimension* indicates the dimension of the geometry. The point geometry uses *pos* instead of *posList* to define its coordinates and does not require the dimension attribute, note that *coordinates* may still be used as already mentioned. A polygon geometry may have an interior boundary and an exterior boundary, in which case the polygon has a hole defined by the interior boundary. The polygon presented above does not have a hole, as only has an exterior boundary defined by *gml:exterior* which is encoded by means of *gml:LinearRing*. Even though we limited ourselves to the three simple and basic geometries: point, line and polygon, GML supports more complex geometries also, as defined in GML3 core schema.

Feature collections

GML provides a way of encoding a set or collection of features. For instance, to represent the buildings in a city as a feature collection, a feature collection *City* containing the *Building* features can be defined. There are two ways of defining a feature collection, using the property *featureMember* or using the property *featureMembers*.

Using *featureMember*:

```
<app:City>
  <app:name>Enschede</name>
  <gml:featureMember>
    <app:Building>
      ...
    </app:Building>
  </gml:featureMember>
  <gml:featureMember>
    <app:Building>
      ...
    </app:Building>
  </gml:featureMember>
</app:City>
```

Using *featureMembers*:

```
<app:City>
  <app:name>Enschede</name>
  <gml:featureMembers>
    <app:Building>
      ...
    </app:Building>
    <app:Building>
      ...
    </app:Building>
  </gml:featureMembers>
</app:City>
```

3.1.4 GML core and application schemas

GML allows its users to create geographic features that represent real-world objects, providing the structure and instructions of how these objects should be encoded [LBTR04]. The real-world objects that one may need to represent may consist of roads, buildings, hospitals, hotels, streets, and so on. They are various and their meaning may differ depending on the application. To avoid misunderstanding among users, before creating a feature, there should be an application schema in which the feature type is defined. GML provides a framework to support the creation of application schemas, the GML core schemas. GML core schemas make use of XML Schemas to define the types of GML object and their

properties, declare the XML elements and XML attributes that represent those objects, and their respective properties.

Currently, there are 28 core schemas defined in GML 3 for which detailed explanation can be found in GML 3.2.1 encoding standard [Ope07]. Note that the 28 schemas include the three core schemas defined in GML 2 (namely *feature.xsd*, *geometry.xsd* and *xlinks.xsd*). Considering the size of the GML 3.2.1 encoding standard, a number of GML profiles have been created to provide GML users with restricted subsets of GML core schemas to ease GML adoption. Here is a list of some created and published GML profiles [Ope07]:

- GML Point Profile: limited to the point geometry.
- GML Simple Features profile (GML-SF): supports the point, line and polygon geometries and their respective collection geometries. This profile can be compared with an earlier OGC specification SF-SQL (Simple Features for SQL), except that GML Simple Features profile supports three-dimensional coordinates and metadata, which are not supported in SF-SQL.
- GML common Coordinate Reference Systems profile: can be used to support encoding the commonly used Coordinate Reference Systems and their corresponding coordinate conversion.

GML application schemas are developed on basis of GML core schemas; the latter provide the types of feature that can be created in GML, the geometric types and non-geometric property types. To avoid the frequent problem of naming while developing an application schema, the namespace should be used to express the application context. For instance, one would use the *urb* prefix for an application related with urban planning or *agr* for an application in context of agriculture (See Section 2.1.1 for more on XML Namespaces). As application schemas are based on GML core schemas, one must choose the appropriate core schemas and import them into one's application schema (because it is not needed to import all the 28 core schemas if they are not required for the concerned application). GML has a number of rules that should be respected through the process of an application schema, which are described in GML specifications [Ope07]. The schemas used to describe the basic component of GML (GML object) can be found in Appendix, A.1.

3.2 Syntax of XQuery extension for GML

As we have already seen in Section 2, XQuery is a more appropriate query language to query XML data (based on XML data model) 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 returnType
```

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 representation. Thus, any node containing geometric properties which is valid in GML, can be passed to this function regardless of its application schema. This can be illustrated by the following GML document fragments:

Example 1: *building1.gml*

```
<Building gml:id = "id01">
  <gml:position>
    <gml:Point srsName="ReferenceSystemNumb">
      ...
    </gml:Point>
  </gml:position>
</Building>
```

Example 2: *building2.gml*

```
<app:city>
  <app:Building gml:id = "Blg01">
    <app:name>Four Times Square</app:name>
    <type>Commercial office</type>
    <gml:extentOf>
      <gml:Polygon srsName="ReferenceSystemNumb">
        ...
      </gml:Polygon>
    </gml:extentOf>
  </app:Building>
</app:city>
```

Note that these two features are defined in different ways: the first one has a point geometry as its geometric property whereas the second has a polygon as geometric property. Moreover for the second example, a namespace has been

used (with *app* as prefix) and it has non-geometric properties in addition to geometric properties. The fact that these features have different application schemas should not affect the function, therefore we may have:

```
geoxml:geometry(doc("building1.gml")//Building/*:Position)
geoxml:geometry(doc("building1.gml")//Building/*:Point)
geoxml:geometry(doc("building2.gml")///*:Building/*:Polygon)
```

The above would return respectively: "POINT(...)", "POINT(...)" and "POLYGON(...)", i.e. points and a polygon geometries in well-know text (WKT) representation. Using the character "*" instead of a namespace prefix indicates that the query will affect the element regardless its namespace. The only thing that is required is to select a right node that is a valid GML element with geometric properties, by means of the XML navigation mechanism offered by XQuery. The spatial queries resulting from this syntax can be compared with SQL spatial queries as shown in Table 3.1. More sample queries are presented in Section 3.2.2 showing the application of different spatial functions in XQuery.

Table 3.1: 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

3.2.1 Data description

The GML data used in the spatial queries presented in the next section, consist of the data of Enschede city for which the application schemas are given in Appendix B. The data set contains:

The neighbourhood boundaries data (neighborhood.gml): consist of features of type *buurt2003*, with *BU_2003* and *BU_NAAM* (unique identifier and name of the neighbourhood, respectively) as non-geometric properties and *polygon* as geometric property.

The land use data (landuse.gml): consist of features of type *enstopolanduse*, with *TDN_CODE*, *TYPE* (type of land use) as non-geometric properties and *polygon* as geometric property.

The buildings data (buildings.gml) consist of features of type *enscadBuildings*, *ID* (building unique identifier) as non-geometric property and *polygon* as geometric property.

The roads data (roads.gml) consist of features of type *roads*, with *WVK_ID* (road unique identifier), *WEGBEHEERD* (road category), *WEGNUMMER* (road number), *STT_NAAM* (street name) as non-geometric properties and *linestring* as geometric property.

The data about places with fire risk (riskpoints.gml): the places with fire risk are places such as gas stations, fireworks sales and hazardous industries. These data consist of features of type *riskpoints*, with *ID* (unique identifier), *TYPE* (place category) as non-geometric properties and *point* as geometric property.

The data of some schools of Enschede (schools.gml) consists of features of type *school*, with *NR* (school unique identifier), *NAME_OF_TH* (school name) as non-geometric properties and *polygon* as geometric property.

The schemas presented in Appendix B depict the node-tree structure of their corresponding GML document (the hierarchy of the node elements).

3.2.2 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), and geometry constructors and accessors.

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($geom as xs:string*[=wkt*]) as xs:string`
4. `geoxml:numGeometries($geom as xs:string*[=wkt*]) xs:integer`

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, the geometry type and the number of geometries 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:geometryFromText($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()*,

Functions for spatial relationships

To check the spatial relationship between two geometries, a number of functions is needed namely *geoxml:intersects()*, *geoxml:crosses()*, *geoxml:touches()*, *geoxml:contains()*, *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
```

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 xs:string* [=wkb*])
    as xs:string* [=wkb*]

2. geoxml:union($geomset as item*)
    as xs:string* [=wkt*]

3. geoxml:intersection($geom1 as xs:string* [=wkb*],
    $geom2 as xs:string* [=wkb*])
    as xs:string* [=wkb*]
```

Examples of spatial queries using XQuery

In this section, we present a number of tentative examples of spatial queries using XQuery. The data used in the following queries were described in Section 3.2.1.

1. Return the SRID used for and the total area of Enschede city.

```
LET $neigh:=doc("neighborhood.gml")//*:buurt2003
RETURN
<enschede>
  <srid>
    {geoxml:srid(geoxml:geometry(($neigh//*:polygonProperty)[1]))}
  </srid>
  <area>
    {fn:sum(
      FOR $n1 in $neigh
      RETURN geoxml:area(geoxml:geometry($n1//*:polygonProperty))
    )}
  </area>
</enschede>
```

2. Return the length of the road of Oldenzaalsestraat.

```
LET $road := doc("roads.gml")
           //*:roads[*:STT_NAAM="Oldenzaalsestraat"]
RETURN geoxml:length(geoxml:union($road//*:LineString))
```

In the query above the function *geoxml:union()* is an aggregate function that takes a set of linestring geometries.

3. List the names of the schools in Enschede, and the name and area of the neighbourhood in which each school is located.

```
FOR $s IN doc ("schools.gml")//*:schools,
  $n IN doc ("neighborhood.gml")//*:buurt2003
LET $school := geoxml:geometry($s//*:polygonProperty)
LET $neighb := geoxml:geometry($n//*:polygonProperty)
WHERE
  geoxml:Within ($school, $neighb)
RETURN
  <schoolEnschede>
    <schoolname>{$s//*:NAME_OF_TH}</schoolname>
    <neighbourhoodName>{$n//*:BU_NAAM}</neighbourhoodName>
    <neighbourhoodArea>{geoxml:area($neighb)}</neighbourhoodArea>
  </schoolEnschede>
```

4. List names of all the schools that are within a distance of 250 meters from a place with fire risk.

```
FOR $s IN doc ("schools.gml")//*:schools,
  $rp IN doc ("riskpoints.gml")//*:riskpoints
  LET $school := geoxml:geometry($s//*:Polygon)
  LET $riskpt := geoxml:geometry($rp//*:pointProperty)
WHERE
  geoxml:distance($school, $riskpt) <= 250
RETURN $s//*:NAME_OF_TH
```

Or alternatively:

```
FOR $s IN doc ("schools.gml")//*:schools,
  $rp IN doc ("riskpoints.gml")//*:riskpoint
  LET $school := geoxml:geometry($s//*:polygonProperty)
  LET $riskpt := geoxml:geometry($rp//*:pointProperty)
WHERE
  geoxml:Intersects (geoxml:buffer($school, 250), $riskpt)
RETURN $s//*:NAME_OF_TH
```

5. Return the road of Hengelosestraat as a single geometry of type linestring.

```
MODULE NAMESPACE funx="http://www.newfunx.com/myfunctions"
DECLARE FUNCTION unionAggr($arg1 as element(),
                           $arg2 as element()) as element()
{
  RETURN geoxml:union($arg1, $arg2)
};
<gml:featureMember>
  FOR $r IN doc("road.gml")
    //*:roads[*:STT_NAAM="Hengelosestraat"]
  LET $rnew := funx:unionAggr($rnew,
    geoxml:geometry($r//*:LineString))
  RETURN $rnew
</gml:featureMember>
```

The function *unionAggr()* was declared to act as an aggregate union function using the binary function *geoxml:union*. This example illustrates another way of applying the aggregate function *geoxml:union()* used in the second example.

6. Return the road of Hengelosestraat as a single geometry collection. Note that here the function *geoxml:union()* is an aggregate function.

```
declare namespace gml="http://www.opengis.net/gml";
LET $r:=doc("road.gml")//*:roads[*:STT_NAAM="Hengelosestraat"]
RETURN
<gml:featureMember>
{
  geomxml:GeomCollectionFromText (
    geoxml:asText(geomxml:union($r//*:lineStringProperty)),
    geoxml:srid(geoxml:geometry($r//*:lineStringProperty))
  )
}
</gml:featureMember>
```

7. Return points where the road of Hengelosestraat meets with other roads (where it is touched or crossed).

```
LET $h := geoxml:geometry(doc("road.gml")
    //*:roads[*:STT_NAAM="Hengelosestraat"]
    //*:lineStringProperty)
FOR $r IN doc ("road.gml")//*:roads
    LET $road := geoxml:geometry($r//*:lineStringProperty)
WHERE (geoxml:crosses($road, $h)) OR
    (geoxml:touche ($road, $h))
RETURN
    <pointGeom>
        {geoxml:intersection ($road, $h)}
    </pointGeom>
```

8. Delete all risk points within the city neighbourhood.

```
FOR $rp IN doc ("riskpoints.gml")//*:riskpoints,
    $n IN doc ("neighborhood.gml")//*:buurt2003
    LET $pt := doc("riskpoints.gml")//*
WHERE
    geoxml:within (geoxml:geometry($rp//*:pointProperty),
        geoxml:geometry($n//*:polygonProperty))
RETURN
    DO DELETE (doc("riskpoints.gml")//*:riskpoints)
```

9. Construct a point from a string and insert the point into the document riskpoints.gml.

```
LET $pt := geoxml:pointFromText("Point(50869,469509)", 28992)
DO INSERT $pt
AS LAST INTO doc("riskpoints.gml")//*:FeatureCollection,text" "
```

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) [GO]. 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). This implementation of the GEOXML library was carried out with the support of the members of the database group of the University of Twente.

Chapter 4

Design of a distributed geoprocessing system

A distributed processing system 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 such 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 [FS07].

We will design a distributed environment for a geoprocessing prototype system that can compute the annual average of soil loss caused by precipitation. To do so, a model for soil erosion assessment is integrated with spatial database and web technologies. Such prototype system can support the decision-making activities of the land managers, planners, farmers and others while making land use planning for environmental conservation purposes. To estimate 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 as shown in Figure 4.1. Having each node equipped with MonetDB/XQuery, will allows us to use the Remote Procedure Call (RPC) mechanism to ensure the interprocess communication.

In this chapter, we discuss in details the design of our system prototype taking into account the chosen scenario, a model for soil erosion assessment described in Section 4.1. The requirements of our system prototype are defined in Section 4.2 and the system architecture is presented in Section 4.3.

4.1 Estimating soil erosion risk in Rwanda - The Sebeya watershed

4.1.1 Scenario description

The soil loss caused by water erosion is a major concern of environmental protection. The water erosion consists of the detachment and transportation of soil

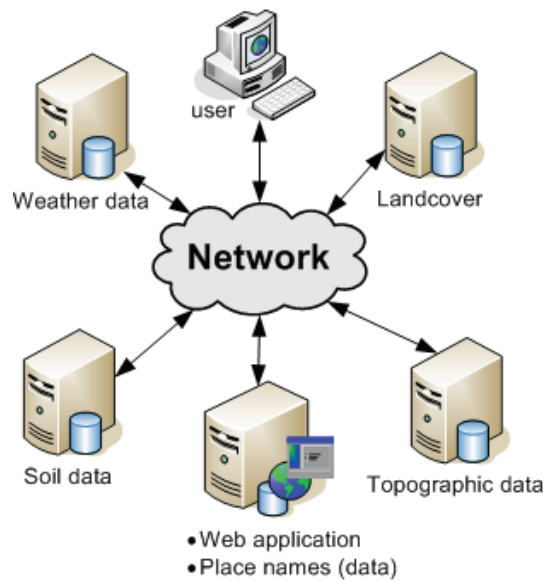


Figure 4.1: Overview of the system prototype

particles from one location to another by surface water deposited by precipitation. It decreases the quality of soil because the soil loses its organic matter that is transported with the topsoil. This has an unfavorable impact on the growth of plants thus reduces the crop productivity [Lal94, NSCE99]. For a country like Rwanda that has a high population density from which the majority make their living from agriculture, with steep slopes and heavy rains [CL96], the water erosion is a threat to agricultural production. Besides the decrease of the agricultural production, water erosion is the cause of many other detrimental effects on the environment such as water quality degradation in rivers, lakes or streams provoked by the eroded soil particles. In the present research, the water erosion risk was assessed within the watershed of Sebeya river and its affluent Pfunda located in the Western Province of Rwanda.

The magnitude of the water erosion is mainly affected by parameters of rainfall, soil type, topography and land cover. The intensity of rain drops and the flow of water on bare ground added to the steep hillside may cause naturally water erosion, but water erosion is accelerated by human activities (changing the land cover) such as overgrazing, deforestation, etc. Decision-makers need to monitor these effects so that they can make environmental conservation planning or determine the conservation practice to be adopted. The Universal Soil Erosion Equation (USLE) (Section 4.1.2, Equation 4.1) is mostly used to assess the soil erosion risk for a particular site and it takes into account all the aforementioned parameters that affect the water erosion. The USLE model can be applied to assess the soil erosion risk for agricultural as well as for non-agricultural sites.

A distributed processing environment is an ideal application infrastructure for a model like USLE, considering that it involves a number of processing tasks (processes required to retrieve USLE input parameters) and a number of data sets. A distributed processing environment offers the possibility not only to use

data stored on different nodes but also use more than one node (processor) for a single processing task, regardless of location of the involved nodes. And this entails a number of advantages, among other things the performance and data sharing. Since in a distributed processing environment there is more than one processor, an opportunity arises to improve the system, distributing (properly) the processing workload among the available nodes.

4.1.2 Universal Soil Loss Equation(USLE)-modified

The USLE is a simple and empirical model that was designed to predict the long-term average annual soil loss caused by sheet and rill erosion. The sheet and rill erosion are types of water erosion. The sheet erosion consists of soil transported in uniform thin layers by the flow of runoff water and is caused by raindrops. The rill erosion is also caused by raindrops, and it occurs when the flowing runoff water is accumulated into small channels. The runoff water occurs when the rainfall intensity exceeds the water-absorption capacity of the soil. For more details about the characteristics of sheet and rill erosion, refer to [Hil98].

To compute the annual average soil loss, the USLE 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 4.1.

$$A = R * K * L * S * C * P \quad (4.1)$$

where

A : A is the annual average soil loss expressed in $t\ ha^{-1}year^{-1}$ (tone per hectare per year) with $1\ t = 1000\ kg$ and $1\ ha = 10000\ m^2$.

R factor: R factor is the rainfall erosivity factor. It is expressed in $MJ\ mm\ ha^{-1}\ h^{-1}\ year^{-1}$ (mega joule millimeter per hectare per hour per year) since it represents conventionally the product (EI_{30}) of the rainstorm energy (E) and the maximum 30-minutes intensity (I_{30}). It depicts the erosive effect of the rain drops to the soil loss.

K factor: K factor is the soil erodibility factor and is expressed in $t\ h\ MJ^{-1}mm^{-1}$ (tone hour per mega joule per millimeter). The K factor determines the ability of a soil to resist erosion.

L factor: L is the slope length factor and is expressed in meter.

S factor: S is the slope steepness factor and is unitless. The steepness of slope affect more the soil loss than the slope length. L and S factors are often considered as one topographic factor LS.

C factor: C factor is the cover-management factor and is unitless. It depends on the type of the condition of the cover on the soil surface (the cover may prevent the soil loss). If the soil is well protected, the C factor is low because in that case the soil is less susceptible to erosion.

P factor: The P factor is the soil conservation practice factor and is based on the soil conservation practice applied on the landscape. If there is no soil conservation practice applied to the soil surface, the value of P factor is assigned to 1, otherwise it is assigned to a lower value. The P factor was not used therefore the output of the model will represent the worst-case scenario that is a better basis for future planning decisions. Normally soil conservation practice may consist of contours, strip cropping or terraces.

As indicated by its naming, USLE is a universal model and can be used under any specified environmental parameters even though it has been developed under the soil and climatic conditions of the United State of America [WS98]. Note that it has been slightly modified in some case studies to fit well the local environment conditions or special conditions [Lal94]. USLE model was also slightly modified in the current research, so that it can be adapted with the environment conditions of Sebeya watershed taking into consideration the upstream and downstream effects. The modified model of USLE applied to assess Sebeya watershed is the Equation 4.2. The procedure is as follows: once all the factors values have been determined, the gross erosion index ($E = R K L S C$ derived from the conventional USLE model) is computed and thereafter E is calibrated using the following equation to get the gross erosion rate:

$$A = (19.6 E) + 14.9 \quad (4.2)$$

The above equation was derived from a number of measurements of USLE factors values taken directly on ground using the Rainfall Simulator, with the purpose of calibrating the model taking into account the local environment conditions. The correlation between the measured values and the values obtained using the modified model Equation 4.2 was computed to prove the quality of the model correctness, with a value of Pearsons correlation coefficient (R) equal to 0.92 that to say 92% of the values fit well with the model and only 8% did not. The model described above should not be confused with MUSLE (Modified Universal Soil Equation).

In some previous studies [EEB06, FH02, AMNS04], the soil erosion models (specifically USLE model) were combined with GIS technology to predict soil loss. This integration facilitates the identification of the locations threatened by soil erosion risk specifying their potential spatial boundaries and spatial patterns; thus it can help the planners to make decisions about what measures should be taken to protect or preserve the environment. In our research, the USLE model is integrated with spatial database (XML database) and web technology to provide a web-based tool that can support planners to make their decisions concerning the land use system and/or soil conservation practices to be applied. The system prototype shall compute the annual average soil loss given the data stored in distinct XML databases.

The USLE achieves a better estimation of the annual soil loss over a long period for small areas at farm yields scale than for large-scale areas [WS98, FH02] because the parameters used to define its factors vary significantly from storm to storm and their fluctuation within larger areas is not spatially correlated. To estimate the soil loss of a large area using the USLE model, it is recommended to divide it into smaller areas. For instance, USLE has been applied at

watershed or catchment scale level in a number of researches, subdividing the watershed or the catchment into smaller areas such as homogeneous plots, or subbasins or cells of a regular grid [FH02, AMNS04]. In the current research, USLE was applied at a watershed scale, using regular grid, notably the watershed of Sebeya, localized in the Western province of Rwanda.

Apart from the scale effect, the accuracy of the USLE model can also be affected by other parameters. The United States Department of Agriculture (USDA) erosion researches showed that the soil loss estimation using USLE was most accurate for medium texture-soil, with slope lengths of less than 400 feet (approximatively 120 meters), slope steepness of 3 to 18 percent and consistent cropping and management system [WS98]. It should be pointed out that USLE can not be used to estimate the deposition, i.e. the amount of eroded soil accumulated onto a given Earth surface.

Determining USLE-modified factors values

Wishmeier and Smith [WS98] provided a guideline to determine the numerical values of each of the factors of USLE model, deriving them from different data sets.

1) Rainfall erosivity factor (R)

The rainfall erosivity factor represents the amount of soil loss due to the climatic conditions. To determine the rainfall erosivity factor, the Fournier Index 4.3 approach was adopted, “an erosivity index for river basins on the basis of the relationship between suspended load in rivers and climatic data and relief characteristics” as defined in [Lal94]. This approach was suggested by Rown-tree [M.82] during a study carried out in Kenya, as a more effective approach to estimate the local erosivity in tropical catchments than the conventional ones based on 30-minutes maximum intensity (refer to [WS98] for more details). Fournier Index is calculated using the following equation:

$$C = \frac{p^2}{P} \quad (4.3)$$

where C is the climate index (Fournier index) in mm, p is the total amount of rainfall in the wettest month (in mm), which is April (for the country of Rwanda) and P is the annual rainfall amount expressed in mm. The input data used to computed the rainfall erosivity, were recorded by 183 meteo-stations dispersed within the country (from 1976 to 1996 thus 20 years) then interpolated using a geostatistical technique: Kriging with Elevation data (Kriging with external Drift).

2) Soil erodibility factor (K)

The soil erodibility factor depends on the soil type (its structure, permeability, organic matter content, particle size distribution), which means that a site may present a high rate of soil erosion just because of the characteristics of its soils (the erodibility parameter does not depend on other parameters). The soil

erodibility consists of the average long-term measurements of soil loss response to rainfall erosivity. The values of K factor, used for this research were measured by soil scientists from University of Ghent and MINAGRI (Ministry of Agriculture and Animal Resources in Rwanda). The measured K factor values were assigned to soil texture accordingly (as shown in Table 4.1) to create a pedological data set.

Table 4.1: Soil erodibility (K factor values in $t h M J^{-1} m m^{-1}$)

Soil type	Erodibility factor	K factor score
Lakes	0	0
Andisols	0.01 - 0.08	0.04
Histosols	0.08 - 0.12	0.10
Mollisols	0.08 - 0.13	0.11
Entisols	0.10 - 0.14	0.12
Inceptisols	0.10 - 0.15	0.12
Alfisols	0.12 - 0.16	0.14
Ultisols	0.13 - 0.17	0.15
Urban zone	0.17 - 0.19	0.18

3) Topographic factor (LS)

The topographic factor affects the water erosion concretely as the velocity of the runoff water increases with the slope, and the more the runoff water accelerates the more soil is transported. The topographic factor is expressed in terms of slope length and slope steepness:

Slope length factor (L): The slope length factor is computed using the equation below:

$$L = (\lambda/22.13)^m$$

where λ is the slope length in meters, and m is the slope length exponent [WS98]. For a slope angle θ :

$$\begin{cases} m = 0.2 & \text{for } \theta < 1\% \\ m = 0.3 & \text{for } 1\% \leq \theta < 3\% \\ m = 0.4 & \text{for } 3\% \leq \theta < 5\% \\ m = 0.5 & \text{for } \theta \geq 5\% \end{cases}$$

For the current case study, we used $\lambda = 22.13m$.

Slope steepness factor (S) : To calculate the effect of the slope steepness factor on soil erosion in Sebeya watershed, the approach of Nearing [A.97] was chosen, as he provides us with an equation adapted for steep slopes $> 25\%$ (see equation below); and the concerned watershed is located in the highlands region with steep slopes.

$S = -1.5 + 17/(e^{(2.3-6.1 \sin \theta)})$ where θ is the slope angle in degrees. The slope angles used, were extracted from a digital elevation model (DEM)

with a 30m contour interval. The slope steepness is unitless and was expressed in percentage.

4) The cover and management factor (C)

The cover and management factor determines the effect of the land use/cover (notably the cropping and management practices) on soil erosion [WS98]. More vegetation cover decreases the intensity of rain drops by breaking their fall before they splash on Earth surface, and it also affects the flow of runoff water. To determine the values of C factor differs from place to place, because the C value for a particular site depends on the rainfall erosivity of that site in relation to its land cover. The C factor of Sebeya watershed was determined according to C values (See Table 4.2) described by Cohen, Shepherd and Walsh [CSW05] in a research project carried out in Kenya . These values were assigned to the vegetation/land cover type accordingly, the land cover data as input.

Table 4.2: C factor scores for five categorical cover classes

Cover class	% Vegetation cover	C factor score
1	0-20%	0.8
2	20-40%	0.5
3	40-60%	0.2
4	60-80%	0.1
5	80-100%	0.01

USLE-modified processing steps

All the USLE factors mentioned above are acquired from distributed spatial data sets and can be communicated through GML. The aim of this erosion assessment scenario is to build a geo-web application prototype that runs in a distributed environment to compute the annual average soil loss. Since the USLE factors vary by location, their retrieval and management may 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 4.2):

1. A request of locating the site to be assessed may be sent to *Node 1* (containing the place names) or to *Node 5* (containing the land cover data) from user machine. The user chooses to locate the site of interest by names of places then the request is sent to *Node 1*.
2. 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 .

3. Before computing A, all USLE factors should be retrieved, to do so the following tasks are executed:
 - 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 (the shipped geometry). Then, the annual and monthly (for the wettest month) rainfall values are retrieved and used to compute the Fournier index (the value of R factor).
 - 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.
 - 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 retrieved 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, then C values are assigned to their corresponding vegetation cover.
4. After computing all the needed factors, the annual average of the soil loss is computed (*Node 2*).
5. The computed annual average of the soil loss is sent back to *Node 1*.
6. *Node 1* presents the computed annual average of the soil loss back to the user.

Figure 4.2 illustrates the order of processing steps followed to calculate the annual average soil loss.

4.2 Requirements definition

The requirements definition is the first step of an application development life cycle and its purpose is to clarify what the application should fulfill [JBR99]. The requirements of an application depend on its users. Our prototype application will have one type of user, who can use it as a decision-making support tool in the domain of environmental protection, especially for soil erosion. The requirements definition will be categorized into two types: functional and non-functional requirements.

Functional requirements : The functional requirements define the behavior of the application. They specify what the application should do without mentioning how it should do it:

- The prototype shall compute and retrieve the erosion variables from different spatial repositories stored on the nodes within our distributed system.

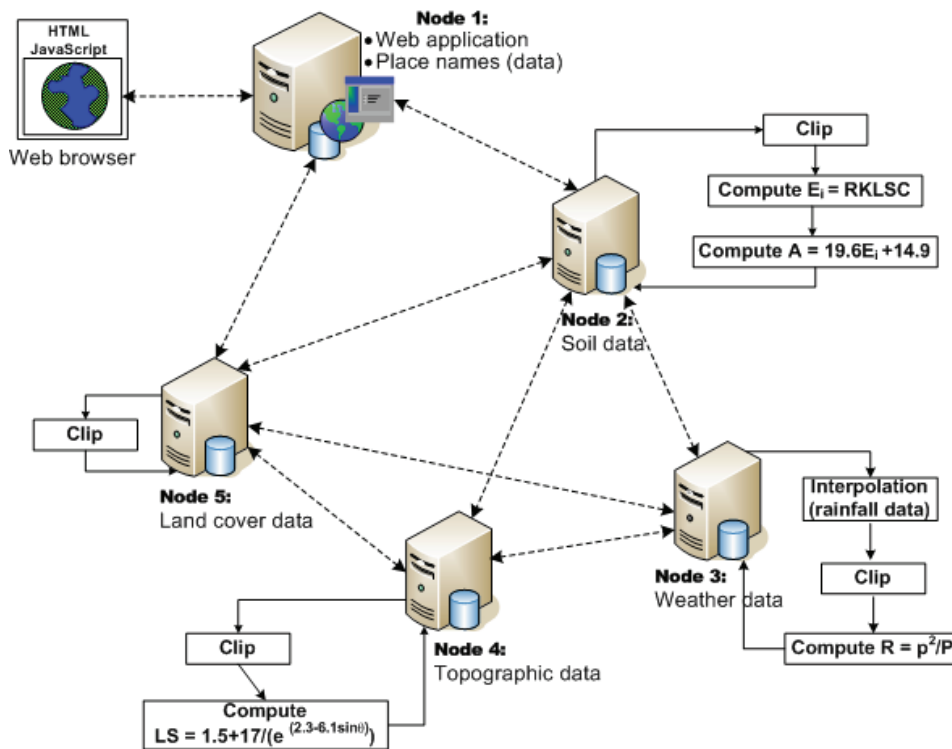


Figure 4.2: Processing steps for calculating the annual average of the soil loss

- The prototype shall compute the annual average of the soil loss.
- The prototype shall return the regions (as geometries) by their soil erosion rate namely Very low, Low, Moderate, High and Very high.

Non-functional requirements : The non-functional requirements place constraints upon the application, they specify how it will be implemented defining its performance and development requirements:

- The prototype shall provide a web-based user interface.
- The prototype shall provide the output of geoprocesses (as well as other processes) within an adequate response time.
- The prototype shall run in a distributed environment, and all the nodes within the distributed environment will be working on base data sets stored in XML database.

4.3 System architecture

To design our prototype, we will use 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 [Jos07]: “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 will adopt 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. 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 [Ree79, MAS⁺03].

The model supports the storage, retrieval of data from the database system and their maintenance. MonetDB/Xquery will be used as database system of our prototype system. The view provides an interface to accommodate the interaction between the user and the application. 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.

4.3.1 The model

The model consists of the application data stored in a database management system (DBMS). In the current research, an XML database system was used to store and manage the data in XML-based format (namely GML and XML documents). To access the data, the data access services will be used, as mentioned in Section 4.3.3, allowing to retrieve and operate on the data.

4.3.2 The view

The view provides a user interface for the users of the application. The user interface allows the user to input data into the application and get back the returned information (processed data). Our user interface will allow the user to select the region for which the soil erosion risk is to be estimated and get back the results. To design our user interface we use an internet browser; a software application that is able to access a web server and interpret the client code (HTML and JavaScript, for our research) to display the returned information.

4.3.3 The controller

The controller consists of the business logic, which is the most important part of the system as it ensures the communication between the data and interface of the application, and provides the system services. Considering that a distributed geoprocessing system consists of a number of nodes that intercommunicate to carry out a task (spatial or non-spatial task). The RPC approach will be used to enable those nodes communicating by means of a call, a consumer

node will be able to call a remote function and execute it as if the function was local. Each system function will be encapsulated as a service and accessed using XRPC approach, applying SOAP and HTTP protocols. Moreover, the business logic allows the services (encapsulated functions) to access the data by means of the data access services.

XQuery Remote Procedure Call

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 to the ordinary XQuery function call (or procedure call) [ZB07]. 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 to integrate XQuery data sources with web services and SOA.

Basic XRPC concepts

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) is a protocol used for transferring files on the World Wide Web (WWW or simply Web), that means that it supports each request made on the Web. HTTP consists of rules that “provide a standardized way for computers to communicate with each other,” (as mentioned in [Won00]) using the client-server model (illustrated in Figure 4.3). It describes how the requests are sent by the client (such as a web browser) and how the server responds to those requests over the Web.



Figure 4.3: Client HTTP request - Server HTTP Response (Adapted from [GTS⁺00])

Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) is an XML-based protocol that allows applications to exchange information based on XML-messaging (using XML documents) over a network [GTS⁺00]. SOAP defines the structure of exchanged messages in (XML format) and is platform- and programming language-independent (See Figure 4.4). A SOAP message consists of a well-formed XML document containing the following elements:

- An envelope element: It is a mandatory and top element of the XML document element that represents a SOAP message.
- A header element: It is an optional element that contains information about how the message should be processed (such as authentication or routing information).
- A body element: It is a mandatory element that contains the information to be delivered.
- A fault element: It is an optional and child element of the body element. It contains errors that occurred while processing the message.

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Figure 4.4: The structure of a SOAP message

As SOAP is not a transfer protocol, it should be used with a transport protocol (such as HTTP) to ensure the transfer of the message. SOAP can be used for one-way messaging or for RPC implementation. In fact, one goal of SOAP was to support message exchange using procedure calls which is why SOAP is mainly used for remote procedure calls (See SOAP Representation in [W3C03]).

Remote Procedure Call (RPC)

Remote Procedure Call (RPC) is a mechanism used to construct distributed systems, enabling an application to call a function (or a procedure) stored on a different computer over the network using the request/response concept [LJD01]. The remote function is called as a local procedure except that the function call is sent through the network, from one application system to another, where it is executed. Then the called application (server) sends back the returned results to the caller (client). When a client has sent a request to a server, it blocks (suspends execution), waiting for the server to finish the execution of request and send back the response. The flow of communication in RPC is illustrated in Figure 4.5, where the reader should note that the communication between the client and the server is supported by two stubs (the so-called client and server stubs). John [Blo92] defined a stub as “a communication interface that implements RPC protocol and specifies how messages are constructed and exchanged.”

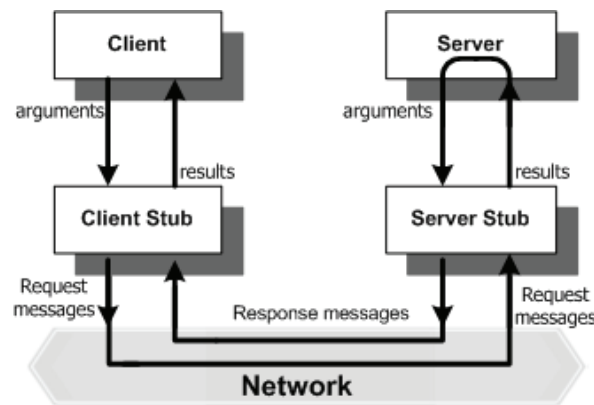


Figure 4.5: Remote Procedure Call communication (Adapted from (Blo92))

The XRPC language extension

XRPC was designed to extend XQuery with distributed queries, supporting remote function calls (or function shipping). XRPC is not not only used for XQuery but supports also XQUF queries (queries that update the data). XQuery remote function calls have this syntax: `execute at {Expr}{Fun(ParamList)}` as already presented in 2.6.3. This syntax can be illustrated by an example of a function *bounds()* that returns the spatial extent (envelope) of a school geometry (refer to data described in Chapter 3) given its name. The function is defined as follows:

```

module namespace b="bbox";
declare namespace gml="http://www.opengis.net/gml";
declare function b:bounds ($geomname as xs:string) as element()*
{
  let $g:=doc("schools.gml")//*:schools[*:NAME_OF_TH=$geomname]
  return
  <gml:Polygon srsName="{($g//@srsName)[1]}">
    {$g//*:Box/*:coordinates}
  </gml:Polygon>
};

```

Considering that the above function is defined in *geom.xq* module stored at *clark* node. The function *bounds()* can be executed locally (*localhost*) from a remote node (*clark*). To execute it locally, the following code will be used:

```

import module namespace b="bbox" at
"http://clark:50001/export/geom.xq";
<schoools>
{
  execute at {"xrpc://localhost:50001/xrpc"}
  {b:bounds("Helmerhoek")}
}
</schoools>

```

The above function call returns:

```
<gml:Polygon xmlns:gml="http://www.opengis.net/gml"
srsName="EPSG:28992">
  <gml:coordinates>
    250363.030018,467865.605999 250374.512079,467876.800950
  </gml:coordinates>
</gml:Polygon>
```

Note that XRPC is used as protocol for the execution of the remote function. As already mentioned, XRPC uses SOAP for messaging and HTTP as transport protocol which allows us to use it for web services. But SOAP XRPC should not be confused with SOAP RPC because they are different. 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 are examples of an XRPC Request 4.6 and the corresponding XRPC Response 4.7 for the same example described above.

```
<?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 4.6: An example of an XRPC Request message

```
<?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:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:schemaLocation="http://monetdb.cwi.nl/XQuery
  http://monetdb.cwi.nl/XQuery/XRPC.xsd">

  <env:Body>
    <xrpc:response xrpc:module="bb"
      xrpc:method="bounds">
      <xrpc:sequence>
        <xrpc:element>
          <gml:Polygon
            xmlns:gml="http://www.opengis.net/gml"
            srsName="EPSG:28992">
            <gml:coordinates>
              250363.030018,467865.605999
              250374.512079,467876.800950
            </gml:coordinates>
          </gml:Polygon>
        </xrpc:element>
      </xrpc:sequence>
    </xrpc:response>
  </env:Body>
</env:Envelope>
```

Figure 4.7: XRPC Response message

Chapter 5

Implementation of a distributed geoprocessing system prototype

The implementation of our system prototype relies on the logical architecture designed in Chapter 4. A SOA approach was adopted to make available the geoprocessing services in a distributed environment through service orchestration. Considering that our system prototype is structured into three parts, namely the model, the view and the controller, this chapter discusses in detail the implementation of each part. Section 5.1 describes the implementation of the model and Section 5.2 that of the user interface. The implementation of the business logic, the tools and techniques used to implement it are described in Section 5.3.

5.1 Model implementation

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. To implement our system prototype, six data sets were used, namely:

- rainfall data (Rain_erosivity.gml)
- soil data (Soil_erodibility.gml)
- topographic data (Topography.gml)
- land cover data (Vegetation_cover.gml)
- watershed boundary (Sebeya_watershed.gml)
- administrative sector boundaries (sectors.gml)

The application schemas of these GML documents were created using the Altova XMLSpy tool, they are provided in Appendix C. The data management was carried out by means of the document management functions built-in in

MonetDB/XQuery. For instance, to add the land cover data into our database, the function *pf:add-doc()* was used as follows:

```
pf:add-doc("D:\data\gml\Vegetation_cover.gml", "cover.gml", "usle")
```

where *D:\data\gml\Vegetation_cover.gml* is the URI to the location of the GML document, *cover.gml* is the logical name under which the document is saved in the database and *usle* is the name of the collection in which the document is stored.

5.2 User interface implementation

The user interface is implemented in such a way that it allows the user to provide input data to the system and get back the output. Since the main functionality of our system prototype is to compute the annual soil loss caused by rainfall water given a specific site, the interface allows the user to specify the site for which the soil loss is to be computed. 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 5.1. After submitting the form, the user retrieves the resulting data displayed in a text area. The returned data is encoded in GML, and the application schemas used to validate it is accessible through the link provided on the user interface.

The user interface allows the user to visualize the output as a map, with some limitations at present. The map on the top-right of the user interface as shown in Figure 5.1, consists of the output map and the bottom-right consists of maps of the different input parameters, namely rainfall, land cover, slope and soil. More details about this functionality of map visualization are given in the Section 5.3.3. The user interface was implemented using HTML, JavaScript a client-side scripting language, and a stylesheet language CSS (Cascading Style Sheets).

5.3 System prototype functionality

The main functionality of our web-based prototype system is to compute the annual average of soil loss caused by precipitation using the USLE-modified model, 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 erosion index and gross erosion rate.

5.3.1 Tools and techniques used

A SOA approach was adopted to make the application more flexible, with reusable and loosely coupled processes. To achieve this, the involved processes mentioned above, were modeled using the Business Process Management (BPM) approach. The BPM provides tools and techniques used to analyse, design and

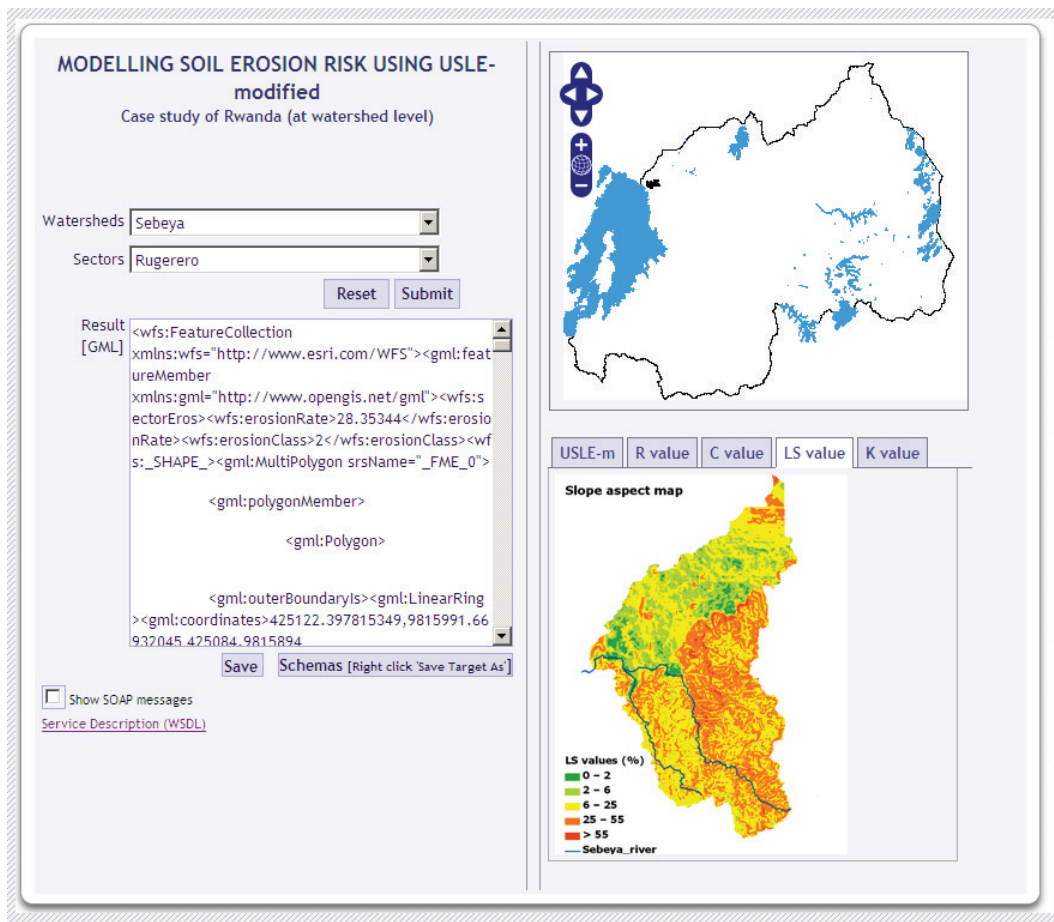


Figure 5.1: User interface

manage business processes. The business processes consist of a set of structured tasks or activities that together produce a service and meet a given business need, that means, that they define how a service is carried out to fulfil its goals [Jos07].

Programming languages

XQuery: XQuery was used to define the functions required to implement our system prototype (see Chapter 2 for more on XQuery).

XRPC: XRPC was used to support remote function calls (see Section 4.3.3 for more on XRPC).

BPEL: BPEL or WSBPEL (Web Services Business Process Execution Language) is an XML-based language used for composing and orchestrating services that result in Web service [Dia07].

WSDL: WSDL (Web Service Definition Language) is an XML-based language used to describe Web services and how to locate them [W3C07a].

Business process modeling tools

The business process modeling tools are used to model business processes defining their workflow, and describe and publish the processing services [Jos07]. In our research, *eClarus Business Process Modeler for SOA Architects* (eClarus BPMSOA) system was used to model our processes and their workflow, using BPMN (Business Process Modeling Notation) standard. The resulting business process diagram is shown in Figure 5.2. This system was chosen because of its feature functionality, its system requirements that are not too demanding and its user-friendliness. Moreover, it supports different standards for business process modeling such as BPMN, BPEL, and Web services standards such as WSDL and UDDI.

Business process modeling techniques

Currently, there are a number of specifications to support business process modeling such as the BPMN standard, used in the current research. The BPMN is a standard developed by the Business Project Management Initiative, with the goal of providing a standardized and readily understandable graphical notation, to represent the processes and their workflow. It specifies the graphical notation used to draw a business process diagram (BPD) and its semantics. The semantics of the BPMN graphical notation are used to bridge the BPDs with other specifications such as BPEL, used to implement Web services.

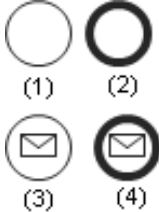


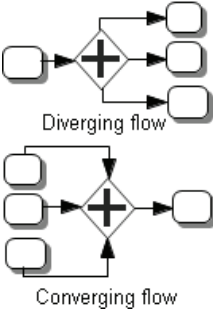
The BPMN specification does not describe how BPMN is translated or mapped to BPEL. But the translation of BPMN to BPEL has been done in several researches (such as [Whi05] and [OvdADtH06]) and has been implemented in several business process management systems such as *eClarus BPMSOA*, used in the current research. The BPD drawn for our system prototype is illustrated in Figure 5.2. It represents the workflow of the processes, detailed in Section 4.1.2, involved in the provision of service to the users of our system prototype. The description of the graphical notation elements used is given in Table 5.1. We only describe the graphical notations used in our research, while more details can be found in [Obj07].

Considering that the adoption of a SOA approach consists of defining and making available the services, so that they can be accessed independently in standardized way, SOA can be coupled with BPM to implement a flexible and effective application. That means that BPM tools and techniques can be used to implement a SOA-enabled application.

5.3.2 The soil erosion processing service

The implemented processing service for assessing the soil erosion, starts by selecting the site of interest. When a user selects the watershed and sector names and submits the form, a geometry boundary is returned and used as input to the functions (or processes) for retrieving the required factor values. Once all the factors values are retrieved, they are used as input to the function for computing the erosion index and finally, the erosion index is used to compute

Table 5.1: Modeling elements used in Figure 5.2

Notation	Description
 <p>(1) (2)</p> <p>(3) (4)</p>	<p>These graphical notations are called <i>Events</i> and are used to represent the events that occur during the flow of process, such as the start and end of a process. The start and end events of a process are represented by a circle with a thin line (1) and a circle with a thicker line (2), respectively. The inside of the circle denotes an optional trigger for a Start Event or an optional result for an End Event. In our case, it is the arrival of a request message that starts the process (3), while a response message is sent back to the calling participant at the end of the process (4).</p>
	<p>This graphical notation is called <i>Activity</i> and is used to represent a task performed in a business process.</p>
	<p>This graphical notation is called <i>Sequence Flow</i> and is used to indicate the order of execution of the activities. A <i>Sequence Flow</i> must have one source and one target.</p>
 <p>Diverging flow</p> <p>Converging flow</p>	<p>The graphical notation represented by a diamond is called a <i>Gateway</i>. It is used to control the <i>Sequence Flow</i> within a business process. Gateways allow to converge and diverge the <i>Sequence Flow</i> and to make use of decisions within a business process. The inside of the diamond denotes the type of Gateway. For instance, in our BPD (Figure 5.2), there is a plus sign inside the diamonds, which denotes that they are Parallel Gateways. Parallel Gateways are used to create and synchronize the parallel flows. For the converging flow, the process continues after receiving a signal from all the parallel flows. For the diverging flow, the process selects a signal from each parallel flow.</p>

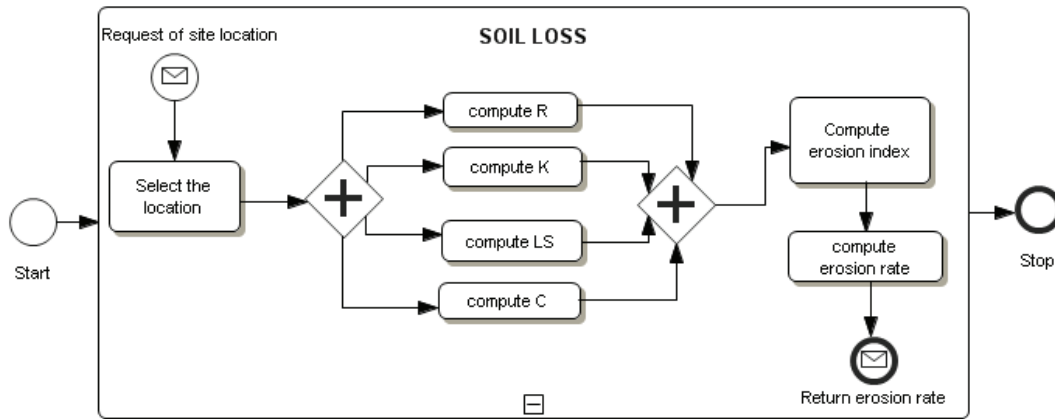


Figure 5.2: Process workflow for computing the annual average of soil loss

the gross erosion rate of the selected site. For instance, the process of retrieving the land cover factor values is handled by the XQuery function below:

```

module namespace er="erosFunctions"; declare function er:landcover
($watershed as xs:string,$sector as xs:string) as element()*
{<wfs:FeatureCollection xmlns:gml="http://www.opengis.net/gml"
xmlns:wfs="http://www.esri.com/WFS">
{
let $s:= exactly-one(geoxml:geometry(doc("sector.gml")
//*:sectors[*:SECT_NAME=$sector and *:BV_NAME=$watershed]
//*: _SHAPE_))
for $cover in doc("cover.gml")//*:Vegetation_cover
let $c:=exactly-one(geoxml:geometry($cover//*: _SHAPE_))
where geoxml:intersection($s,$c)!="GEOMETRYCOLLECTION EMPTY"
return
<gml:featureMember>
<wfs:landcover>
<wfs:cvalue>exactly-one(data($c//*:CVALUE))</wfs:cvalue>
<wfs:_SHAPE_>
er:GeometryFromWKT(geoxml:intersection($s,exactly-one($c)))
</wfs:_SHAPE_>
</wfs:landcover>
</gml:featureMember>
}</wfs:FeatureCollection>
};

```

The function *er:landcover()* is defined inside the module *geom.xq* located on our MonetDB/XQuery server (<http://127.0.0.1:50001/export/geom.xq>). It uses *er="erosFunctions"* as namespace and requires two arguments, the watershed name and the sector name, to locate the site of interest. To retrieve the land cover factor values that correspond with the selected site, the function *geoxml:intersection()* is used to intersect the boundary of the selected site and the land cover data. Since the function *geoxml:intersection()* returns a geometry in well-known text format, the function *er:GeometryFromWKT()* is defined to convert that geometry into GML format. The functions for retrieving the other USLE factors, namely

the rainfall, slope and soil factor values are also defined in the module *geom.xq* as *er:rainfall()*, *er:slope()* and *er:soil()*, respectively. The function for calculating the erosion index and the gross erosion rate calls these four functions, and below is its algorithm:

Algorithm soilErosion

Input: The watershed and sector name of the site of interest

Output: The gross erosion rate of the selected site in GML format

1. Declare variable *rvalues*, *cvalues*, *lsvalues*, and *kvalues*
2. Call the function *er:rainfall()* with the watershed and sector name as arguments and assign the returned result to the *rvalues* variable.
3. Call the function *er:landcover()* with the watershed and sector name as arguments and assign the returned result to *cvalues* variable.
4. Call the function *er:slope()* with the watershed and sector name as arguments and assign the returned result to *lsvalues* variable.
5. Call the function *er:soil()* with the watershed and sector name as arguments and assign the returned result to *kvalues* variable.
6. Calculate the erosion index multiplying *rvalues*, *cvalues*, *lsvalues* and *kvalues* where their respective geometries intersect.
7. Calculate the gross erosion rate using Formula 4.2.
8. Return the gross erosion rate.

5.3.3 Visualisation of the results

The results returned by the implemented processing service are displayed in a text area of the user interface as a GML object that can be visualized in any WFS/GML viewer. The user interface provides the button *Save*, which allows the user to save the output on disk, and a link through which the application schema that validates the output can be downloaded. For instance, Figure 5.3 shows the results returned if the user chooses Sebeya watershed and Rugerero sector as site of interest (visualised using ArcMap).

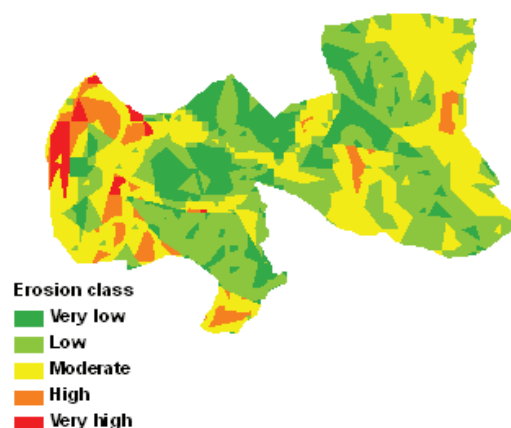


Figure 5.3: Visualising the GML results in ArcMap

5.3.4 Web service orchestration

A service consists of a function that performs a task (or many tasks); it is self-contained and independent. A service can be called a Web service if it is accessible over the Internet, so it should be described to provide meta-information about how it can be accessed. This description must provide the message formats used, since Web services use XML-based messaging mechanism to communicate with each other. To describe Web services, the WSDL standard is used [W3C07a]. Moreover, a Web service is published, which makes it discoverable, i.e. that it can be found through a UDDI (Universal Description, Discovery and Integration) interface. Publishing Web services is optional and is done using the UDDI standard.

As mentioned above, 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. For instance, in our case, we have a business process to assess the soil loss caused by erosion that involves a number of other processes to be accomplished. The BPEL standard is used to orchestrate our Web service, using *eClarus BPMSOA* system. Before orchestrating our Web service, we described it according to the WSDL 1.1 standard. To do so, we created a WSDL document composed by the following elements:

- `<types>` element describes the used data types.
- `<message>` elements describe the message used, they define the data elements of an operation. For instance, the message named *siteLocationRequest* contains *watershedName* and *sectorName* elements used by the operation *siteLocation*, as shown in Figure 5.4, a fragment of the WSDL document.
- `<portType>` elements describe the operations carried out by the Web service. They define the input and output message required by the operation.
- `<binding>` elements describe the communication protocols used by each of the operations of the Web service.
- `<service>` elements describe the services of the Web services, specify the ports, bindings and location of the services.

Based on the BPD created to model the workflow of processes (Figure 5.2), 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 notation specified while drawing the BPD. The resulting BPEL document is given in Appendix D. It is composed of `<variables>` element and `<sequence>` element. The `<sequence>` element contains four elements, namely `<receive>`, `<invoke>`, `<flow>` and `<reply>`. Note that all these elements are sub-elements of the `<process>` element, used to define the Web service being orchestrated (named *SoilErosionAssessment* in our case, as shown in Appendix D.1).

A BPEL document is composed of the following elements:

```

<wsdl:message name="siteLocationRequest">
  <wsdl:part name="watershedName" type="xsd:string"/>
  <wsdl:part name="sectorName" type="xsd:string"/>
</wsdl:message>
:
<wsdl:portType name="LocateSitePT">
  <wsdl:operation name="siteLocation">
    <wsdl:input message="usle:siteLocationRequest"/>
    <wsdl:output message="usle:siteLocationResponse"/>
  </wsdl:operation>
</wsdl:portType>
:
<wsdl:binding name="sitelocationBinding" type="usle:LocateSitePT">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="siteLocation">
    <soap:operation soapAction="http://www.example.org/uslewsdl/siteLocation" style="rpc"/>
    <wsdl:input>
      <soap:body use="literal" namespace="http://www.example.org/uslewsdl/"
        parts="watershedName sectorName" type="soap:tbody"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" namespace="http://www.example.org/uslewsdl/"
        parts="siteBoundary" type="soap:tbody"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
:
<wsdl:service name="sitelocationService">
  <wsdl:port name="usle:LocateSitePT" binding="usle:sitelocationBinding">
    <soap:address location="http://127.0.0.1:50001/export/geom.xq"/>
  </wsdl:port>
</wsdl:service>

```

Figure 5.4: A fragment of the WSDL document

- `<variables>` element defines the variables used to define the sequence flow, namely the messages used when services are communicating.
- `<sequence>` element contains the elements used to define the tasks to be performed sequentially.
- `<flow>` element defines the parallel tasks, i.e., tasks to be performed concurrently.
- `<receive>` element defines the starting operation, it specifies the name of that operation and its input variable (message).
- `<invoke>` element defines the called service.
- `<reply>` element defines the end of the process and the output of the process.

After setting up the BPEL document for *SoilErosionAssessment* process, the process was then executed on the orchestration engine provided by *eClarus BPMSOA* system.

Chapter 6

Discussions, conclusions and recommendations

In this thesis, we designed and implemented a distributed geoprocessing system prototype 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.

6.1 Discussions

In this section we outline and discuss the answers to the research questions put forward in Section 1.2.2. This discussion recapitulates the achievements of our research in accordance with the addressed research objectives, and the limitations and problems encountered.

1. *Can an XML database system be used to manipulate the amount of spatial data involved in a typical geoprocessing workflow?*

To manipulate the spatial data involved in a typical geoprocessing workflow using an XML database system, the following were required: a standard for encoding the spatial data and a query language to query and manipulate the spatial data within an XML database. GML was used as spatial data encoding standard, since it is based on XML and can be stored and manipulated in an XML database management system, as detailed in Chapter 3. For the query language, we used XQuery. After enquiring into what XQuery offers to retrieve and manipulate XML data, in Chapter 2, we showed that XQuery needs to be extended with spatial semantics and proposed a syntax for spatial functions in XQuery based on the GML data model (Chapter 3). The proposed syntax was used to implement a few spatial functions in an XML database management system (namely MonetDB/XQuery), as an extension of XQuery to support spatial functionality. The issue of the amount of spatial data involved in a typical geoprocessing

workflow was addressed in Chapter 5, where we described the implementation of our geoprocessing system prototype that involves a number of spatial datasets using an XML database system as back-end.

2. *How to optimize the geoprocesses on GML data sources stored in an XML database?*

This question was addressed throughout our research while writing and executing queries in XQuery over GML data. To optimize the geoprocesses on GML data sources stored in an XML database, we used large GML documents instead of using many small documents to store our data in the database. This allowed us to avoid increasing the FLWOR nesting level, as the more you select documents the more the FLWOR nesting level increases. The FLWR expression are described in Section 2.4. A more reliable way to optimize the geoprocesses on GML data sources stored in an XML database would be using spatial indexes, but these are not yet implemented in the XML database system used.

3. *How to design a geoprocessing service orchestration to implement faster web geoprocessing applications?*

To design a geoprocessing service orchestration for implementing faster web geoprocessing applications, we used the MVC architectural pattern adapted to SOA principles. This approach allowed us to separate the application functionality and data from the presentation, as described in Section 4.3. The proposed approach was applied for designing and implementing a web geoprocessing application for assessing the soil erosion caused by rainfall, in a distributed environment. After identifying the processes involved in soil erosion assessment in Section 4.1.2, we modeled them using BPM tools and techniques which facilitate the implementation of a SOA-enabled application. Furthermore, we provided our geoprocessing service (assessing soil erosion) through Web service orchestration, defining the sequence flow of the involved processes as described in Section 5.

Limitations and problems

The limitations and problems encountered during our research consist of:

- The XML database system used, namely MonetDB/XQuery, currently has a very limited spatial functionality and it lacks a number of features. For instance, the new front-end does not currently support recursive functions.
- We could not use the OpenGIS Web service, since they are not supported in the database system used.

6.2 Conclusions

The objectives of this research were achieved and the research questions answered. A distributed system prototype, that provides web geoprocessing service, was designed and implemented using an XML database system as back-end and GML as data encoding standard. We proposed a syntax for spatial queries in XQuery to support the manipulation of GML data stored in an XML database, based on GML data model. A scenario was chosen, namely assessment of soil erosion caused by rainfall, to apply the proposed design and implementation approaches. After describing and analysing the requirements of the assessment of soil erosion caused by rainfall, we proposed a suitable system design combining the MVC architectural pattern with SOA principles. The required geoprocesses were implemented using XQuery and provided through Web service orchestration. 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.

6.3 Recommendations

For further research projects to improve the outcome of this research, we recommend the following:

- Implementation of OpenGis web services (such as Web Map Service, Web Feature Service, Catalog Service, ...) using an XML database system as back-end.
- The implementation of a complete library of spatial functions in MonetDB/XQuery.
- The implementation of spatial indexes in MonetDB/XQuery XML database.
- As an improvement of the implemented system, we recommend a study for the 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.

Appendix A

A.1 Schema for basic GML objects

The following is used to define a basic GML object in a schema:

```
<elementname="AbstractGML" type="gml:AbstractGMLType"
abstract="true" substitutionGroup="gml:AbstractObject"/>

<complexTypename="AbstractGMLType" abstract="true">
  <sequence>
    <group ref="gml:StandardObjectProperties"/>
  </sequence>
  <attribute ref="gml:id" use="required"/>
</complexType>
<group name="StandardObjectProperties">
  <sequence>
    <element ref="gml:metaDataProperty" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="gml:description" minOccurs="0"/>
    <element ref="gml:descriptionReference" minOccurs="0"/>
    <element ref="gml:identifier" minOccurs="0"/>
    <element ref="gml:name" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</group>
```


Appendix B

Application schemas of the used GML data (Enschede data)

B.1 Application schema for neighbourhood data

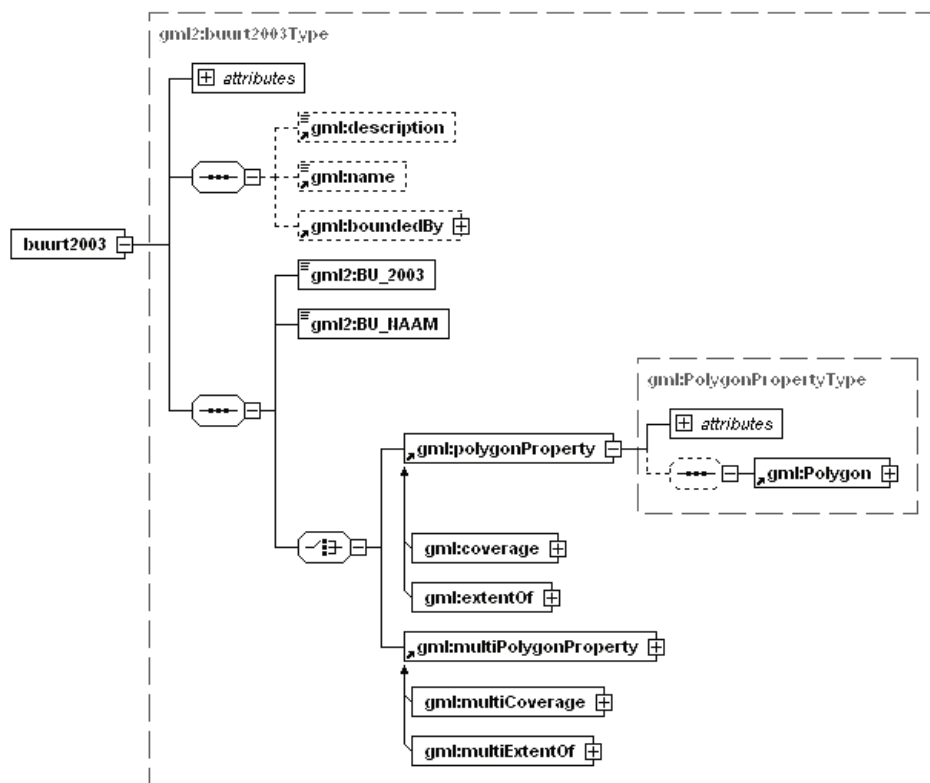


Figure B.1: Application schema for neighbourhood data

B.2 Application schema for land use data

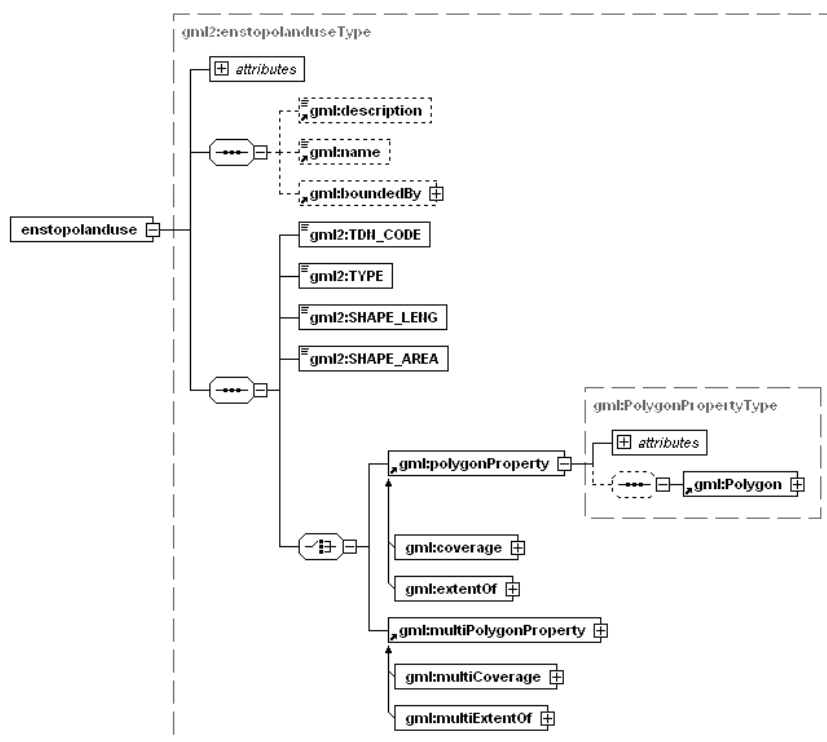


Figure B.2: Application schema for land use data

B.3 Application schema for buildings data

B.4 Application schema for roads data

B.5 Application schema for risk points data

B.6 Application schema for schools data

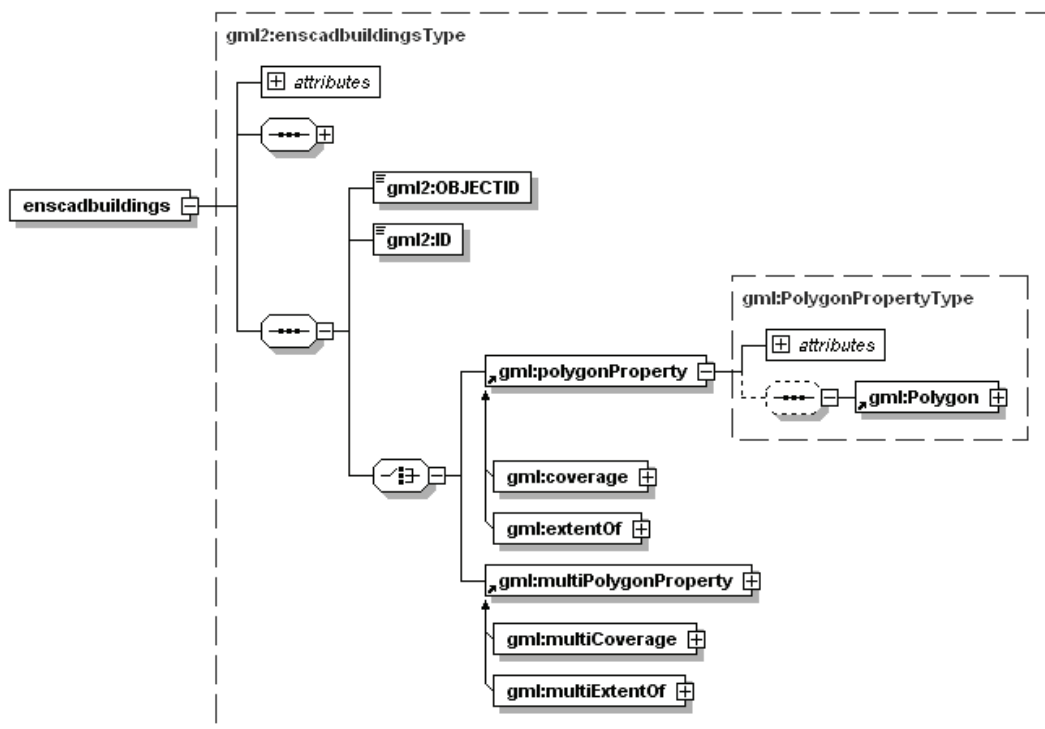


Figure B.3: Application schema for buildings data

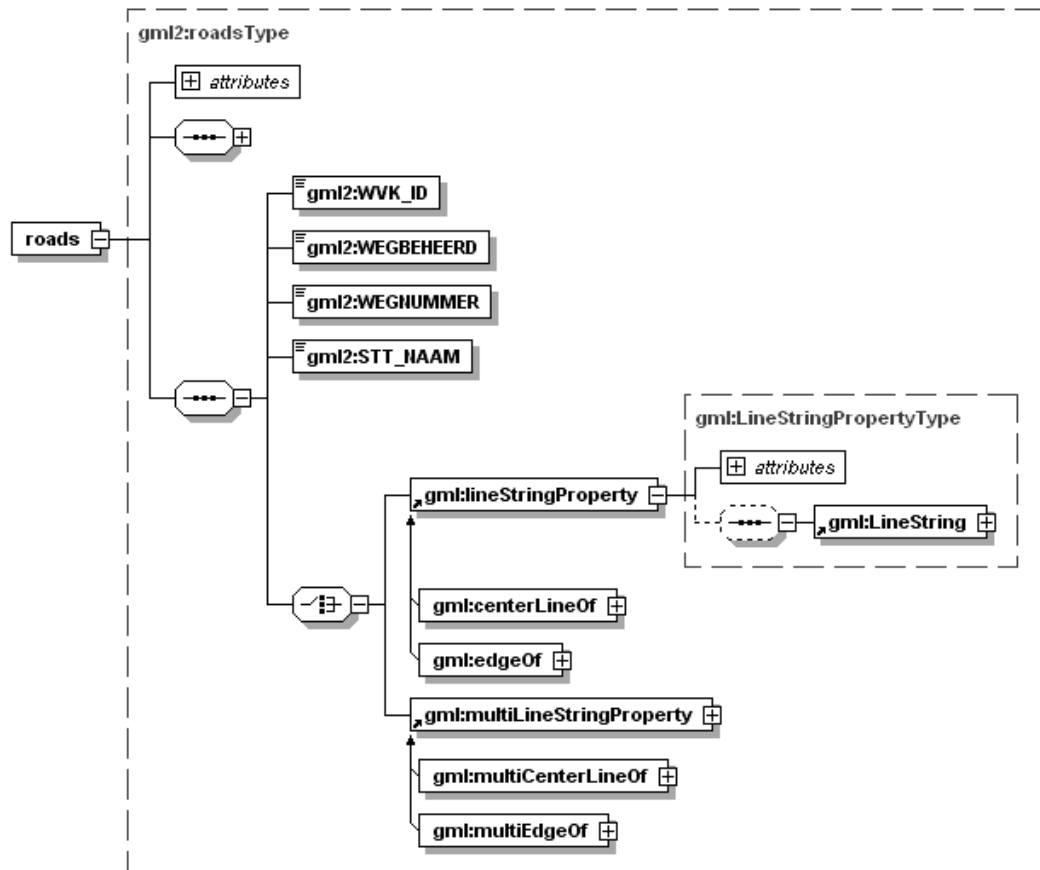


Figure B.4: Application schema for roads data

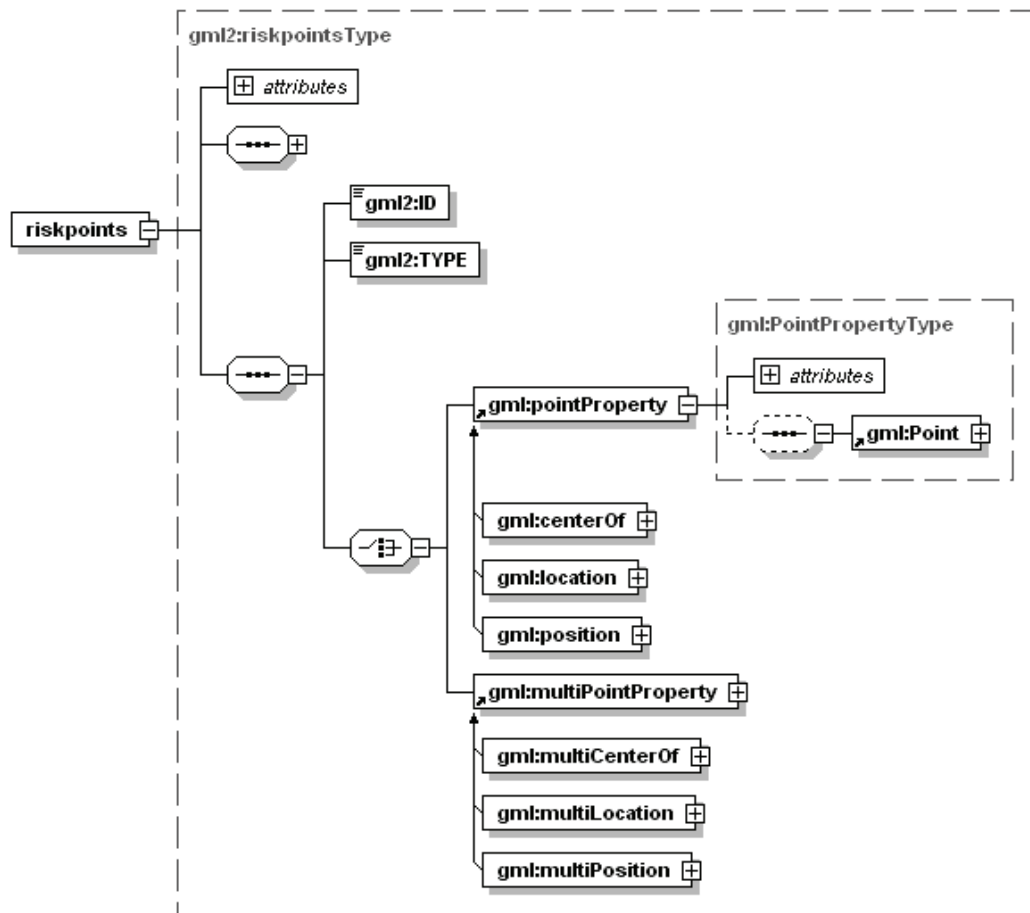


Figure B.5: Application schema for risk points data

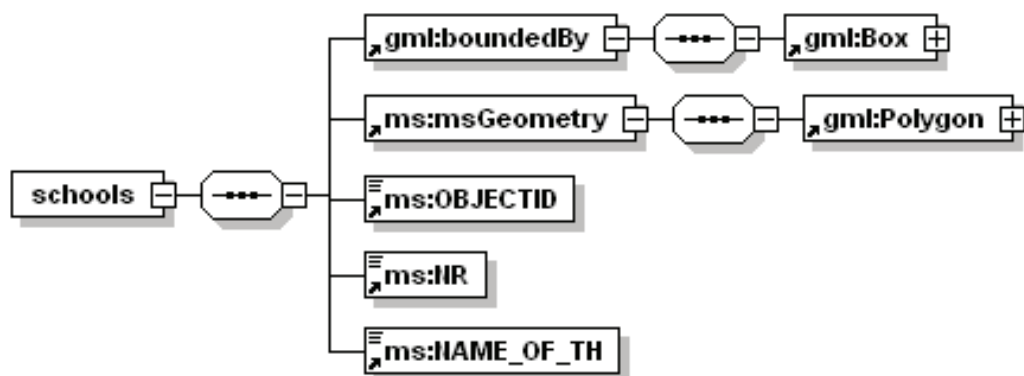


Figure B.6: Application schema for schools data

Appendix C

Application schemas of Sebeya watershed

C.1 Application schema for rainfall data

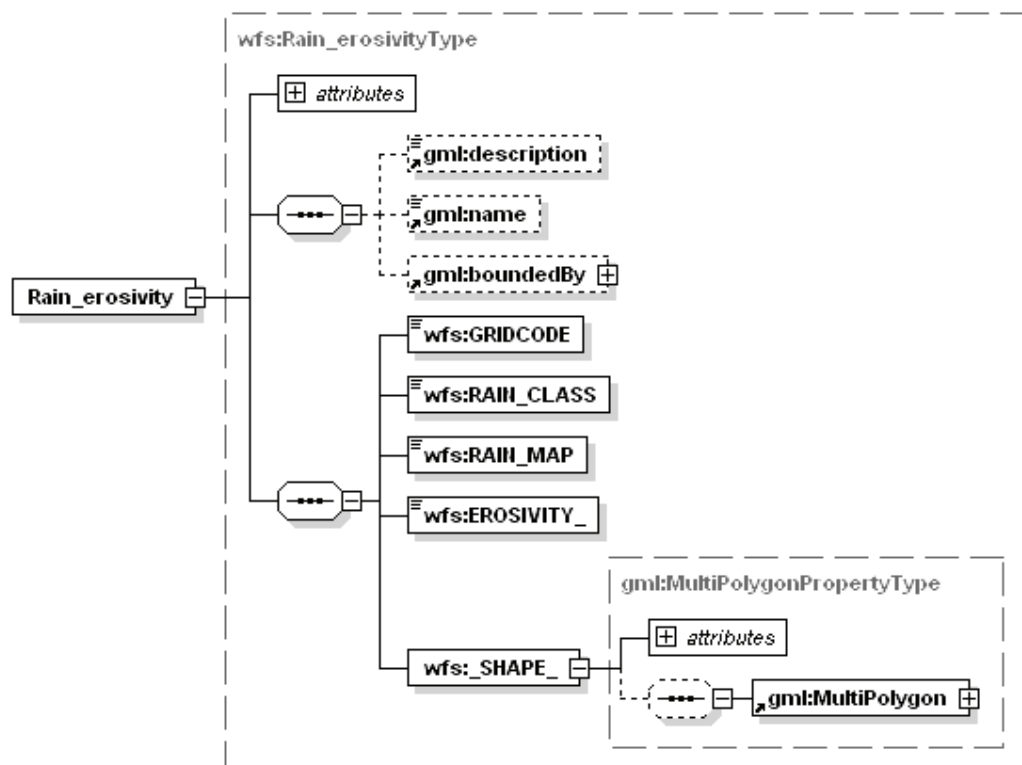


Figure C.1: Application schema for rainfall data

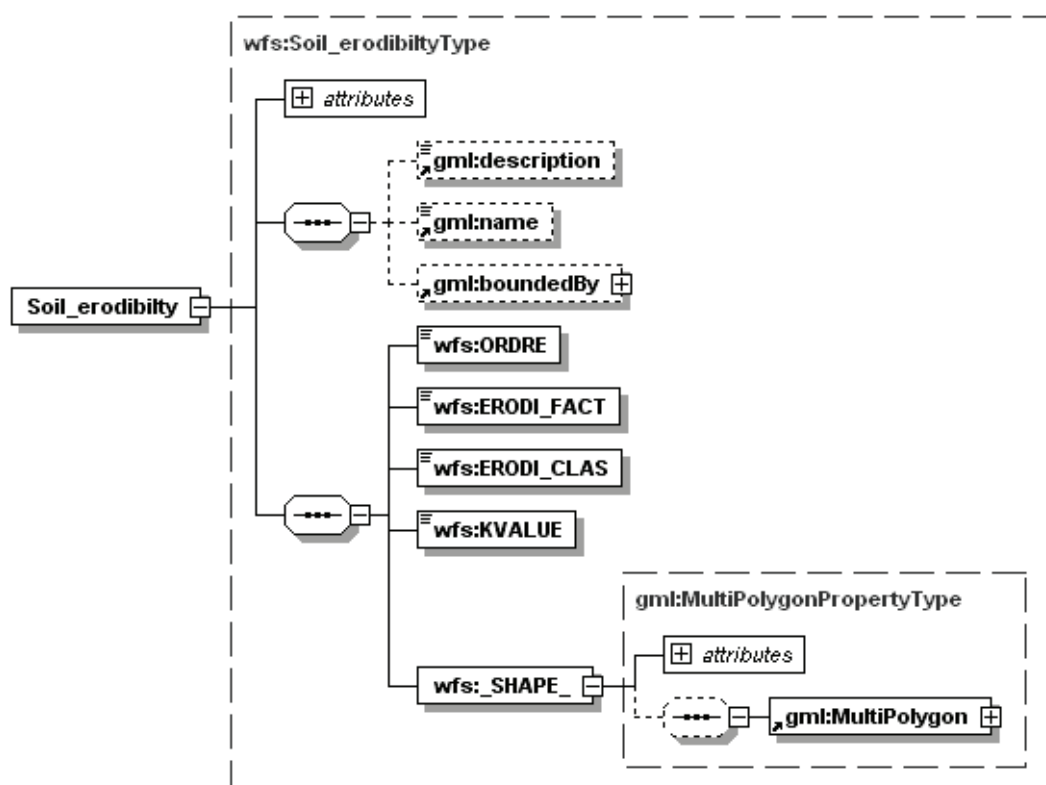


Figure C.2: Application schema for soil data

C.2 Application schema for soil data

C.3 Application schema for topographic data

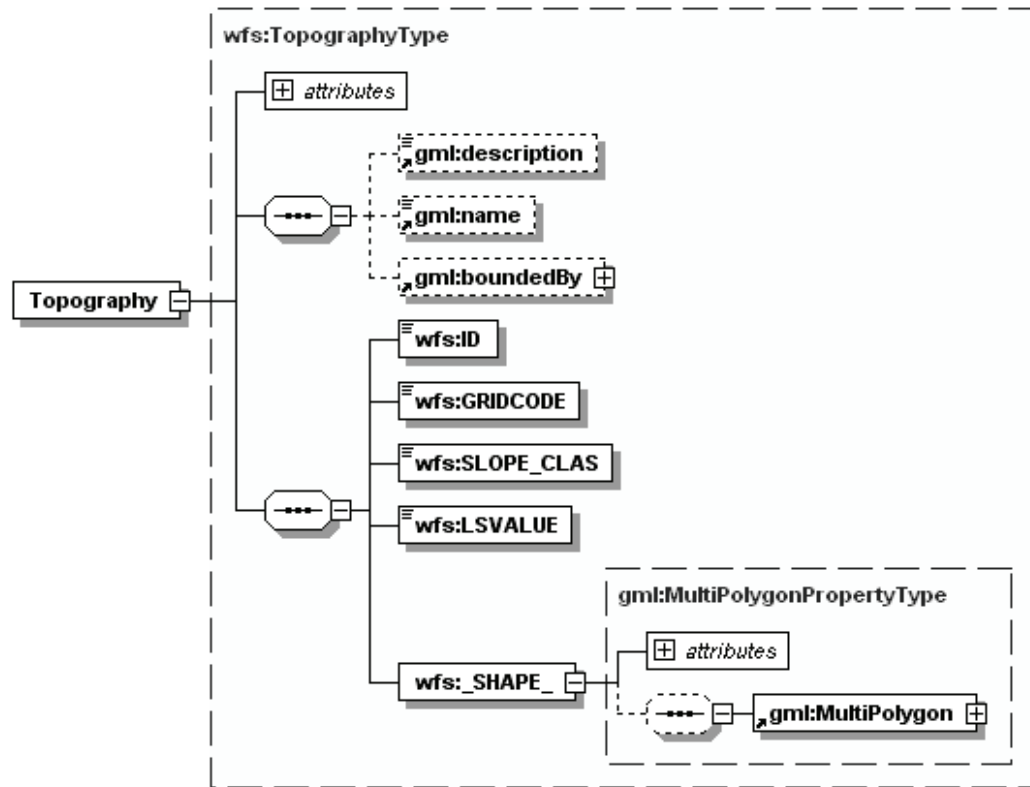


Figure C.3: Application schema for topographic data

C.4 Application schema for land cover data

C.5 Application schema for watershed boundary data

C.6 Application schema for sectors boundaries data

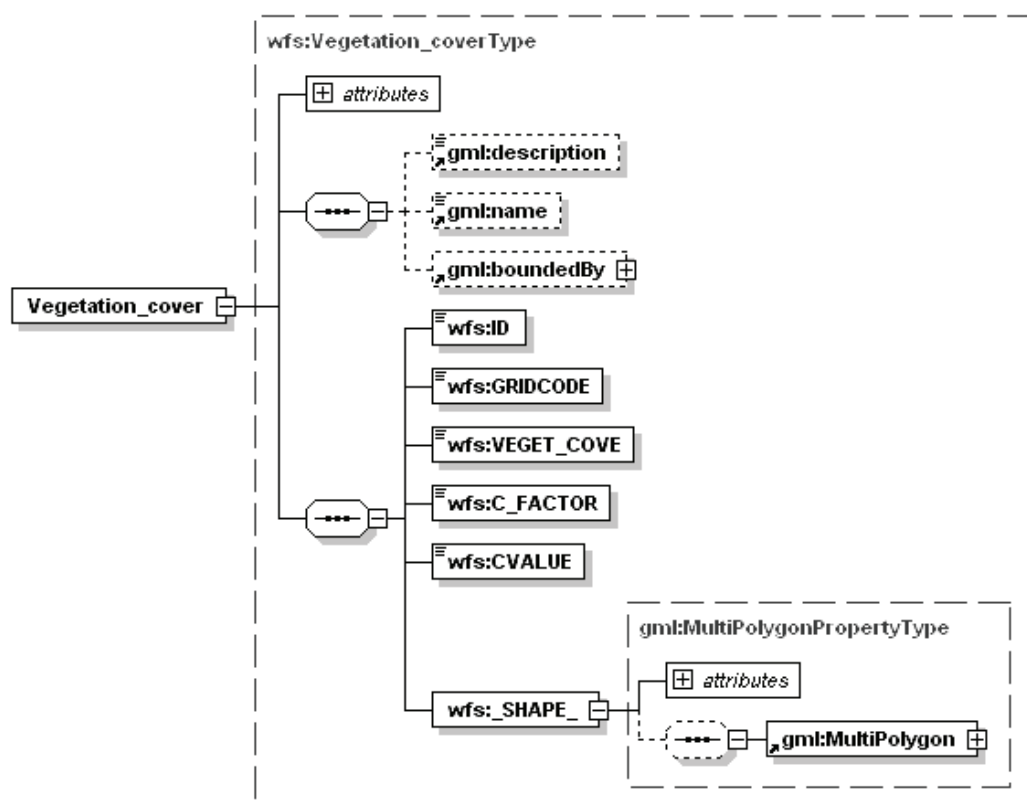


Figure C.4: Application schema for land cover data

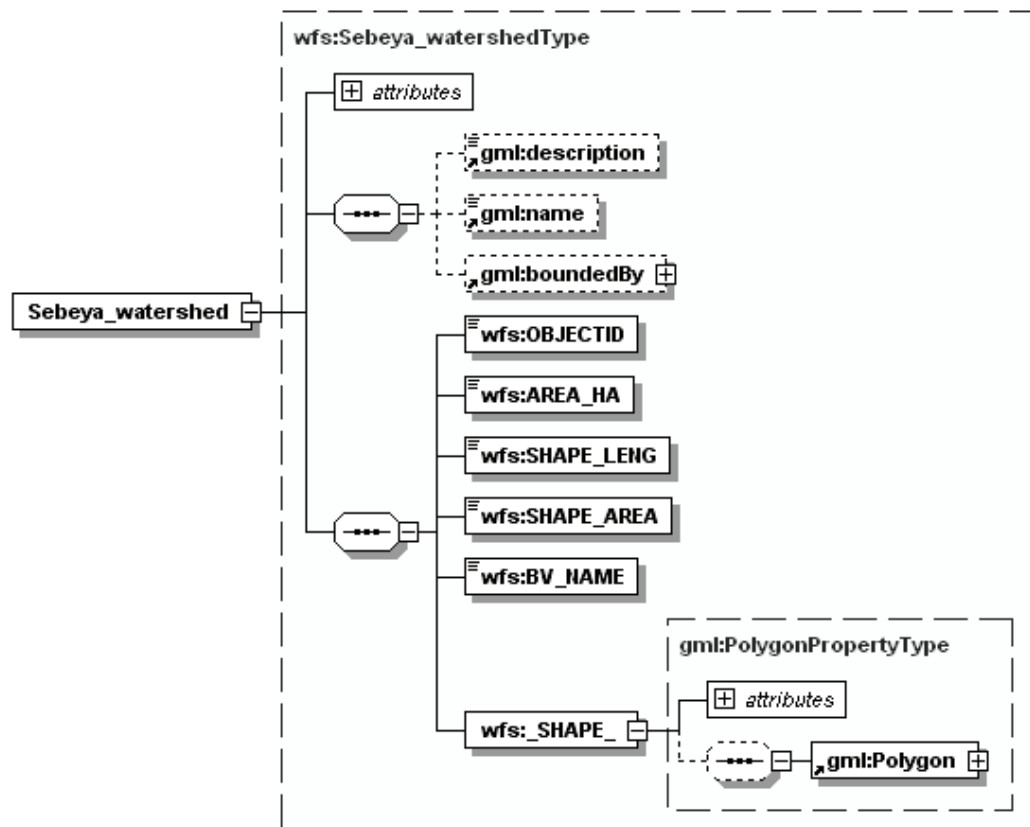


Figure C.5: Application schema for watershed boundary data

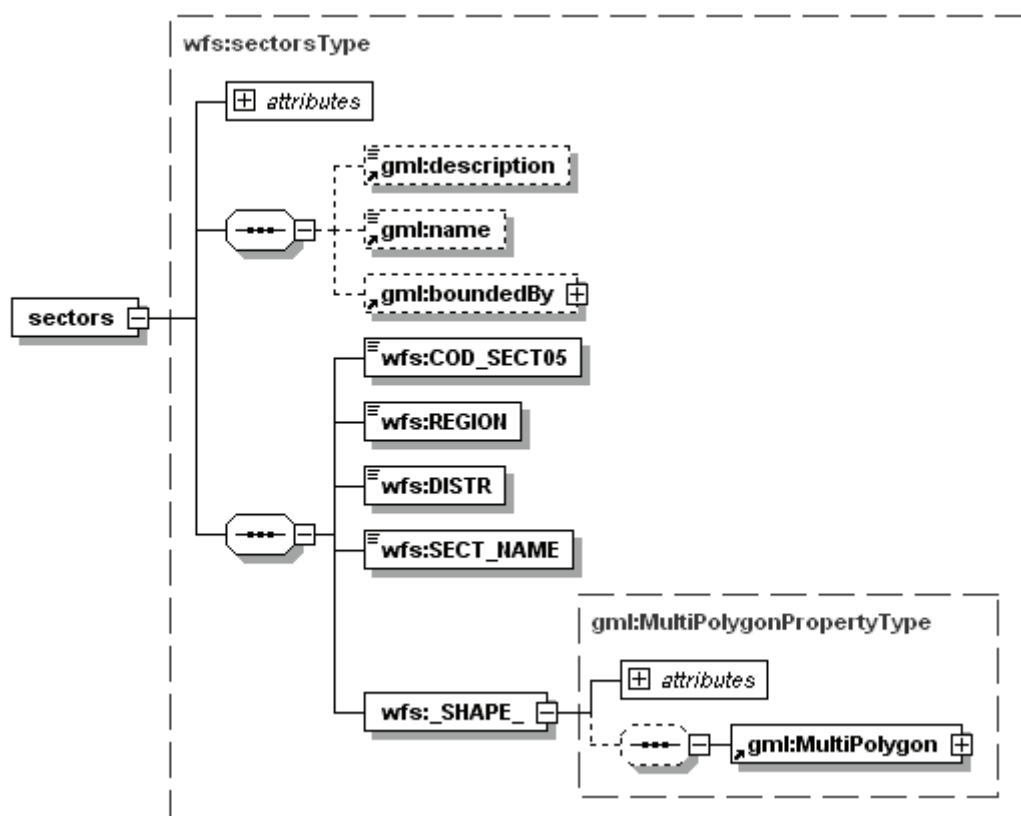


Figure C.6: Application schema for sectors boundaries data

Appendix D

Web service orchestration

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable" xmlns:usle="
http://www.example.org/uslewsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" expressionLanguage
="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0" name="SoilErosionAssessment" queryLanguage="
urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0" suppressJoinFailure="yes">
  <bpel:variables>
    <bpel:variable messageType="usle:siteLocationRequest" name="siteLocationRequest"/>
    <bpel:variable messageType="usle:siteLocationResponse" name="siteResponse"/>
    <bpel:variable messageType="usle:computeR" name="computedRvalue"/>
    <bpel:variable messageType="usle:computeC" name="computedCvalue"/>
    <bpel:variable messageType="usle:computeK" name="computedKvalue"/>
    <bpel:variable messageType="usle:computeLS" name="computedLSvalue"/>
    <bpel:variable messageType="usle:computeE" name="computederosionIndex"/>
    <bpel:variable messageType="usle:computeA" name="computederosionRate"/>
    <bpel:variable messageType="usle:requestE" name="requestSoilErosion"/>
    <bpel:variable messageType="usle:siteLocationResponse" name="siteLocationResponse"/>
  </bpel:variables>
  <bpel:sequence>
    <bpel:receive name="Request_of_site_location" createInstance="yes" variable="siteLocationRequest"/>
    <bpel:invoke name="Select_the_location" inputVariable="siteLocationRequest" outputVariable="
siteLocationResponse"/>
    <bpel:flow>
      <bpel:invoke name="compute_K" inputVariable="siteLocationResponse" outputVariable="
computedKvalue"/>
      <bpel:invoke name="compute_LS" inputVariable="siteLocationResponse" outputVariable="
computedLSvalue"/>
      <bpel:invoke name="compute_C" inputVariable="siteLocationResponse" outputVariable="
computedCvalue"/>
      <bpel:invoke name="compute_R" inputVariable="siteLocationResponse" outputVariable="
computedRvalue"/>
    </bpel:flow>
    <bpel:invoke name="Compute_erosion_index" inputVariable="requestSoilErosion" outputVariable="
computederosionIndex"/>
    <bpel:invoke name="compute_erosion_rate" inputVariable="computederosionIndex" outputVariable="
computederosionRate"/>
    <bpel:reply name="Return_erosion_rate" inputVariable="computederosionRate"/>
  </bpel:sequence>
</bpel:process>
```

Figure D.1: BPEL document

Bibliography

- [A.97] Nearing M. A., *A Single, Continuous Function for Slope Steepness Influence on Soil Loss*, Soil Science Society of America **61** (1997), no. 3, 917–919.
- [ABS00] S. Abiteboul, P. Buneman, and D. Suciu, *Data on the web : From relations to semistructured data and XML*, Morgan Kaufmann, 2000.
- [AMNS04] Elena Amore, Carlo Modica, Mark A. Nearing, and Vincenza C. Santoro, *Scale effect in USLE and WEPP application for soil erosion computation from three Sicilian basins*, Journal of Hydrology **293** (2004), no. 1-4, 100–114.
- [BGM⁺05] Peter Boncz, Torsten Grust, Stefan Manegold, Jan Rittinger, and Jens Teubner, *Pathfinder: relational XQuery over multi-gigabyte XML inputs in interactive time*, Tech. Report INS-E0503, CWI, Amsterdam, The Netherlands, 2005.
- [BGvK⁺05] Peter Boncz, Torsten Grust, Maurice van Keulen, Stefan Manegold, Jan Rittinger, and Jens Teubner, *Pathfinder: XQuery–The Relational Way*, VLDB ’05: Proceedings of the 31st International Conference on Very large data bases, VLDB Endowment, 2005, pp. 1322–1325.
- [BGvK⁺06] Peter A. Boncz, Torsten Grust, Maurice van Keulen, Stefan Manegold, Jan Rittinger, and Jens Teubner, *MonetDB/XQuery: a fast XQuery processor powered by a relational engine*, Proceedings of the 2006 ACM SIGMOD international conference on Management of data, Chicago, IL, USA (New York, NY, USA), ACM Press, June 2006, pp. 479–490.
- [Blo92] John Bloomer, *Power Programming with RPC*, O’Reilly, 1992.
- [BMR05] Peter Boncz, Stefan Manegold, and Jan Rittinger, *Updating the Pre/Post plane in MonetDB/XQuery*, Proceedings of the ACM SIGMOD/PODS 2nd International Workshop on XQuery Implementation, Experience and Perspectives, 2005.
- [BQK96] Peter A. Boncz, Wilko Quak, and Martin L. Kersten, *Monet and its Geographical Extensions: a Novel Approach to High-*

- Performance GIS Processing*, In Proceedings of the International Conference on Extending Database Technology (EDBT), Springer-Verlag, 1996, pp. 147–166.
- [CG01] J. E. Córcoles and P. González, *A specification of a spatial query language over GML*, GIS '01: Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems (New York, NY, USA), ACM, 2001, pp. 112–117.
- [CL96] Daniel C. Clay and Laurence A. Lewis, *Land Use, Soil Loss and Sustainable Agriculture in Rwanda*, International Development Collaborative Working Papers RW-FSRP-RR-01, Department of Agricultural Economics, Michigan State University, 1996.
- [CRZ03] Akmal B. Chaudhri, Awais Rashid, and Roberto Zicari, *XML Data management: native XML and XML-enabled database systems*, Addison Wesley, 2003.
- [CSW05] Matthew J. Cohen, Keith D. Shepherd, and Markus G. Walsh, *Empirical reformulation of the universal soil loss equation for erosion risk assessment in a tropical watershed*, *Geoderma* **124** (2005), no. 3-4, 235–252.
- [Dat06] Database Promotion Center, *G-XML project*, 2006, Available at <http://www.dpc.jpipdec.or.jp/gxml/contents-e/>.
- [Dia07] Diane Jordan and John Evdemon, *Web Services Business Process Execution Language version 2.0*, Oasis standard, Organization for the Advancement of Structured Information Standards, 2007, Available at <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>.
- [EEB06] Emrah H. Erdogan, Gnay Erpul, and Ilhami Bayramin, *Use of USLE/GIS Methodology for Predicting Soil Loss in Semiarid Agricultural Watershed*, *Environmental Monitoring and Assessment* **131** (2006), no. 1, 153–161.
- [FCOLB07] Anders Friis-Christensen, Nicole Ostländer, Michael Lutz, and Lars Bernard, *Designing service architectures for distributed geoprocessing: Challenges and future directions*, *Transactions in GIS* **11** (2007), no. 6, 799–818.
- [FH02] Okan Fistikoglu and Nilgun B. Harmancioglu, *Integration of GIS with USLE in assessment of soil erosion*, *Water Resources Management* **16** (2002), 447–467.
- [FS07] Theodor Foerster and Bastian Schäffer, *A client for distributed geo-processing on the web*, Proceedings of Web and Wireless Geographical Information Systems : 7th International Symposium (J.M. Ware and G.E. Taylor, eds.), Springer-Verlag, November 2007, pp. 252–263.

- [GO] Jody Garnett and Brent Owens, *Spatial validation — Academic References*, Tech. report, Refractions Research, Inc., Available at http://vwfs.refractions.net/docs/Spatial_Validation_Academic.pdf.
- [Gru02] Torsten Grust, *Accelerating XPath location steps*, SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data (New York, NY, USA), ACM Press, 2002, pp. 109–120.
- [GTS⁺00] David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, and Sailu Reddy, *HTTP: The Definitive Guide*, O'Reilly, 2000.
- [HCDL06] Chia-Hsin Huang, Tyng-Ruey Chuang, Dong-Po Deng, and Hahn-Ming Lee, *Efficient GML-native processors for web-based GIS: techniques and tools*, GIS '06: Proceedings of the 14th annual ACM international symposium on Advances in Geographic Information systems (New York, NY, USA), ACM, 2006, pp. 91–98.
- [Hil98] Daniel Hillel, *Environmental soil physics*, Environmental Soil Physics, 1998.
- [JBR99] I. Jacobson, G. Booch, and J. Rumbaugh, *Unified software development process*, Addison Wesley, 1999.
- [Jih06] Guan Jihong, *GQL: Extending XQuery to query GML documents*, Geo-Spatial Information Science (2006), 118–126.
- [Jos07] Nicolai M. Josuttis, *SOA in Practice: the Art of Distributed System Design*, O'Reilly, 2007.
- [KCK⁺03] Howard Katz, Don Chamberlin, Michael Kay, Philip Wadler, and Denise Draper, *XQuery from the experts: A guide to the W3C XML Query Language*, Addison-Wesley Longman Publishing Co., Inc., 2003.
- [KGH07] Christian Kiehle, Klaus Greve, and Christian Heier, *Requirements for Next Generation Spatial Data Infrastructures-Standardized Web Based Geoprocessing and Web Service Orchestration*, Transactions in GIS **11** (2007), no. 6, 819–834.
- [Kie06] Christian Kiehle, *Business logic for geoprocessing of distributed geodata*, Computers & Geosciences **32** (2006), no. 10, 1746–1757.
- [Lal94] Rattan Lal (ed.), *Soil erosion research methods*, 2nd ed., Soil and Water Conservation Society and St. Lucie Press, 1994.
- [LBTR04] Ron Lake, David Burggraf, Milan Trninic, and Laurie Rae, *Geography Mark-Up Language: Foundation for the Geo-Web*, John Wiley & Sons, 2004.

- [LJD01] Simon St. Laurent, Joe Johnston, and Edd Dumbill, *Programming Web Services with XML-RPC*, O'Reilly, 2001.
- [M.82] Rowntree K. M., *Rainfall erosivity in Kenya: some preliminary considerations*, Proceedings of the Second National Workshop on Soil and Water Conservation in Kenya, no. 42, 1982, pp. 1–19.
- [MAS⁺03] James McGovern, Scott W. Ambler, Michael E. Stevens, James Linn, Vikas Sharan, and Elias K. Jo, *A practical guide to Enterprise Architecture*, Prentice Hall, 2003.
- [Mon08] CWI, *MonetDB/XQuery Reference Manual*, 2008, Available at <http://monetdb.cwi.nl/Assets/xqrymanual/xqrymanual.pdf>.
- [NSCE99] Darrell Norton, Issac Shainberg, Larry Chihacek, and J.J. Edwards, *Soil quality and soil erosion*, ch. 3, pp. 36–56, Soil and Water Conservation Society, 1999.
- [Obj07] Object Management Group, *Business Process Modeling Notation v1.2*, OMG Specification, Object Management Group, 2007, Available at <http://www.omg.org/docs/dtc/08-02-07.pdf>.
- [Ope05] Open Geospatial Consortium Inc., *GML simple features profile*, OpenGIS Standard, Open Geospatial Consortium Inc., 2005, Available at portal.opengeospatial.org/files/index.php?artifact_id=15201.
- [Ope07] Open Geospatial Consortium, Inc., *OpenGIS Geography Markup Language (GML) Encoding Standard*, OpenGIS Standard, Open Geospatial Consortium, Inc., 2007, Available at <http://www.opengeospatial.org/standards/gml>.
- [OvdADtH06] Chun Ouyang, Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede, *Translating BPMN to BPEL*, Available at <http://eprints.qut.edu.au/3615/1/3615.pdf>, 2006.
- [Pat08] *Pathfinder/Tijah*, 2008, Available at <http://dbappl.cs.utwente.nl/pftijah/>.
- [PRT06] Eric Pardede, J. Wenny Rahayu, and David Taniar, *XML-Enabled Relational Database for XML Document Update*, Advanced Information Networking and Applications, 2006. 20th International Conference on **2** (2006), 205–212.
- [Ree79] Trygve Reenskaug, *The original MVC reports*, Tech. report, Department of Informatics, University of Oslo, 1979.

-
- [SYU99] Takeyuki Shimura, Masatoshi Yoshikawa, and Shunsuke Uemura, *Storage and retrieval of XML documents using Object-Relational Databases*, In Database and Expert Systems Applications, Springer, 1999, pp. 206–217.
- [Ten03] W.T.M.S.B. Tennakoon, *Visualization of GML data using XSLT*, Master's thesis, International Institute for Geo-Information Science and Earth Observation, Netherlands, February 2003.
- [Teu00] Jens Teubner, *Pathfinder: XQuery compilation techniques for relational database targets*, Ph.D. thesis, TU München, 2000.
- [Vat02] Ranga Raju Vatsavai, *GML-QL: A spatial query language specification for GML*, 2002.
- [W3C03] W3C, *SOAP Version 1.2 Part 2: Adjuncts*, W3C recommendation, World Wide Web Consortium, June 2003, Available at <http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#soapforrpc>.
- [W3C04a] ———, *XML Information Set (second edition)*, W3C recommendation, World Wide Web Consortium, 2004, Available at <http://www.w3.org/TR/xml-infoset/>.
- [W3C04b] ———, *XML Schema Part 0: Primer second edition*, W3C recommendation, World Wide Web Consortium, 2004, Available at <http://www.w3.org/TR/xmlschema-0/>.
- [W3C04c] ———, *XML Schema Part 1: Structures second edition*, W3C recommendation, World Wide Web Consortium, 2004, Available at <http://www.w3.org/TR/xmlschema-1/>.
- [W3C04d] ———, *XML Schema Part 2: Datatypes second edition*, W3C recommendation, World Wide Web Consortium, 2004, Available at <http://www.w3.org/TR/xmlschema-2/>.
- [W3C06] ———, *Namespaces in XML 1.1 (second edition)*, W3C recommendation, World Wide Web Consortium, 2006, Available at <http://www.w3.org/TR/xml-names11/>.
- [W3C07a] ———, *Web Services Description Language (WSDL) Version 2.0*, W3C recommendation, World Wide Web Consortium, 2007, Available at <http://www.w3.org/TR/wsdl20>.
- [W3C07b] ———, *XML Path Language (XPath) 2.0*, W3C recommendation, World Wide Web Consortium, January 2007, Available at <http://www.w3.org/TR/xpath20/>.
- [W3C07c] ———, *XQuery 1.0: An XML Query Language*, W3C recommendation, World Wide Web Consortium, January 2007, Available at <http://www.w3.org/TR/xquery/>.

- [W3C07d] ———, *XQuery 1.0 and XPath 2.0 Data Model*, W3C recommendation, World Wide Web Consortium, 2007, Available at <http://www.w3.org/TR/xpath-datamodel/>.
- [W3C07e] ———, *XQuery 1.0 and XPath 2.0 formal semantics*, W3C recommendation, World Wide Web Consortium, 2007, Available at <http://www.w3.org/TR/xquery-semantics/>.
- [W3C07f] ———, *XQuery 1.0 and XPath 2.0 Functions and Operators*, W3C recommendation, World Wide Web Consortium, 2007, Available at <http://www.w3.org/TR/xpath-functions/>.
- [W3C07g] ———, *XSL Transformations (XSLT) Version 2.0*, W3C recommendation, World Wide Web Consortium, 2007, Available at <http://www.w3.org/TR/xslt20/>.
- [W3C08a] ———, *Extensible Markup Language (XML) 1.0 Fifth Edition*, W3C recommendation, World Wide Web Consortium, 2008, Available at <http://www.w3.org/TR/xml/>.
- [W3C08b] ———, *XQuery Update Facility 1.0*, W3C recommendation, World Wide Web Consortium, 2008, Available at <http://www.w3.org/TR/xquery-update-10/>.
- [Whi05] Stephen A. White, *Using BPMN to Model a BPEL Process*, Tech. report, International Business Machines Corporation, 2005.
- [Won00] Clinton Wong, *HTTP Pocket Reference*, 1st ed., O'Reilly, 2000.
- [WS98] Wishmeier and Smith, *Predicting Rainfall Erosion Losses – A guide to conservation planning*, 1998.
- [ZB07] Ying Zhang and Peter Boncz, *XRPC: interoperable and efficient distributed XQuery*, VLDB '07: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB Endowment, 2007, pp. 99–110.