User interface for the integration of GIS components

Francisco Julián Gómez Benítez
April, 2002
User interface for the integration of GIS components

by

Francisco Julián Gómez Benítez

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

Chairman Prof. Dr. Wolfgang Kainz
First supervisor Ir. Rob Lemmens
Second supervisor Dr. Ir. Rolf de By
External examiner Prof. Dr. Ir. P. van Oosterom
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

There is a recent tendency in software development to move from monolithic applications to small, portable and reusable pieces of software called components. In the present research this term is adopted to refer to the building blocks of Geographic Information Systems (GIS) analysis: GIS functions and GIS datasets, which correctly assembled in process chains can help to solve generic geographic questions of GIS users. To complement existing Online GIS systems, these components can be made available in a distributed environment letting many new potential users to benefit of GIS functionality. In such a distributed environment it becomes necessary to have proper metadata to identify the components and proper user interfaces that enable the discovery and integration of available components.

In this work some foundation GIS functionalities were developed as independent executable files. These GIS function components together with some selected datasets treated as components, were described with relevant metadata elements in Extensible Markup Language (XML) files. Next, an ‘integrating’ user interface was developed which implemented component discovery and their ‘smart’ integration in valid process chains by using the knowledge stored in the XML files. This user interface was tested for the solution of a generic geographic example question and proved successful in reaching a valid solution.

It has been found that it is possible to offer GIS functionality in the form of separate components and that the use of XML is straightforward in modelling metadata. XML also proved excellent to enable the discovery and assembly of GIS components by the developed interface in which users can pose and solve their geographic questions.

Keywords

GIS components, software components, Process chains, XML Metadata, GIS interoperability, Online GIS, Open GIS
Contents

Abstract i

List of Figures vii

1 Introduction 1
  1.1 Motivation ................................................. 1
    1.1.1 Components in Geographic Information Systems .... 1
    1.1.2 Online GIS ........................................... 2
    1.1.3 GIS datasets ......................................... 3
  1.2 Problem definition ....................................... 3
    1.2.1 Objective ............................................ 4
  1.3 Thesis outline .......................................... 4

2 Background 5
  2.1 Underlying concepts ..................................... 5
    2.1.1 GIS functions, operations or services? ............ 5
    2.1.2 Process chains ....................................... 6
  2.2 Previous components experiences ......................... 7
    2.2.1 Open GIS service architecture .................... 7
    2.2.2 Agent based architecture for imagery and geospatial pro-
        cessing .............................................. 8
    2.2.3 GeoServNet .......................................... 8
    2.2.4 A CORBA-based interoperable GIS ................. 9
    2.2.5 Interoperable Geo Information Systems (IOGIS) .... 10
    2.2.6 Universal Description, Discovery and Integration (UDDI) 
        ............................................... 10
    2.2.7 Method ManageMent .................................. 10
    2.2.8 Geography Network .................................. 11
  2.3 Final remarks ............................................ 11

3 GIS function components 13
  3.1 Select ..................................................... 14
Contents

3.2 Projection .......................... 15
3.3 Buffer ................................ 15
3.4 FeatureInPolygon .................... 16
3.5 CalArea ............................... 17
3.6 SumField ............................. 17
3.7 MapDisplay ......................... 18
3.8 ArcObjects ........................... 20
3.9 Final remarks ....................... 20

4 Metadata .................................. 21
4.1 XML, Extensible Markup Language ........ 21
4.2 The XML Schema ..................... 22
4.3 Metadata for GIS functions ............. 23
  4.3.1 General node ...................... 24
  4.3.2 Detailed node .................... 25
  4.3.3 Access node ....................... 28
  4.3.4 Procedure for generation of GIS Functions metadata files . 29
4.4 Metadata for GIS datasets .............. 29
  4.4.1 Use of ESRI metadata profile ........ 30
4.5 The XML Document Object Model ......... 32
4.6 Final remarks ........................... 32

5 Prototype Integrating Interface for GIS components .......... 35
5.1 Discovery of GIS components ............ 36
5.2 Integration of function and dataset components through forms .. 39
  5.2.1 Argument checking .................. 41
  5.2.2 Generation of intermediate files’ metadata .............. 42
5.3 Creation and execution of scripts files ........ 44
5.4 Process chain diagrams .................. 45
5.5 Transparency for new components .......... 46
  5.5.1 The Script control .................. 47
5.6 Final Remarks ........................... 47

6 Testing the prototype: solving a geographic question ........ 49
6.1 Preparation phase ....................... 49
6.2 Procedure in the interface ............... 50
6.3 Execution of the script .................. 52
6.4 Final remarks ........................... 53

7 Discussion ................................ 55
List of Figures

2.1 Process chain of GIS functions and datasets ............................ 6
3.1 Buffer graphic interface .............................................. 16
3.2 calArea graphic interface ........................................... 18
3.3 MapDisplay graphic interface ....................................... 19
4.1 GIS functions metadata schema—Main nodes— ...................... 24
4.2 GIS functions metadata schema—‘General’ node— ................. 24
4.3 GIS functions metadata schema—‘Detailed’ node— ............... 26
4.4 GIS functions metadata schema—‘Data_Conditions’ node— .... 27
4.5 GIS functions metadata schema—‘Access’ node— ................. 28
5.1 Architecture of the Integrating interface for GIS components ... 36
5.2 Discovery of GIS Functions Window ................................. 38
5.3 Discovery of GIS Datasets Window ..................................... 39
5.4 Forms for input of GIS function arguments .......................... 40
5.5 Wrong argument type warning message .............................. 42
5.6 Display of metadata for intermediate datasets .................... 44
5.7 Warning for intermediates dataset’s metadata files ............... 44
5.8 Script window .......................................................... 45
5.9 Visual model window .................................................. 46
5.10 Integrating interface for GIS components .......................... 48
6.1 Preparation of GIS function components ............................ 50
6.2 Projection input form .................................................. 51
6.3 First part of the process chain ....................................... 51
6.4 FeatureInPolygon input form ......................................... 52
6.5 Second part of the process chain .................................... 52
6.6 Step by step execution of the bat file ............................... 53
6.7 SumField component’s results window .............................. 54
6.8 Display of relevant intermediate datasets with MapDisplay ... 54
List of Figures

A.1 Select XML file ................................................. 66
A.2 Projection XML file ........................................... 67
A.3 Buffer XML file ................................................ 68
A.4 FeatureInPolygon XML file ................................. 69
A.5 CalArea XML file .............................................. 70
A.6 SumField XML file ............................................ 71
A.7 MapDisplay XML file .......................................... 72

B.1 Select graphic interface ....................................... 74
B.2 Select activity diagram ......................................... 75
B.3 Projection graphic interface .................................. 76
B.4 Projection activity diagram .................................... 77
B.5 Buffer graphic interface ....................................... 78
B.6 Buffer activity diagram ....................................... 79
B.7 FeatureInPolygon graphic interface ....................... 80
B.8 FeatureInPolygon activity diagram ....................... 81
B.9 calArea graphic interface ..................................... 82
B.10 CalArea activity diagram ..................................... 83
B.11 sumField graphic interface .................................. 84
B.12 sumField activity diagram ................................... 85
B.13 MapDisplay graphic interface ............................. 86

C.1 Activity diagram for the Discovery of GIS function components .. 91
C.2 Activity diagram for the creation of a line of script ........... 92
C.3 Activity diagram for the Discovery of GIS dataset components .. 93
C.4 Activity diagram for the Creation and execution of Script files .. 93
Chapter 1

Introduction

1.1 Motivation

During the previous years there has been a tendency in software development towards creating small, portable and reusable pieces of software called components, instead of the traditional big monolithic programs that allow little interaction with other related applications. These components can be combined with other ones in order to create larger applications that tackle the user’s needs.

1.1.1 Components in Geographic Information Systems

Some Geographic Information Systems (GIS) vendors have implemented the concept of software components in their GIS products (based on technologies like COM) as the base programming blocks of GIS, having as a main purpose to give GIS programmers a set of basic ‘GIS objects’ that they can use to create custom applications. These components in themselves do not provide GIS analysis capabilities, they have to be assembled by the programmer using any programming language in order to create customised GIS applications.

Unfortunately, these ‘GIS objects’ are of little use to the common GIS users whose interest is to apply certain common GIS processing functions to give solution to their geographic questions. For this reason, in the present research the term GIS component is applied to indicate small programs that implement common GIS functions like buffering, overlay, optimal path finding etc. and form the building blocks of GIS analysis. These ‘higher level’ components are really of relevance to the GIS user (not only to the GIS programmer).

1Some examples of GIS objects implemented as components are geometry, layer, feature class and record set objects
1.1. Motivation

1.1.2 Online GIS

After its introduction, the Internet has been widely adopted and has experienced an enormous growth, because of this, it has become a very important information and services dissemination medium for many companies. Nowadays GIS functionality has been implemented and it's being offered on many sites over the Internet, so some authors have coined the term ‘Online GIS’ [10] to call this new way of working with GIS.

Most of the existing Online GIS systems are either spatial query systems (publishing of datasets with a viewing interface and some predefined GIS functionality), or map-building programs [10]. These sites offer mainly viewing and exploring capabilities and therefore are intended for the general public and do not offer actual GIS analysis or processing tools, and when they do, they can only be applied on predefined datasets, not on the user's datasets. Therefore, it is still missing on Online GIS to make general GIS analysis tools (the GIS components as explained in 1.1.1) available, so more specialised GIS users that need to perform analytical or specialised tasks, can make use of these components and integrate them in chains to analyse their data. This is the issue on which the present research concentrates.

To have GIS components residing at different locations across the network would allow thousands of potential new ‘online users’ to access them, but also makes it necessary to have brokers that control and coordinate the flow components to the clients [12]. Under a GIS components implementation the users would only pay for the few functions they need to use instead of buying complete GIS software licenses. They would be able to pick the GIS functions that better fit their requirements (robustness\(^2\), price, data formats, etc.) from different providers as they require them [16, 22]. The chosen components would be sent to their web browsers, without needing to locally install software. A well developed implementation of this kind will allow the seamless integration of data and GIS functions, and the access to GIS of many new non-expert users\(^3\) which will only need to be connected to the Internet. Additionally, The approach of ‘Online GIS’ appears to be very appropriate for the actual topology of GIS projects which have moved from being stand-alone projects to multi-agency, multi-disciplinary and multi-software projects [10].

\(^2\)Robustness and price would have a direct relationship, it’s very likely that cheap or free components would be reliable only for small and simple datasets, but expensive components offered by well known producers would have little limitations as to datasets complexity

\(^3\)Some of the non-GIS-expert users that would highly benefit include: insurance, real estate, marketing, product sale and distribution, transportation and utilities companies
1.1.3 GIS datasets

GIS datasets are by nature distributed due to the fact that different data producers reside at different geographic locations. The Internet enables these data producers to offer their information online, in this way the users can access the most current data available. Unfortunately, this creates the burden of having to go through many different Internet sites to collect information on different themes for a single project. This has given origin to data warehouses which are information networks that link, coordinate and manage the geographic information offered online by different sites for a particular region. Data warehouses provide a common interface through which the user can query the resources available (e.g. the USGS-NASA Earth Observation System Data Gateway4).

The availability of data warehouses that easily connect the users to their required datasets helps to avoid the problem of information duplication that is very costly for our society (resources are spent in creating datasets that already exist). The issue of reuse of data is therefore very important.

Because both GIS functions and datasets are tightly integrated in the solution of geographic-related problems, GIS datasets also become important ‘building blocks’ in GIS analysis. For this reason in the present research the concept of GIS components is extended to encompass not only GIS processing functions but also GIS datasets.

1.2 Problem definition

In order to move from the traditional stand-alone GIS to Online GIS, special problems arise due to the spread of software, datasets and processing over the Internet [10], that’s why in the reviewed literature (see 2.2), it can be observed that a lot has been studied and thought as to technology (infrastructure, architecture, communication protocols, processing) to enable the operation of GIS processing over the Internet. Nevertheless, not so much has been studied about the user interface which is a key element for the implementation of GIS components, even some authors [11] have identified the need to concentrate on this issue. This element is the focus in the present research. The integrating interface is necessary for the discovery of GIS components and their integration in a meaningful way in order to solve a user’s specific geographic questions. The term ‘discovery’ goes beyond plain ‘search and find’, including more intelligence in the detection of GIS components of interest for the user.

A very important issue linked to the user interface is the user’s expertise,

4see: http://edcimsw.cr.usgs.gov/pub/imswelcome/plain.html
1.3 Thesis outline

being necessary very different interfaces according to different expertise levels (GIS programmer, GIS user and GIS viewer are three generic expertise levels). In the present research, the emphasis will be put on the ‘GIS user’ who applies GIS analysis functions to increase his understanding about a particular phenomenon. Therefore, this user has some basic GIS knowledge.

1.2.1 Objective

The main objective in this research is then to design a system that implements discovery of GIS dataset and function components, and that facilitates basic service chaining (see 2.1.2), and to test it for the solution of general geographic problems. This objective further includes the development of some generic GIS function components and metadata. The system makes use of some existing GIS datasets.

The user interface of this system will be developed using Visual Basic because it’s intended to be used as a prototype for a future implementation that works online (the online implementation either as a browser helper application or as an applet should come in as a later research stage). In this first stage the emphasis is put on detecting which is the functionality that is considered essential and to implement it on the system. That is why Visual Basic has been chosen, due to the fact that it allows the quick production of applications.

1.3 Thesis outline

In Chapter 2 some additional basic concepts that support this research will be introduced, together with some previous experiences and research found in the literature about GIS components on the Internet. In Chapter 3 the GIS function components, which were developed in order to test the interface, are explained. They were developed using Visual Basic and arcObjects and implement seven common-use GIS functions. In Chapter 4, the set of metadata which was chosen and implemented to identify GIS function and Dataset components is presented. This metadata was implemented using XML, the extensible markup language. The last two elements (components and metadata) conform the foundation on which the integrating interface is built. The integrating interface functionality is dealt with in Chapter 5. Finally, in Chapter 6 the integrating interface is demonstrated for the solution of a generic geographic question.
Chapter 2

Background

2.1 Underlying concepts

Some important concepts that support the present research are subsequently explained.

2.1.1 GIS functions, operations or services?

In the reviewed literature dealing with online GIS, the terms GIS functions, GIS operations and GIS services are sometimes used indistinctively to refer to GIS processing over the network, so a definition of the terminology that will be used in this thesis will be given.

GIS functions is a term that refers to the analytic capabilities of GIS, which allow the GIS users to ‘extract information from data’ [6]. Operation is a rather more generic term that is used for transformations having a name and some parameters [5]. Operations encompass GIS functions, but can also include other more generic functions like format conversion, which do not necessarily involve GIS analysis.

Service is a more generic term that sits on top of the operations, a GIS service on the Internet can make use of GIS functions and datasets to attain different goals, so we can speak about viewing services, querying services or discovery services.

Because in the present research the processing components were defined as “small programs that implement common GIS processing functions” (the ‘building blocks’ of GIS analysis—see 1.1.1—), the term GIS function components will be used to refer to them. In addition, because GIS processing and datasets are two elements so tightly integrated in GIS analysis (see 1.1.3), for the last ones the term GIS dataset components will be used. The processing programs together with the datasets comprise the more generic term GIS components.
2.1. Underlying concepts

2.1.2 Process chains

With a given set of GIS functions available (with certain granularity), any geographic question that the user has must be broken up into a work-flow or process-chain of both GIS functions and datasets connected in a sequence such that at the end a solution is achieved. In such a sequence the input of one function corresponds to the output of another one and so on, forming a chain (see Figure 2.1). According to [5], a service chain is “a sequence of services where for each adjacent pair of services, occurrence of the first action is necessary for the occurrence of the second one”. The concept of process chains naturally extends to Online GIS systems where both GIS functions and datasets spread over the network, supporting the OpenGIS project’s goal that states: “the user’s work flow extends across the net, accessing geoprocessing services in a distributed, cooperative way to accomplish some application-specific work objectives” [4].

![Figure 2.1: Process chain of GIS functions and datasets](image)

In a first approach to process chains it is the user (based on his knowledge and experience in the use of GIS functions), who has to find out the right sequence of components that solves his problem. In this setting, the user can be given extra information about specific components (supporting his right choice), but at least basic knowledge of GIS is required. This approach is called in [5] the ‘user defined’ or ‘transparent’ chaining.

Another approach is to store predefined ‘common-use’ process chains of interest for many potential users so they can be accessed and reused. The users will be able to search for and access process assemblages that meet their requirements and that have proven to give valid results. These predefined sequences of functions can then be customised by the users by inserting their particular datasets and parameters at the right positions in the chain. This second approach is more appropriate for less expert users.
The processing chains approach is enforced if the different GIS function components produced by different vendors follow a common definition of functionality, inputs and outputs. In this way, the components could be easily interchangeable, allowing the user to pick different functions from different vendors as convenient. An important issue here is the *interoperability* between components, taking into account issues like file formats and data structures. In the present research the term interoperability means that the GIS function components are able to interchange data, and therefore can be connected in process chains. When an open standard for geographic data is agreed on by software and vendors (the vision of the Open GIS consortium [4]), the issue of interoperability would be solved. In practice, before this ideal is attained, interoperability between components must be checked during the creation of service chains.

The operation of GIS software components in service chains proves especially useful for chains of GIS functions that require little intermediate user interaction and that need to be run several times using different parameters. It is less suitable for extensive user-interaction functions such as topology cleaning.

### 2.2 Previous components experiences

In the reviewed literature, some prototype GIS components experiences have been reported together with some recommendations for the creation and development of Open GIS systems. They serve as an starting point for the present research. These experiences are summarised together with their results in the following sections.

#### 2.2.1 Open GIS service architecture

In accordance to its vision of “interoperable geoprocessing software throughout the global information infrastructure” [4], the Open GIS consortium proposes on the *abstract specification on Services Architecture* [5] to divide the functionality of GIS software in *geographic services* with some purposes like “to enable interoperable data services through standardisation” and “to support the development of service catalogs through metadata”. In this way GIS users can be enabled to operate with distributed resources over an open environment.

This abstract specification explains how these geographic services must be able to be combined in *service chains* to achieve larger tasks (see 2.1.2), how they must be interoperable (so they can freely exchange spatial information) and even specifies a taxonomy for services with the following main categories: human interaction services, information management, workflow management,
2.2. Previous components experiences

geographic processing (spatial, thematic and temporal), communication services and system management.

This specification also deals with other important issues of distributed GIS such as the computational architectures necessary for the distributed GIS system, emphasising the use of multi-tier client-server architectures. The issue of cross-platform interoperability is also addressed, here they proposed to use communication services (like the Object Request Broker—ORB—) that “allow the services to interoperate across different networks, hardware platforms, operating systems and programming languages”.

When dealing with the development of open GIS systems this specification is an important reference to be consulted.

2.2.2 Agent based architecture for imagery and geospatial processing

AIGA [13] is a Java-based architecture for distributed GIS and image processing. This architecture is designed to answer generic geographic queries posed by any non-expert user (e.g., “how has the border between countries A and B changed?”). It’s based on processing agents that address low level algorithms used in image processing and GIS, and information retrieval agents that allow to discover other agents on the network. These agents are described in RDF (Resource Definition Framework) encoded in XML files. Their implementation includes a so-called ‘communication page space’, a repository where agents exchange messages and store and discover knowledge about previous analyses performed. For the discovery of agents they use an information retrieval algorithm that ranks found agents according to their relevance for the user’s geographic query.

The user interface of their Java-based implementation consists of a window displaying a map of the world with a text box where the user can input a geographic question. The relevant agents (or sets of agents) discovered are displayed to the user together with their relevance (a percentage). Once an agent is chosen, a window is displayed showing the agent’s approach to solve the problem, including required datasets, computational steps and processing strategy. The user can make adjustments and submit for processing.

2.2.3 GeoServNet

GeoServNet [16, 22] deploys geo-processing software components over the Internet and allows clients to assemble them using web-browsers. This implementation is based on Java technology. The server side contains a web server,
a component description server and a data catalog.

The components’ metadata are registered in a meta-info database. This enables the users to search this database and connect to the actual components. The meta information contains some user identification information together with the software component interface. The component registration is done through a form in HTML that populates the meta-information database in the server.

They developed some useful Geo-processing services that can be accessed through standard web-browsers. Some of them are: ‘Geospatial Display Services’ (viewing and querying of data), ‘Geodata Access services’, ‘Data transformation services’ (coordinate transformation) and ‘Map annotation and symbolisation services’. These geoprocessing tools are offered to the client via Java Remote Method Invocation (RMI) or the Java Applet Extensions.

The main user interface of this implementation is provided by a Java applet called ‘GeoEye’ which is sent to the user’s browser and has the following visual elements: map display, layer control, status bar, map overview window, attribute display window and operation buttons. There is another interface that allows the user to select other components to extend the ‘GeoEye’ applet. Here, the available components are displayed under different categories (data access, data format conversion, data processing and data analysis). The users can see the description of the different available components and choose the ones they need. Once chosen, the newly selected components are added to the ‘GeoEye’ applet as additional operation buttons. In this way, the users can execute them from the main interface.

2.2.4 A CORBA-based interoperable GIS

The authors of [12] propose to use distributed computing infrastructures, in particular the ‘Object management architecture’ (OMA) and the ‘Common object request broker architecture’ (CORBA), to integrate distributed Geographic information systems composed of heterogeneous components into one overall system or open GIS. According to the authors, CORBA proves excellent as enabling communication infrastructure.

As to the graphical user interface they propose one that includes: display windows (to show maps), a query manager (to input commands to the system), an object manager (fetches datasets from repositories and stores them locally) and a services manager (to find and call services from the different participating systems). Metadata, which enables search and data integration, is stored in a special catalog.
2.2.5 Interoperable Geo Information Systems (IOGIS)

This is a German research project dealing with uniform access to heterogeneous and distributed datasets. One of its subprojects at the University of Bonn [2] involved access to distributed paleoecologic data stored in different formats and systems.

The user interface of their integrating application is called the ‘Geo-client’, a client application that supports the assembly of plug-in components according to the user’s demands, allowing the access to diverse datasets and geoscientific methods. Different components (provided by a ‘Geo-Server’) are visualised as buttons in the Geo-client. The user can query generic drilling information using a visual query component and afterwards will be provided with additional components (as additional buttons) when he wants to visualise details of particular types of drilling data.

Because in this project the issues of platform independence and access to diverse software systems are of great importance, they used the Java platform to implement the components, CORBA as middleware and the ‘Object request broker’ is used as communication infrastructure.

2.2.6 Universal Description, Discovery and Integration (UDDI)

Universal Description, Discovery and Integration (UDDI) is a specification for distributed Web-based information registries of Web services [18]. This specification allows programmers to discover and use services that different companies publish on the Web.

The approach of UDDI is a distributed registry of services and description implemented in a common XML format (so consistency is ensured). The UDDI business registry is a publicly accessible, multiple-nodes registration service. An ‘UDDI business registration’ is an XML file describing a business and its related web services. This information has three main components: ‘white pages’ (general contact information), ‘yellow pages’ (industrial categorisation) and ‘green pages’ (technical information about the services offered by the business). The great advantage of UDDI is that it provides search engines with standardised formats for service discovery.

2.2.7 Method ManageMent

Method ManageMent, MMM [11] is a distributed computing infrastructure that supports the ‘service-oriented’ model in which the users make their data available to GIS services centres which perform the necessary operations and send
the results back to them. The prototype is based on Java applet technology, a CORBA implementation is reported to be under consideration.

The system gives the user the possibility to search for GIS services, execute them and register new ones. The registry contains ‘meta objects’ to wrap and describe data and computational services. These ‘meta objects’ can be stored either as XML documents or as relational database tables.

### 2.2.8 Geography Network

The ‘Geography Network’ of ESRI [9] provides an infrastructure for the sharing of geographic information, it provides access to a large collection of data published by many data publishers. As to applications, it offers ‘geoservices’, i.e. common geo-processing functions (like geocoding and buffering) that can be used by developers to build their custom distributed applications where most of the processing in done in remote systems.

Through the geography network URL it is possible to check which ‘geoservices’ are currently on line, currently you can see some geocoding services provided by ESRI.

### 2.3 Final remarks

These existing experiences provide a ground for the implementation of GIS components, a lot has been thought as to communication infrastructure and platform-independent development, even some attempts of actually publishing components are beginning to be implemented. Some of them have incorporated different types of user interfaces for the operation of these components. Still, a user interface that gives more insight about existing components and that enables their integration in process chains can further complement these experiences.
2.3. Final remarks
Chapter 3

GIS function components

The GIS function components are the first important resource in this research. A GIS function component consists of an ‘exe’ file that takes some input (generally including datasets and parameters), performs some GIS analysis and finally produces some output either in the form of a new dataset or a report. The functions developed in the present research address some common use GIS vectorial functions.

It was necessary to develop the GIS function components in-house due to the fact that this type of programs are still not publicly available and additionally, to have more control on the desired GIS functionality at this research stage. They were built using Arc Objects (see Section 3.8) and VisualBasic. Although they’re not open components due to the technology used (VisualBasic works in MS-Windows platforms, Arc Objects works with ESRI GIS formats), they serve the purpose of the design of the integrating interface. The function components were developed taking into account that it had to be possible to assemble them forming larger processes or process chains, being able to pass data between each other (see Section 2.1.2).

Due to time constraints only seven function components were developed and their functionality was planned to address common use GIS vector analysis functions. The granularity$^1$ of the components was chosen such that each component addresses a single GIS vector operation. The data format used for these components is the shapefile$^2$ format. This format is very popular between GIS users and is additionally straightforward to use due to the fact that both geo-

---

1Granularity has to do with the ‘size’ of the software components which can vary from the very basic GIS functions like buffer or overlay, to more complicated aggregation of basic GIS functions specific to certain fields of knowledge like landslide risk, environmental sensitivity or best site location functions.

2This is a data format developed by ESRI to store geometric and attribute information for vectorial GIS datasets.
metry and associated attributes are stored in files.

All the function components (except for MapDisplay) were developed with two user interfaces. First a ‘visual’ one that consists in a form with controls\(^3\) with which the user can interact, give input and obtain output in a more or less interactive way. This user interface also provides a `pictureBox` control where the user can preview input and output shapefiles. The other user interface was developed to accept arguments in a ‘command-line’ way. So if the program is launched using the command prompt, the necessary arguments can be typed after the name of the program. Using this user interface it is easy to form chains of procedures using scripts in *.bat files. In this way a function component can take as input the output given by a previous one and so on, giving solution to the user's problem in one run (see Figure 2.1).

Next, the functionality of the GIS functions that were programmed as separate components is explained. More detailed programming issues, including explanation of the ArcObjects used can be found in Appendix B.

### 3.1 Select

This component takes a shapefile as input and makes a selection of features based on an SQL (Structured Query Language) condition given by the user. It outputs a new shapefile with the selected features with all the original attribute values. This component falls into the category of GIS Query functions\(^4\). The arguments for this function are:

- **Input File**: Complete path to the dataset from which features will be selected
- **Where Condition**: 'WhereClause' part of the SQL condition to perform the selection. For example: `NAME='Colombia'`
- **Output File**: Name of the output file that will contain the selected features.

If the arguments are not given in the command line, a form will be displayed where the user can visualise the input file, the selected features and output file created. Specific programming information and the graphic interface can be found in section B.1 on page 74.

---

\(^3\)ActiveX Controls that form the building blocks of the Windows user interface [14], like text boxes, command buttons and combo boxes

\(^4\)The categories presented in this work are a personal choice from those presented in GIS textbooks[6] and those one normally encounters as menu headers in GIS and Remote Sensing commercial software
3.2 Projection

This component takes a shapefile in geographic coordinates and a projected coordinate system definition as input, and outputs a new shapefile in projected coordinates in the specified system. This component assumes that both the geographic and projected coordinate systems are based on the same ellipsoid, i.e. no datum shift is performed. If the geographic and projected system are based on different datums, then the output of this component will only be accurate for small-scale based datasets, for large scale datasets the ‘datum shift’ must be performed. This component falls into the category of GIS Coordinate transformation functions. The arguments for this function are:

- **Input File**: Complete path to the input dataset in geographic coordinates. It is advisable that the shapefile has a .prj file associated (where its geographic system is defined).

- **Coordinate system code**: Number that indicates the ESRI code of the projected coordinate system. (esriSRProjCSType [7]).

- **OutputFile**: Name of the output file in projected coordinates.

When all the arguments are not supplied in the command line, a form will be displayed where the user can input the arguments and visualise the files. This form and specific ArcObjects’ programming details for this component is found in section B.2 on page 76.

3.3 Buffer

This component takes a shapefile and outputs a new dataset that consists of the union of the buffers around each individual feature at the given distance. The output polygon feature will have as unique attribute the buffer distance. This component falls into the category of GIS Vector Analysis functions.

The arguments for this function are:

- **Input File**: Complete path to the dataset around which the buffer will be calculated.

- **Distance**: Buffer distance in the map’s coordinate units (according to the coordinate system definition).

- **Output File**: Name of the output file that will contain the buffer.
3.4 FeatureInPolygon

This component takes as input a polygon shapefile (the ‘containing’ one) and a second shapefile (‘contained’ one: polygon, line or point) and outputs a new shapefile with those features of the ‘contained’ file that are contained by the union of the ‘containing’ polygons. The output will contain the contained features’ attribute values. This component falls into the category of GIS Vector Analysis functions. The arguments for this function are:

- Input Containing: Complete path to the input polygon dataset. ‘Containing’ dataset

One condition associated to this component is that the input file must be in projected coordinates (‘degrees’ is not a distance unit).

If the arguments are not given in the command line, a form will be displayed where the user can visualise the input file and the created buffer (Figure 3.1). Specific ArcObjects’ programming details is found in section B.3 on page 77.

Figure 3.1: Buffer graphic interface
Chapter 3. GIS function components

- **Input Contained**: Complete path to the input dataset from which features will be selected. ‘Contained’ dataset

- **Output File**: Name of the output file that will contain the selected features.

  Two conditions associated to this component is that both the ‘input containing’ and ‘input contained’ dataset must be in the same coordinate system and have the same units. Otherwise no real ‘containment’ can be assessed.

  If the arguments are not given in the command line, a form will be displayed where the user can visualise the input files and select the contained features. Specific programming information and the graphic interface can be found in section B.4 on page 79.

### 3.5 CalArea (area calculation)

This component takes a polygon shapefile as input and outputs the same one with a new calculated area field. The area is calculated in the shapefile’s coordinate system definition, therefore, if it is in geographic coordinates, it must be projected first before calculating the area. This component falls into the category of GIS Measurement functions.

The arguments for this function are:

- **Input File**: Complete path to the input polygon dataset of which the area will be calculated.

- **Area Field**: Name of the new field that will hold the area values. If it already exists, its record values will be updated.

  Two conditions associated to this component are that the input file must be a polygon geometry and that it must be in a projected coordinate system, otherwise no area can be calculated.

  If the arguments are not given in the command line, a form will be displayed where the user can visualise the input polygon file and assign the area field name (Figure 3.2). Specific programming information can be found in section B.5 on page 81.

### 3.6 SumField (fields statistics)

This component takes a shapefile as input together with one of its numeric fields and outputs a summary of the statistics of this field (including count,
3.7 MapDisplay

sum, standard deviation, mean, maximum and minimum). These statistics are reported in a message box. This component falls into the category of GIS Measurement functions. The arguments for this function are:

- Input File: Complete path to the input dataset containing the field whose statistics will be calculated.

- Field Name: Name of the filed whose statistics will be calculated.

If the arguments are not given in the command line, a form will be displayed where the user can visualise the input file and the field to summarise. This form and specific programming information is found in section B.6 on page 83.

3.7 MapDisplay (display of shapefiles)

This component was built to address the data visualisation function, and therefore complements all the previous components which are not intended for visualisation but only for data processing. The data visualisation component
normally forms the main user interface of common GIS software. In the present approach, the user can use this component to visualise results obtained with any of the previous components.

Figure 3.3: MapDisplay graphic interface

MapDisplay can display one or several shapefiles and contains general display tools like: opening and closing of layers, zooming, panning, changing of layer’s symbology, examination of attributes, examination of layer’s coordinate system and changing of the display’s coordinate system (Figure 3.3). It also displays the map coordinates of the mouse’s current position in a label control. It was build with arcObject’s ‘Map Control’⁵.

This function has a unique argument which is the InputFile. This is a repeatable argument which means that the user can give several files as input in the command line to display together.

Specific programming information is found in section B.7 on page 85.

⁵The use of this object requires to have an ESRI ArcGis licence
3.8 ArcObjects

ArcObjects is the development platform for ESRI’s ArcGIS software. It consists of a large library of COM objects, and therefore it can be used by any COM-compliant programming language like VisualBasic. ArcObjects allows to customise the ArcGIS applications, but more importantly, it enables the development of independent programs like the GIS functions presented in this chapter.

The ArcObjects are grouped in subsystems that encompass the different aspects of geographic information management and analysis. Some of them are: Geodatabase, Raster, Geometry, Spatial Reference and Network. These subsystems include classes and interfaces documented in object model diagrams.

The use of arcObjects for the development of GIS function components enabled their fast production due to the fact that many GIS functionalities are offered as methods in their objects, so the programmer can directly use these functionalities and give them shape instead of having to write the basic algorithms. Additionally, GIS data management (access to datasets, creation of new datasets) is greatly facilitated, it is performed through the ArcObjects so the programmer doesn’t have to worry about the ‘shape’ of the physical files.

3.9 Final remarks

Being independent *.exe programs that accept arguments, these components can be integrated in chains of commands using bat files put together according to the user’s GIS processing requirements. This approach can be very cumbersome because the user must be aware of all the arguments, proper order, and possible restrictions that apply to each argument. In many cases, the user has to follow trial-and-error process before being able to get valid results using this approach. Additionally, complete paths of both GIS function components and Dataset components must be provided, increasing the source of possible errors. The integrating interface approach taken in this research (see Chapter 5) eliminates this cumbersome procedure and enables much smoother creation of bat files by guiding the user in the procedure of argument input and validating this input.

The following chapter will explain the metadata approach that was chosen to be implemented in order to allow the discovery of GIS function and dataset components and their integration.

---

6COM stands for ‘Component Object Model’, it is Microsoft’s standard for component-based system development
Chapter 4

Metadata

In order to enable the integration of GIS components a set of metadata attributes must be recommended which identifies available GIS function and dataset components. Metadata is the additional background information that describes the relevant characteristics of the components like descriptive information, quality, and conditions of use [6]. This background information enables other users (different than those who produced them) to use the components. In addition, metadata is paramount to coordinate data coming from multiple origins [10]. Two types of metadata attributes can be identified: human-readable metadata, intended for display to the user, and metadata for automation (machine-understandable), not necessarily human-readable and used by component discovery brokers and automation programs.

4.1 XML, Extensible Markup Language

XML, the Extensible Markup Language offers a platform-neutral view of data and allows hierarchical relationships between attributes to be represented in a natural way. XML provides a simple method for data structuring and organisation (based on tags written in text files) reducing the problems of data incompatibility.

The World Wide Web consortium has an ‘XML Activity’ which works for the development of XML products specifications, guidelines, software and tools. According to the XML Activity Statement [20], the number of XML applications is growing everyday giving it a more important role in the exchange on data on the Web. XML is supporting the definition of data-exchange protocols, especially on electronic commerce and it’s also allowing the introduction of metadata, so producers and consumers on the web can easily find each other.

In addition to the previous advantages of XML, some services identifica-
tion specifications have adopted XML as their metadata language (see Section 2.2.6). XML offered a structured way to store metadata in comparison to plain text files from which metadata values are hard to fetch. RDF (the resource description framework) was also studied, but it was considered very specific for web resources and not so straightforward as plain XML to model the desired aspects of GIS components. These facts supported the choice of XML as metadata format to identify GIS components in the present research. Additionally, for Dataset components there exists a thorough metadata implementation in XML, part of which was utilised in this work (this will be discussed in Section 4.4).

4.2 The XML Schema

The XML schema [19] is a new approach that replaces DTD (Document Type Definition) for the definition of XML files’ rules and allows to easily represent different characteristics of data like hierarchy, repeatability, obligatory or optional character, choice, attributes, data types and valid values. Most importantly, it permits the validation of newly created XML files against a definition to ensure consistency.

An XML Schema consists of components such as type definitions and element declarations. These are used to assess the validity of data stored in XML files that use a schema definition.

The use of XML editors like Altova’s XML Spy\(^1\), allows the easy creation of XML schema files showing the user the different elements through a ‘tree-like’ visual interface, so the user can immediately and globally visualise different hierarchy levels, repeatability and optional character of elements, choice nodes, data types, enumerations, and the other elements that compose a schema file. In this way the user can make sure that the schema being designed fits the nature of the data being modelled. Additionally, XML editors offer a very natural, easy and straightforward way to visualise and edit the metadata stored on *.xml files. Most importantly, they force the user to check the validity\(^2\) of newly created files against the XML schema, therefore ensuring that the information stored in XML files is complete and valid.

\(^{1}\)(www.xmlspy.com)
\(^{2}\)An XML file is valid if it follows the definition and rules stated on the XML schema file in which it is based
4.3 Metadata for GIS functions

Metadata is relevant for GIS functions in order to describe their functionality and other characteristics important for the user like conditions for use, required inputs and produced outputs.

In particular, ‘machine-understandable’ metadata must allow GIS components applications (like the integrating interface presented in Chapter 5) to find out how to access the components, and which are their inputs and outputs (and in which format). Using the previous information, the GIS components can be integrated in a meaningful way.

The choice of metadata attributes to identify GIS function components was a process of trial and error: initially a metadata structure was proposed and implemented. Afterwards, during the development of the integrating interface, new metadata elements had to be added to further support the integration of GIS function components with Dataset components. During this trial and error process the recommendation of the OpenGIS Consortium about service metadata [5] was taken into account for this choice of metadata attributes.

In a broad outline, according to the OpenGIS Service Architecture specification, the ‘Service metadata’ must allow the use of catalogs or search engines to discover services (or GIS functions, using the terminology of this research). They also add that ‘the metadata record’ must provide the user with enough information to ‘invoke’ the service. In this section the term ‘user’ will refer not only to a human user but (especially) to an electronic user as a broker or search engine.

In addition to the OpenGIS recommendation, only the attributes really needed for the interface were chosen to be implemented, in order to avoid the creation of a very extensive set of metadata attributes (which in practice results in many empty values). It must also be considered that for an actual implementation in an Internet-setting it is likely that many more metadata attributes will need to be added.

The metadata schema developed for the GIS functions has three main elements identifying three different information categories. The following diagram shows them.

The schema diagrams shown in this section obey the following symbology: the element’s data types are indicated under the element’s name, possible enumerations (lists of valid values) are indicated under the data type, optional elements are drawn with a dashed line and repeatable elements have the \((1..\infty)\) or \((0..\infty)\) symbols below them (required or optional respectively). This is the symbology used by the XML editor utilised in this work: Altova’s XML Spy.
4.3. Metadata for GIS functions

4.3.1 **General node**

This metadata element contains general identification information of the component like the operation name and keywords. The following diagram shows the elements that conform this node, they are subsequently explained.

- **Operation_Name**: The name of the GIS function component. Generally coincides with the GIS operation it implements.

- **Category**: The Category of GIS functions to which the component belongs. This is very important for organisation of components (for instance, when a broker finds components, they can be displayed to the user grouped by category). The list of values defined for this element are: Vector Analysis, Coordinate transformation, Data preparation, Query, Measurement, and Display. This list can be further enlarged (for instance adding raster operation and remote sensing categories), but for the components available in this research it suffices as it is.
• Platform: The operating system under which the component can be executed. In this research it always takes the value of ‘Windows’.

• Operation_Description: Description of the component’s functionality. It must clearly inform the human user in few words what are the results that can be obtained with it.

• Creation_Date: This element gives the user an idea of how up-to-date the component is.

• Version: It states the state of revision of the component. It also allows the user to choose those components that are considered to be more ‘stable’.

• Keywords: Repeatable element, Keywords are very important for the processes of search and discovery, in this way the user can find the components without knowing its exact name, but using synonyms or related terms.

4.3.2 **Detailed node**

This element contains detailed operation information about the component. It also contains the information that is used by the integrating interface to correctly assemble GIS functions and datasets. The conforming elements are shown in the following diagram and later explained.

• **Data_format**: File Format of the datasets for which the component was developed to operate on. In the present research, all the GIS function components operate only on ‘Shape file’ format.

• **Arguments**: This repeatable element contains the list of arguments (datasets and parameters) that the component takes for its operation. These arguments are the different elements that are typed on the command line after the component’s name. The Arguments are described using the following child elements and attributes.

  – **Name**: Identification name of the argument. This name must be unique within the component. This name is also used for the definition of data conditions (see below).

  – **Type**: Specifies whether the argument refers to a dataset or to a parameter and which is the format of this parameter. It has the following list of allowed values: ‘Dataset’, ‘Number’ and ‘String’.

  – **Order**: The consecutive order in which the argument must be input into the command line.
4.3. Metadata for GIS functions

- **Description**: A concise and complete explanation of the type of information the user must give as an argument.

- **Direction**: Indicates if the argument is given to the function component as an input or if it is produced as an output. The list of allowed values is: ‘IN’ and ‘OUT’.

- **Required (attribute)**: A boolean value that indicates whether the argument is optional or mandatory.

- **Repeatable (attribute)**: A boolean value that indicates whether the argument can have multiple values.

- **Data_conditions** (node): This node contains the different conditions or restrictions associated with dataset arguments. These data conditions allow the integrating interface to check correct matches between GIS function components and Dataset components. Additionally, for output dataset arguments, it specifies the relevant metadata attributes they will get based on the attributes of input datasets (this is a very important issue, because it allows to know in advance the metadata for output datasets, allowing their use in processing chains before they are actually written to disk).

The data conditions associated with an argument can involve only
Figure 4.4: GIS functions metadata schema—'Data_Conditions' node—

that same argument and be of the format:

.CoordSys="Geographic" Which means that the coordinate system of the present argument must be “Geographic”

In this case the data condition is formed with three additional elements: Attribute, any of the pertinent dataset component attributes (coordinate system type CoordSys, format Dformat, coordinate system name CoordSysName, geometry DGeometry, data structure DSstructure, units DUnits or path, see 4.4.1); Comp comparison operator, which is either ‘=’ or ‘!=’—not equal—; and Condition (a string).

Or on the other hand, it can involve other arguments and be of the format:

.CoordSys=Input1.CoordSys Which means that the coordinate system of the present argument must be equal to the coordinate system of the argument called ‘Input1’—which must have been previously defined—

In this case the data condition is made up of four additional elements: Attribute, Comp, Parameter (the name of another previously defined parameter\(^3\)), and Attribute2 (again any of the pertinent dataset component attributes).

The two previously mentioned options were modelled in the XML schema file using a choice structure. See in Appendix A, page 65 the actual data conditions defined for the GIS function components

\(^3\)This condition was enforced using the ID and IDREF types in the schema definition, so the user can only select as valid value any of the other already given parameter names

27
in the XML files.

- Prerequisites: This repeatable element contains a list of other GIS functions that must be operated on the Dataset before applying the present one.

### 4.3.3 Access node

This element gives information about the way how the component can be acquired or accessed. It includes therefore information about the component’s provider, the ‘digital’ address where it can be found, conditions and restrictions. In the present research, the number of elements implemented on this node was reduced to those strictly necessary for the operation of the interface (and others which were included only for illustration). In a public distributed implementation the access element must further include elements that address issues like fees, ordering instructions, license conditions, etc.

The following diagram shows the elements that conform this node, they are subsequently explained.

![Diagram showing the elements of the Access node](image-url)

- Address: This is the metadata attribute that allows the user to physically (or digitally) access the component and be able to execute it. In the present research, it corresponds to the physical address (on the hard disk or the Local Area Network—LAN—) where the component lies. In an Internet-setting it would correspond to a Uniform Resource Locator (URL) address plus instructions to obtain the component electronically.

- Conditions: The conditions of use of the component. In this element the
information about licensing would be included. For the GIS function components of the present research there are no conditions.

- Restrictions: In an actual implementation, this node should contain legal and security constraints associated with the component. For the programs of the present research there are no restrictions.

- Provider: Contains general information about the provider of the component allowing the user to come in contact with it. It includes elements like Name (the organisation providing the component), Contact Person (name of a person in charge of this component in the organisation) and email (of this contact person).

### 4.3.4 Procedure for generation of GIS Functions metadata files

Once the elements that conform the metadata identification were selected, the first step was to create an XML schema file. In this schema file, the different metadata elements were grouped in a hierarchical way, attributes were specified together with child element, enumerations, types and number of occurrences. The diagrams shown in the previous section are the graphic representation of the XML schema file developed in this research for the identification of GIS components.

Once the XML schema file is ready, new XML files can be created based on it. Using the XML editor software, the actual component XML files where populated filling in the specific details corresponding to each one of the components. Every time an XML file was edited, it was checked for validity against the XML schema (ensuring data consistency). For each GIS function component’s *.exe file, an accompanying *.xml file with the same name was created. The reader can look into the actual xml files generated for the seven function components, in Appendix A, page 65.

### 4.4 Metadata for GIS datasets

A GIS dataset is a representation, a model of the real world. The production of any particular dataset involves processes of selection, abstraction, generalisation, classification and simplification. This ‘model’ of the real world is therefore approximated and all the assumptions and limitations involved with its production must be informed to the users of the information [17].

Additionally, when the volume of data becomes bigger and bigger (which is the case now that the Internet is being used for data distribution), it becomes paramount to have proper documentation fully describing the datasets offered
by a producer. In this way potential users of information are able to find the
datasets they need, evaluate if they fit their needs and if so, access them. Met-
tadata is the means how this ‘data about the data’ is reported to the user.

Metadata adds value to a dataset, it makes it ‘usable’ or better said: ‘reus-
sable’, otherwise no user can risk to use a dataset of which nothing is known
about its origin, the purpose for which it was produced, quality, accuracy, pro-
ducer, etc. Unidentified datasets are just a pile of elements and coordinates
with unknown meaning and which cannot be reused.

4.4.1 Use of ESRI metadata profile

ESRI’s ArcCatalog software implements metadata for identification of datasets.
This metadata is called the ESRI profile of the Federal Geographic Data Com-
mittee (FGDC) standard [8]. The FGDC standard was developed in order to
allow potential users of information to determine the availability of a set of
geospatial data, determine its fitness for an intended use, determine the means
of accessing it, and successfully transfer it [3]. This standard was published on
1994 and further modified in 1998 based on user feedback. The (ISO) document
19115, Geographic Information-Metadata was built using this standard as its
foundation [8]. The ESRI profile extends the FGDC standard in order to provide
information about additional elements relevant for ESRI software users.

For the present research it was chosen to work with this existing imple-
mentation due to its facility for metadata generation (in ArcCatalog it’s only
necessary to select a dataset and press the metadata folder to generate an XML
file for the dataset, many attributes are also automatically populated), its XML
format compatible with the metadata for GIS function components, and most
importantly, because the metadata attributes required in this research for the
coupling between datasets and components, are contained within the many me-
tadata attributes that ESRI implements.

Additionally, the use of an existing set of metadata (which is based on an
international standard), also supports the issue of data sharing (which becomes
limited when every organisation develops it own standards).

The ESRI profile provides metadata in the following categories: identifica-
tion, data quality, spatial data organisation, spatial reference, entity and attribu-
tute, time period, distribution, citation, contact and metadata reference informa-
tion. This metadata profile is very extensive and will not be explained here.
Instead, the attributes chosen for the purpose of this research are subsequently
explained (the names shown in sans-serif correspond to the element names in
the ESRI metadata).
• Dataset Description (ESRI name: Abstract). This field contains a summary of the dataset content.

• Name (ESRI name: ‘File or Table Name’, ftname). The name of the dataset.

• Address (ESRI name: ‘Online linkage’, onlink). The name of an online computer resource that contains the dataset. In the present research, it corresponds to the physical address (in the hard disk or the LAN) where the dataset component lies.

• Data format (ESRI name: ‘Native Dataset Format’, natvform). The native ESRI terminology describing the format of the data. The general value of this attribute for most datasets used for this research is ‘Shapefile’. For Spatial Analyst’s raster files its value is ‘Raster Dataset’.

• Data structure (ESRI name: ‘Direct Spatial Reference’, direct). This attribute describes the data structure the dataset uses to represent the geographic information. The possible values for this attribute are either ‘Vector’ or ‘Raster Dataset’.

• Geometry representation: This attribute describes the geometry definition of the dataset (ESRI name: ‘ESRI Feature Geometry’, efeageom under ‘Point and Vector Object Information’, ptvcinf—for vector files—; and ‘Raster Type’, rasttype under ‘Raster Object Information’, rastinfo—for raster files—). In the case of vector files its value is either ‘Polygon’, ‘Point’, ‘Polilyne’, ‘Triangle’ or ‘Annotation’. For Spatial Analyst’s raster files it is ‘Grid Cell’.

• Coordinate system type: This attribute indicates if the dataset is in geographic coordinates or in projected coordinates, its value is either ‘Geographic’ or ‘Projected’ (this is defined under the ‘Horizontal Coordinate System Definition’, horizsys element).

• Coordinate system name: The name of the geographic coordinate system or the projected system—for projected datasets—. (ESRI name: ‘Projected Coordinate System Name’, projcsn—for projected datasets—; ‘Geographic Coordinate System Name’, geogcsn—for datasets in longitude and latitude—).

• Units: This attribute represents either the ‘Geographic Coordinate Units’, geogunit; or the ‘Planar Distance Units’, plandu.
4.5 The XML Document Object Model

- Attributes information (only vector). An attribute is a defined characteristic of an entity (ESRI name: attr). For the present research, only the number of attributes is fetched.

As it was mentioned before, only those attributes which were found to be enough and necessary for the purpose of the design of the integrating interface were chosen. The choice of a big set of attributes would have become cumbersome and a demanding activity in itself (for datasets, many additional metadata attributes would need to be collected and input). Therefore, for further refinement for instance in the implementation of a public integrating interface on the Internet, other important dataset attributes must be considered, e.g., coordinates of the covered geographic area, positional accuracy, attribute accuracy, validity period, status, attribute types and names, ellipsoid information. Additionally, in a public implementation, all the issues of access to the information (process of order, license fees, digital transfer information, etc.), must be taken into account.

4.5 The XML Document Object Model

In order to access programmatically from within VisualBasic programs the metadata attribute values stored in XML files, the XML Document Object Model (DOM) was used. This is a language-neutral interface that allows developers to create applications that access and update the content, style and structure of XML documents [15].

The Microsoft implementation of XML, DOM, can be added as a reference in Visual Basic project (Microsoft XML, msxml.dll). It implements some objects that have properties and methods that allow the programmer to navigate along the different document's nodes and access the data stored in any *.xml file. This Visual Basic reference strongly supported the design of the integrating interface, allowing the easy run-time reading of metadata stored in XML files both for user display and for processing.

4.6 Final remarks

As it was explained in this chapter, XML provides a 'layer' that sits on top of the GIS components providing some semantic knowledge about them which would otherwise be hidden and not described. Without this 'layer' the GIS functions are reduced to unusable pieces of unknown code that nobody except the owner knows how to use, and the GIS datasets are reduced to coordinates with unknown structure and meaning.
The metadata attributes defined in this chapter and stored in XML files are the foundation for the GIS components integrating interface which will be described in the following chapter. In that chapter it will be explained how human-readable metadata is used to display information to the user to guide his input and how machine-understandable metadata is used by the interface to enforce the integration between GIS function and dataset components.

In an Internet implementation, the metadata of GIS components from users around the world could be stored on servers that would act as central catalogues for the discovery of components by other users.
4.6. Final remarks
Chapter 5

Prototype Integrating Interface for GIS components

In Chapter 3 it was seen how GIS functionality can be split in different independent executable programs, instead of having one unique monolithic GIS program. This is a first necessary step in order to implement an Online GIS components system (see 1.1.2): to divide GIS software in ‘building blocks’ that can reside (together with GIS datasets) in different parts over a distributed network environment. In this way many different GIS software providers can make available their GIS function components to many potential users over the Internet.

A first important functionality that must be developed so users can benefit of the availability of GIS function and dataset components on the Internet is the Discovery of these components. The relevant GIS components of interest for a particular user must be discovered from possibly hundreds of available ones. This is an important issue due to the fact that the tremendous growth of the Internet has made finding specific and useful information a difficult and lengthy process.

Once relevant GIS components are discovered, now the idea is that GIS users are provided with the possibility to create valid process chains of GIS functions and datasets that result in solutions to their geographic problems (the importance of process chains is explained in Section 2.1.2). These process chains must be able to be run in this distributed environment which is the Internet, and their results must be presented to the user.

The essential characteristics mentioned in the last two paragraphs, which are necessary for the integration of GIS components were implemented in a

\[\text{footnote}{The word user, used throughout this chapter refers to the human user that interacts with the interface}\]
prototype interface in Visual Basic according to the research objective (see Section 1.2.1), these functionalities are explained in the following sections. The design of this integrating interface strongly relies on both the developed GIS function components and the XML metadata (Chapters 3 and 4). Figure 5.1 displays the way how the identified functionalities were organised and implemented in the prototype, they are explained in the following sections.

**5.1 Discovery of GIS components**

The purpose of the *discovery* functionality is to find relevant GIS function and dataset components to the user, and to display the corresponding ‘human readable’ metadata in order to guide his choice.

This functionality is performed by a VB procedure that searches for XML files\(^2\) stored in any drive of the local area network (this is a local network implementation) and for each XML file found, reads relevant nodes in the XML files to identify if it corresponds to GIS function or dataset components as de-

\(^2\)The file searching capabilities in VB are provided by using the FileSystemObject (Microsoft Scripting Runtime reference). The access to XML files is possible by using the Microsoft XML document object model, see section 4.5
fined in sections 4.3 and 4.4. This is important because not all XML files refer to GIS components; in order to be identified as valid GIS components metadata files, they have to follow the following conditions: For GIS functions, the first element in the XML file (the root element) must be called ‘component’ (see Figure 4.1). For GIS datasets metadata, which is based on the ESRI implementation (see Section 4.4.1), the condition are: first, that the root element is called ‘metadata’ and second, the XML file must contain a valid value for the onlink attribute (this corresponds to the physical address where the dataset is stored), this field is only populated when the user ‘tells’ ArcCatalog to create the XML metadata file for the particular dataset.

Once discovered, the components metadata are displayed to the user in a table implemented as a VB Grid control. The human-readable attributes that were chosen to be displayed here (from the many available in the XML files) are only those considered to really drive and determine the user’s choice between one or another component. It is considered sound at this discovery phase not to overload the user with many metadata attributes when he just needs to make a fast choice. In case the user is interested in examining more detailed metadata (to refine his choice), this could be displayed on demand for certain chosen components (this additional functionality is not available for the present implementation). The diagrams in figures C.2 and C.3 explain the actions performed by the developed discovery procedures.

The XML attributes chosen to be displayed to the user are the following:

- For GIS function components:
  - Category: The components are displayed grouped by a predefined set of categories, see Section 4.3.1. In this way some order is introduced so the user can more easily visualise his components of interest.
  - Name of the component.
  - Description: Really important, for that reason this text must be short and clear.
  - Number of arguments (datasets and parameters) the function has, see Figure 4.3
  - Location (Physical location in the network): relevant because this is a local area network implementation.
  - Data format: It refers to the dataset file format that the component can process. This attribute helps the user to select those function components that better accommodate to his particular GIS implementation. Once a real Open GIS data format is agreed on by GIS
5.1. Discovery of GIS components

software producers, the data format will eventually stop being a restriction for the user.

- Platform: This attribute refers to the OS platform in which the component was design to operate. When platform-independent languages like Java are used for the development of the functions, this attribute stops being a restriction to the user.

- For GIS dataset components: Name of the dataset, Description, Location, Format (native format), Structure (data structure), Geometry definition, Coordinate system, Coordinate system name, Map Units and Number of attributes. These attributes are explained in Section 4.4.1.

As it was mentioned at the end of Section 4.4.1, there are other important dataset's metadata necessary for a real Internet implementation which were left out in this research. When they are available, some of them will also be relevant and will need to be displayed to the user to help make a better choice (e.g. metadata about license fees and data quality attributes).

Figures 5.2 and 5.3 show the implemented windows displaying ‘human readable’ metadata for discovered GIS function and datasets components.

![Figure 5.2: Discovery of GIS Functions Window](image)

It is worth noting that visualisation of discovered Datasets is not directly provided by the integrating application, it is provided as a GIS function component (the MapDisplay component) that can be applied to any of the datasets if the user wants to geographically explore them.

Another relevant element that can be added to this part of the interface (specially relevant in a real Internet implementation), is the search by Keyword, in order to narrow down the found components according to the user interests.
This is straightforward to implement by using the 'Keywords' node from the GIS function components' XML files (Section 4.3.1). Additionally, the search can be further refined by allowing the user to specify ranges (or lists) of values of interest for the different human-readable attributes mentioned before. The results of the search could be rated, just as it works with popular Web search engines. So far this element was not implemented due to lack of time and to the fact that for the small quantity of components available in the present research it was considered not so relevant.

5.2 Integration of function and dataset components through forms

After the GIS function and dataset components are discovered and displayed to the user, the next step is to integrate them meaningfully, using the process chain concept introduced in Section 2.1.2. The execution of these process chains must result in solutions to the user’s geographic questions. From the process chain approaches presented in that section, the first one (‘user defined’) was chosen to be implemented in the interface. This means that it is the user (based on some basic knowledge in GIS operations) who has to mentally design and subsequently build a process chain that solves his question (using the discovered components). This type of user is very common, specially in the application domain, i.e. all those users from different disciplines that resort to GIS to analyse their specific problems. An example of such a user is a scientist performing risk assessment studies by analysing many different thematic maps. On the other hand, the major of a city that needs to quantify the effects of an explosion, is
typically a user that doesn’t fall within this category.

In the present implementation, the integration of components begins when the user double clicks one of the discovered GIS functions from the corresponding window (Figure 5.2). In that moment a form is displayed which allows to input the arguments the function requires, as shown in Figure 5.4. This form is built at run time based on the information stored in the function’s XML metadata file (see 5.5). When an argument is repetitive, a ‘+’ sign displayed at the right of the argument allows the input of multiple values in the text control (see the mapDisplay box in Figure 5.4).

![Figure 5.4: Forms for input of GIS function arguments](image)

Two very important characteristics of this form are: first, that it gives the user insight by clearly displaying each argument’s name and description, decreasing thus the chance of giving wrong arguments or arguments in the wrong order; and second, that for arguments that correspond to datasets, the program restricts the user to only select elements from the list of discovered GIS dataset components. Additionally, the selected datasets are checked in conformity with the inherent restrictions associated to the different GIS functions (this ‘type checking’ is carefully explained in Section 5.2.1). Another useful characteristic is that the user doesn’t have to type complete dataset names and paths, instead, when the user selects one of the found Datasets, it’s path and name are automatically input to the form.
Chapter 5. Prototype Integrating Interface for GIS components

After the ‘Add to script’ button is selected, four important actions are taken:

- Generic number and string types are checked (explained in Section 5.2.1).
- A new line of script is generated and displayed in the ‘integration’ window (see Figure 5.8), the script contains the command lines that will perform the desired actions (see Section 5.3).
- A visual representation of the function and its arguments is displayed in the ‘Visual Model’ window (see Figure 5.9).
- if there are output dataset arguments, a new row is generated in the GIS dataset window corresponding to the newly created dataset (this is explained carefully in Section 5.2.2).

See the diagram in Figure C.1 on page 91 for an overview of the developed VB procedure which corresponds to the activities explained in this Chapter.

5.2.1 Argument checking

The GIS function components work with arguments of three different types (Dataset, number and string as was defined in 4.3.2). Of interest is the ‘Dataset’ type because it normally has associated restrictions depending on the particular function. These ‘data restrictions’ must be checked because the user can provide arguments of wrong types which would result in run-time errors (for instance to give a point dataset as input to the area calculation function). Because the idea with the integrating interface is that the functions form part of larger process chains, it is very important to avoid run-time errors, due to the fact that the failure to execute any script line of a process chain results in many other subsequent lines not executed and therefore a solution can not be achieved.

The types of datasets on which a specific function operates are defined as a subset of the dataset type which complies with certain attribute conditions. For instance, the Projection function operates on a ‘geographic dataset’, which is a subset of the ‘dataset’ type with the following attribute condition:

\[
\text{Dataset.CoodSys} = \text{‘Geographic’} \]

Each time the user selects a dataset, its metadata attributes must be fetched. For this reason, a Dataset type was defined as a class in VB with the following attributes: Name, Descrip (description), Address, DFormat (file format), DStructure (data structure), DGeometry (geometry), CoordSys (type or

\[3\text{The reader can check all the specified conditions in the actual GIS function's XML files in Appendix A}\]
5.2. Integration of function and dataset components through forms

coordinate system), CoordSysName (name of the coordinate system), DUnits (planar units) and nAttr (number of attributes). The values for these attributes for a particular dataset are fetched from its corresponding XML file (see Section 4.4.1).

This type checking based on attribute conditions is performed explicitly by the integrating interface before the function is actually executed. Every GIS function argument has some data restrictions specified in the XML file (the ‘Data_conditions’ node, see Figure 4.4), these restrictions are checked at the moment the user selects a dataset argument, with the help of the script control object in Visual Basic (see Section 5.8). In case the user selects a non-valid argument, a warning message is passed as shown in Figure 5.5.

In this way the integrating interface and the XML files act as a ‘protecting layer’ that ensures the correct assignment of arguments, reducing the possible occurrences of run-time errors during the execution of script files.

For arguments that refer to a coordinate system code, because these codes correspond to a predefined list of integers (the ESRI list of projected coordinate systems: esriSRProjCSType [7]), a connection to an external Access database is performed to check that the given argument is a valid value in the list and additionally to fetch other associated information like the coordinate system name and the units.

Arguments of the generic number and string types are checked when the user select the ‘Add to script’ button. In case a wrong type of argument has been input, the user will be prompt to enter it again.

5.2.2 Generation of intermediate files’ metadata

As it was explained in Section 2.1.2, the way how complex geographic questions are solved is by creating process chains of GIS components. In these chains the output dataset of one function becomes the input of another one(s) and so forth as can be seen in Figure 5.9. As a consequence, there will be a number of intermediate datasets which are created before arriving at the solution. For this
reason, it is very important to make these intermediate datasets available to be selected as arguments of GIS functions while the user is creating the chain (before they are actually written to disk, they are only created after the bat file is run—Section 5.3—). Additionally, as it was explained in Section 5.2.1, in order to properly integrate GIS functions and datasets, the values of some of the dataset metadata attributes are necessary (data format, data structure, geometry, coordinate system type and name and units). Therefore, these attributes must be generated somehow for these intermediate datasets, so they can be further used as arguments of the GIS functions.

The metadata values of the output files depend on the metadata values of input files and on the applied GIS function. For instance in the select function, the output file's geometry definition will be the same as that of the input file, but for the buffer function the output file's geometry definition will always be 'polygon' (independent of the input file's geometry definition). The definition of this metadata for output files is specified in each function's XML file under the 'Data conditions' node (see Figure 4.4). For instance, the two foregoing conditions are written as follows (all the specified conditions in the actual GIS function’s XML files can be examined in Appendix A):

- For the select function's OutputFile argument:
  \[ \text{DGeometry} = \text{InputFile.DGeometry} \]

- For the buffer function's OutputFile argument:
  \[ \text{DGeometry} = '"'Polygon'"' \]

Based on the above conditions, the temporary metadata is created with the help of the script control in visual basic (see Section 5.8), and a new row is displayed at the end of the list of the discovered GIS datasets window, so they can now be selected as arguments of functions in the process chain. Figure 5.6 shows how this temporary metadata is displayed in the 'discovered datasets' window.

Because the metadata explained in this section is only temporary (i.e. intermediate dataset to not have associated XML files), for important intermediately created datasets which could be of relevance for future analysis the user needs to manually generate the corresponding metadata XML files. Otherwise, they won’t be able to be discovered by the interface in future sessions. A warning box tells the user to do so when he decides to exit the application, see Figure 5.7.
5.3. Creation and execution of script files

The script files are the materialisation in the form of command lines of the process chains. Each command line contains the name and path of a GIS function component followed by a number of arguments that include GIS dataset components. A complete script file contains all the commands necessary to solve a geographic question.

As the user is building the process chain, the corresponding script lines are generated and displayed in a summarised form (without displaying complete paths) in a window called ‘integration’. In this way the user is aware of the commands that will be further executed.

A very important functionality (to be further implemented) in order to support the idea of reusable process chains (see Section 2.1.2) is the saving of script files. In this way, successfully created process chains can be reused either by the user who created it or by other users that need to perform the same or another similar analysis.

When the user presses the ‘Execute Script’ button (see Figure 5.8), a bat
file is written to disk with the script commands in their complete form, i.e., with complete paths and names. Subsequently, this bat file is executed. In this way, all the commands are performed, intermediate datasets are created and written to disk and after execution, a solution to the user geographic task will be achieved and displayed.

5.4 Process chain diagrams

Although a script file contains all the commands to be executed in order to solve a geographic question, this is usually not enough for the user to correctly understand and mentally visualise the geographic analysis that is being performed. For this reason, it was considered very important to add a new window in order to graphically visualise the process chain as it’s being created, as shown in Figure 5.9.

In this window a diagram is drawn in which the GIS function components are displayed as yellow ovals, the dataset components as blue squares, other arguments as white squares and newly created datasets are shown as green squares. They are connected by lines showing the processing order. In this way, at every moment the user gets a clear view of the process chain being built, allowing to refine or correct possible errors.
5.5  **Transparency for new components**

A very important issue that was kept in mind while programming the interface is that it couldn’t have any specific or previous knowledge about any of the GIS function components to be discovered, the knowledge about them is only acquired at run-time when the function components are ‘discovered’ and the metadata files associated with them are read.

The only requirement for this discovery, is that the metadata files obey the rules stated for the GIS functions metadata (they must be valid XML files according to the defined XML schema, see sections 4.2 and 4.3). In this way any new added components (for example, a new Raster function developed by certain software producer) can be immediately integrated and used within the integrating interface as long as it is accompanied with its corresponding metadata file. The W3C organisation on its ‘Semantic Web’ document states that “tomorrow’s programs must be able to share and process data even when these programs have been designed totally independently” [21].

The same applies to GIS datasets metadata. Of course it is impossible to have previous knowledge about datasets (which vary and change every minute, specially on a world-wide-web setting). Therefore, only the metadata associated to each dataset gives knowledge to the interface in run-time, allowing it to properly integrate it with the GIS function components. Again, the only requirement is that the metadata is arranged according to the rules stated beforehand. The interface is therefore **transparent** to the separate components (GIS functions and Datasets) that can be integrated by it.

This **transparent** character of the interface proved to be a programming challenge, due to the fact that some steps that would normally be done at design time (like form design, generation of code), must be performed at run time. For instance, the forms that allow the user to input function’s arguments had to
be generated in run-time according to the GIS functions’ metadata; the code to check the data conditions associated with the different arguments against the dataset’s metadata had to be generated in run-time as well. The generation of forms was achieved in VB by using a dynamic form and arrays of controls from which new controls are loaded at run time as needed (for instance, the two different run-time forms in Figure 5.4 are generated by a unique design-time form); the run-time generation of code could be implemented with the help of the script control (see following section).

5.5.1 The Script control

The script control is an Active X component that can be added to any Visual Basic project. It allows the creation and evaluation of code expressions at run-time. It accepts VBScript expressions and other ActiveX compatible scripting languages like JScript. VBScript is an interpreted and typeless language recommended for medium sized applications [14].

This component proved essential for the run-time evaluation of the data conditions stored in XML files in order to do type checking and to generate metadata for intermediate datasets. The script control allowed to generate and run VBScript code based on these conditions.

5.6 Final Remarks

The developed prototype implements those functionalities which were considered essential for a GIS components integrating interface: discovery of components, intelligent integration of GIS functions and GIS datasets in process chains, and creation and execution of script files. In addition to the implemented ones, there are other additional functionalities to further improve the interface which were identified during the final stage of this research work and for that reason couldn’t be implemented (search by keyword, saving of script files). It is important anyway that all these additional functionalities were identified and explained, so together with the implemented ones they serve as a conceptual design for a future distributed implementation.

Figure 5.10 shows the final graphic interface of the prototype developed during the present research. The programming details corresponding to the VB development of the Integrating Interface can be examined in Appendix C.

In the following chapter it will be shown how this interface can be successfully used for the solution of a generic geographic question.
5.6. Final Remarks

Figure 5.10: Integrating interface for GIS components
Chapter 6

Testing the prototype: solving a geographic question

In this chapter a general geographic question is posed, taking into account the available GIS data and functions and the possible needs of a generic application user. The necessary steps to solve the problem using the GIS components integrating interface presented in the previous chapter will be explained together with the way how the program will achieve a solution by assisting the user in selecting the right GIS function and data components.

The chosen generic question is an estimation of human related contamination in the Amazon river. This will be performed by calculating the total population of cities in the vicinity of this river. Generic geographic problems like this are of interest to many potential users of Online GIS systems.

6.1 Preparation phase

The first preparatory step to be performed before we can begin our analysis is to make the GIS function components’ exe files available somewhere in the local area network. The same applies to the GIS dataset components. Next, as it was stated in Chapter 4, to allow the integration of GIS components they need to be first identified with metadata. As a consequence, all the input datasets that will be used in the analysis must be identified with corresponding XML files (as explained in Section 4.4). All the important metadata attributes must be collected and populated, and the paths to the datasets must be verified to ensure proper access. The same applies to the GIS function components, their corresponding XML files must be created, validated and stored, see Figure 6.1.

1The world datasets in shapefile format provided with ESRI’s arcGIS software were used as input information
6.2 Procedure in the interface

In this way the integrating interface will be able to discover and access the GIS components. It is worth noting that these preparatory steps are to be done by the system provider, not by the end-user.

6.2 Procedure in the interface

The first step to be performed with the interface by the end-user is to discover the available GIS functions and datasets from any of the connected drives of the LAN. They will be subsequently displayed in windows as those shown in figures 5.2 and 5.3.

Second, with the available set of GIS functions and the world datasets, the GIS problem is broken down into 'atomic actions' by the GIS user as follows:

- The initial datasets needed for the solution of the problems are chosen: the world rivers and world cities datasets (in shapefile format).

- The world rivers and world cities datasets, originally in geographic coordinates, must be projected into the World Mercator coordinate system (ESRI code 54004), by using the Projection GIS function component. Using the interface, the user just needs to double click the Projection function from the GIS functions window. A form is displayed and each argument displays a description. Then the user just selects and double clicks the corresponding datasets from the grid control, they are then automatically added to the form as arguments of the GIS function (see Figure 6.2). While the user is selecting the arguments, the interfaces performs argument checking is so it is ensure that the input files to project are effectively in geographic coordinates. The output files’ names (cities_proj.shp,
rivers.proj.shp) are automatically added to the datasets window as temporary datasets so they can be used in the rest of the analysis (see Figure 5.6). The visual representation of the process chain is also displayed, see Figure 6.3.

- The Amazon river is selected from the projected World rivers dataset using the Select function and the SQL condition: Name='Amazon' (rAmazon.shp).
- The river's corridor of influence is generated by calculating a buffer of 80 km around the Amazon river, using the buffer function (bAmazon.shp). This distance was arbitrarily selected and depends on the user requirements.
- The cities that fall within the corridor of influence (citiesAmazon.shp) are selected by applying the FeatureInPolygon function, using the buffer and the projected World cities as input arguments (see Figure 6.4).
- Because our final goal is to estimate the total population potentially contaminating the river, then the sumField function is used to totalise the
6.3 Execution of the script

After the process chain has been defined, the next step is to execute the script; the integrating interface will create and run the bat file, the command lines will then be executed one by one and some of the executed programs will output results in windows as can be observed in Figure 6.6.

Population field for the affecting cities.

- Finally, to graphically visualise the results, the MapDisplay component is used to display the relevant intermediate datasets: the Amazon river, its corridor of influence and the affecting cities (with some background information like country boundaries).

Figures 6.3 and 6.2 show the process chain diagram with the analysis to be performed.
After the bat file is executed, the `sumField` component’s window is displayed, showing the desired results. Here, the statistical information of the `population` field for the affecting cities is displayed. As it can be seen in Figure 6.7, the number of cities potentially contaminating the river is three, and their total population is 1 215 270 inhabitants. Other statistics are also given. The solution to our geographic question is achieved!

Additionally, the visualisation component `MapDisplay` is displayed at the end of the analysis, showing some of the relevant intermediate datasets used in the analysis, giving the user a visual insight of the steps performed in the process chain, see Figure 6.8.

### 6.4 Final remarks

In a similar way as the foregoing, many other generic geographic questions can be solved using the approach and the interface presented in this research. The only limitation is the availability of suitable GIS functions and datasets required for the specific problem in mind. For instance, by adding GIS Raster func-
6.4. Final remarks

Figure 6.7: SumField component’s results window. This is a generic statistics function, that’s why the window displays some values of no interest for this particular problem (only Count and Sum are relevant)

tions the GIS analytical possibilities shown in this chapter would be strongly expanded.

Application domain users could benefit from this or a similar interface to pose, solve and visualise their GIS related problems. Such a system could be made available initially on an local area network environment so big organisations using GIS can benefit from it. The next step would be to implement it on the Internet, bringing GIS analysis to Online GIS.

Figure 6.8: Display of relevant intermediate datasets with MapDisplay
Chapter 7

Discussion

As it is mentioned in Section 1.2 there is a lot to examine as to enabling technology when talking about Open or Online GIS\(^1\). But the research objective here was to concentrate on the design of a prototype visual user interface for an (hypothetical) Online GIS analysis system. For this reason a choice was made to develop a prototype that works in a local way and not to worry about the communication problems inherent to an Internet implementation nor about the issue of portability of GIS function components.

The final version of this prototype was presented and discussed in Chapter 5 together with an example of its application to solve a generic geographic problem, Chapter 6. As the reader must have discovered in these chapters, what lies at the bottom of this implementation is ‘simply’ execution of command lines of GIS functions with arguments in script files. Because the execution of script files is not something new in GIS software, it is important to give some attention now to those novel and relevant issues in the operation of GIS presented in this work, which are considered the contribution of this research in the field of Geoinformatics.

The first relevant contribution, is to show that it is possible to divide the functionality of GIS in completely separate and independent executable programs. This in contrast with the traditional monolithic GIS software that groups a predefined set of GIS functionality ‘under the same roof’. The GIS building blocks as presented in this research do not offer maximum GIS analysis possibilities in themselves, but it is the fact that they can be assembled forming process chains what increases their analysis possibilities. What is very special about this division of GIS in building blocks, is that GIS users can be-

---

\(^1\)The reader is encouraged to take a look at another M.Sc. research performed parallel to the present one where some of the enabling technologies for GIS components are presented and analysed[1].
nefit from individual separate programs of different vendors which assembled together offer the best solution to fulfill their GIS analysis needs. This goes together with the ideas presented in Section 1.1.2 about the possible change in the way how GIS software is used: if these building blocks are available in the Internet, then the users will only pay according to the actual use they make of the different programs and sporadic GIS users will pay for this sporadic use and won’t need to buy complete GIS licences anymore.

A second important contribution is the use of XML metadata files as the ‘connecting phase’ between the GIS user and the components. XML metadata files like the ones presented in this research enable a straightforward discovery of GIS function and dataset components, give the user insight about the discovered components and add intelligence to the assembly of process chains. This enables the creation of meaningful process chains, taking into account the knowledge about GIS function explicitly stored in XML files. This is considered a very important added value if we were to compare this implementation to command-based GIS software programs. The agreement on a standard in the identification of GIS components would allow new components developed by diverse producers to be used by integrating interfaces like the one presented in this research. Additionally, XML metadata gives the possibility to have central catalogs on Internet servers so users can easily discover relevant components fitting their needs. The current situation in Online GIS is that there are many available services offered, but the mechanism that allows potential users to discover them is still missing.

The present research proposed to implement GIS analysis by integrating the two previously mentioned elements: GIS building blocks and XML metadata using an approach that combines input forms for the available functions that give the user textual information about arguments to input, argument checking specially of dataset arguments and flow diagrams that give the user visual insight about the process chains being performed. Although the visual process chaining approach has been used in some well known GIS and Remote Sensing software (ERDAS, ArcView’s model builder), it is the combination with the forms and the argument checking what enforces the meaningful integration between GIS functions and datasets, avoiding also the rising of run-time errors which would become cumbersome to manage in an Internet environment.

---

2According to the definition presented in section 2.1.1
Chapter 8

Conclusions

In accordance to the research objective, the essential characteristics of a user interface for the use of GIS components were first identified and then implemented in a prototype using Visual Basic. It implements the main functionality identified to be important: *discovery of GIS components, creation and validation of process chains and execution of these process chains*.

This interface was also tested for the solution of a generic geographic question and proved to give a useful solution. It is clear that for the definition of process chains in the interface, previous basic knowledge of GIS functionality is required: it is the user's task to select the right functions (and right sequence), that give solution to his question. This approach could be improved by adding more visual cues that support less expert users, but it also has to be noted, that if the goal is to offer an interface for users with no knowledge about geospatial processing, the approach would then need to be completely different than the one presented in this research.

There are other additional functionalities which were identified but couldn't be implemented due to time limitations; these, together with the implemented ones, form a good starting point for future research work in this topic.

The general conclusions that were found in the course of this research are the following:

- Basic GIS functionality as targeted in this research work, can be offered in the form of independent functional components. These components can be assembled in process chains in order to offer GIS analysis solutions.

- The separation of GIS software in functional units requires accurate descriptions of offered functionality and data conditions. This can be effectively achieved with a flexible and structured description technique such as XML.
• XML files, and particularly the XML schema proved to be a very natural and straightforward way to model metadata. The user can easily visualise hierarchies between the different metadata attributes, also repeatable elements are easily identified. Certain validation rules can also be enforced (for instance, enumerations of acceptable values can be specified), improving the reliance of the information stored in them. These benefits are better exploited when using a good XML editor package. Additionally, it is also straightforward to access specific attributes from XML files by using Visual Basic and the Document Object Model reference.

• Any additional function components can be immediately used within the integrating interface as long as they are accompanied with corresponding metadata files structured according to the metadata definition (the XML schema). It must be noted that these functions are not still publicly available and therefore need to be developed in-house. Additionally, not all GIS functionalities are straightforward to implement as components in process chains, this is an approach that works very well for functions requiring little user-interaction (functions requiring only definition of input data and parameters). It is not appropriate for functions that demand extensive user interaction, like topology cleaning.

• Metadata about GIS functions proved to be excellent for the control of correctness of user input arguments, enforcing a correct integration with GIS datasets. Data restrictions of arguments could be specified in the metadata files and later could be checked by the integrating interface.

• Some existing GIS components experiences have successfully dealt with important issues like platform independency, component cataloguing, distributed architecture and have even implemented diverse user interfaces. The integrating interface developed during the present research complements these existing experiences especially in two points: its easiness for the assembly of process chains, which allows to pose and solve complex geographic problems by their decomposition in ‘building blocks’; and the intelligent integration between GIS functions and datasets thanks to the checking of argument conditions and metadata attributes.

• Existing metadata standards for geographic information are very extensive and can be cumbersome to handle and to populate (ending many times being only partially used). The use of a subset of attributes from one of them proved to be enough to ensure the correct integration of GIS datasets with GIS function components and in addition, more straightforward
Chapter 8. Conclusions

to populate and handle.

- ESRI’s ArcObjects and Visual Basic proved to be very good resources for the quick development of GIS function components. Very important and common use GIS functionality is already implemented as methods in some of the available Objects, so the programmer doesn’t have to program these algorithms. It is only required to ‘organise’ these methods according to the requirements. Two limitation are that only ESRI data formats can be used and that some objects require the end-user to possess a local ArcGIS licence.

- The possibilities that an integrating interface like the one presented in this work offers for the solution of geographic questions depends on the availability of suitable GIS function and dataset components.

The following recommendations for further research and refining of a GIS components interface were identified during the course of the present work:

- The GIS analysis possibilities of the integrating interface presented in this research were limited to some GIS vector analysis functions, they can be further extended by adding more functions, for instance, Raster analysis function components.

- To improve the interface for the discovery of components, including search by keyword and conditions on other relevant fields like category, data format, platform (for GIS function components) and geographic coverage, coordinate system, format, description (for Dataset components), the results given by the search could also be rated according to their matching of the given conditions. In this way the discovery becomes more in accordance with the concept that was presented in Section 1.2 of “detection of GIS components of interest for the user”.

- To add user manipulation to the visual modelling interface (so far this interface is only illustrative and one cannot interact with it). The user can be enabled to edit or modify parts of big process chains, by changing or adding the desired GIS function and datasets. In this way there will be more control on the process chains to execute.

- To develop GIS function components which are portable to other operating systems. This is of importance when the goal is to reach users over the Internet. The GIS function components presented in this work can be shared between Windows users (except for MapDisplay which requires an
ArcGIS licence). The use of Java technology enables the development of platform-independent function components.

- To implement the integrating interface in a real Internet environment, so GIS components can also be found when they reside on different servers on the network. The main elements of the present application would be maintained: GIS building blocks, XML metadata, integration based on data conditions and display of process chains. It will be necessary to study the problem of access to components through URLs and the execution of process chains in this distributed environment.

- To consider the design of a GIS components interface intended for users with less GIS or geographic knowledge than the ‘application GIS user’ considered for the design of the present interface. This can be achieved by adding more visual cues to the interface, for instance the GIS functions could further include iconic representations that visually show the users the results that can be achieved with it, helping the novice user in his selection. Also some more ‘general public’ names for the functions can be added as keywords (e.g. ‘corridor’, ‘vicinity’ for ‘buffer’), in this way the users will be able to pose geographic questions in talking language (e.g. “which are the cities on the vicinity of the Amazon river”) and get from the interface the most relevant components that can help to answer the question.

- It is also important to analyse and foresee which would be the effects for GIS users that rely on GIS analysis on the Internet. Although this is an idea that sounds very promising, it is important to note that slow and unreliable Internet connections in some countries, could become a limitation for an implementation of GIS analysis in Online GIS.
Bibliography


Appendix A

XML metadata files for GIS function components

In the following pages, the actual XML files that were created for each one of the GIS function components are presented. There is a separate page for each XML file. The different attributes (defined and explained in Section 4.3) and their values are displayed in a hierarchical way according to the XML schema on which they are based. These files were created using the XML Spy program (Altova).
**General**

<table>
<thead>
<tr>
<th>Operation_Name</th>
<th>select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Query</td>
</tr>
<tr>
<td>Platform</td>
<td>Windows</td>
</tr>
<tr>
<td>Operation_Descr</td>
<td>Selects features using a SQL clause and creates a new dataset with the selection using the same attributes of the original features.</td>
</tr>
<tr>
<td>Creation_Date</td>
<td>2001-09-27</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Keywords</td>
<td>(3)</td>
</tr>
</tbody>
</table>

| Data_format | Shape file |

**Arguments**

<table>
<thead>
<tr>
<th>Required</th>
<th>Repeatable</th>
<th>Name</th>
<th>Type</th>
<th>Order</th>
<th>Description</th>
<th>Direction</th>
<th>Data_Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td>InputFile</td>
<td>Dataset</td>
<td>1</td>
<td>Complete path to the dataset from which features will be selected</td>
<td>IN</td>
<td>Data_Conditions (1)</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>Where</td>
<td>String</td>
<td>2</td>
<td>WhereClause part of the SQL condition to perform the selection. Of the type NAME = 'Colombia'</td>
<td>IN</td>
<td>Data_Conditions (2)</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>OutputFile</td>
<td>Dataset</td>
<td>3</td>
<td>Name of the output file that will contain the selected features</td>
<td>OUT</td>
<td>Data_Conditions (7)</td>
</tr>
</tbody>
</table>

**Data_Conditions**

<table>
<thead>
<tr>
<th>(1) Attribute</th>
<th>(1) Comp</th>
<th>(1) Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFormat</td>
<td>=</td>
<td>shapefile</td>
</tr>
</tbody>
</table>

**Access**

<table>
<thead>
<tr>
<th>Address</th>
<th>D:arcobjects/components/select.exe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditions</td>
<td>none</td>
</tr>
<tr>
<td>Restrictions</td>
<td>none</td>
</tr>
</tbody>
</table>

**Provider**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordSys</td>
<td>InputFile</td>
</tr>
<tr>
<td>CoordSysName</td>
<td>InputFile</td>
</tr>
<tr>
<td>DFormat</td>
<td>InputFile</td>
</tr>
<tr>
<td>Style</td>
<td>InputFile</td>
</tr>
<tr>
<td>Units</td>
<td>InputFile</td>
</tr>
<tr>
<td>OldPath</td>
<td>InputFile</td>
</tr>
</tbody>
</table>

**Conditions**

none

**Restrictions**

none
XML metadata files for GIS function components

### COMPONENT

**General**
- **Operation_Name**: Projection
- **Category**: Coordinate Transformation
- **Platform**: Windows
- **Operation_Description**: Takes a file in geographic coordinates and creates a new one in projected coordinates in a defined projected coordinate system
- **Creation_Date**: 2001-10-08
- **Version**: 1.0
- **Keywords**: (4)
  - projection
  - reference
  - projected coordinates
  - geographic coordinates

**Detailed**

**Data_format**: Shape file

**Arguments**

<table>
<thead>
<tr>
<th>Required</th>
<th>Repeatable</th>
<th>Name</th>
<th>Type</th>
<th>Order</th>
<th>Description</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td>InputFile</td>
<td>Dataset</td>
<td>1</td>
<td>complete path to the input dataset in geographic coordinates</td>
<td>IN</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>ProjCoorSysCode</td>
<td>Number</td>
<td>2</td>
<td>Number that indicates the ESRI code of the projected coordinate system (esriSRProjCStype)</td>
<td>IN</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>OutputFile</td>
<td>Dataset</td>
<td>3</td>
<td>Name of the output file in projected coordinates</td>
<td>OUT</td>
</tr>
</tbody>
</table>

**Data_Conditions**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Comp</th>
<th>Condition</th>
<th>Parameter</th>
<th>Attribute2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordSys</td>
<td>=</td>
<td>Projected</td>
<td>ProjCoorSysCode</td>
<td>CoordSysName</td>
</tr>
<tr>
<td>DFormat</td>
<td>=</td>
<td>InputFile</td>
<td>DFormat</td>
<td></td>
</tr>
<tr>
<td>DGeometry</td>
<td>=</td>
<td>InputFile</td>
<td>DGeometry</td>
<td></td>
</tr>
<tr>
<td>DStructure</td>
<td>=</td>
<td>InputFile</td>
<td>DStructure</td>
<td></td>
</tr>
<tr>
<td>DUnits</td>
<td>=</td>
<td>ProjCoorSysCode</td>
<td>DUnits</td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>=</td>
<td>InputFile</td>
<td>Path</td>
<td></td>
</tr>
</tbody>
</table>

**Access**

<table>
<thead>
<tr>
<th>Address</th>
<th>Conditions</th>
<th>Restrictions</th>
<th>Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>D:\arcobjects\components\Projection.exe</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>
## Appendix A. XML metadata files for GIS function components

### FeatureInPolygon XML file

<table>
<thead>
<tr>
<th>Data Format</th>
<th>Arguments</th>
<th>Data Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Containing Dataset</strong></td>
<td>1</td>
<td>True, False</td>
</tr>
<tr>
<td><strong>Input Contained Dataset</strong></td>
<td>2</td>
<td>True, False</td>
</tr>
<tr>
<td><strong>Output File</strong></td>
<td>3</td>
<td>True, False</td>
</tr>
</tbody>
</table>

**Description:**

Creates a new dataset with the features of a polygon, line or point dataset that are contained by another polygon dataset (Union of the polygons).

**Data Format:** Shape file

**Arguments:**

- **Input Containing Dataset:** Complete path to the input polygon dataset.
- **Input Contained Dataset:** Complete path to the input dataset from which features are selected.
- **Output File:** Name of the output file of the contained features.

**Data Conditions:**

- **CoordSys:** Name of the datum of the containing dataset.
- **CoordSysName:** Name of the datum of the containing dataset.
- **DFormat:** Format of the containing dataset.
- **DGeometry:** Geometry of the containing dataset.
- **DStructure:** Structure of the containing dataset.
- **DUnits:** Units of the containing dataset.
- **Path:** Path of the containing dataset.

**Address:**

D:\arcobjects\components\FeatureInPolygon.exe

**Conditions:**

None

**Restrictions:**

None

**Provider:**

D:\XML\FeatureInPolygon.xml 01/24/02 12:13:47
(c)1998-2001 Altova GmbH   http://www.xmlspy.com

Figure A.4: FeatureInPolygon XML file
Figure A.5: CalArea XML file
Appendix A. XML metadata files for GIS function components

Registered to Rob Lemmens (ITC)

Figure A.6: SumField XML file
**Component:**

- **xmlns:** http://www.w3.org/2001/XMLSchema-instance
- **xsi:noNamespaceSchemaLocation:** D:\XML\component.xsd

**General**

- **Operation_Name:** mapDisplay
- **Category:** Visualisation
- **Platform:** Windows
- **Operation_Description:** Component to display multiple shape files. Contains general display tools like: opening and closing layer, zooming, change of symbology, view of attributes, view and change of display's coordinate system
- **Creation_Date:** 2001-10-08
- **Version:** 1.0

**Keywords**

- text
- visualiser
- visualize
- display
- viewer

**Detailed**

- **Data_format:** Shape file

**Arguments**

- **Required:** true
- **Repeatable:** true
- **Name:** InputFile
- **Type:** Dataset
- **Order:** 1
- **Description:** complete path to the dataset to display
- **Direction:** IN

**Access**

- **Address:** D:\arcobjects\components\mapDisplay.exe
- **Conditions:** none
- **Restrictions:** none

**Provider**

- **Name:** ITC
- **Contact_person:** Julian Gomez
- **Email:** gomez@itc.nl
Appendix B

GIS function components development

In this appendix some programming details of the GIS function components developed for the present research will be presented. The components were developed in Microsoft Visual Basic 6.0 (professional edition), using ESRI’s ArcObjects (see Section 3.8). The programs’ activity diagrams will be displayed to help the reader understand the logic and the usage of the different ArcObjects. The available documentation for ArcObjects [7, 23] was of key importance in the achievement of the desired programming goals.

In other to be able to use the ArcObjects within a new VB project, the ‘ESRI Object Library’ reference must be loaded (esriCore.olb, present on the bin directory of ArcGIS 8.1). This will allow the use of all the functionality of the ArcObjects to develop applications that make use of the ArcGIS technology. Additionally, in order to add file management features to the programs (like file deleting and overwriting), the ‘Microsoft Scripting Runtime’ (scrrun.dll, present on the WindowsNT directory) reference must also be added to the new VB project.

The GIS function components’s visual interface is composed of a main form where the user can visualise the datasets and input the function’s arguments (only previewing-kind of visualisation). The controls that make up the forms in the different components will be explained. In addition to the main form, the components also have a non-visual interface for the case when then arguments are given in the command-line, this is implemented as a module where code is written separately from the form code. When the function is used in ‘off-line’ mode, the arguments must be separated by spaces. For this reason spaces are not allowed inside the arguments.

In all the cases (except for mapDisplay), the start-up object in the Visual-
**B.1. Select**

Basic program is the sub Main() routine of the module, so first the presence of arguments is checked, and if not all of them are present then the main form is displayed, otherwise the program will execute and produce results without showing the form.

**B.1 Select (selectmap_noform.vbp)**

The description of the component's functionality and arguments can be found in Section 3.1. Figure B.1 shows the graphic interface for this component.

![Select graphic interface](image)

Figure B.1: Select graphic interface

The controls types and names are the following:

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CommandButton</td>
<td>cmdOpen</td>
</tr>
<tr>
<td>2</td>
<td>CommandButton</td>
<td>cmdSelect</td>
</tr>
<tr>
<td>3</td>
<td>CommandButton</td>
<td>cmdCount</td>
</tr>
<tr>
<td>4</td>
<td>TextBox</td>
<td>txtFile</td>
</tr>
<tr>
<td>5</td>
<td>TextBox</td>
<td>txtSelect</td>
</tr>
<tr>
<td>6</td>
<td>TextBox</td>
<td>txtCreate</td>
</tr>
<tr>
<td>7</td>
<td>PictureBox</td>
<td>picMap</td>
</tr>
<tr>
<td>8</td>
<td>CommandButton</td>
<td>cmdExit</td>
</tr>
<tr>
<td>9</td>
<td>Form</td>
<td>MyMap</td>
</tr>
</tbody>
</table>
The main steps and arcObjects used in this component are the following: (Figure B.2 shows the corresponding activity diagram)

- A `queryFilter` is used to select the features in the input dataset.
- A `featureCursor` is populated with the selected features
- A new `featureClass` is created as a copy of the input `featureClass`, so the original attributes can be stored there.
- The selected feature and attributes are added to the new feature class.

![Select activity diagram](image-url)
B.2 Projection (NEWProjection.vbp)

The description of the component’s arguments and functionality can be found in Section 3.2. Figure B.3 shows the graphic interface for this component.

![Projection graphic interface](image)

The controls types and names are the following:

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CommandButton</td>
<td>cmdOpen</td>
</tr>
<tr>
<td>2</td>
<td>TextBox</td>
<td>txtFile</td>
</tr>
<tr>
<td>3</td>
<td>TextBox</td>
<td>txtDist</td>
</tr>
<tr>
<td>4</td>
<td>TextBox</td>
<td>txtCreate</td>
</tr>
<tr>
<td>5</td>
<td>CommandButton</td>
<td>cmdSelect</td>
</tr>
<tr>
<td>6</td>
<td>PictureBox</td>
<td>picMap</td>
</tr>
<tr>
<td>7</td>
<td>CommandButton</td>
<td>cmdExit</td>
</tr>
<tr>
<td>8</td>
<td>Form</td>
<td>MyMap</td>
</tr>
</tbody>
</table>

The main steps and arcObjects used in this component are the following: (Figure B.4 shows the corresponding activity diagram)

- The input shape file is assigned to a feature Class.
- A SpatialReferenceFactory is used to create a ProjectedCoordinateSystem.
Appendix B. GIS function components development

- A new featureClass is created with the same fields and geometry definition of the input file.
- The shape property of each feature of the input featureClass is projected to the new coordinate systems and it’s written to the output feature class.
- All the input file’s attribute values are written to the output featureClass.

![Diagram](image)

**Figure B.4: Projection activity diagram**

**B.3 Buffer (buffer2.vbp)**

The description of the component’s functionality and arguments can be found in Section 3.3. Figure B.5 shows the graphic interface for this component.
The controls types and names are the following:

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CommandButton</td>
<td>cmdOpen</td>
</tr>
<tr>
<td>2</td>
<td>TextBox</td>
<td>txtFile</td>
</tr>
<tr>
<td>3</td>
<td>TextBox</td>
<td>txtDist</td>
</tr>
<tr>
<td>4</td>
<td>CommandButton</td>
<td>cmdCreate</td>
</tr>
<tr>
<td>5</td>
<td>TextBox</td>
<td>txtCreate</td>
</tr>
<tr>
<td>6</td>
<td>PictureBox</td>
<td>picMap</td>
</tr>
<tr>
<td>7</td>
<td>CommandButton</td>
<td>cmdExit</td>
</tr>
<tr>
<td>8</td>
<td>Form</td>
<td>MyMap</td>
</tr>
</tbody>
</table>

The main steps and arcObjects used in this component are the following:

1. The input shape file is assigned to a featureClass and its constituting features are assigned to a featureCursor.

2. A cycle is performed in which the geometry of each individual feature is generalised and simplified (otherwise errors of complex geometry arise). Afterwards, the buffer at the given distance is calculated and also simplified.

3. As new features are processed, the union of the buffer polygons is calculated.
Appendix B. GIS function components development

A new featureClass is created with the same spatial reference of the input feature class and with a numeric attribute to hold the buffer distance.

The final buffer polygon is added to the new feature class. The buffer distance is added as its only attribute.

Figure B.6: Buffer activity diagram

B.4 FeatureInPolygon (FeatureInPolygon.vbp)

The description of the component’s functionality and arguments can be found in Section 3.4. Figure B.7 shows the graphic interface for this component.
B.4. FeatureInPolygon

The controls types and names are the following:

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CommandButton</td>
<td>cmdOpenPol</td>
</tr>
<tr>
<td>2</td>
<td>CommandButton</td>
<td>cmdOpenPnt</td>
</tr>
<tr>
<td>3</td>
<td>CommandButton</td>
<td>cmdCount</td>
</tr>
<tr>
<td>4</td>
<td>TextBox</td>
<td>txtPolyg</td>
</tr>
<tr>
<td>5</td>
<td>TextBox</td>
<td>txtPoint</td>
</tr>
<tr>
<td>6</td>
<td>TextBox</td>
<td>txtCreate</td>
</tr>
<tr>
<td>7</td>
<td>PictureBox</td>
<td>picMap</td>
</tr>
<tr>
<td>8</td>
<td>CommandButton</td>
<td>cmdExit</td>
</tr>
<tr>
<td>9</td>
<td>CommandButton</td>
<td>cmdSelect</td>
</tr>
<tr>
<td>10</td>
<td>Form</td>
<td>MyMap</td>
</tr>
</tbody>
</table>

The main steps and arcObjects used in this component are the following: (Figure B.8 shows the corresponding activity diagram)

- The input shape files are assigned to two feature Classes.

- The features that form the ‘containing’ feature class area assigned to a featureCursor and a cycle is performed in order to calculate the union of the polygons using the ‘union’ method of the ITopologicalOperator interface.
The contained features are selected and assigned to a Feature cursor by using a SpatialFilter and setting its ‘Geometry’ property equal to the union polygon.

A new featureClass is created as a copy of the input ‘contained’ featureClass, so the original attributes can be stored there.

The contained features and attributes are added to the new feature class.

---

**B.5 CalArea (calArea.vbp)**

The description of the component’s functionality and arguments can be found in Section 3.5. Figure B.9 shows the graphic interface for this component.
The controls types and names are the following:

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CommandButton</td>
<td>cmdOpenPol</td>
</tr>
<tr>
<td>2</td>
<td>CommandButton</td>
<td>cmdCreate</td>
</tr>
<tr>
<td>3</td>
<td>TextBox</td>
<td>txtPolyg</td>
</tr>
<tr>
<td>4</td>
<td>TextBox</td>
<td>txtArea</td>
</tr>
<tr>
<td>5</td>
<td>PictureBox</td>
<td>picMap</td>
</tr>
<tr>
<td>6</td>
<td>CommandButton</td>
<td>cmdExit</td>
</tr>
<tr>
<td>7</td>
<td>Form</td>
<td>MyMap</td>
</tr>
</tbody>
</table>

The main steps and arcObjects used in this component are the following: (Figure B.10 shows the corresponding activity diagram)

- The input shape file is assigned to a feature Class.
- If the area field name given by the user doesn't exist, a new Area field is created and it's added to the feature class using the IWorkspaceEdit interface.
- The features that compose the feature class are assigned to a feature-Cursor and a cycle is performed in order to calculate the area of each individual polygon, using the Area property of the IArea interface. This
area value is added to the newly created area field (or to the existing specified area field if it already exists).

- The total area of the feature class and area units are reported on a message box.

![Diagram](image)

Figure B.10: CalArea activity diagram

### B.6 SumField (sumField.vbp)

The description of the component’s functionality and arguments can be found in Section 3.6. Figure B.11 shows the graphic interface for this component.

The controls types and names are the following:
The main steps and arcObjects used in this component are the following: (Figure B.12 shows the corresponding activity diagram)

- The input shape file is assigned to a feature Class.

- It is checked that the field name given by the user exists in the fields collection (using the findField method of the IFields interface). It is also checked that it corresponds to a numeric field.

- The BaseStatistics coclass is used to calculate the statistics: a feature cursor and a cycle are used to add all values of the corresponding attribute to the base statistics.

- The IStatisticsResults interface is used to report the main statistics.
Appendix B. GIS function components development

in a message box (sum, count, maximum, minimum, mean and standard deviation).

B.7 MapDisplay (mapDisplay.vbp)

The description of the component’s functionality and arguments can be found in Section 3.7. In order to incorporate common-use GIS visualisation tools, the Map Control of ESRI must be added to the project in addition to the ‘ESRI Object Library’ reference. It is therefore necessary to load the ‘ESRI ArcObjects Control 8.1’ component (afcontrols.ocx). The Map Control has a set of properties, methods and events that facilitate the implementation of visualisation
tools and allow the easy development of stand-alone applications using ArcObjects, but it requires to have a local ArcGIS licence. Figure B.13 shows the graphic interface for this component.

![MapDisplay graphic interface](image)

Figure B.13: MapDisplay graphic interface

The controls types and names are the following:

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Toolbar</td>
<td>Toolbar1</td>
</tr>
<tr>
<td>2</td>
<td>ImageList</td>
<td>ImageList1</td>
</tr>
<tr>
<td>3</td>
<td>MapControl</td>
<td>MapControl1</td>
</tr>
<tr>
<td>4</td>
<td>ComboBox</td>
<td>cmbLayers</td>
</tr>
<tr>
<td>5</td>
<td>CheckBox</td>
<td>chkQuery</td>
</tr>
<tr>
<td>6</td>
<td>ComboBox</td>
<td>cmbFields</td>
</tr>
<tr>
<td>7</td>
<td>ComboBox</td>
<td>cmbComp</td>
</tr>
<tr>
<td>8</td>
<td>TextBox</td>
<td>txtValue</td>
</tr>
<tr>
<td>9</td>
<td>Label</td>
<td>lblCoord</td>
</tr>
<tr>
<td>10</td>
<td>Form</td>
<td>Form1</td>
</tr>
</tbody>
</table>

Because this is a *visualisation* function the only user interface it has is the form with which the user can interact. Additionally, it also accepts arguments
given in the command-line (the files to display), but the form will always be displayed regardless. The main functionality offered in this component and the corresponding arcObjects used are the following:

- The user can open shapefiles and select them interactively in a dialog box thanks to the IGxDialog interface. The selected shape files are added as Feature Layers to the map control.

- Using the chkQuery check box, the user can select features from the active layer using SQL. The cmbFields combo box is automatically filled with all the available attributes from the active layer.

- Using the ArcGIS’ AfCommands library, some common-use GIS commands were added to the component, including: the ‘identify’ tool, the ‘query’ tool that allows to query interactively, and the ‘table of contents’ tool, which allows the user to define layer properties, including symbology. In order to use it, the ‘ESRI AfCommands(VB)’ reference must be added to the project.

- The ‘OnMouseDown’ event of the map control was used to program the zoom and pan tools. The ‘OnMouseMove’ event allows the display of map coordinates on the label control.

- The ISpatialReference and ISpatialReferenceDialog interfaces together with the SpatialReference property of the map control, allow the user to set the view’s coordinate system.
Appendix C

Integrating Interface development

In the following pages some programming issues corresponding to the development of Integrating Interface presented in Chapter 5 will be discussed. It was developed using Microsoft Visual Basic 6.0 (professional edition). The different windows that compose the interface will be presented together with activity diagrams of the main procedures.

In order to enable the VB application to read and create files, the ‘Microsoft Scripting Runtime’ reference (scrrun.dll) must be added to the VB project. Likewise, to enable the access to XML files and attributes, the ‘Microsoft XML’ reference must be loaded, allowing the use of all the functionality of the Document Object Model (see Section 4.5). As to VB components, in order to make possible the generation and evaluation of code at run-time, the Microsoft Script control must be attached to the project. The Microsoft FlexiGrid control must also be added to enable the display of records in a tabular form.

The Integrating Interface’s visual interface is composed of seven windows (VB forms) as was presented in Chapter 5. It was developed using the multiple document interface (MDI), which enables the development of applications composed by multiple simultaneously active windows. These windows and their controls are listed below.

- **MDIForm1**, MDI parent window. This form is a container for all the other windows and contains the menu (see Figure 5.10, page 48). The following forms (except for frmDir) are ‘child windows’ of MDIForm1.

- **Form1**, window for the display of discovered GIS Function components (see Figure 5.2, page 38). It is comprised of the following controls:
• *FrmData*, window for the display of discovered GIS Dataset components (see Figure 5.3, page 39). It includes the following controls:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSHFlexGrid</td>
<td>Grid1</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>lblResults</td>
<td></td>
</tr>
<tr>
<td>ScriptControl</td>
<td>ScriptControl1</td>
<td></td>
</tr>
</tbody>
</table>

• *FrmAttr*, ‘Template’ form for the user to input the GIS function arguments. In design time this form has only one of each type of control (labels, textbox), the required controls are loaded at run-time. In Figure 5.4, page 40 there are two examples of run-time forms. This form includes the following controls:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Labels(0)</td>
<td>Array of labels displaying the argument’s names</td>
</tr>
<tr>
<td>TextBox</td>
<td>TextBoxes(0)</td>
<td>Array of textboxes where the user types the argument’s value</td>
</tr>
<tr>
<td>Label</td>
<td>lblAdd(0)</td>
<td>Contains the ‘+’ sign, used for repeatable arguments</td>
</tr>
<tr>
<td>CommandButton</td>
<td>cmdAdd</td>
<td>Adds the currently given arguments to the script</td>
</tr>
<tr>
<td>Label</td>
<td>lblDesc</td>
<td>Label at the bottom of the form that displays the argument’s descriptions</td>
</tr>
</tbody>
</table>

• *frmDir*, window that prompts the user to select a LAN drive to start the search of components. It includes the following controls:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriveListBox</td>
<td>drvPath</td>
<td></td>
</tr>
<tr>
<td>DirListBox</td>
<td>dirPath</td>
<td></td>
</tr>
<tr>
<td>CommandButton</td>
<td>cmdOK</td>
<td></td>
</tr>
<tr>
<td>CommandButton</td>
<td>cmdCancel</td>
<td></td>
</tr>
</tbody>
</table>
Appendix C. Integrating Interface development

- **frmInteg**, form for the display of script lines (see Figure 5.8, page 45). It consists of a ListBox and a CommandButton.

- **frmModel**, window for the display of process chains or visual models (see Figure 5.9, page 46). It includes two PictureBox controls, vertical and horizontal ScrollBars.

In the following pages, the activity diagrams corresponding to the main functionalities offered through the previously mentioned windows (i.e., Discovery of GIS functions, Discovery of GIS Datasets, Creation of process chains and Execution of script files) will be presented (figures C.1, C.3, C.2 and C.4). These functionalities are carefully explained in sections 5.1, 5.2 and 5.3.

![Activity diagram for the Discovery of GIS function components](image)

Figure C.1: Activity diagram for the Discovery of GIS function components
The user double clicks one of the discovered GIS function components (Grid control in Function components form) 

Fetch component’s name, path and # of arguments from the Grid control 

Read XML metadata records corresponding to the function’s arguments (name, description, type, data conditions) 
Store in a user-defined array 

Display frmAttr (argument input form) 
Load a number of controls according to the number of arguments 
Assign labels according to argument names and descriptions 

The user inputs argument values either by directly typing or by double clicking one of the discovered datasets (Grid control in Dataset components form) 

Clean values of argument input form 
Reset ScriptControl 

[Dataset arguments] 
Read XML metadata records corresponding to the selected Dataset’s properties, assign to a user-defined variable and add this variable to the ScriptControl 

[Non-Dataset arguments] 
Generate code for a procedure that checks each one of the Data Conditions associated to the argument against the Dataset’s properties 

Send the code to the ScriptControl and run 

The user presses the ‘Add to script’ button 

[all conditions are satisfied] 
[missing arguments, numeric or string types not correct] 
[all arguments were input, numeric and string types are correct] 
[all conditions are satisfied] 

Display error message specifying the non-fullfilled condition 

Generate code for a procedure that assigns meta-attribute values for output datasets based on the Data Conditions and the metadata values of the input datasets 

Send the code to the ScriptControl and run 

Print meta-attribute values (in green color) in the last row of the GridControl in the Discovered Datasets window 

Generate a script line based on the values given by the user 
Display in a listBox in the Integration Form 

Draw GIS function and argument boxes in the Visual Model window 

Figure C.2: Activity diagram for the creation of a line of script
The user selects the 
>Discovery >GIS Datasets menu

A form is displayed for the user to select a path in the LAN

Look recursively for XML files in the selected path
Return found files into a Dictionary object

[XML file: ObjectRoot='Metadata'
AND node('onlink') exists]
Read metadata records from the corresponding nodes in the XML file.
Store them in a user-defined variable

[No more elements in the Dictionary]
[There are more elements in the Dictionary]

Print the metadata values in the GridControl
(one row per GIS Dataset found)

Figure C.3: Activity diagram for the *Discovery of GIS dataset components*

The user presses the 'Execute Script' button (frmInteg)

A *.bat* temporary file is created in disc with the script lines from the listBox

The *.bat* file is executed using the Shell command

Figure C.4: Activity diagram for the *Creation and execution of Script files*