Designing and Performing Geographic Analysis Processes with GISCASE

Cirano IOCHPE, Guillermo N. HESS, Cláudio RUSCHEL, Alécio P. D. BINOTTO, Márcia A. S. de ALMEIDA, Luciana V. da ROCHA[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{ciochpe,hess,claudior,abinotto,marcia,vargas}@inf.ufrgs.br

Abstract. *This work presents GISCASE, an UML CASE tool based on the conceptual framework GeoFrame-A to aid GIS users to model their geographical analysis processes. Starting from a graphical specification of the geographic process, the tool generates an XML intermediate code free of platform and then creates the specific programs for a GIS system. In addition, GISCASE is entirely developed using free software and technologies, independent of GIS platform and has a modular architecture, which means it is easy to develop modules to support a wide range of GIS Software.*

Resumo. *Este trabalho apresenta o GISCASE, uma ferramenta CASE UML baseada no framework conceitual GeoFrame-A para auxílio aos usuários de SIG na modelagem de seus processos de análise geográfica. A partir de uma especificação gráfica do processo geográfico, a ferramenta gera um código XML intermediário independente de plataforma, criando programas específicos para um sistema de SIG. Adicionalmente, o GISCASE é todo desenvolvido utilizando software e tecnologias livres, independente de plataforma de SIG e tem sua arquitetura dividida em módulos, o que significa que é simples desenvolver módulos com suporte a uma vasta gama de programas de SIG.*

1. Introduction

Users of Geographic Information Systems (GIS) may come from different professional areas, such as geography, geology, urban planning, etc. This leads to a scenario where these users may not be familiar with the database and process modeling. Furthermore, they may know very well a specifi GIS software. However if the GIS platform is changed the process and data design may be lost.

For these reasons, there is a need for models and tools to aid the definition and design of both geographic data and geographic analysis processes in a language that all kind of users may understand. The tools used in software engineering more specifically those used in Geographic Database (GDB) design can be considered as candidates to supply this need. Some related works proposed data models and frameworks to guide the GDB design, such as MADS [Parent et al. 1998], GeoFrame [Rocha et al. 2001] and GeoOOA [Köesters et al. 1996]. However, these data models and frameworks provide support only for the description of static components of the GIS, i.e., the data. Therefore, there is a gap in terms of models and tools for the dynamic GIS components, that is, the processes. Even though some GIS software provide tools for the processes definitions they are platform dependent, designed for specific GIS software.

Therefore, we propose GISCASE, a software tool to guide the design of Geographical analysis Processes (GP), based on a conceptual framework called GeoFrame-A [Ruschel et al. 2005]. The goal is to provide a graphical interface, using a formal semantics, to allow users that are not very familiar to the GIS language to define their own processes for different GIS softwares, as GISCASE has a platform independent architecture. From this formal definition, it generates source code compatible with the chosen platform.

The paper is organized as follows. In section 2 we present an overview of the GeoFrame-A framework, the GISCASE basis. In section 3, we directly present the GISCASE architecture, especially the Graphical Editor, GPtoXML and XMLtoGIS modules. In section 4, the implementation of the proposed technique using the TerraLib GIS API is described. Finally, in section 5, we discuss the results of our work, conclusions and directions for future work.

2. The framework GeoFrame-A

GeoFrame is an object oriented conceptual framework which makes use of the Unified Modeling Language (UML) [Fowler 2000]. The GeoFrame allows the GDB design, by providing a set of classes to be instatiated, specialized and extended (that covers the geographical data features, such as spatiality and temporality), beyond the descriptive attributes of the non-geographic data. Continuous efforts were made recently by Ruschel (2003) who developed GeoFrame-A (GeoFrame with Action), extending GeoFrame to support geographic analysis processes modeling.

In GeoFrame-A, specifically in the class diagram where the spatio-temporal and descriptive data are designed, the GP is represented as an *Activity Class*, defined by UML 2 specification. Taking advantage of the UML concepts, the behavior of an *Activity* is characterized as a sequence of subordinated units where each individual element is an *Action*. A stereotype like a gear identifies such class, while other stereotypes show the spatial representation.

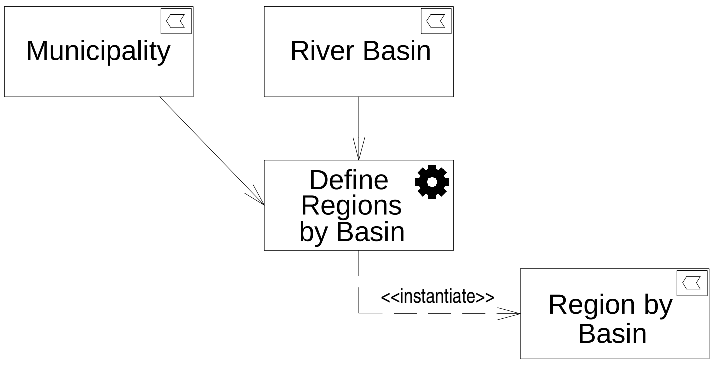


Figure 1. A class diagram using GeoFrame-A

Figure 1 presents a simple process definition using a class diagram with GeoFrame-A. This process uses spatial classes represented through polygons

(Municipality and River Basin) to define a new kind of region (Region by Basin) which considers both political and natural aspects.

Moreover, the UML activity diagram details the GP specification and we can define data and control flows during the GP, as well as the inputs needed for each operation and the outputs expected. In addition, data is represented as object nodes and operations as activity nodes.

The GeoFrame-A definition included a research and a classification of the basic geoprocessing operations. Thus, the Geoframe catalog define operations that deals only with non-spatial attributes, like *Selection*, and others that deals with spatial attributes, such as *Spatial Selection*, *Buffer*, *Overlay*, *Dissolve* and *Classification* [Ruschel 2003].

Figure 2 details the process presented in Figure 1 through an activity diagram. The notation used is compatible with the GISCASE tool. In this diagram, one can read that the objects of the Municipality class are aggregated through the attribute "Id_AdmRegion" (such attribute would identify in which Administrative Region a municipality is contained). The prefix "DS" identifies the Dissolve operation, that erase the border line between polygons with equal values for a specified attribute. Next, the result of this operation is intersected with the objects of the class River Basin. The geometric intersection between these objects is performed by the geoprocessing operation Overlay (identified with the prefix "OV"), using the Boolean operator "AND". The final result is a set of objects that represent a new administrative region definition (class "Region by Basin").

Both the field and object visions of a geographic phenomenon spatial representation were considered. For each operation, the entry parameters were described. The geographic analysis operations are, in general, made with long (and even complex) algorithms. However, many of the operations defined in GeoFrame-A are already implemented in common GIS software, but probably not exactly with the same names. That happens because GeoFrame-A is entirely GIS independent and the operations name may vary from one GIS to another.

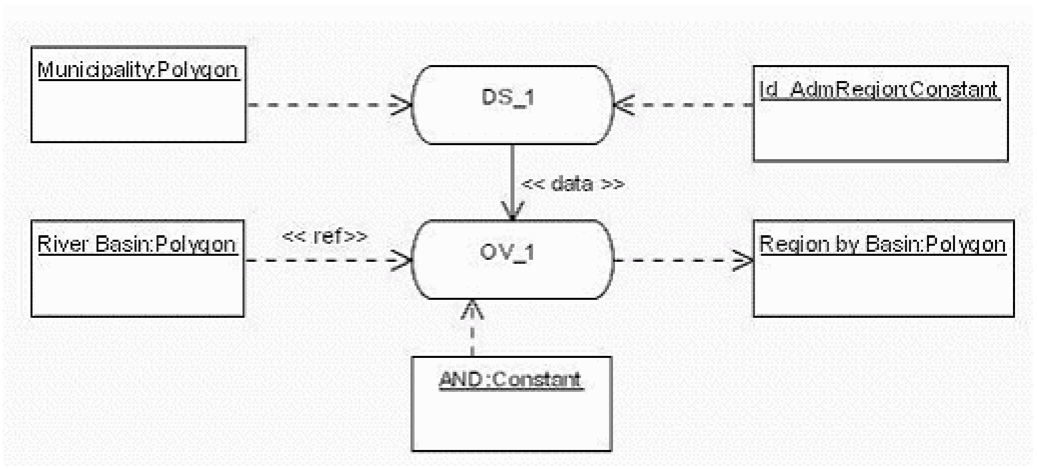


Figure 2. An activity diagram compatible with the GISCASE tool

3. GISCASE´S Architecture

The activity diagram semantics in a conceptual level is enough to perform an automatic conversion to computer systems if some conventions were used. In GISCASE, it was defined to consider that instance of spatial classes are represented through table register in a geographic database. On the other hand, it was considered too that the operations defined in the activity diagram have a corresponding implementation available on a GIS software API.

To automate such processes, we developed the GISCASE tool, which creates those programs for the user automatically. All the concern for the user is to define his process and then choose the desired GIS API. It consists on three main modules (Graphical Editor, GPtoXML and XML to GIS), as Figure 3 illustrates.

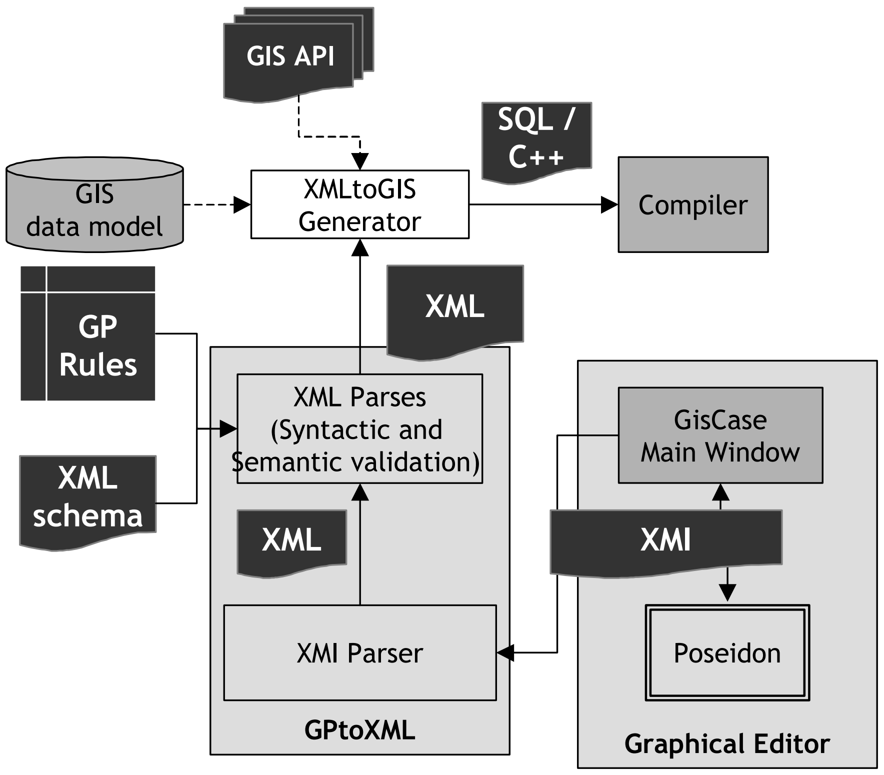
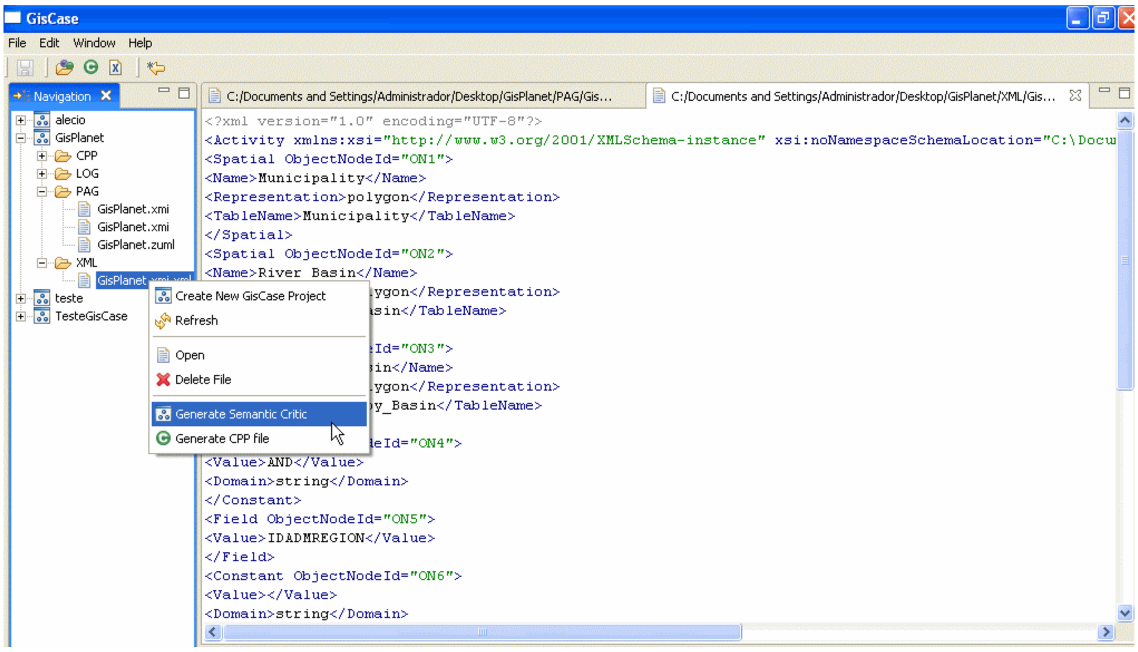


Figure 3. GISCASE software architecture

These modules are managed by a main interface (Figure 4), where the user can view (and even edit) the results of each module. Also through this interface, the user can define the database system and the GIS API that should be used.



```
<?xml version="1.0" encoding="UTF-8"?>
<Activity xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\Docu
<Spatial ObjectNodeId="ON1">
<Name>Municipality</Name>
<Representation>polygon</Representation>
<TableName>Municipality</TableName>
</Spatial>
<Spatial ObjectNodeId="ON2">
<Name>River Basin</Name>
ygon</Representation>
sin</TableName>

Id="ON3">
in</Name>
ygon</Representation>
y_Basin</TableName>

eId="ON4">
<Value>AND</Value>
<Domain>string</Domain>
</Constant>
<Field ObjectNodeId="ON5">
<Value>IDADMREGION</Value>
</Field>
<Constant ObjectNodeId="ON6">
<Value></Value>
<Domain>string</Domain>
```

Figure 4. A screenshot of the GISCASE interface

3.1. Graphical Editor

The Graphical Editor is an external tool that can be activated by GISCASE, in which the user creates its GP graphically, using the UML activity diagram.

GISCASE is capable of using generic graphical editors or CASE tools that have, at least, the basic features of an activity diagram. The editor must be able to distinguish Action Node and Object Node, and also represents flows between them. Each flow must be identified as a data flow or as a control flow through stereotypes, such as their roles. Other control flow elements, such split nodes, must be available.

Another requisite for the editor to be used as the GISCASE's graphical interface is the capability of generating code in an XML like language. The reason for that is the simple fact that the XML language (and the languages based on it) is actually a standard for coding and sharing information [Bradley 2002].

As, in general, UML CASE tools already support and implement the OMG XML Metadata Interchange (XMI), it was chosen as the output needed from the graphical editor. However, the choice for the XMI was due to some other features: The XML Metadata Interchange Format (XMI) specifies an open information interchange model that is intended to give developers working with object technology the ability to exchange programming data over the Internet in a standardized way, thus bringing consistency and compatibility to applications created in collaborative environments (2003). Furthermore, its main purpose is exactly the feature we were looking for, which is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF) in distributed heterogeneous environments.

3.2. GPtoXML Parser

The GPtoXML Parser is the module responsible for translating the geographical analysis process (GP) into an eXtensible Markup Language (XML) encoding. This module, at first, translates the XMI file generated by the Graphical Editor through a XMI Parser. This parser must be customized for each specific graphical editor.

The relevant information (elements) is coded into a new XML file, which is based on a defined XML-Schema. The XML schema may be seen as the XML encoding for the GeoFrame-A. All the components of the GeoFrame-A model are covered by the XML-Schema. In the same way that the GP objetcts nodes and actions are instances of the GeoFrame classes, the XML encoding for the GP is an instance of the XML-Schema we defined.

The schema has two basic components, the Object Nodes and the Actions. The Object Nodes are the inputs as well as the outputs of each one of the operations of the activity described by the GP. They can be spatial data, database descriptive fields, constants or even variables.

The actions are the operations performed during the process. They can be defined to be executed over the descriptive part of the data, or over spatial features. Furthermore they can be unary or binary. An action references the Object Nodes that are its inputs and output. An action may have at least one input (or more) and exactly one output.

As a GP may be composed by more than one action, it is important to distinguish the dependencies between the actions. In some cases it is only a matter of order, that is, one action has to occur before or after another but they are independent in terms of data. In other cases there is also a data dependency, which means that the result of an action is the input for the next action. For that reason, when designing the GP in the graphical editor the user has to specify the flow type when linking two actions. By default it is a control (order) dependency (control flow). If it is a data dependency (data flow), a stereotype <<data>> has to be included.

When the parser finds a control flow, it only puts the actions in the specified order. When a data flow is found, a new Object Node is created as the output of the first action and the input of the second action.

Once in XML the GP is parsed to check its semantic correctness, that is, verify if the actions were correctly defined as well as the inputs and outputs (object nodes). This verification has nothing to deal wit the XML file validation. The GP may be valid, that is, correct when faced against the XML-Schema, but semantically incorrect. This occurs when the user specifies incorrect parameter (object nodes) as inputs or output for an action. The semantic checker verifies:

- If the number of inputs (object nodes) for each action is correct;
- If the data type of the inputs and output are all correct. For example, the overlay operation requires two spatial data and one constant as inputs and generates a spatial data as output;

- If the spatial data specified as inputs and outputs are geometrically consistent. For example, in a buffer operation, the output has to be always a polygon.

After the semantic validation, a log file is created to allow the user to check theerrors, if there are some.

3.3. XMLtoGIS Generator

The GP described in XML allows the storage as well as its interchange on a standard format. However, none of the GIS solutions support the definition of a GP in that format. Thus, this module translates the GP into a specific program for the desired GIS. To do that, the XMLtoGIS Generator considers that all data required in the process definition relies in a geographic database, i.e., a database system that can handle both spatial and non-spatial data.

Following this idea, for each GIS software, it makes necessary the development of a specific library which maps the operations defined in GeoFrame-A to the operations offered by the GIS software API. Therefore, the module XMLtoGIS generates the source-code that makes reference and calls these API operations, passing all the necessary parameters in a compatible order of process definition. As a result, only the operations used by the defined process are included in the source-code.

As a final step, the source code generated must be compiled and linked with the required GIS libraries, or interpreted within the GIS software environment. When the program is executed, the process will read the required data in the database, perform the geographic operations as defined, and store the results in existing or new tables. Note that the actual names of tables and fields must be used in the activity diagram, so it can be recognized in the process execution and the intermediate data generated in the process execution is deleted at the end of the process.

4. Implementation

To implement a first prototype of GISCASE, we decided to make use of an existing UML CASE tool called Poseidon (2005). The reason for choosing Poseidon was because our entire CASE Tool must be based on free software and also because it is possible to export the diagram into the XML Metadata Interface (XMI), enabling to parse this XML based file. We are using the version 3.1 of the Community Edition and it is activated from our tool as GISCASE subprocess when the user wants to graphically edit the activity diagram.

At the moment, our tool supports one API called TerraLib (2005) (a Brazilian geographic library). This library is based on an on-going project and classified as open source, free software and developed completely as object-oriented. The main goal of TerraLib is to supply the absent of libraries that covers the growing treating complexity of spatial information.

4.1. Operations

In the first implementation of the GISCASE's prototype only the representations of point, line and polygon are supported, as a limited number of operations, which are:

- GcSelection: select objects by attributes;
- GcRegionSelection: select objects that fall inside a region defined by a polygon;
- GcSpatialSelection: select objects that have a spatial relationship to a reference set of objects;
- GcBuffer: creates a polygon with a specified distance from input objects;
- GcOverlay: creates a new set of objects through the overlay of two set of objects;
- GcDissolve: creates a new set of objects through the generalization of input objects;
- GcCentroid: return the centroid of a polygon;
- GcAlgebra: in this implementation performs a SQL Update command;
- GcDistance: calculate the distance of two geometries.

These operations are already implemented in the TerraLib API and tested in TerraView (2005), a desktop GIS built for the TerraLib. The TerraView is operated through a menu interface and does not have a macro language for processes execution yet. Hence, GISCASE offers to the TerraView users a way to generate executable programs for frequently executed processes.

The GISCASE was almost entirely implemented in Java, using the Eclipse IDE. The choice for the Java language was due it is free, platform independent and easy to parse XML files.

As mentioned, we developed specific source code for translating the XML GP definition into TerraLib classes and methods, organized as a library named "GcGeoOperations for TerraLib". For each operation that could be defined in the activity diagram, a function was written calling the respective TerraLib function. This code was written in C++ because it is the language of the TerraLib API.

The final product of the GISCASE is a source code, also in C++ that makes reference to this library.

The reason for creating this intermediate layer is simple. When extending the GISCASE to cover other GIS APIs, the only effort will be on creating specific libraries for the new APIs. The library functions will have the same names for the same operations, and thus it will not be necessary to change the code generated by GISCASE.

4.2 Case Study

To verify the functioning of the GISCASE tool, we ran tests with all the operations implemented, in different scenarios. We tested the operations alone as well as combined with other operations. Also GP incorrect definitions were tested. In all cases the GISCASE results were correct.

To illustrate the process of generating a TerraLib source code from a graphical specification, the GP used here is the one presented by Figure 2.

The listing presented in Table 1 shows the GP's XML code generated by the GPtoXML Parser module.

Table 1. Example of XML code generated by GISCASE

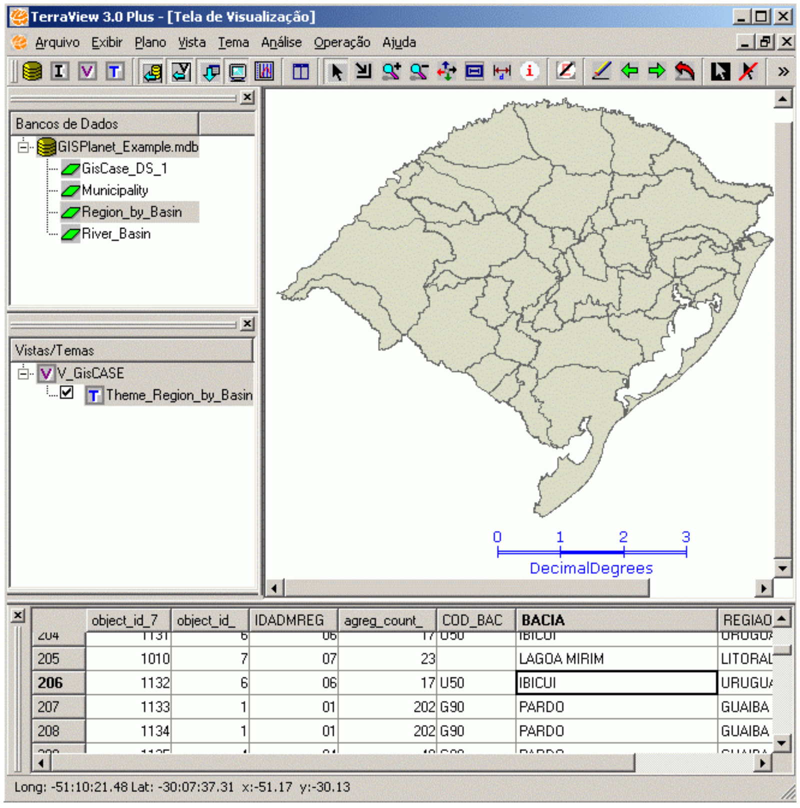
```
<?xml version="1.0" encoding="UTF-8"?>
<Activity xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="C:\Documents and Settings\Hess\My
        Documents\UFRGS\projetos\GISCase\pag2.xsd">
        <Spatial ObjectNodeId="ON1">
                <Name>Municipality     </Name>
                <Representation>polygon</Representation>
                <TableName>Municipality</TableName>
        </Spatial>
        <Spatial ObjectNodeId="ON2">
                <Name>River_Basin</Name>
                <Representation>polygon</Representation>
                <TableName>River Basin</TableName>
        </Spatial>
        <Spatial ObjectNodeId="ON3">
                <Name>Region by Basin</Name>
                <Representation>polygon</Representation>
                <TableName>Region_by_Basin</TableName>
        </Spatial>
        <Constant ObjectNodeId="ON4">
                <Value>AND</Value>
                <Domain>string</Domain>
        </Constant>
        <Constant ObjectNodeId="ON5">
                <Value>IdAdmReg</Value>
                <Domain>string</Domain>
        </Constant>
        <Spatial ObjectNodeId="GisCase_AC2">
                <Name>GisCase_DS_1</Name>
                <Representation>polygon</Representation>
                <TableName>GisCase_DS_1</TableName>
        </Spatial>
        <Action ActionId="AC1">
                <Name>OV_1</Name>
                <Input InId="ON2" Ref="1"/>
                <Input InId="ON4"/>
                <Input InId="GisCase_AC2"/>
                <Output OuId="ON3"/>
        </Action>
        <Action ActionId="AC2">
                <Name>DS_1</Name>
                <Input InId="ON1"/>
                <Input InId="ON5"/>
                <Output OuId="GisCase_AC2"/>
        </Action>
</Activity>
```

The listing of Table 2 is part of the C++ source code corresponding to the GP, as defined in the XML file presented in Table 1, generated by XMLtoGIS Generator module. This main procedure creates the connection with the database and calls the GcDissolve and GcOverlay functions with the necessary parameters. The code for establish the connection with the database is also specific to the TerraLib API, which requires the use of its own classes, like "TeDatabasePortal", for instance. The code that implements the GISCASE operations is read from the "GcGeoOperations for TerraLib" library and appended in the same text file of the main procedure.

Table 2. Example of C++ code generated by GISCASE

```
int main(){
    string dbname = "C:\GeoDB\GisCase.mdb";
    TeAdo* db = new TeAdo();
    db->close();
    if (!db->connect("localhost","","",dbname,0)) {
       cout << "Error in ADO connection: " << db->errorMessage() << endl;
       cout << endl << "Press Enter";
       db->close();
       getchar();
       return 1;
    }
    cout << "Database connection successfull: '" << dbname << " ' !";;
    TeAdoPortal* dbPortal = new TeAdoPortal(db);
    TeDatabasePortal* portal = dbPortal;
    if (!portal)
        return false;
    bool val;
    val = GcDissolve("Municipality","GisCase_DS_1","IdAdmReg",db, portal);
    if (!val)
        cout << "Error in function GcDissolve!" << endl;
    val = GcOverlay("GisCase_DS_1","River_Basin","Region_by_Basin","AND",db,
                     portal);
    if (!val)
        cout << "Error in function GcOverlay!" << endl;
    portal->freeResult();
    db->close();
    return 0;
}
```

After compiling this code using the TerraLib API in the Visual C++ IDE and executing it, the result can be visualized in Figure 5. It is a screenshot of the TerraView displaying the layers generated by the execution of the GP defined in GISCASE.



	object_id_7	object_id_	IDADMREG	agreg_count_	COD_BAC	BACIA	REGIAO
204	1131	6	06	17	U50	IBICUI	URUGUA
205	1010	7	07	23		LAGOA MIRIM	LITORAL
206	1132	6	06	17	U50	IBICUI	URUGUA
207	1133	1	01	202	G90	PARDO	GUAIBA
208	1134	1	01	202	G90	PARDO	GUAIBA

Long: -51:10:21.48 Lat: -30:07:37.31 x:-51.17 y:-30.13

Figure 5. A screenshot of TerraView's interface displaying the result generated by GISCASE

5. Conclusions and Future Work

GIS users are not used to formally design geographic process because each software platform has its own interface and specific set of commands. GeoFrame-A and GISCASE aim to offer a model and a tool to aid a typical GIS user to design geographic process in a high abstraction level. An important characteristic of the GISCASE architecture is that it uses free software principles, but it can also be used to the development of wrappers to commercial GIS libraries. The initial implementation using the Terralib API complements a set of application developed in Brazil using this library [TerraLib 2005].

Another contribution is the definition of a XML schema able to define a geographic process. Furthermore, by storing the geographic process definition in an XML file it is possible to share it easily, especially over the Internet.

We also want to emphasize the relationship with the TerraLib developer team, which we could contribute to validate the library and suggest some alternatives for the functions.

As future works one intends to implement other geographic analysis operations, as well extends the tool to generate code in other GIS software platforms. It is also in progress an interface to allow the user to define his GP in a formal grammar using a map algebra [Camara et al. 1994], compile it and generate a XML file compatible with GISCASE [Martins 2005]. In this way the user would choose between this language and the UML diagram.

Acknowledgments

This paper is part of the research project “GISCASE: A CASE tool for geoprocessing application development”, sponsored by CNPq (Brazilian National Research Council), in the “Program of Research and Technological Development in Free Software”.

References

Bradley, N. (2002), The XML companion, Addison Wesley, 3rd edition.

Camara, G., Freitas, U., Cordeiro, J. “Towards an Algebra for Geographic Fields”. In Proc. VII Brazilian Symposium on Computer Graphics and Image Processing,Curitiba, 1994. p. 205-212.

Fowler, M. (2000), UML Distilled: a brief guide to the standard object modeling language, Addison Wesley, 2nd edition.

Köesters, G., Pagel, B., Six, H., (1996), “GeoOOA: Object Oriented Analysis for Geographic Information Systems”. In: Proc. ICRE 96 - 2nd International Conference on Requirements Engineering, Colorado.

Martins, T. E. F. (2005), “Um Compilador para a Linguagem TerraMap”. Computer Science Bachelor Degree Diplomation Work, UFRGS, Porto Alegre (in portuguese).

Parent, C. et al. (1998), "Modeling Spatial Data in the MADS Conceptual Model". In: Proc. SDH 98 - International Symposium on Spatial Data Handling, Vancouver, Canada.

Poseidon web site (2005), http://www.gentleware.com.

Rocha, L. V., Edelweiss, N., Iochpe, C. (2001), "GeoFrame-T: A Temporal Conceptual Framework for Data Modeling". In: Proc. 9th ACM GIS, Atlanta, p. 124-129.

Ruschel, C. (2003), Extending the Framework GeoFrame for suporting Geographic Analysis Processes. Porto Alegre: PPGC-UFRGS. Master Degree Dissertation (in portuguese). www.inf.ufrgs.br/~ciochpe.

Ruschel, C., Iochpe, C., Lisboa Filho, J., Rocha, L. V., (2005) "Designing Geographic Analysis Processes on the Basis of the Conceptual Framework GeoFrame". In: Proc. ICEIS 2005 - 7th International Conference On Enterprise Information Systems, Miami, p. 91-97.

TerraLib Oficial Web Documentation (2005), http://www.terralib.org

TerraView Project (2005), http://www.dpi.inpe.br/terraview.

XML Metadata Interchange Specification Version 2.0 (2003). http://www.omg.org/docs/formal/03-05-02.pdf.