Using the Internet to Access Geographic Information: An Open GIS Interface Prototype

Frederico Torres Fonseca

Clodoveu Augusto Davis Jr.

PRODABEL - Empresa de Informática e Informação do Município de Belo Horizonte S.A.
Av. Presidente Carlos Luz, 1275
31230-000 - Belo Horizonte - MG - Brazil
Phone: +55(31)984.5156 - FAX: +55(31)224.0022
e-mail: fred@pbh.gov.br
clodoveu@pbh.gov.br

Summary

Interfaces to access geographic data can be designed in a number of ways. When the design takes into consideration interoperability concerns, the software's architecture becomes one of the main issues. In the interfaces that intend to be interoperable, the design often includes a module that is responsible for the interaction with the user, and another which implements the connection to the GIS. The distribution of tasks between the user interface and the connection module will be discussed here. The design of an interface that allows the user to gain access to geographic data stored in one or more GIS, using the Internet as a communications medium, will be proposed as an alternative.

The architecture of this proposed solution is presented, along with its weak and strong points. In this approach, direct manipulation and menu selection tools are used to build the user interface. The connection to the GIS has been implemented to reach a single source of geographic information, but given the interoperability features included in the interface's design, the potential to access several different GIS is assured. The use of Java, a multiplatform, object-oriented language, and its potential to build geographic interfaces based on an object model, is also discussed.

Objectives

This project has started with the objective of creating "middleware" to allow an user to gain access to geographic data from several different sources, using a simple object-based model. The intention was to provide the widest possible range of users with easy and inexpensive access to public GIS data. The initial concept called for the development of an interface between the user and the GIS. Initially, this interface would only provide basic data access, and should evolve in the future to support the definition and implementation of more specialized services. However, the ability to connect simultaneously to different GIS platforms has been considered important to the success of the project.

As a development platform, the Java language has been chosen. Java includes features that provide for easy client-server computing, Internet connections and standard database access, in a multiplatform, object-oriented environment. Furthermore, using Java allows for the development of highly interactive visual interfaces [Shn92][Mor88], including the possibility of visualizing geographic objects as vectors or images. This can be used to overcome some of the limitations of current GIS interfaces: [ER95] observes that, for a class of systems that has visual characteristics as striking as a GIS, the number of non-visual interfaces is impressive.

Java applets can be made available through the Internet (or any Intranet), cutting down on software maintenance and distribution costs. The Internet also provides an excellent medium for distributed applications, as required by the project's objectives.

GIS Interfaces

[OM96] identify four essential items in an architecture for a GIS interface:

1. The integration of the interface with the GIS;

2. The description of the main architecture modules, specifying their functionality and interoperability;

3. The intermediate representation model and its mapping to the GIS model;

4. The division of tasks between the GIS and the interface.

Each one of these items will be discussed next.

The Integration of the Interface with the GIS

An interface to access geographic data generally is backed by a GIS, and therefore must communicate with it somehow. The classification of integration styles (strong and weak) of the interface to the GIS, proposed by [Voi94], clarifies this point and concludes that weak integration is better, since it ensures GIS independence. [OCM97] observes the independence from the GIS as a feature present in most modern architectures ([Voi91], [AYA+92], [PMP93], [Voi94], [Rig95], [OM95]). That is, the interface is stable even when the GIS is changed, or when an additional GIS becomes available to the user. [Shn92] warns that even small changes in the interface can lead to unpleasant and dangerous mistakes. Greater changes, which are normally what happens when a GIS is replaced, will require heavy training and will disturb the user in many ways [KWM+92]. [OM95] emphasizes other benefits from GIS independence, including the worldwide tendency towards open systems, the improvement on the functionality of each component of the solution (interface and GIS) and also the possibility of progressively adding new services and functions. The necessity for the development of a conversion module between the GIS and the interface can be perceived as an implementation difficulty. This can be complicated, and depending on the GIS the development of low-level communication routines can be necessary. [OM95] also highlights the necessity for a rich data model, representing concepts that are present in several GISes.

In the strong integration, the interface is an integral part of the GIS, and therefore there is a very strong dependence between them. As an advantage, there is the fact that the interface is able to manipulate the objects in the same way they are stored, leading to a greater efficiency in manipulation and exhibition operations. But the disadvantages are great, such as the enormous difficulty in incorporating new GISes to the system, and the dependence on the (not always available, and often not good enough) user interface tools provided by the GIS developer.

Functionality and Interoperability of the Main Architecture Modules

The architectures of the interfaces presented in [Voi94], [OM95] and [PR95] show some similarities. All of them have a user interaction module, a database connection module and a module dedicated to converting geographic objects from GIS' format to the interface's format and back.

The user interaction module generally depends on the operating system, or on operating system tools, in order to be able to control user keyboard or mouse actions. This module also generates the presentation of the results of user queries.

The conversion module translates the data format used by the GIS into the interface's representation format. It is also the responsibility of this module to convert user actions into queries formulated in a language accepted by the GIS. The connection task is also executed by this module, such as in [OM95] and [Voi94]. [OM95] presents a special module to deal specifically with the user's view of the data.

Mapping to the GIS Model

Most of the times, the interface has its own representation model, and each GIS that supports the interface, in case there is more than one, certainly will have its own model. Therefore, it is necessary to create a mapping between the representations contained in the GIS to the respective representations in the interface.

[Voi94] states that the mapping should be made in a higher level, and must allow for the conversion of one representation (GIS) to the other (interface). This must be accomplished through a model that perceives the vision both sides have of the problem. It is also observed that this mapping is not 1 to 1, but *m* to *n*. [Voi95] suggests that this model can be defined using a functional language such as FGD [BP92].

[OM95] presents a module set aside solely for this function, the data model module, which is the responsible for providing the user with a vision of the underlying GIS, according to the data model in use. It is also stressed the idea that the data model should be broad enough to accomodate concepts used in different GISes.

The Division of Tasks Between Interface and GIS

It is very important to be able to clearly separate the functions of the interface itself from the tasks that are the exclusive responsibility of the GIS. Consider the following situation: a set of objects is currently represented in the user's display screen, and he needs to obtain specific data (spatial or not) about an object that is already on the screen. Since it has been displayed, the object could also, depending on the underlying architecture, be present in some intermediate data structure on the memory. The temptation to leave this task to the interface is great, but it must be observed that the responsibility for identifying objects on the screen is the GIS' responsibility, not the interface's. The interface must again request the object that has been identified by, say, the mouse click, to the GIS. Even though this operation should be transparent to the user, it is fundamentally important to establish the limits of the interface's functionality. There is no consensus around this point in the literature. [Voi95] considers that the tendency is to build increasingly faster, more efficient and more intelligent tools, and that as many tasks as possible should be designated to the interface, even though the general lack of standards for the representation and interchange of geographic data constitutes a great impediment to the wide adoption of this solution. On the other hand, [OM96] considers that not having a clear division of tasks generally causes code redundancy, since the interface functions are repeated several times for each type of analysis. Furthermore, this creates difficulty to the maintenance and evolution of the GIS. [Fer92] also observes that the GIS developers are starting to rewrite their code in order to create an isolation between the user interface and the algorithms and processing functions.

Summary

It is clear that the choice of the architecture will strongly influence the ease of development of the interface as well as its interoperability capability. As to the development, it can be emphasized that, in a loosely integrated architecture, several user interface construction tools can be used, resulting in an interface that is more flexible and therefore more adaptable to the user's needs. As to interoperability, a desirable feature in the GISes of the next generation [Cam95], it becomes clear that an architecture that adequately defines the roles of each interface module, as well as the role of the GIS, and also incorporates a representation that is adequate to the user's mental model, has a greate chance of being successful. Such an architecture will isolate the user from the specific implementation problems of the underlying GISes.

The Architecture

A general architecture for the interface will be proposed next. The idea is to create mechanisms to translate user actions into geographic database operations, treating information coming from the database to allow for visualization and other forms of manipulation. These mechanisms assume a clear separation between the part of the system that deals directly with user interaction, and another that communicates with the GIS (Figure 1).

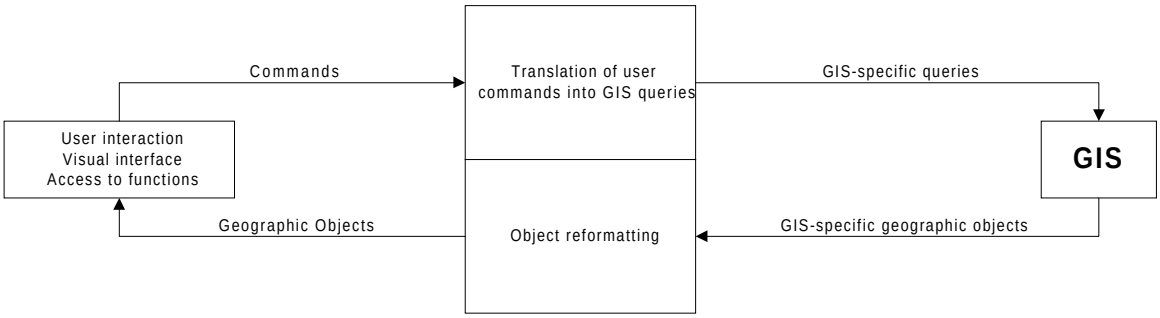

Figure 1 – Conceptual model for the interface

In a conventional database, information can be displayed basically in the same format they are stored, with no need for changes in their structure or abstraction level. GIS, on the other hand, store information in more complex structures, that need to be shaped into a more convenient display format [PR95]. Furthermore, different GISes use different geometric structures to represent objects, thus requiring geographic information retrieval code that must be specifically developed for each GIS. Likewise, there is no single standard for geographic query languages, and a certain degree of translation is required in order to achieve generalized access to GIS data.

The conceptual schema presented in Figure 1 evolved into the physical model presented in Figure 2, with the incorporation of other guidelines for this work: development using Java, in order to accomplish platform independence, and the access to the geographic database through the Internet. Therefore, in the physical model it was decided that the interface would be implemented with two modules. The first one, called **Manipulator**, is the responsible for direct user interaction. This module will receive commands from the user, and will send these commands to the other module, called **Extractor**. The latter will execute all necessary communication with the GIS, and will return any retrieved objects to the **Manipulator**, which in turn will present them to the user. The objects that are returned from the **Extractor** are standardized, in a way that is similar to the object model that is defined by the OpenGIS Consortium [MB96]. This standard is independent from the GIS that is used to manage the geographic data, and it is the **Extractor**'s task to implement methods that will interpret and decode information retrieved from the GIS, and to rearrange everything into the standard object format. With this, the **Manipulator** becomes completely independent from the GIS from which objects are being extracted, while the **Extractor** needs to be specifically developed to interact with a specific GIS. In this conception, the **Extractor** functions as a filtering and standardizing layer, acting on data coming from or going to the GIS. Extending this concept a bit further, it becomes feasible to access several geographic databases, either separately or simultaneously, without the need to inform neither the **Manipulator** nor the user. This effectively incorporates to the architecture the possibility of interoperating different GISes, since the user gets indistinguishable, simultaneous access to information maintained by each one of them.

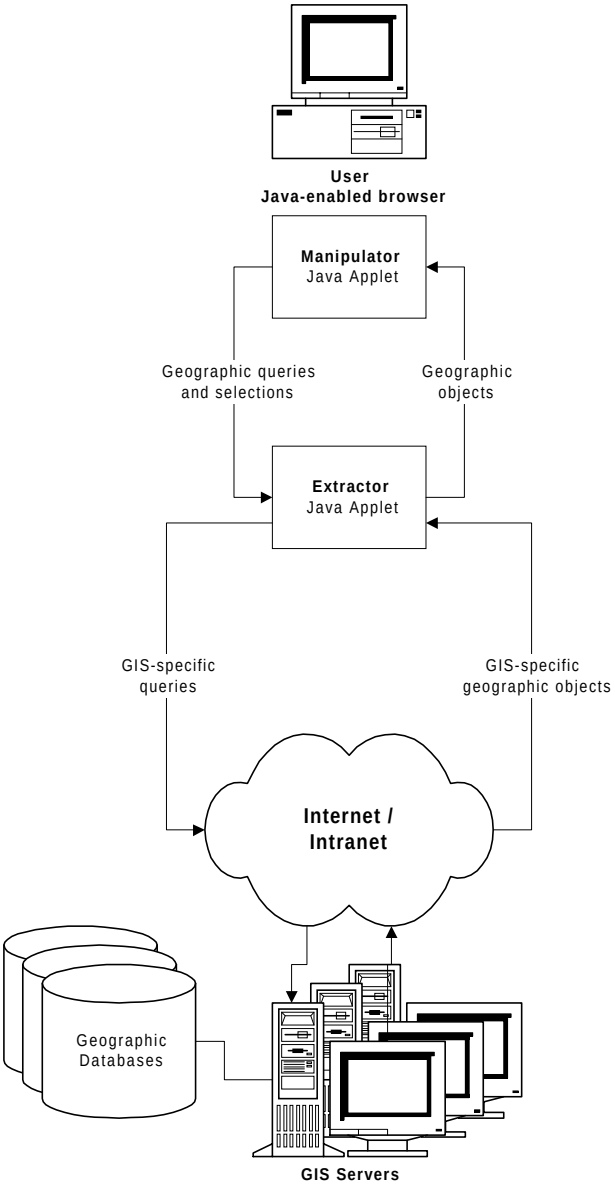

Figure 2 – Physical model for the interface

According to the classification proposed in [Voi94], notice that the **Manipulator** is loosely coupled to the GIS, since it is isolated from the GIS by the **Extractor**. Now, the Extractor is strongly coupled to the GIS, since it needs to know details about the way the GIS works, in order to fulfill the translation of commands and the reformatting of objects. With this, it becomes possible to combine the advantages of each coupling model, while preserving the desired isolation between user interface and GIS retrieval code.

It is also important to notice that in the proposed architecture, the interface does not take over geoprocessing tasks, that are the responsibility of the GIS, and which belong to the GIS' functionality. These functions can be accessed through the mechanisms already proposed, keeping the interface simple and compact. Considering that the intention is to use the Internet as a medium to access the GIS, even with all its congestion and response time problems, it is fundamentally important to avoid the execution of spatial processing and information retrieval tasks by the interface (which is a really great temptation, considering the great processing power and memory capacity of today's average client microcomputer). This option may lead to some

performance loss, but it can be counterbalanced by the advantages that emerge from the stability and independence of the interface.

The high interoperability potential of this solution could only be achieved using the object-oriented software development principles recommended by [Mey88]. That article indicates that routines containing an oversized share of knowledge about the overall system have a very high potential for producing problems. If this happens, adding any new type, or modifying an already existent type, can have an impact over each routine in the system. [Mey88] quotes "*Ne sutor ultra crepidam*, the shoemaker should not look into anything beyond the shoe. This is one of the software project principles." As an example, observe that a routine that performs a rotation on objects of the type *Shape* does not need to know anything about the classes that derive from *Shape*. This principle becomes clear in the geographic object drawing routine that will be described in detail further ahead. That routine can be understood as the capability for each object to draw itself on the screen. The implementation is specific for each type of geographic object, following the method established by the superclass that defines geographic objects.

The Extractor

The **Extractor** module has been developed as a Java class. This class is the responsible for extracting geographic objects from the GIS, sending them over to the **Manipulator**. Since the object formats used by the GIS and by the **Manipulator** differ, this class is also responsible for converting objects from the GIS into the **Manipulator**'s format.

As implemented, the **Extractor** is totally dependent on the source of data, and it must be rewritten, modified or specialized for each update that occurs on the GIS that is coupled to the system. Its implementation can be extended to provide access to more than one GIS, using a set of parameters that indicate from which GIS each layer of information should be retrieved. So, when called by the **Manipulator**, the **Extractor** will return a instance of the desired class according to the parameters. Let's say that the **Manipulator** needs a layer that is stored on `GIS_03`. The parameters will be the name of the layer and the number of the GIS. Both are part of the parameter set. According to these parameters, **Extractor** will return to **Manipulator** a class that is derived from itself, `Extractor_03`, that knows how to deal with `GIS_03` and will provide all the service that **Manipulator** needs. Since `Extractor_03` is a specialization of **Extractor**, it provides all the services that its superclass provides, and therefore **Manipulator** doesn't have to know if it is dealing with **Extractor** itself or one of its subclasses.

General Operation

The Extractor needs to obtain some information before initiating its operation:

- What is the current geographic space, defined as a rectangle using two geographic points, and corresponding to the current zoom window limits;
- What are the currently activated layers, that is, layers that will be visible.

The first information can be retrieved from the class `geographic_space`, that contains all information on the coordinates domain, as well as the definition of the current geographic space. The second information is kept by the **Manipulator**, and will be sent over to the **Extractor** whenever geographic object retrieval services are requested.

In order to execute fundamental operations, such as zooming, two different kinds of selection are combined: a spatial selection of objects within or over the border of a given rectangle, and a selection of objects that belong to one of the activated layers.

The **Extractor** implements the following selection services:

- retrieving and sending all geographic objects from all requested layers within a given rectangle;

- given a coordinate pair, retrieve all objects from the currently active layers that are located on that point or that contain that point;

- given an alphanumeric identifier (see geographic object definition), retrieve the associated geographic object.

Caching

When the Internet is involved, one should always think of mechanisms to avoid excessive network traffic. The proposed architecture facilitates the implementation of local buffers or caches that will increase the performance of the application. The necessity of these buffers at the client side of an Internet application is mentioned in [Cam95]. [OCM97] mentions that this task is usually performed by the DBMS, but in the case of a GIS application it should be performed by the interface.

Since extracting geographic objects from the database is the **Extractor**'s task, therefore it should also be responsible for caching. The factors that can influence the architecture of such cache are many, among them the storage capacity of the client machine and the probability of repeated request of objects, based on proximity or relationship to already retrieved objects. A very simple cache, based only on proximity, and taking advantage of the spatial indexing logic, was implemented. The important thing here is to demonstrate the necessity of this kind of service, and to show how it can be implemented without further impact on the **Manipulator,** while preserving the interoperability potential of the proposed architecture. As a result, we achieve the benefits of weak coupling, keeping the **Manipulator** unaware of the data retrieval mechanisms, and the benefits of strong coupling, since the **Extractor** can benefit from its specific knowledge of the GIS to implement the caching mechanism.

The Manipulator

The **Manipulator** is responsible for the interaction with the user. Through the **Manipulator** the user will be able to execute operations like object selection, zooming, searching, thematic mapping, and so on. The user can double-click a mouse button and select a geographic object, and can also use the mouse to select a menu command or perform a zoom.

The user requests are sent by the **Manipulator** to the **Extractor**. The **Extractor** translates the requests to a format that the GIS can understand. When the **Extractor** receives the answer from the GIS, it prepares from it a response to the **Manipulator**, in the form of geographic objects. These objects are members of a class called `geographic_object`. The exhibition of the results will be produced by the **Manipulator**, using the methods available within the `geographic_object` class. As a result, the **Manipulator** code needs to deal only with user interaction and object display.

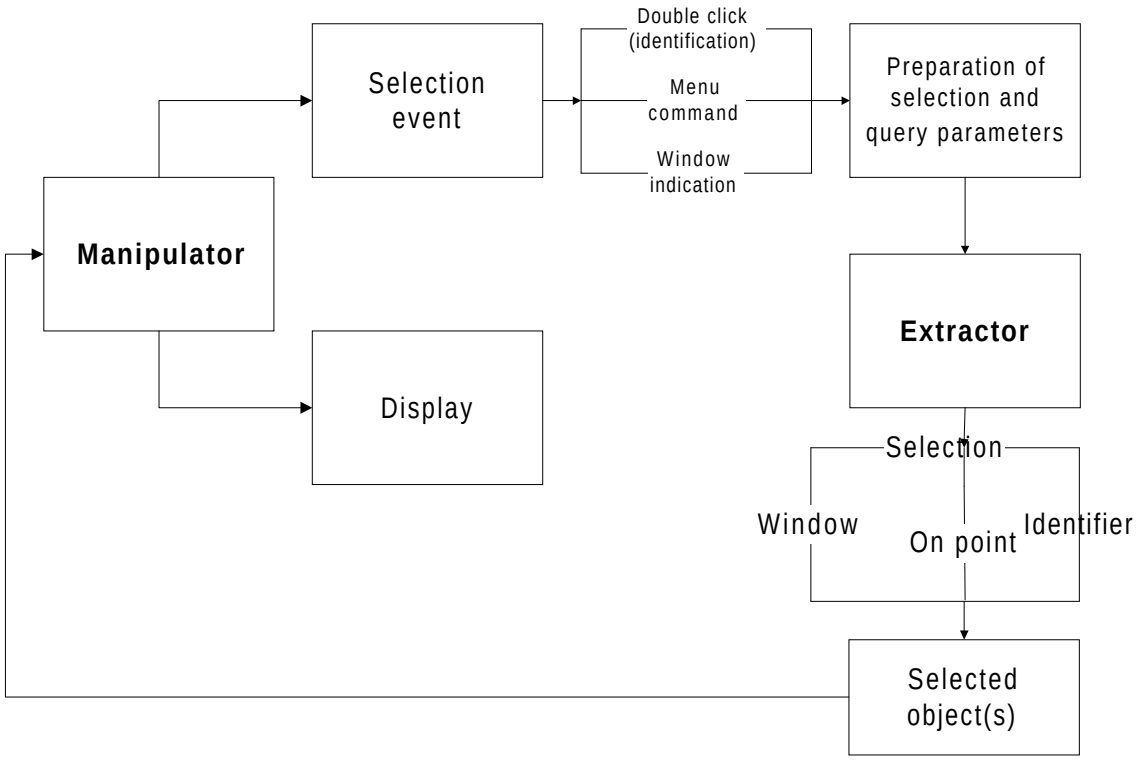

Figura 3 - General Manipulator working schema

The clear distinction between the tasks of the Manipulator (user interaction and exhibition of results) and the extractor (communication between the GIS and the Manipulator) follows one of the basic principles mentioned in [OM95]: a greater degree of specialization of the software modules will facilitate the integration of new modules in the future, and the progressive incorporation of additional services and functions.

OpenGIS and Geographic Objects

Standards facilitate the sharing, integration and transference of data. Although the most common standards are the ones directed to data transfer, there are others that regard specification languages and metadata [CCH+96]. One of these standards is under development by the OpenGIS consortium. The OpenGIS data schema defines two basic types of data: *feature* and *coverage*. These types correspond to the ones described by [Cou92] and [CFS94]: geo-fields and geo-objects. We will focus here on the *feature* type (geo-object), that corresponds to the geographic objects that have been implemented. It contains three components that are specified in [MB96]:

- **geometry:** a description of the geometry of the phenomenon with an associated spatial and/or temporal reference model;
- **semantics:** a description of how the phenomenon is defined in the lexicon of a particular group;
- **metadata:** contains information needed to position the phenomenon in the context of the application environment user community.

The geographic object actually implemented follows partially the OpenGIS guidelines. Some simplifications were made, such as making use of a unique reference system and disregarding a temporal reference system.

The geographic objects are implemented as an abstract class called `geographic_object`, reflecting only OpenGIS *features*, and saving the implementation of *coverages* for later. To do this the class `geographic_object` was specialized as `vectorial_geographic_object`. The latter was further specialized in three new classes: `Polygon`, `Line`, and `Point`. Basic services, provided by the `geographic_object` class, include methods that enable a geographic object to draw itself on the screen, to draw its identifier on the screen and to return a list of its attributes. These methods are only declared in the abstract class, and are actually implemented by the subclasses.

When a new GIS is made available to the interface, `geographic_object` needs to be specialized, and the defined methods must be implemented. The `extractor` class must also be specialized to communicate with the new GIS. Of course, if any of the available classes can fulfill the needs of the new GIS, they can be used.

So, we can see that the integration between the **Manipulator** and the **Extractor** is achieved through the use of geographic objects. The **Manipulator** just has to know how to work with them, or one of the subclasses. Thus, interoperability is achieved and the stability of the interface (the manipulator) is assured.

Conclusion

Results achieved with the initial implementation of the above described interface indicate that the proposed architecture, even though it carries the burden of translating data between the GIS and the user interface, has benefits that overcome its limitations. Among the main benefits, we can mention the adherence to open systems concepts, the concentration on the basic functional roles of each component (**Manipulator**, **Extractor** and GIS), and the possibility to progressively incorporate new services and functions. Furthermore, the strong coupling between the **Extractor** and the GIS brings the opportunity for optimization and performance gains, due to specific knowledge of the GIS internal routines, while respecting the basic interoperability rules through the weak coupling between the **Manipulator** and the **Extrator**.

References

[AYA+92] Abel, D., Yap, S., Ackland, R., Cameron, M., Smith, D. e Walker, G., “Environmental Decision Support System Project: an Exploration of Alternative Architectures for Geographical Information Systems”, in International Journal of Geographical Information Systems, 6 (3), 1992.

[BP92] Breunig, M. e Perkhoff, A., “Data and System Integration for Geoscientific Data”, in Proceedings of the International Symposium on Spatial Data Handling, 1992.

[Cam95] Câmara, G., “Models, Languages and Architectures for Geographic Databases”, Ph.D. Thesis, National Institute of Spatial Research (INPE), 1995 (in portuguese).

[CCH+96] Câmara, G., Casanova, M.A., Hemerly, A.S., Magalhães, G.C., and Medeiros, C.B., “Anatomy of Geographic Information Systems”, Computing Institute, University of Campinas, 1996 (in portuguese).

[CFS+94] Câmara, G., Freitas, U., Souza, R., Casanova, M., Hemerly, A.,e Medeiros, C., “A model to cultivate objects and manipulate fields”, in Proceedings of the Second ACM Workshop on Advances in GIS, Gaithersburg, Maryland, USA, 1994.

[Cou92] Couclelis, H., “People Manipulate Objects (but Cultivate Fields): Beyond the *Raster*-Vector Debate in GIS” in Proceedings of International Conference on GIS – From Space to

Territory: Theories and Methods of Spatial Reasoning, Springer Verlag Notes in Computer Science 639, 1992.

[ER95] Egenhofer, M., J. and Richards, J., "Visual Map Algebra: a Direct-Manipulation User Interface for GIS.", Visual Database Systems 3, Visual Information Management, Proceedings of Third IFIP 2.6 Working Conference on Visual Database Systems, Spaccapietra, S. and Jain, R., eds, London, Chapman & Hall, 1995.

[Fer92] Ferreira, J., "User Interfaces for Geographic Information Systems", in [KWM+92], 1992.

[KWM+92] Kuhn, W. and Willauer, L. and Mark , D. M. and Frank, A. U. (Eds), "User Interfaces for Geographic Information Systems: Discussions at the Specialist Meeting", National Center for Geographic Information and Analysis, 1992.

[MB96] McKee, L. e Buehler, K. (eds), "The Open GIS Guide", in http://www.opengis.org/guide, Open GIS Consortium, Inc., Massachusetts, 1996.

[Mey88] Meyer, B., "Object Oriented Software Construction", New York, NY, Prentice Hall, 1988.

[Mor88] Morris, B., "CARTO-NET: Graphic Retrieval and Management in an Automated Map Library", Special Libraries Association, Geography and Map Division Bulletin, 1988.

[OCM97] Oliveira, J. L., Cereja, N. and Medeiros, C. B., "Intermediate Interface Model for Geographic Information Systems", in Proceedings of GIS Brasil 97, 1997 (in portuguese).

[OM95] Oliveira, J. L. and Medeiros, C. B., "A Direct Manipulation User Interface for Querying Geographic Databases", Technical Report DCC-95-08, Computer Science Department, University of Campinas, 1995.

[OM96] Oliveira, J. L., Medeiros, C. B., "User Interface Issues in Geographic Information Systems", Technical Report IC-96-06, Computer Science Department, University of Campinas, 1996.

[PMP93] Pissinou, N., Makki, K. e Park, E., "Towards the Design and Development of a New Architecture for Geographic Information Systems", in Proceedings 2nd International Conference on Information and Knowledge Management, USA, 1993.

[Rig95] Rigaux, P., "Interfaces Graphiques pour Bases de Données Spatiales: Application à la Représentation Multiple", PhD thesis, CEDRIC - Conservatoire National des Arts et Metiers, 1995.

[Shn92] Shneiderman, B., "Designing the User Interface: Strategies for Effective Human-computer Interaction", New York, NY, Addison-Wesley Publishing, 1992.

[Voi91] Voisard, A., "Towards a Toolbox for Geographic User Interfaces" in Design and Implementations of Very Large Databases (SSD), Zurich, 1991.

[Voi94] Voisard, A., "Designing and Integrating User Interfaces of Geographic Database Applications", in Proceedings of 1994 ACM Workshop on Advanced Visual Interfaces, pp 133-142, 1994.

[Voi95] Voisard, A., "Mapgets: A Tool for Visualizing and Querying Geographic Information", in Journal of Visual Languages and Computing, 1995

Acknowledgements

The authors would like to thank to Prodabel, that supported Frederico T. Fonseca Master Degree program work, that resulted in the development of the interface described here, and also the Computer Science Department of the Federal University of Minas Gerais that provided funds for the presentation of this paper.