TOWARDS A VISUAL LANGUAGE
TO IMPROVE USER ACCESS TO
GEOGRAPHICAL INFORMATION SYSTEM
FUNCTIONALITIES

Internal Report C91/07

February 20, 1991

I. Campari, F. Paternò,
R. Scopigno
Towards a Visual Language To Improve User Access
To Geographic Information System Functionalities

I. Campari, F. Paternó, R. Scopigno

Istituto CNUCE
Consiglio Nazionale delle Ricerche
Via S. Maria, 36
56100 Pisa, Italy
Send correspondence to:
Roberto Scopigno
Phone +39 50 593211
E-mail scop@icnucevm.cnuce.cnr.it
Fax +39 50 576751

February 15, 1991
Abstract

In this paper an environment to specify, analyze and execute visual programs for Geographical Information Systems, NetGIS, is described. The lack of sophisticated user interfaces is one of the major drawbacks to Geographical Information Systems (GIS), particularly for people without a sound background in computer science. The use of a visual language approach is proposed here in order to hide the plethora of basic GIS functions while providing ready-to-use tools to solve user's sub-tasks.

NetGIS is based on the module concept, conceived as a software building block that implements a solution to a general subtask and is presented to the user through a structured icon. Complex GIS queries or tasks can be carried out by interconnecting modules into flow networks, using a direct manipulation approach.

1 Introduction

The need for up-to-date user interfaces applied to general purpose Geographic Information Systems (GIS) is a direct consequence of the intrinsic character of these tools. GISs have been developed to satisfy multiple applications in various disciplinary contexts characterized by the common pressure to handle efficiently spatial data [1]. Geographers, geologists, demographers, planners, etc., all need similar tools for spatial data handling, but often they lack computer science knowledge and then they cannot give that innovative step ahead in their activities. Whereas in the past decade GIS development was driven on by the need to extend the functionality and the user interface (UI) was merely considered as an esoteric aspect of the whole system, now usability issues are gaining much attention in the area of GIS [2].

The use of a spatial data base "via" a GIS requires much skill when moving from the conceptual process to the technical procedures of representation. The user activity generally originates from a particular knowledge goal, i.e. a certain type of geo-analysis. The user has to conceive the sequence of sub-tasks which conceptually contribute to fulfilling the goal; then each sub-task must be dealt with by defining the correct sequence of GIS functions. This task-to-functions process is crucial and requires a deep knowledge of how the GIS structures spatial data and how it is possible to access, integrate and visualize these data. The wide variety of data types recorded in digital maps, the complex data structures used to organize them and the range of functionalities available all contribute to increasing the complexity of defining the GIS function path solving a required task.
Direct handling data is becoming increasingly important in all areas of GIS. Consequently the barrier that computer science and the complex function-oriented design of GIS impose needs to be overcome. Technical procedures of spatial data handling must therefore be made easier, whereas the basic conceptualization activities are still the application specific expert’s responsibility. A sophisticated graphical user interface (GUI) can greatly reduce some problems, by creating the bridge to join the geo-scientist user’s needs to the computer-based instrument.

User interfaces have substantially evolved in the last few years. The great improvement in hardware/software visual technology brought dramatic changes to the way in which users interact with programming environments and applications [3]. The old approach based on single screen and command-typing has been largely replaced by a multi-windows, menu, icons and mouse approach, where the direct manipulation of a synthetic and metaphorical world greatly helps the user to build the correct conceptual model of the functionality and the behaviour of the application. The use of metaphors based on gestural input and manipulation of icons can affect the capacity of the user to acquire or infer the conceptual meaning associated with each function or action of the system more simply than dealing with bytes, files and other computer-related concepts.

All these concepts and paradigms are common to a large number of user interfaces, especially in the area of personal or workstation based applications. A substantial effort has been made to introduce these concepts into the GIS area, either as research proposals or into current or forthcoming versions of commercial systems. Among the former, Frank suggested a way of defining a flexible and easy-to-use query language, conceived as a tool for general interaction with spatial data [4]. Recently there have been many papers on the extension of query languages to geographic data processing and considering user interface issues [5] [6] [7] [8]. Rhind, Raper and Green recently proposed UGIX [9], a layered-based model of user interaction with GIS. This proposal highlights the tasks orientation, while the interface to the GIS is based on a spatially extended version of SQL.

However, the design of GIS UIs involves considerably more than the dialog improvements that are generally supported by a basic GUI. A traditional GUI only makes a translation of the command syntax into graphical cues, leaving the function-orientation of the system unchanged. However, in our opinion, reducing the intrinsic complexity of the task-to-function process should be one of the main points: simply substituting of the textual representation of the command language with a graphical representation may not be enough, in fact a global modification of the user-system dialogue may be needed.
In Section 2 we propose the use of a visual language approach as a support in the Task-to-Function process. The visual language, NetGIS, is presented in detail in Section 3: the language is specified and then the visual environment is described. Section 4 shows some phases of a work session. Finally, some concluding remarks are reported in Section 5.

2 Visual Languages as a Support in the Task-to-Function Process

When the user is not a computer professional, a critical issue is how to support the specification of the procedural skeleton solving the user’s inquiries [10]. GIS user’s inquiries often cannot be fulfilled by using a single functionality or command of the system, but involve the choice of a set of cooperating commands, i.e. the definition of the composition of functions required to achieve the particular task. The definition of the procedural task may become critical for people who are not used to giving solutions in algorithmical terms, as is the case with most interdisciplinary GIS users. Therefore, a critical point is the current lack of tools that can help the user to specify the procedural flow. Most of the tools generally provided by current commercial GIS are command languages interpreters, showing again GIS orientation towards computer or programming aware users. The "on fly" specification of an algorithm, while interacting with a complex couple such as computer and GIS, is thus a task that often discourages the naive non-programmer GIS user. Programming expertise must not be a must while using a GIS to solve a non-routine task, and nor is the collaboration with a "programmer", conceived as a computer-aware interpreter between the GIS system and the user, the right solution. In the latter case, the lack of a common language between the "geographer" and the programmer could cause misunderstandings or incorrect use, from a geographical point of view, of the system. The availability of tools for "on fly" design of the procedural flows should be one of the key research themes in the design of UI to GISs.

A similar need for tools arose in the area of software engineering and programming language environments, where visual editing environments have been developed to support the programmer, making it simpler to write syntactically correct and more readable programs [11]. The first step was to define a template-oriented approach to the design of syntax-driven editors. The use of a visual approach leads to such an extreme that the entire program could be codified
by visual techniques alone, discarding the textual version in favour of a pictorial or diagrammatic representation. In this approach, the interaction with a two-dimensional graphic notation replaces the conventional textual specification and editing of the program.

There is currently much interest in Visual Programming and Visual Languages as they seem a powerful tool for people who are not expert in programming to access to the functionalities of an application. There have been proposals to categorize this kind of language into different classes [12]: languages for visual programming using a visual representation for language constructs and for objects not inherently visual that will appear to the user by graphical representations; iconic languages to process visual information (i.e. icons, pixmap or images); languages to support visual interaction with a linear representation of language constructs and visual representation for not inherently visual objects manipulated; languages to process visual information, textual languages used to manipulate images and similar data, common in the field of image processing, office automation and robotics.

The main developments in the field of visual programming fall into two classes. On the one hand there is the creation of visual environments to specify and debug programs, more generally in the software engineering field. On the other hand there is the evolution of languages to manipulate visual information, to support visual interaction and to program with visual expressions. In the second field, there have already been some proposals to use visual programming to specify user interfaces or to interact by visual expressions with systems of different types. In the Computer Graphics and Visualization field, ConMan [13] has been proposed as a connection manager which develops dynamic connections between predefined modules that are visually represented on the screen and that support different types of interaction and graphic visualization. A similar approach has been also used in the design of the AVS [14] and apE [15] systems. In a different application field but with the same philosophy, VILD has been defined as a visual language for the access to an object-oriented multi-media database system [16]. The same approach can be also used to deal with GIS usability issues previously highlighted.

If the target is to overcome the use of a textual command language, the visual programming approach lets users take advantage of the functionality of the system without having to become a programming specialists each time they encounter a new problem [17].

We propose to hide the canonical GIS command language by using a visual language based on modules building blocks connected in procedural flow diagrams. The module is defined as an elementary software component, implementing a medium level functionality and characterized by the stream of input and output data. Modules are represented on the screen by icons and
the user can interactively build flow networks by connecting the output ports of some modules to the input ports of other modules. Using a data-flow metaphor the user builds an application by choosing modules from the set of previously defined modules and interconnecting them into a direct acyclic graph, whose nodes are the functional modules and whose arcs are I/O port interconnections.

The use of diagrams to represent computational flow is widely accepted and with current graphical technologies effective and efficient iconic diagram editors and interpreters can be developed. Diagrams are only considered to be effective when they represent the relationships between a limited number of elements, in other words, when they give a high level representation of a procedural flow. This is a well-known limitation of the diagram as representation medium. Modules are conceived here as a higher abstraction level entity than the low level GIS commands. The module, internally defined as a procedure of GIS commands, implements a meta-function belonging to the user conceptual model of a generical GIS rather than a simple graphical representation of a canonical GIS command.

3 The NetGIS Enviroment
NetGIS provides a visual environment for the specification, analysis and execution of GIS tasks. The language has been built on some basic entities described following an object-oriented approach [18]: the module, the link and the network.

The architecture of the NetGIS system is described in Fig. 1. It is based on a graphical editor which allows the user to specify an expression of the visual language by direct manipulation techniques. The editor then checks the syntax of the visual program specified by the user and interacts with a generical traditional GIS to execute it; the GIS results will appear on the screen in GIS-managed windows.

The module represents a meta-function, or task, built as a sequence of commands of the GIS. It is characterized by one or more input and output data flows. The module is conceived as a building block by which the abstraction level of the GIS interface can be increased. Traditional GIS interfaces are closer to the application implementation than to the user conceptual world. Such low level interfaces allow more flexible specifications but are usually complex. NetGIS, on the other hand, allows users to specify their requests in a way closer to their own conceptual model of the GIS. The module is therefore conceived as a basic processing element and it is visualized by an interactive iconic representation, the frame (Fig. 2). The frame, a rectangular area, represents the class which the module belongs to (name and icon fields) and contains a set of interactive fields. One or more interactive fields represent the communication ports, by which the module receives input data or sends results; each of these I/O ports has a specific data type associated with it. Other interactive fields are mover, info and more: the first changes the module position on the screen, the second allows the user to ask for information on the module and its related task, the third is used to select the parameter values associated with the
Figure 3: An example of a network.

instantiated module. The transformation of the input data operated by a module may depend on the actual setting of a parameter list; the user selects the more field to display the module parameters dialog panel, in order to modify or simply to show the parameter default/current values.

A link represents a connection among two modules; it introduces a logic and temporal ordering among modules and determines the visual adjacency. A link is visualized as a line connecting I/O ports. Links are interactively created by the user; they can be selected and deleted. If a module is moved around all the links connected to it are redrawn automatically. There are two types of links: point-to-point and broadcast, the first indicates a communication between two modules, the second shares out the output of a module to a set of modules.

The network is a valid composition of modules and links forming a directed acyclic graph. As an example, the network in Fig. 3 selects two input maps \((map_1, map_2)\), using two set modules which transmit these selected maps to the following modules; the buffer function is applied to \(map_2\) and then an overlay module overlaps \(map_1\) and the buffered \(map_2\) obtaining the final map that is visualized on the display (display module) while the report module stores some results of interest in a different form.
3.1 Language definition

Two approaches have been investigated for the formal definition of NetGIS. In the first one, the language is defined using traditional textual grammars, and the expressions of the textual syntax are mapped to expressions of the graphical syntax. The second uses a Picture Layout Grammar, a specific grammar type which can directly define the graphical syntax of the language.

In the first approach (Fig. 4) the grammar for textual NetGIS is defined as a common BNF grammar. The grammar is defined with the usual 4-tuple \( \{ N, T, S, P \} \) with \( N \) the set of non-terminal symbols, \( T \) the set of terminal symbols, \( S \) the initial symbol and \( P \) the set of derivation rules. The non terminal symbol \( M_{ij} \) represents modules with \( i \) input ports and \( j \) output ports; a terminal \( l_o \) is associated with each link connected to an output port, and analogously \( l_i \) is associated with a link connected to an input port; the sequence \( l_i l_o \) therefore represents a link that joins two modules. The terminals \( set, edit, topology \) and so on, represent specific VGIS modules.
\[ \mathcal{N} = \{ \text{IN, IN', M_{01}, M_{10}, M_{11}, M_{21}, OUT, OUT', PROG, PROG'} \} \]
\[ \mathcal{T} = \{ \text{set, edit, topology, buffer, near, overlay, display, report, l_i, l_o, (), [ ], ||} \} \]
\[ \mathcal{S} = \text{PROG} \]
\[ \mathcal{P} = \{ \begin{array}{l} 0. \text{PROG} \rightarrow (\text{IN}) \text{OUT}; \\
1. \text{IN} \rightarrow (\text{IN})_i M_{11} l_o; \\
2. \text{IN} \rightarrow (\text{IN})_i M_{11} l_o; \\
3. \text{IN} \rightarrow M_{01} l_o; \\
4. \text{OUT} \rightarrow [(\text{PROG'}) || \text{OUT'}]; \\
5. \text{OUT} \rightarrow l_i M_{10}; \\
6. \text{PROG'} \rightarrow (\text{IN'}) \text{OUT}; \\
7. \text{OUT'} \rightarrow (\text{PROG'}) || \text{OUT'}; \\
8. \text{OUT'} \rightarrow (\text{PROG'}); \\
9. \text{IN'} \rightarrow (\text{IN'})_i M_{11} l_o; \\
10. \text{IN'} \rightarrow (\text{IN'})_i (\text{IN})_i M_{21} l_o; \\
11. \text{IN'} \rightarrow l_o; \\
12. M_{01} \rightarrow \text{set}; \\
13. M_{11} \rightarrow \text{edit} \mid \text{topology} \mid \text{buffer}; \\
14. M_{21} \rightarrow l_i \text{overlay} \mid l_i \text{near}; \\
15. M_{10} \rightarrow \text{display} \mid \text{report}; \end{array} \}
\]

The syntax analysis process is described in Fig. 4. At first, a mapping process converts a visual program into textual format. The textual string is built by replacing each module with the associated terminal (i.e. set, edit, topology, etc. terminals); the network is linearized by using an algorithm that visits the graph in left-to-right, top-down order. The textual string obtained by the mapping function applied on the network in Fig. 4 is:

\[ S = (((\text{set } l_o)_i \text{ buffer } l_o), (\text{set } l_o)_i l_i \text{ overlay } l_o) || (l_o)_i \text{ report}) || (l_o)_i l_i \text{ display}).] \]

A direct approach was also used to define the NetGIS syntax. It is based on Picture Layout Grammars (PLG), a particular type of grammar introduced by Golin and Reiss [19] and based on the Attributed Multiset Grammars (AMG). A multiset is a set where elements can be repeated and the AMG is similar to a context-free grammar but the right-hand side of the productions are considered to be unordered collections of symbols, rather than strings. Each AML production is a triple \((R, S, C)\), where \(R\) is a rewrite rule, \(S\) is a semantic function which computes the attribute values of the left-hand side from the attributes values of the right-hand
side, $C$ is a constraint defined over the right-hand side attributes which indicates when the production can be applied.

*PLG* are defined as AMG where a production corresponds to a *picture composition operator* and the attributes represent spatial information on a picture element (terminal or aggregate). A sample PLG production is therefore:

\[
A \rightarrow \quad Op(B, C) \quad \text{(rewrite rule)}
\]

\[
A.attrib = Func_{Op}(B.attrib, C.attrib) \quad \text{(semantic function)}
\]

\[
\text{where:} \quad Pred_{Op}(b.attrib, C.attrib) \quad \text{(constraint)}
\]

Terminal symbols of the NetGIS PLG are the basic graphical elements of NetGIS (links and modules); the attributes are defined on each element, to describe its pictorial and spatial aspect, and the productions use spatial operators to describe element composition.

Each NetGIS PLG symbol (both terminal and nonterminal) has a set of spatial attributes (Fig. 5). Module symbol attributes are two points that bound the rectangular area of the frame, the position and the type of each I/O port. A link symbol has attributes that describe its end points' spatial position.

A set of operators working over NetGIS symbols was defined:

- **TouchesLeft**(B, C) where B is a link, C is a network and $(r_x(C) = init_x(B)) \land (b_y(C) < end_y(B) < t_y(C))$;
- **PointsTo**(B, C) where B is a link and C is a network and $(l_x(C) = end_x(B)) \land (b_y(C) < end_y(B) < t_y(C))$. 

---

10
Figure 6: Evaluation of operators.

- AdjacentTo(B,C) where B,C are networks;
- Contains(B,C) where B,C are networks and \((l_x(B) < l_x(C)) \land (r_x(C) < r_x(B)) \land (b_y(B) < b_y(C)) \land (t_y(B) > t_y(C))\);

In Fig. 5 these operators are evaluated on some terminal and composite symbols.

An example of a production from the NetGIS PLG, defined in [20], is:

\[
LINK \rightarrow \text{ PointTo(LINK}_0,\text{ MODULE});
\]

\[
\begin{align*}
\text{LINK}.\text{OutPortType} &= \text{LINK}_0.\text{OutPortType}; \\
\text{LINK}.\text{RightEnd} &= \text{LINK}_0.\text{RightEnd}; \\
\text{LINK}.\text{InPortType} &= \text{LINK}_0.\text{InPortType}; \\
\text{LINK}.\text{InPortCoord} &= \text{LINK}_0.\text{InPortCoord};
\end{align*}
\]

where \(\text{CheckConnection(LINK}_0,\text{ MODULE)\)}

Comparing the two approaches to visual syntax description is not easy. Picture Layout Grammars facilitate a more formal definition of visual language syntax as a whole, without requiring mapping functions which generally embody most of the visual cues of the language.

One aspect that is not covered by either of the approaches is how to cope with the possible dynamic aspect of a visual language (VL). Interactive VLSs can be defined as languages which allow programs with an interactive specification, i.e. programs whose syntactic content can not be completely described by a static visual representation of the program itself. NetGIS resembles such an interactive VL if we look to the module parameter setting and how it works for. Although the actual parameter values are not directly represented their values affect the syntactical correctness of the network and its associated semantics. The user, using the visual
editor, can assign values to the parameters in an interactive way, by selecting the more field of a module frame and working with the dynamically visualized dialog window. Interactively hiding/showing part of a specification could be conceived of as an important potentiality of VL (or, at least, of the supporting VL environments) and this would be difficult to describe using the consolidated syntax description techniques.

3.2 The visual editor

NetGIS provides a graphical environment to specify visual programs and to control their executions. The system supports:

- graphical editing, with direct manipulation specification of NetGIS programs; the editing tool is syntax-driven and so it makes impossible to build pictures which do not belong to NetGIS;
- storing and retrieving the specified NetGIS programs on/from secondary storage;
- interpretation of NetGIS programs; a textual representation of the visual expression is derived from the picture; it is checked against static semantic requirements and then executed on the underlying GIS;
- help functionalities, to guide the user and to allow self-explanation of the system functionalities.

A NetGIS program consists of a network composed of modules organized in a direct acyclic graph, and therefore only a partial ordering between modules is expressed in the visual program. While the user specifies the network, the editor visualizes the current network, manages a data structure that represents the network and performs syntactic checks such as compatible data types of connected I/O ports and parameter settings. The textual string can be later simply inferred from this data structure, providing a linear string of elements with a total ordering; this ordering could be followed in the execution phase as well. The graphical to textual mapping is therefore performed during the network editing transparently to the user. Further syntactical controls are applied before the execution on the textual string to detect other errors, e.g. the missed definition of a link on a I/O module port.

The visual editor has been implemented using a X toolkit, XView [21]. The visual editor layout is composed of three areas (Figure 7):
Figure 7: NetGIS editor layout.

- a horizontal pull-down menu that specifies the main commands (File, Edit, Run, Help, etc.);
- a vertical menu where the link and the module classes available to the user are represented by interactive icons;
- a work area where the user can specify NetGIS programs by direct manipulation.

The network specification proceeds by subsequent instantiations of modules and links. The following set of available modules defined to obtain a (not exhaustive) subset of G.I.S. sub-tasks is:

- Set, for interactive selection of a map in the geo-cartographical data base.
- Edit, is used to modify or to create a map.
- Topology, generates the correct topology of the flat data received in input.
- Buffer, generates the buffer areas for the elements in the input map.
- Near, computes the distance between each point in the first map and the nearest line or point in the second input map.
• Overlay, builds the overlapping between maps.

• Display, visualizes the carto-thematic values of a map on the screen.

• Report, builds, and prints on request, a report file on the contents of the map.

Selecting one of the icons in the vertical menu creates a new instance of a module and visualizes it on the work area, at a position chosen by the user. The instantiation of a new link works in the same way: the user firstly selects the link icon on the vertical menu and then selects the two I/O ports that have to be connected by the link.

By instancing links the user defines logical and temporal orderings between modules. During this phase syntactical controls are applied to avoid specifying inconsistent connections. If there is an incorrect specification (i.e. two links entering in the same input port, or a link connecting two ports with different data type) an error message is generated by the editor and the instantiation is rejected.

The current network could be modified by deleting and/or adding modules, by altering the position of the modules, or by redirecting some links, thereby changing the logical ordering between modules. By selecting interactive fields of each instantiated module information can be visualized on its application task or the actual values of the module parameters, whose values can also be changed. At each moment the specification of the current NetGIS program can be saved for further use.

The design of the system followed the object-oriented paradigm, providing good abstraction mechanisms and optimum extendibility by using inheritance, data hiding and reusability. Links, modules and networks are the basic object classes of the system. A module is an object with two categories of attributes: status and appearance. The status attributes are: the class; the high level semantic task related; a set of parameters, to modify or customize the actual semantic function associated with each instantiated module; the execution state (ready, executed, running or data waiting); links-connected, a boolean to indicate if associated connections have been defined; the links list, the list of the defined connection; the list of the I/O ports. The appearance attributes include the dynamic information for drawing the graphical items associated with the module. The methods associated with the general module class are: creation, deletion, visualization, attributes update, selection and execution.
3.3 Execution

The execution of a visual NetGIS program can be driven under two different policies: user-driven and dataflow.

In the user-driven policy, the system gives users full control over the next module to be executed, asking him for a step-by-step explicit selection of the next module. If the chosen module is in executable state (i.e. if it has the data input ready on the input ports) the request is satisfied; otherwise, the system communicates that selection has been incorrect and asks for a new selection to continue or close the run. Moreover, the user can dynamically interact with the network at execution time too, modifying the execution ordering of the modules depending on the partial results, or stopping the execution of the network if the resulting data are not useful.

Under the dataflow policy the system executes a network by selecting one of the total orderings allowed by the topology of the specified network: execution is thus completely automatic.

4 A work session example

Figure 7 shows the layout of the system at startup time.

In the example in Fig. 8 the programmer has instantiated five modules and has correctly connected some of them. Then he/she has tried to specify an erroneous link: the editor detects this syntactic error and an error message is generated.

In the example in Fig. 9 the user selection of one more field caused the visualization of the corresponding parameter setting dialog window. The user can now interactively modify the current parameter values shown; the module internal status will be consistently updated.

5 Concluding remarks

This paper describes NetGIS, a visual language and an associated programming environment proposed to reduce the complexity of the general GIS application interface. The paper addresses the complexity of the Task-to-Function process, that is how to fulfill a specified user
Figure 8: Erroneous specification of a link.

Figure 9: The parameter setting dialog window is shown.
task using the low-level functionalities provided with the GIS. The proposed language, based on the module concept, is an attempt to deal with this problem by defining an intermediate abstraction level, half way between the task and the function levels.

A prototype was developed, using an object-oriented approach in the definition of the application structure. An object-oriented design of NetGIS is particularly opportune: the language is defined as a set of interacting objects - modules and links - whose interconnection specifies a procedural flow graph. The result is a visual programming environment, easy to use and closer to the conceptual model of the user than the usual graphical GIS interfaces. The prototype uses ESRI's ArcInfo as its underlying GIS [22]; an AML (the ArcInfo command language) subroutine was thus defined for each module to implement the sub-task which the module represents.

The power of the NetGIS approach clearly depends on the correct definition of the basic modules set. A tentative set is defined in this paper, and it has been implemented in the prototype, but further work must be done. In order to increase flexibility, a possible extension of the system can be directed to the development of a tool for the extension of the set of basic modules, allowing users to define their own custom set of modules. The definition of new module classes and their integration in the programming environment should be possible without knowledge of the software implementation of the system.

Acknowledgments

We are grateful to Carlo Magnarapa, Ernesto Bronzini and Sandro Mazzotta for valuable discussions and for their support in the system software specification. This project was partially funded by grants from the Strategic Project MEDASE of the Consiglio Nazionale delle Ricerche.

References


