EUROGRAPHICS '94 / M. Daehlen and L. Kjelldahl (Guest Editors), Blackwell Publishers

*Volume 13, (1994), number 3*

# A Semantics-based Approach for the Design and Implementation of Interaction Objects

F.Paterno', A.Leonardi

CNUCE - C.N.R.
Via S.Maria 36 - 56100 Pisa - Italy
paterno@vm.cnuce.cnr.it

## Abstract

*Although tools for developing graphical user interfaces are becoming increasingly popular, they do not usually highlight precisely which key elements developers should take into account. This still entails considerable expertise in developing user interfaces. In this paper we present an approach to overcome these problems. Our approach is based on a model for interaction objects and a corresponding design space. This is supported by a toolkit where the available interaction objects are initially classified by their semantics whereas in most current toolkits they are investigated by their appearance. This facilitates designers and developers in identifying the interactors needed in order to obtain an Interactive System supporting user tasks.*

**Keywords:** Design of Interactive Systems, Interaction Object Model, User Interfaces Development, Task-oriented Interactive System Modelling.

## 1. Introduction

There is a general agreement that developing user interfaces is still a difficult task. Various limitations in current tools are described in [11], for example appearance-driven design of the available interaction techniques or poor support for their application semantics (by "application semantics" we mean what an interaction object provides in the direction of the application). Generally speaking, current User Interface Systems provide a wide range of interaction objects which are designed with poor abstractions and with a lot of low level details, particularly in their appearance. This generates two types of difficulties:

- One is related to identifying which interaction object is most suitable for implementing the interaction desired.
- The other is in developing complex User Interface Systems by trying to compose interaction objects with not very clear semantics and with poor support for their composition in order to exchange information among them.

To overcome these problems we propose a new approach based on some results related to analytical work on formal system modelling [3] [4] [18]. Our aim is to obtain an engineering model for the design and implementation of interaction objects, which we call *interactors.*
Our approach takes the viewpoint of researchers and developers of Interactive Systems, and is addressed to other researchers and developers, along with users of Interactive Systems.

In our case, interactors are defined by an architectural model, aimed at:

- Obtaining a methodology which allows designers to easily define the set of interaction objects needed by taking into account the user's perspective. This is achieved firstly with a task analysis which is then refined into a hierarchy of objects. This entails characterising interaction objects in terms of the user task which they are able to support.
- Providing a template which explicitly indicates to developers the main components and relationships which must be considered in designing and implementing interaction objects.
- Encapsulating their general dynamic behaviour by indicating the relationships among the components in terms of access to their functionalities. Each specific interactor can then be obtained by refining the general model. This adds further constraints to the dynamic behaviour and refines data types and processing.
- Improving the reusability of interaction objects through a clear, structured approach.

Our approach is supported by a toolkit whose hierarchy is organised so as to clearly identify the range of the possible solutions that designers need to assess when deciding which interaction object to use in a specific development of Interactive Systems.
We decided to characterise each interactor in terms of the basic task which it supports by extending the approach in [5]. This makes it easier to translate user tasks into calls to the functions of the system available. In fact, interaction objects are characterised in terms of the processing they support (selecting a value, activating an application function or determining a location, and so on), whereas in most current toolkits they are defined by their appearance.
We then show an approach that supports an interactor refinement, primarily by refining the corresponding task.
We define a task in terms of accesses to the functionality of a system in order to modify its state or to query it. Tasks depend on the part of an Interactive System they refer to:

- *presentation-related tasks* do not require access to the functional core (i.e., the set of application functionalities which are independent of the interaction objects used to interact with the user). They can only be performed by the UIS (User Interface System) component of an Interactive System; for example, by changing the layout of the windows on the screen or selecting a visualised object;
- *application-related tusks* require access to the functional core (i.e., the part of an Interactive System which is independent of the components used to interact with the user); for example, the functionality of a file system or of a network.

## 2. Related Works

In [11] interaction techniques are characterised on the basis of their semantic effects rather than their appearance. This method was used to develop the ACE environment [12], where a limited number of abstractions is provided.
Humanoid [14] was developed so that all features should be visible and changeable by designers, which means that designers can control all aspects of an interface design. Five semi-independent dimensions were thus identified: application semantics, presentation, behaviour, dialogue sequencing, and action side-effects. UIDE [6] is an interesting model-based environment where in the application model the designer describes the functionality of an application as a set of actions which are associated with pre- and post-conditions, parameters and parameter constraints. When an application object has been created and registered, its corresponding presentation object is displayed. The UIDE runtime architecture is designed so that application action semantics is used to guide the dialogue sequencing. In UIDE the interaction objects are dependent on widgets of current toolkits which communicate by call-backs with a dialogue manager. This requires low level support for communicating with higher levels, and it may create problems in mapping application objects with widgets with not well defined semantics.
None of these proposals has a specific architectural model for interaction objects. This is a serious drawback because developers have no clear indication about the main aspects to take into account when deciding how to structure and choose interaction objects.

Existing architectural proposals include PAC [1], Theseus [9], Dialogue Cells [20] and Abstract-View-Link [7]. We differ from these works in that we focus on the problem of characterising interaction objects in terms of their semantic effects. We also make more explicit distinctions about their input and output functionalities along with their relationships. Smalltalk [8] has a similar architecture to our interactors, however in our approach each component has more explicitly defined functionalities and interfaces with the other components. This allows us to overcome a problem of Smalltalk MVC which is that the components are too tightly coupled. In addition, we have defined a set of composition operators for our interactors [19].
Myers [16] developed a group of interactors to identify meaningful abstractions to facilitate developing user interfaces. These interactors try to encapsulate the behaviour of typical interactions. All the specific interactions are then obtained by giving values to a long list of parameters. Our approach is different because we start from a unique general model for interactors. We then refine specific interactors by working on the design space that has been identified and which has been implemented in the hierarchy of the interaction objects available. By assessing the design space designers can easily identify the interactor that they need, and if it has not yet been supported they can easily find out where it should be in the hierarchy and how to refine it.
Chiron [21] is a system which separates the application code from the user interface code by attaching user interface agents, called artists, to application abstract data types. Our interactors have a similar task to these artists, but being more structured they are easier to reuse and to implement.

## 3. The Architectural Model

The need for a new architectural model arose from the desire to explicitly identify input and output functionalities between the user and the application side, and indicate their relationships by exploiting previous experience in graphics and user interface systems. In addition, we want to provide multi-layer networks of interactors, by hierarchically composing input or output parts of interactors. This means that the input of the input part or the output part of an interactor, is not always provided by the user or the application, respectively, but it may be provided by one or more other interactors.
Our architectural model is a refinement of a previous model, formally specified in [18], into the Sather object-oriented programming language [17]. This model was originally inspired by the architecture of the Computer Graphics Reference Model [10]. Then it was modified to satisfy specific requirements in the design and implementation of interaction objects. Thus, for example, the input part is performed by one component alone. This allows us to maintain a clear distinction between the input and output components and their relationships, and also to simplify the architectural model and to have components with a similar load of processing.

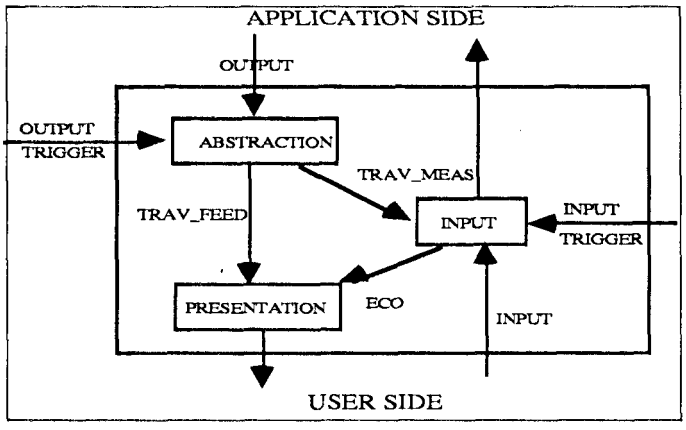


Figure 1: The Architecture of the Interactors Implemented.

The model has three main components (Figure 1). Each component identifies important aspects in modelling and designing interaction objects. This means that this decomposition identifies a 3D design space for interaction objects, with one axis associated with each basic component (Abstraction, Presentation, Input). We will now describe these components in terms of the questions they should answer.

- *Abstraction* contains the description of the data which should be visualised; it answers the following questions:
*a1) What information should be graphically represented?* Its state can contain both application data or graphical entities for further graphical refinement (e.g. projection of a 3D scene into a 2D space, removing hidden lines, etc.).
*a2) How should this information be rendered?* Given a set of data there are many ways of rendering it. For example, a set of values can be graphically described by a pie chart or a bar chart, or by a function; while a 3D graphical scene can be rendered by identifying different parameters for the projection.
*a3) When should rendering be performed?* The output trigger performs this task. There may be cases where the answer is simply whenever some new output data is received.
*a4) Where does it receive output data from?* Here we can find either another interaction object or the functionality of an application object.

- *Input* performs the input processing from the user toward the application; it answers several questions:
*i1) What input is received from the user side?* Here the data types received from the user side are indicated.
*i2) What higher level input is produced toward the application side?* The data type which is generated by the input processing of the interaction object considered is indicated. This is the information which mainly identifies the application semantics of an interactor.
*i3) When should input be provided to the application side?* The input trigger performs this task. Once again, the answer may be whenever some new data is received.
*i4) What information does it provide for feedback?* Here the input state and/or the last input received from the user side can be indicated.
*i5) Where should input data be received from?* Here we can either find user-generated actions or other interaction objects.
*i6) Where should the result of the input be sent to?* Here there is either another interaction object or the functionality of an application object.

- *Presentation* defines the appearance of the interaction object which is the result of the output processing composed with the echo of the input state. It answers the following questions:
***p1)*** *How should feedback of the user-generated input be represented?* There are many methods here *too;* for example, all the possible ways to give feedback of the current selected element in a menu (outline rectangle, cross sign, reverse video rectangle, etc.).
*p2) Where should the result of the output be sent?* Here there is either another interaction object for further graphical refinement or a window of the underlying window system.

The above questions can be divided into two groups: one related to identifying the appearance and behaviour of a specific interactor, and the other questions about *where* to send or receive data. The answers to the second group of questions are provided when the logical structure of the UIS (which is defined by composing instances of interactors) is defined (there is an example in Figure 5). In fact, at that particular time information flows are defined among interactors, and among interactors and application functionalities. Different interactors can be thought of as performing different transformations on the graphical entities at different abstraction levels (see the Computer Graphics Reference Model [10] as an indication of abstraction levels).

As an example of the answers to the previous questions, let us take the case of a pop-up menu used for selecting a font. In this case the answers are (without considering the "where" questions and by using the indexes *a1, a2* and so on to map questions and answers):

- for the Abstraction:
a1) a set of names of available fonts which are received from the Application side;

a2) by a set of rectangles with a label inside which are associated by the Abstraction to each received font;
a3) when the button on the mouse is pressed while the cursor is on an area associated with a font selection;

- for the Input:
i1) the position of the cursor;
i2) the font selected;
i3) when the button on the mouse is released while the cursor is in the menu;
i4) the font currently selected by the cursor while the button on the mouse is being pressed;

- for the Presentation:
p1) highlighting the current selected font by inverting foreground and background colours.

The implementation of the architectural model was performed in the Sather [12] programming language. Sather supports efficiency, strong typing, multiple inheritance and garbage collection. These features were useful in the implementation. We implemented three hierarchies of objects, one for each main component. The interaction objects are obtained by multiple inheritance of one object from each hierarchy, and by defining the methods which indicate the interactions among the components.
Sather programs are compiled in C programs, and can call C programs and vice versa. This guarantees high portability and easy access to the functionality of underlying window systems such as X.
The interaction objects have three general purpose functions (create, resize, configure–connections). They also have four functions which define the relationships among the components that determine its dynamic behaviour (Figure 1). These functions are:
- *create*, to create a new instance;
- *resize*, which updates the appearance and processing of the interaction object depending on the new window coordinates;
- *configure–connections*, that indicates which interactors should receive the results of both the input and output sides.
- *input trigger*, which can be considered as a Boolean function. When this is verified the processing function of the Input component is called and provides it with the indication of the interactor or the application object to which the result of the function should be given;
- *input*, which calls the *echo* function of the input object with the input data received as a parameter and gives its result to the *highlight* function of the presentation object in order to echo the current input state;
- *output trigger*, another Boolean function which, when verified, applies two functions to the Abstraction object: *trav_meas* to provide updated information of its state to the input component, *trav_feed* to provide similar information to the presentation component which can pass the updated description of the appearance either to another interactor or to the window system for visualisation;
- *output*, which receives new output data from the application or other interaction objects, and adds them to the state of the interaction object.

Besides being modelled in terms of interactors, i.e. objects which interact with the users, the UIS is also modelled in terms of *controllers*, i.e. objects which interpret and distribute events of the underlying window systems to interactors, and which control the dynamic allocation and deallocation of interactors on the screen. The functional core is designed in terms of *application objects* which can follow different paradigms, and *conceptual objects* which provide an interface between interactors and application objects.

## 4. The Design of the Hierarchy of the Available Interaction Objects

Once we have defined the general architecture for interaction objects, the problem is then how to design the set of its instances to make them available to developers of user interfaces.
The advantage of performing interactors by multiple inheritance of an instance of the three components is that in this way the implementation clearly reflects the architectural model. This makes it easy to explore the underlying design space in the evaluation process of the most suitable interaction objects to support the current task.

There is a general code template in the root of the interactors hierarchy that encapsulates the general dynamic behaviour. This code is implemented by the four functions described above. Different inherited abstraction, presentation and input components can then be linked. For example, if we want to change the type of graphical representation of the application data visualised by the interaction object being considered, only the abstraction object which is inherited needs to be changed (this entails changing one line of code).

We consider the application semantics of an interactor strongly characterised by the data type which its Input component delivers toward the application as being a result of the interaction with the users.

Our approach is to highlight the application semantics of each interaction object, as this is the most important result of the user interaction in terms of modifying the state of the Interactive System. This makes it easier to translate the solution in terms of user tasks into a collection of interacting objects that support access to system functionalities.

The most important element in the refinement of the available hierarchy of interaction objects is thus the Input component: so the first levels of our hierarchy are defined in terms of their application semantics.

Figure 2 shows the first levels of the hierarchy. We start with very general and basic tasks which were selected by exploiting previous experience in the input model for standard graphics systems and its extensions [2]:

- *Quantifier,* which allows the user to select a value:
- *Text,* which allows the user to input a portion of text;
- *Position,* which allows a graphical point to be selected;
- *Choice,* which enables the user to select from a set of visualised elements.

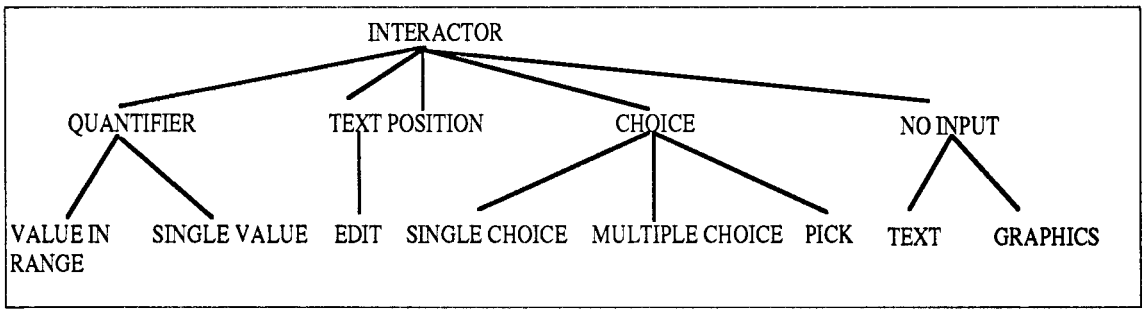


Figure 2 The First Levels of the Hierarchy of Interactors.

These are the most general and important basic tasks which we identified. We also consider the possibility of only having output objects *(No Input).* In addition, there are refinements which always depend on semantic considerations thus obtaining more refined tasks associated with the corresponding interactor. The *Quantifier* can be refined, for example, into the *Value in Range* which means selecting a value in a specific interval (further refined, for example, in a *Scrollbar* or in a *Cycle Button,* for providing a value by cycling among a set of values) or into the *Single value* which allows the user to generate a value (further refined, for example, in a *Button* or in *Type–in* for directly typing a value). *Choice* can be refined into *Single choice,* which allows a value to be chosen from a set of visualised predefined values of the same type; *Multiple choice,* which allows multiple values to be chosen from the set of visualised predefined values of the same type; and *Pick,* which allows the user to choose one element from a set of predefined values of different types and with different appearances.

A further refinement can be made on the basis of presentation and behaviour (Figure 3). The following are examples of a presentation-based refinement (taken from the wide set of choices which we have implemented) of a single-choice object: a rectangular menu, a pie-chart or a bar-chart. They only differ in terms of the type of graphical attribute used to encode the information which has to be visualised [15] (rectangles with the same size, angles or rectangles with a size which is proportional to the data value, respectively).

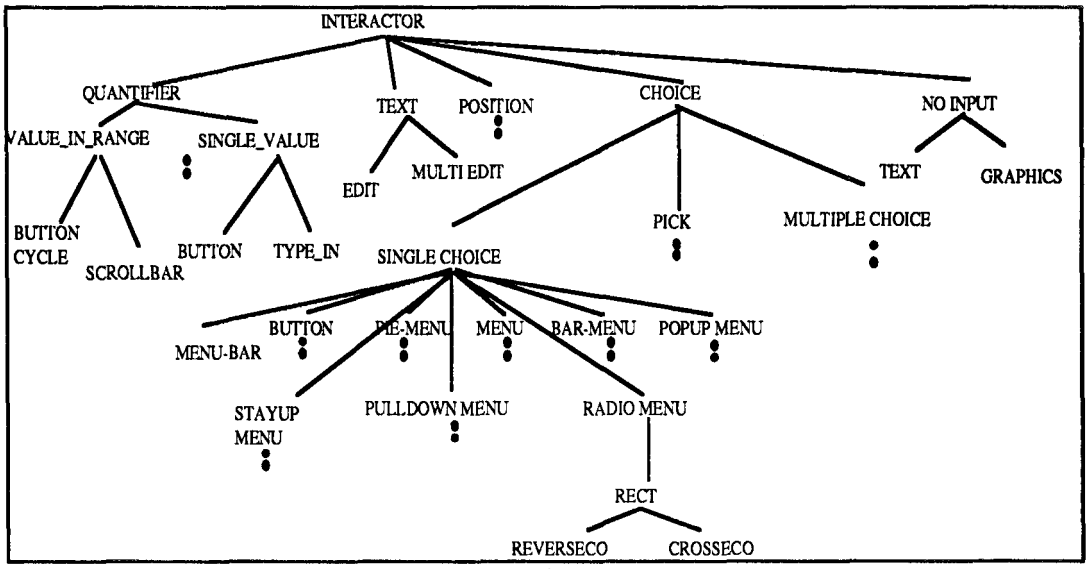


Figure 3: Further Levels of the Interactors Hierarchy.

Pop-up, stay-up and pull-down menus are examples of behavioural-based refinements of a choice object (these are valid for both single and multiple choice). In the first the output trigger is user-generated and after producing one input value the object is deactivated and disappears. The stay-up menu has an output trigger generated by the user but it can be used to generate multiple input data before disappearing from the corresponding window. The pull-down menu is similar to the pop-up menu but is activated when the button of the mouse is pressed while the cursor is on a specific button on a menu-bar.

Finally, there is a refinement which depends on the type of feedback generated: for example, inverting foreground and background colours of the selected element or putting a cross beside it.

In the toolkit we have defined the hierarchy of entities which are objects that can be exchanged among interactors. The first levels of the hierarchy of entities are typical general purpose data types, and it is also possible to refine specific application-oriented objects for better communication with the functional core.

In this approach to developing user interfaces the designer performs task decomposition until his goal is refined in terms of basic tasks which can be associated with interactors. The basic tasks are then refined into the available interaction objects, unlike in the UAN [8] approach where they are refined in a sequence of user actions and system feedback. The interactors are chosen by navigating in the tree of available objects. This tree is organised into logical layers based on the aspects discussed above: application semantics, attributes used for the presentation and type of feedback.

Figure 4 shows a navigation in the tree of the interactors available. At any time it is possible to go down by one level in the tree (the menu on the left indicates the possible current child) and to get information on the components of the currently selected class of interactors.

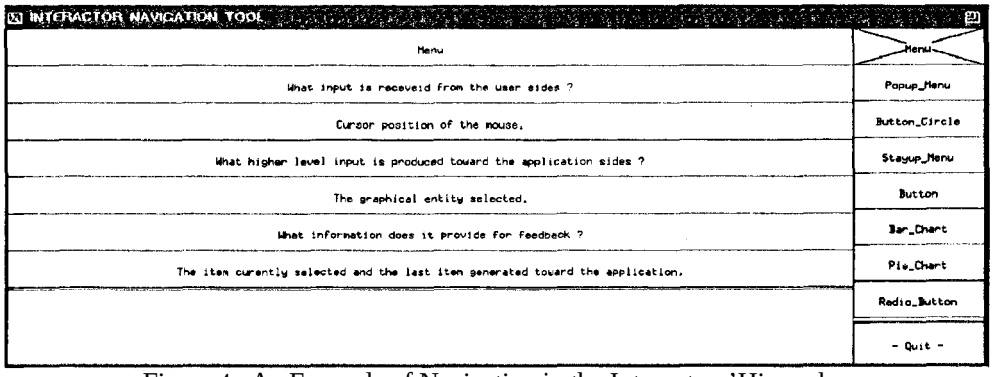


Figure 4 : An Example of Navigation in the Interactors' Hierarchy.

## 5. An Example

We now describe an example of an application of our approach. We consider a small subset of an Interactive System which was developed to allow users to interact with GIS (Geographical Information Systems) functionalities. We describe the part which allows the user to select one province in order to visualise it and the borders of its boroughs. Users can also indicate whether the labels related to each borough have to be visualised or not. They can then select one or more boroughs, and related information will appear in a specific stay-up menu.
The first phase consists in performing a task analysis so that what the user wants to perform is decomposed in terms of the basic tasks which the toolkit supports.
In our example the user's tasks are:
T1) to select one province in Tuscany;
T2) the boroughs of the selected province should then be visualised;
T3) by selecting the graphical representation of one of them related information should appear;
T4) visualising the labels with the names of the boroughs should be under the user's control.

We now have to associate each task with an interactor whose semantics can support it. T1 is supported by a Menu interactor which allows the user to select one element from a set of elements in the same class (Provinces) and with the same appearance (rectangles with font identifiers inside). T4 can be supported by a Radio Button because there is mutual exclusion among the possible choices and one choice must always be selected, T2 needs the result of the interactions associated with T1 and T4 in order to gain access to the functional core to retrieve the request data. This means that in order to have a UIS able to support the tasks described we need to organise it into a two-layer organisation (Figure 5). T3 is supported by a Pick Interactor because different boroughs have graphical representations with different shapes. When the Pick interactor produces an input result, one interactor to visualise information related to the selected borough has to be visualised.
Thus we can start with three interactors at the lower level: Pick, Menu, Radio and one at the higher level (File–System). The Menu allows the user to indicate which province to visualise, and Radio if the labels with the names of the provinces also have to be visualised or not. The input data generated by these two interactors is passed to the File–System interactor, which accesses the data base functionalities, takes the corresponding data and passes it to the Pick interactor. The Pick interactor visualises it and allows the user to graphically select one borough. The result of the selection dynamically enables a Stay-up menu and visualises information related to the borough selected.
File–System is an example of an interactor which does not access directly to the user interface. However, the result of its Presentation is passed to the Abstraction of another interactor in order to receive further processing; it is then visualised.

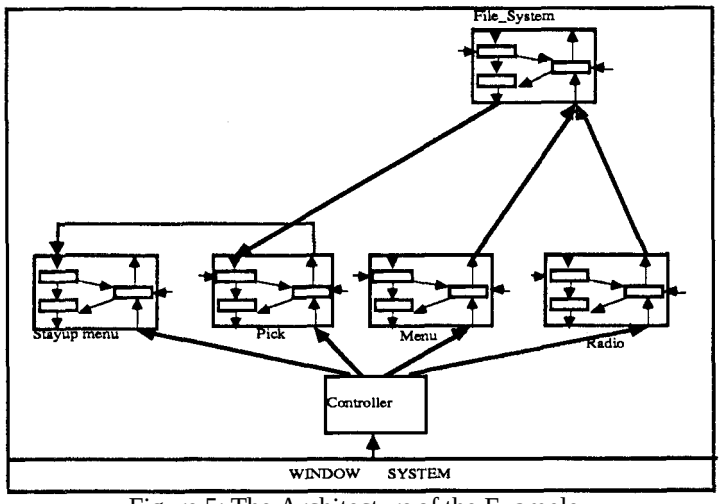

Figure 5: The Architecture of the Example.

We use one Controller object which knows the related window for each interactor, the type of events which it is sensitive to and the compositions among interactors. The next figure is an example of the user interface obtained, both interaction techniques and graphics are used. Its specification is only 30 lines long. On one side, the user has selected the province of Massa-Carrara and the "no labels" element of the Radio Button (which means that no labels with the names of the boroughs should appear). On the other side, the graphical representation of the Carrara borough has been selected and the "with labels" radio button element. This leads to the visualisation of a set of information related to the selected borough on the left side of the user interface and of the names of all the boroughs, respectively.

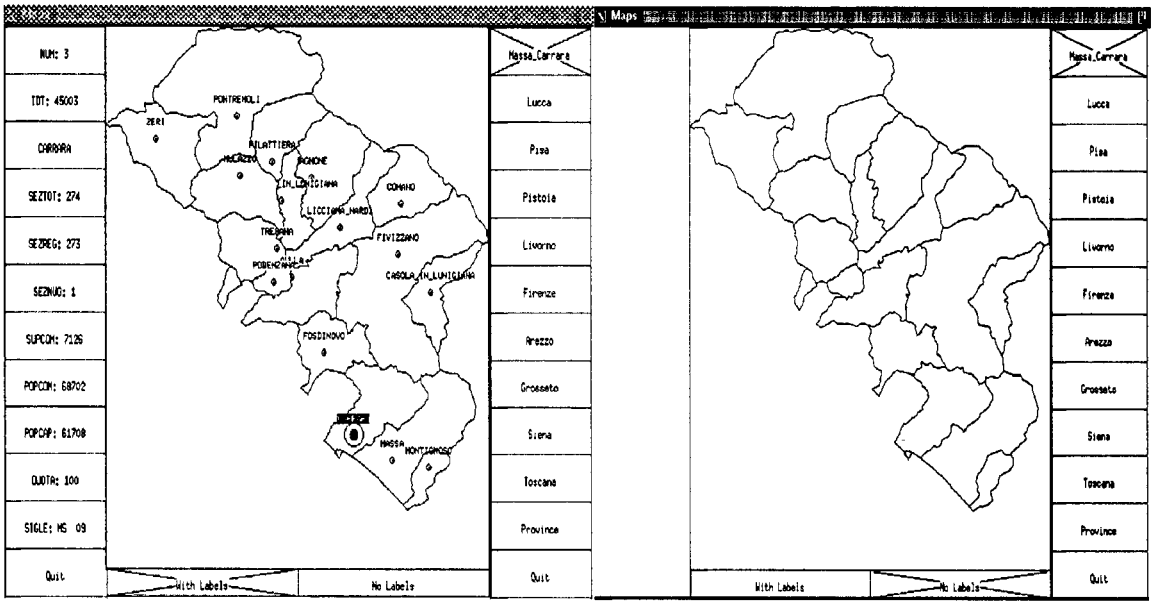


Figure 6: An Example of Interaction with Territorial Data.

## 6. Conclusions

We have presented an approach to facilitating the development of user interfaces. It is based on a clear and structured modelling of the interaction objects available. This allows designers to immediately evaluate whether the interactor considered is suitable or not. Its features include:

- A task-oriented classification of interaction objects;
- An explicit architectural model to improve reusability and refinement;
- A clear description of the bidirectional flow of information between users and applications;
- A hierarchy of available interaction objects incorporating the design space proposed.

Our approach allows designers to model their Interactive Systems in such a way as to reflect the logical organisation of user tasks. The resulting systems are thus easier for the final users.
Future work will be dedicated to building a tool to provide automatic support for the generation of a User Interface from the description of the user tasks to support, performed by a formal notation.

## References

[1] Coutaz, J. PAC, an Implementation Model for Dialogue Design, Proceedings of Interact'87,Stuttgart, September'87, pp.43 1-436.
[2] D.A.Duce, P.J.W. ten Hagen, R. van Liere. An Approach to Hierarchical Input Devices, Computer Graphics Forum, Vol.9, 1990, N.1, pp.15-26.
[3] Duke, D.J., Harrison M.D., Abstract Interaction Objects, Proceedings of EUROGRAPHICS '93, Barcelona, September '93 pp.25-36.
[4] Faconti G., Paterno'F., An Approach to the Formal Specification of the Components of an Interaction. In C.Vandoni and D.Duce, editors, Proceedings EUROGRAPHICS '90 pp.481-494, 1990.
[5] Foley, J.D., Wallace, V.L., Chan, P., The Human factors of Computer Graphics Interaction Techniques. IEEE Computer Graphics Application 4, 11, November 1984, pp.13-48.
[6] Foley, J.D., Kim, W.C., Kovacevic, S., Murray, K. UIDE-An Intelligent User Interface Design Environment. In Architectures for Intelligent Interfaces: Elements and Prototypes. Eds. J.Sullivan and S.Tyler, Reading, MA: Addison Wesley, 1991.
[7] Hill, R.D. The Abstraction-Link-View Paradigm: Using Constraints to Connect user Interfaces to Applications, Proceedings CHI'92 pp.335-342.
[8] Hartson R., Gray P., Temporal Aspects of Tasks in the User Action Notation, Human Computer Interaction, Vol.7, pp.1-45.
[9] Hubner W., de Lancastre M., Towards an Object-Oriented Interaction Model for Graphics User Interfaces, Computer Graphics Forum, Vol.8, 1989, N.3, pp.207-218.
[10] Geneva ISO Central Secretariat. Information processing systems, computer graphics, computer graphics reference model. ISO/IEC DIS 11072, 1991.
[11] Johnson J., Selector: Going beyond user-interface widgets Proceeding of INTERCHI '92 Conference (May 3-7 1992). pp 273-229
[12] Johnson J.A., Nardi B.A., Zarmer C.L., Miller J.R., ACE: Building Interactive Graphical Applications, Communication of the ACM, April 1993, Vol.36, N.4, pp.41-55.
[13] Krasner G. E., Pope S. T., A cookbook for using the Model-View-Control user interface in Smalltak-80 JOOP August\September 1988, pp 26-49.
[14] Luo P., Szekely P., Neches R., Management of Interface Design in Humanoid, INTERCHI '93, pp 107-114
[15] Mackinlay J.,Automating the Design of Graphical Presentations of Relational Information, ACM Transaction on Computer Graphics, Vol.5, N.2, April 1986, pp.110-141.
[16] Myers B.A., A New Model for Handling Input, ACM Transactions on Information Systems, pp.289-320, Vol.8, N.3, July 1990.
[17] Omohundro S. M., The Sather language International Computer Science Institute, 1947 Center Street, Suite 600 Berkley CA, ICSI Internal Report, June 1991.
[18] Paterno'F., Faconti G., On the LOTOS Use to Describe Graphical Interaction, Proceedings HCI Conference 1992, pp.155-173, Cambridge University Press.
[19] Paterno'F., A Theory of User-Interaction Objects, CNUCE Internal Report, December 1993.
[20] Schouten H.G., ten Hagen P.J.W., Dialogue Cell Resource Model and basic Dialogue Cells, Computer Graphics Forum, Vol.7, 1988, N.4, pp.311-322.
[21] Taylor R.N., Johnson G.F., Separation of Concerns in the Chiron-1 User Interface Development and Management System, Proceedings INTERCHI'93 pp.367-374.