Repositorio Institucional de la Universidad Autónoma de Madrid

https://repositorio.uam.es

Esta es la **versión de autor** de la comunicación de congreso publicada en:
This is an **author produced version** of a paper published in:

Conceptual Modeling - ER 2006: 25th International Conference on Conceptual Modeling, Tucson, AZ, USA, November 6-9, 2006. Proceedings. Lecture Notes in Computer Science, Volumen 4215. Springer, 2006. 413-423.

DOI: http://dx.doi.org/10.1007/11901181_31

Copyright: © 2006 Springer-Verlag

El acceso a la versión del editor puede requerir la suscripción del recurso
Access to the published version may require subscription

NOTES FOR THE CONCEPTUAL DESIGN OF INTERFACES

Simone Santini[*]

Abstract

This paper presents a design method for user interfaces based on some ideas from conversation analysis. The method uses *interaction diagram* and it is conceived to design the overall flow of conversation between the user and the computer system in an abstract way, as an architectural prolegomenon to the designer's choice of the actual interface elements that will be used.

1 Introduction.

The art of design is an art of abstraction. A design is the specification of certain structural relations between components; a specification that abstracts, on one hand, from the characteristics of the components that are not relevant for the structure of the whole and, on the other hand, from those structural relations that can be dispensed with at the level of abstraction where the designer is operating at a particular moment. For design is also a process of progressive deepening of the detail: the first sketch that Norman Foster made of his Hong Kong and Shanghai Bank building (in Hong Kong) consisted in a stack of four rectangles hanging, each one separately, from the steps of a ladder-like structure [10]. Each one of these rectangles would in the end become a block of ten stories of the bank, a transition process that required painstaking attention to a plethora of excruciatingly minute details. In the case of Norman Foster, just as in any other case of design of a complicated system, however, it is important to maintain a progressive view of the process, to possess the means to express the evolving design at different levels of detail, from a very generic one in which only the most general and most important traits of the system are represented to the most detailed one.

What is true for the design of a building is true as well for the design of an interface to a complex information system. Just as it happens for other design activities, the design of an interface needs a methodology and a notation that will allow a design to progress from its

[*]Universidad Autónoma de Madrid, Spain and University of California, San Diego, USA

most general lines to its most minute details. The design of interfaces for information systems often fails to develop along these lines, in particular it fails to start at a suitably abstract and conceptual level. Quite paradoxically, this problem is due in part to the great success of standard computer interfaces, in particular to the interfaces based on overlapping *windows* for local programs and to those based on documents and forms for programs invoked through the web. The ubiquity of these interfaces often leads the designer to start thinking about the interface in terms of windows, buttons, data presentation options (or, in the case of web interfaces, in terms of pages, forms, and documents). This, I submit, is the interface equivalent of trying to create a program starting from the specification of its low-level functions, without an architectural design in which the overall organization of the program and the relations among its different functions are established.

That the design of windows or forms is a necessary step in the design of an interface doesn't entail that it should be the first, nor the most abstract. The lack of a suitably high level architectutal design of interfaces is exacerbated by the lack of a suitable design method and formalism: in many cases, interfaces are designed using the same methods and formalisms used for general programming, which these days means, more often than not, using an object oriented design method. For instance, UML, the popular object oriented graphic formalism for object oriented design, has been used in [9], and specialized for web navigation in [2]. Object oriented design has the primary purpose of designing the static structure of a program and the dynamics of the interaction among its components.

In an interface, however, the *process* of interaction is logically prior both to the static structure of the interface and to the dynamic interaction of the components, both of which depend on the more abstract process of interaction. Such a process is not well captured by program design formalisms because it is not itself a part of program design, but the description of a generalized *conversation* between the user and the system. In other words, the use of a program design method forces the designer to focus immediately on the structure of the interface rather than beginning by designing the structure of the conversation that takes place between the user and the information system. It is noteworthy, for instance, that a paper such as [9] on the use of UML for user interface design makes little of no use of the class diagram, that is, with the main conceptual tool of UML. The class diagram is of course present in this approach, but the impression is that it is by no means the main design instrument. The main graphical instrument for interface design in [9] is an interaction diagram not too dissimilar from what is presented here although, in my view, it suffers from two drawbacks. First, it is not properly grounded in a theory of interaction and, secondly, the need to resolve the interface in an object oriented design makes the specification of the interface very much oriented towards the information exchange between the user and the system, rather than on the possibilities of the conversation between the two.

In this paper, I present a method and a formalism for interface design particularly suitable for information systems interfaces, but adaptable to a considerably wider range of situations. The method abstracts from the details and the appearance of the interface to concentrate on the design of the process of conversation that takes place between the user and the system. The types of interaction that are chosen at any particular step constrain the choice of what interface elements are suitable to implement them, so that the method is also very effective as an architectural phase preliminary to the design of the appearance of an interface. The principal formal instrument of the method is the *interaction diagram*, somewhat inspired to conversation analysis [4]. Section 2 will introduce the theoretical basis of the method, which is to be sought in a form of *algebraic semiotics* [1]; section 3 illustrates the interaction diagrams and their use.

2 Semiotic systems.

The essential function of an interface is to allow an interaction between a person and a program, a communicative function that falls in the general area studied by semiotics. In this respect, however, we find ourselves in a conundrum: by its very nature and theoretical foundations, semiotics escapes any attempt at formalization while, in order to design a computer program for a given problem, we need that the problem be completely formalized. The pragmatic solution in this case is to proceed to a programme of "partial formalization," that is, of formalizing only those aspects of communication that lead themselves to formalization. It is clear that, in so doing, many of the interesting aspects of signification will be left out: in particular, all the *semantic* aspects of communication will be left out. But this is not a loss in the present situation, since if an aspect of communication can't be formalized we can't deal with it using a computer, so it can't be the object of interface design. In particular computers can't deal with semantics, unless the semantics is an *epiphenomenon* of the interaction (*episemantics,* or *emergent semantics* [5]), in which case the *syntactic* rôle of the interface is particularly important [7].

The basis for this formalization is constituted by the formal *sign system* derived, with some adaptation, from Goguen's algebraic semiotics.

Definition 2.1. *A sign system is a 5-tuple*

$$\mathfrak{S} = (T, V, \leq_T, F, A) \tag{1}$$

where:

i) T *is the set of sorts (or data types);*

ii) V *is a set of parameter types;*

iii) $\leq_T$ *is a partial order on* T*, called the subsort relation;*

iv) F *is a set of functions and relations on* T *and* V*;*

v) A *is a set of logical statements called the axioms.*

The sorts are those to which the *signs* of the system belong, that is, in the case of an interface, the types of elements and combinations of elements through which the interaction takes place. The parameters capture ancillary information about the sorts (position, color, etc.).

Definition 2.2. *Given two sign systems* $\mathfrak{S}_1 = (T_1, V_1, \leq_{T_1}, F_1, A_1)$*, and* $\mathfrak{S}_2 = (T_2, V_2, \leq_{T_2}, F_2, A_2)$*, a semiotic morphism* $M : \mathfrak{S}_1 \to \mathfrak{S}_2$ *is a collection of partial functions*

$$\begin{aligned} M : T_1 &\to T_2 \\ M : V_1 &\to V_2 \\ M : F_1 &\to F_2 \\ M : A_1 &\to A_2 \end{aligned}$$

such that

i) *if* $\tau_1 \leq_{T_1} \tau_2$*, then* $M(\tau_1) \leq_{T_2} M(\tau_2)$*;*

ii) *if* $p(s_1, \ldots, s_k) \in F_1$*, and* $M(p)$ *is defined, then* $M(p)(M(s_1), \ldots, M(s_n))$*;*

iii) *if* $s = f(s_1, \ldots, s_n)$ *and* $M(f)$ *is defined, then* $M(s) = M(f)(M(s_1), \ldots, M(s_n))$*.*

The general idea of a semiotic morphism is to transform the system $\mathfrak{S}_1$ into $\mathfrak{S}_2$ in a way that preserves (part of) its structure. Before specifying additional properties of morphisms, I will need the following technical definition:

Definition 2.3. *A selector for a sign system $\mathfrak{S}$ is a function $f : \tau \rightarrow \nu$, with $\tau \in T$ and $\nu \in V$ for which there is a set of axioms A' such that adding A' and f to $\mathfrak{S}$ is consistent and defines a unique value $f(x)$ for each such $x : \tau$.*

The definition is a trifle involved but, essentially, it states that f attaches a parameter to each sign of sort τ and that its definition (the axioms in A', which uniquely define it) does not cause the system to become contradictory. I will clarify all this with an example in a short while, but first I need a couple more definitions.

Definition 2.4. *Let $\mathfrak{S}_1 = (T_1, V_1, \leq_{T_1}, F_1, A_1)$, and $\mathfrak{S}_2 = (T_2, V_2, \leq_{T_2}, F_2, A_2)$ be two sign systems and $M : \mathfrak{S}_1 \rightarrow \mathfrak{S}_2$ a semiotic morphism.*

i) *M is axiom preserving if, for each $a \in A_1$, $A_2 \models M(a)$;*

ii) *M preserves a selector f of $\mathfrak{S}_1$ if there is a selector f' of $\mathfrak{S}_2$ such that for every sign of $\mathfrak{S}_1$ for which f is defined it is $f'(M(x)) = M(f(x))$.*

Definition 2.5. *let $\mathfrak{S}_1$ and $\mathfrak{S}_2$ be two sign systems as in the previous definition, and $M, M' : \mathfrak{S}_1 \rightarrow \mathfrak{S}_2$ two semiotic morphisms, then*

i) *M' preserves at least as many axioms as M, written $M \leq_a M'$ if for every $a \in A_1$, if M preserves a then M' also preserves a;*

ii) *M' is at least as inclusive as M, written $M \leq_i M'$ if, for each sign x of $\mathfrak{S}_1$, $M(x) = x \rightarrow M'(x) = x$;*

iii) *M' preserves at least as much content as M, written $M \leq_c M'$, if whenever M preserves a selector of $|frakS_1$, so does M'.*

The inverse of a morphism, the definition of isomorphisms, and the various unicity and definition theorems follow pretty much the expected patterns [6, 1].

2.1 An example

As an inllustration of the principles of semiotic systems, I will present a simple interface for a fictitious image data base. As the purpose of an example is to illustrate the ideas rather than obfuscate them, I have purposedly chosen a very simple example, one for which I will reach conclusions that are, *per se*, quite obvious, and that could be obtained without the complicated machinery of algebraic semiotics. The reader is invited to consider that the purpose of the example is to illustrate the theory rather than to prove its power, and to mentally extrapolate the results to much more complicated examples.

Consider the typical interface of a system for doing content based image retrieval based on *query by example*, which displays a grid of 3×3 images, containing the images most similar to the current query image. This is a semiotic system with three sorts: *image*, *position*, and *group*. An image is composed of an identifier and the actual image data:

```
type image is (id : int × data : unit)
```

The image data are declared of the data type *unit* because, in this particular interface, there are no operations defined on them. A position is a pair of integers:

```
type pos is (row : int × col : int)
```

while the group is an array of images that is, a function that associates an image to each position:

```
type group is pos → image
```

The ordering relations of the system are

image $\leq$ group
pos $\leq$ group

Some functions defined in this system include:

img_at: group $\times$ pos $\rightarrow$ image; given a group and a position in it, returns the image in that position;

ngb : pos $\times$ pos $\rightarrow$ boolean; determines whether two positions are neighbors in the interface;

nxt : pos $\rightarrow$ pos $\cup\perp$; returns the next position to a given one in the textual order (left to right, top to bottom), or $\perp$ is there is no such position;

s : image $\rightarrow (0,1)$; returns the "score" of an image.

Axioms may include:

i) structural axioms of the group, such as *ngb((1,1),(2,1))*, *ntx(1,3) = (2,1)*, $\tau:\mathsf{pos} \Rightarrow 1 \leq \tau.r \leq 3 \wedge 1 \leq \tau.c \leq 3$, the symmetry of ngb, and so on;

ii) unicity axioms, such as

$$\forall p_1, p_2 : \mathtt{pos}.(p_1 \neq p_2 \Rightarrow \mathtt{img\_at}(p_1).\mathtt{id} \neq \mathtt{img\_at}(p_2).\mathtt{id}) \tag{2}$$

iii) scoring axioms

$$\forall p_1, p_2 : \mathtt{pos}.(p_2 = \mathtt{nxt}(p_1) \Rightarrow s(\mathtt{img\_at}(p_1)) \geq s(\mathtt{img\_at}(p_2))) \tag{3}$$

(images are ordered by score),

$$\forall p_1, p_2 : \mathtt{pos}.(p_2 = \mathtt{nxt}(p_1) \Rightarrow \not\exists i.s(\mathtt{img\_at}(p_1)) \geq s(i) \geq s(\mathtt{img\_at}(p_2))) \tag{4}$$

(positions next to each other contains images which are next to each other in the scoring list), and

$$\forall i.(\exists p.s(i) > s(\mathtt{img\_at}(p)) \quad \Rightarrow \quad \exists p'.i = \mathtt{img\_at}(p')) \tag{5}$$

(the interface contains the images with the highest scores).

Consider now a second interface, similar to the first with one exception: the images are laid out in a single file of 6 images. The position is now an integer number between one and six ($\mathtt{pos}' \equiv \mathtt{int}$) and consequently the group ($\mathtt{group}' : \mathtt{pos}' \rightarrow \mathtt{image}'$) is now a function from integers to $\mathtt{image}'$, which is isomorphic to image. A morphism between the two can be defined as a map of types $M : \mathtt{pos} \mapsto \mathtt{pos}'$, $M : \mathtt{image} \mapsto \mathtt{image}'$, $M : \mathtt{group} \mapsto \mathtt{group}'$. The morphism maps functions $M : \mathtt{nxt} \mapsto \mathtt{nxt}'$, and $M : \mathtt{ngb} \mapsto \mathtt{ngb}'$, and so on. Note, however, that some of these mappings are partial functions: for instance, when mapping pos to $\mathtt{pos}'$, not all the positions in the first interface are mapped to positions of the second one, since the first interface has nine position, while the second has only six. Also, only some of the axioms are translated; we have, for instance, $M(\mathtt{ngb}((1,1),(1,2))) = \mathtt{ngb}'(1,2)$, but an axiom such as $\mathtt{ngb}((1,1),(1,2))$ is not translated into any axiom of the second interface, while an axiom such as $\mathtt{ngb}'(3,4)$ of the second interface has no correspondent in the first one. These differences simply reflect the different arrangement of the two interfaces, but there are other discrepancies between the two. One, which we have already noticed, is the partiality of the function $M : \mathtt{pos} \rightarrow \mathtt{pos}'$ due to the fact that the first interface shows nine images, but the second only six. But, suppose the second interface as well displayed nine images: what could we say of the structural differences between the two? Note that the scoring function is a selector (considering the score as one of the parameters of the image position) and that, for each interface, axiom 4 holds: consecutive positions contain images that are consecutive in the score ordering. But for the second interface we can state the additional axiom:

$$\forall p, p'.(\mathtt{ngb}(p,p') \Rightarrow p' = \mathtt{nxt}(p) \vee p = \mathtt{nxt}(p')) \tag{6}$$

(all neighbors are consecutive). With this, and defining for the sake of convenience $Q(p,p') \equiv \nexists i . s(\mathtt{img\_at}(p_1)) \geq s(i) \geq s(\mathtt{img\_at}(p_2))$ one can infer for the second interface

$$\forall p, p'.(\mathtt{ngb}(p,p') \Rightarrow Q(p,p') \vee Q(p',p)) \tag{7}$$

There is no morphism from the second interface to the first that can maintain this axiom: the second interface has, in semiotic terms, a richer structure than the first.

Of course, when designing an interface, additional considerations come into play: for one thing, the order "left-to-right, top-to-bottom" is culturally so rooted in the western civilization that the "vertical" neighborhood relations are hardly perceived, so that the matrix placement is virtually equivalent to the linear one. (Things would be very different, of course, were the interface designed for a multi-cultural environment, a matter of which the blindly western-centric engineers are often blissfully oblivious: this is, however, the possible subject for a separate article.) Considerations such as this one are to be regarded as part of the designer's common sense judgment, of his *being-in-the-word*, so to speak, and, as such, they are beyond the scope of formalization, and will not be considered here. Note, however, that the model that we are using can still be useful to suggest the location of possible problems: in this case, it might suggest to the designer the opportunity to place suitable elements (lines, spacing,...) to "weaken" the vertical organization of the matrix, thereby reinforcing the horizontal order.

3 Interaction diagrams

Consider again the query by example interface of the previous section, and the way it works. The user sees a set of images in what we might call the *display space* of the interface, a space constituted, as we have seen, by the possible instantiations of an algebraic sign system. Out of this configuration, the user operates a selection, which we'll call a member of the *composition space*, constituted by the instantiations of another sign system (in this case a very simple one whose signs are single images). A composition is then translated in a suitable request to the system that, in the case of an information system, takes the form of a query drawn from a *query space*, which can be considered, once again, a sign system: one involved in the communication between the interface and the information system. The information system executes the commands of the query space (since one can say that the query sign system belongs to an imperative *sprachspiel*, which is the active principle of the whole system), and creates a configuration in a suitable *output space*. The output space is often an abstract space and, in order to show some of its results, a further semiotic morphism translates it back in the display space. (In a query by example system, for instance, the output may consist of the whole data base organized in a high-dimensional space.) All this can be specified by a diagram constituted in this way:

$$\begin{tikzcd} Q \arrow[r] & O \arrow[d, "\text{display}"] \\ C \arrow[u] & D \arrow[l, dotted] \end{tikzcd} \tag{8}$$

The four letters are not part of the diagram *per se*, but labels used to distinguish the different semiotic systems involved. Here Q stands for *query (semiotic) system*, O for *output system*, C for *(query) composition system*, and D for *display system*; the arrows indicate semiotic morphisms (which, optionally, can be labeled), and the dotted arrow indicates a user action. It should be noted that this diagram is in many senses orthogonal to design-oriented specifications, even to very abstract ones such as the model-view-controller formalism [3, 8]. In this diagram there is no a *priori* distinction between the functions performed by the user and those performed by the computer system. In this case, for instance, the semiotic morphism between query and output is implemented by a computer (specifically, by a data base) but, should we replace the data base with an archivist that does the research manually, the diagram would not change, since it describes an interaction that is the result of a certain activity, and it is logically prior to the decision of having part of that activity done by a computer.

In this sense, this kind of diagram is the logical equivalent, at the interface design level, of the architectural diagrams collected during the requirement phase of a software project.

A closed loop such as this one indicates an iteratively refined query, and I call it a *context*. To make a comparison, consider querying a relational data base: here the user fills in a form, which is translated into a query and answered; the answer is suitably formatted and displayed. There is no context formation in this case and, introducing opportune symbols for the beginning and the end of the interaction, the diagram looks something like this:

$$\begin{tikzcd} \boxed{s} \arrow[r, dotted] & C \arrow[r] & Q \arrow[r] & O \arrow[r] & D \arrow[r, dotted] & \boxed{e} \end{tikzcd} \tag{9}$$

Where the last arrow corresponds to whatever user acknowledgment closes the query. Looking back at the query by example diagram, two things appear to be missing: a way to start a query and a way to finish it. There are typically two ways to start a query: either the user starts it from outside the loop (e.g. by providing a sample image) or the system proposes an initial configuration (e.g. a display with random images): the two alternatives (including the action that terminates the interaction) are shown in these two diagrams:

$$\begin{tikzcd} & Q \arrow[r] & O \arrow[d] & \\ \boxed{s} \arrow[r, dotted] & C \arrow[u] & D \arrow[l, dotted] \arrow[r, dotted, "\text{OK}"'] & \boxed{e} \end{tikzcd} \qquad \begin{tikzcd} Q \arrow[r] & O \arrow[d] & \boxed{s} \arrow[l] \\ C \arrow[u] & D \arrow[l, dotted] \arrow[r, dotted, "\text{OK}"'] & \boxed{e} \end{tikzcd} \tag{10}$$

The first one assumes that the composition is exactly the same at the beginning of the interaction as it is in the context, which is seldom true: some special interface element is used to make the first image selection. The diagram is then

$$\begin{tikzcd} \boxed{s} \arrow[r, dotted] & C' \arrow[r] & Q \arrow[r] & O \arrow[d] & \\ & & C \arrow[u] & D \arrow[l, dotted] \arrow[r, dotted, "\text{OK}"'] & \boxed{e} \end{tikzcd} \tag{11}$$

The diagram can be extended to allow a more complete interaction. For instance, if the data base has some machinery to determine that the interaction is not converging towards a satisfactory solution, it can offer the user the option to disengage: the user can accept, decide to continue, or start a new context by posing a new query:

$$\begin{tikzcd}
 & & C'' \arrow[d] & D' \arrow[l, dotted, "\text{rest.}"'] \arrow[r, "\text{disengage}"] \arrow[dl, dotted, "\text{cont.}"] & \boxed{e} \\
\boxed{s} \arrow[r, dotted] & C' \arrow[r] & Q \arrow[r] & O \arrow[u, "\text{disengage?}"'] \arrow[d] & \\
 & & C \arrow[u] & D \arrow[l, dotted] \arrow[r, dotted, "\text{OK}"'] & \boxed{e}
\end{tikzcd} \tag{12}$$

*   *   *

This diagrammatic notation allows one to analyze other interesting cases of interfaces. One is that of what I elsewhere called *direct manipulation* interfaces [6], in which the composition is done directly in the display space, so that the systems C and D above coincide. This situation can be represented by collapsing the two systems in a single one

$$\begin{tikzcd}
Q \arrow[rr] & & O \arrow[dl] \\
 & D \arrow[ul] \arrow[loop below, dotted] &
\end{tikzcd} \tag{13}$$

or by joining the two symbols with a double edge (reminiscent of the "=" sign) which states that the two systems are one and the same:

$$\begin{tikzcd}
Q \arrow[r] & O \arrow[d] \\
C \arrow[u] \arrow[r, equal] & D \arrow[l, dotted, bend right]
\end{tikzcd} \tag{14}$$

(The two diagrams are equivalent: the double edge is only a graphic convenience.)

In some cases we want to highlight that some of the sign systems are the union of two parts: in the previous example, for instance, it may be the case that part of the configuration comes from direct manipulation and part from some other kind of interaction. We can divide the composition space in two portions $c = c_1 \oplus c_2$ and join them with a $\cdot\!-\!\oplus\!-\!\cdot$ edge:

$$\begin{tikzcd}
Q \arrow[r] & O \arrow[d] \\
C' \arrow[u] \arrow[r, equal] \arrow[d, dash, "\oplus" description] & D \arrow[l, dotted, bend right] \\
C'' & \boxed{s} \arrow[l, dotted]
\end{tikzcd} \tag{15}$$

Note that C and C' are part of the same semiotic system: if they were two separate systems, each one to which could be used to compose a query, the diagram would have looked like

$$\begin{tikzcd}
\boxed{s} \arrow[r, dotted] & C'' \arrow[r] & Q \arrow[r] & O \arrow[d] \\
 & & C' \arrow[u] \arrow[r, equal] & D \arrow[l, dotted, bend right]
\end{tikzcd} \tag{16}$$

As a final example, consider an interface that deals at the same time with two aspects of an information system---say, with the images and the text associated with them, as the interface that I discussed in [7]. Such a system consists of two separate direct manipulation interfaces that operate on the same query space, a situation corresponding to the following diagram:

$$\begin{tikzcd}
 & F \arrow[dd] & \\
\mathrm{Txt} \arrow[ur] \arrow[loop left, dotted] & & \mathrm{img} \arrow[ul] \arrow[loop right, dotted] \\
 & O \arrow[ul] \arrow[ur] &
\end{tikzcd} \tag{17}$$

In summary, we can give the following definition:

Definition 3.1. *An interaction diagram is a colored graph whose nodes are either sign systems, start nodes (*$\boxed{s}$*), or end nodes (*$\boxed{e}$*), and whose edges are of four colors (types):*

$\longrightarrow$: *a semiotic morphism realized by the system;*

$\dashrightarrow$: *a semiotic morphism realized by the user;*

$=\!=\!=$: *an edge that establishes the identity of two sign systems;*

$-\!\oplus\!-$: *an edge that composes two sign systems into one.*

Subject to the following restrictions:

i) *start nodes only have outgoing edges; end nodes only have incoming edges; all other nodes have at least one outgoing and one incoming edge;*

ii) *every cycle has at least one* $\dashrightarrow$ *edge; the graph that is obtained from the original one by removing all the* $\dashrightarrow$ *edges is acyclic.*

3.1 The diagrams as a design instrument

In the previous examples, the diagrams were used as a semi-formal way of keeping track of the interactions between the user and the information system. In this rôle, they are useful mainly as an informal "thought instrument" to collect and organize requirement: the type of instrument that is more useful on a paper napkin or a blackboard than on a computer screen. What makes diagrams into a more complete design instrument is the presence of sign systems and semiotic morphisms. We developed a design system based on these diagrams in which each one of the nodes of the graph is associated by the designer to the formal specification of a semiotic system given in a suitable language[1]: an interface panel for query by example, for instance, can be associated to the formal specification of a system such as that briefly outlined in section 2. Some of the sign systems are chosen by the designer from a library of available interface elements (the display and the composition space are typically selected in this way), while others derive from a formalization, in semiotic terms, of the system for which the interface is designed, others yet can correspond to special modules created by the designer. Between all these elements, the designer creates the morphisms specified by the $\longrightarrow$ edges of the graph.

The system tries to assist the designer by identifying the locus of possible problems (e.g. an interaction in which the user is required to add too much structure or, conversely, one in which a considerable portion of a structure is lost) and to enforce constraints (both the topological constraints of the definition of the diagram and structural constraints such as the fact that the net structural variation in a cycle must be zero). The design instrument is in its early alpha release at the time of this writing (June 2005), but it is being developed along the directions outlined here. Due to my own technical point of view, the system, even in its final form, will not generate automatically the code for the interface that is being designed.

4 Conclusions

In this paper I presented the embryo of a design method for user interfaces in visual information systems. In this area, an interface is important not only as an access to the system, but also as a way of creating the semantics of the data. The theoretical bases of the

[1]I am happy to report that none of the languages used by this system is based on XML.

method are to be sought in conversation theory and in algebraic semiotics, and its embodiment is in the form of interaction diagrams.

References

[1] Joseph Goguen. An introduction to algebraic semiotics, with applications to user interface design. In Chrystopher Nehaniv, editor, *Computation for metaphor, Analogy and Agents*, Springer Lecture Notes in Artificial Intelligence,. Springer-Verlag, 1999.

[2] Natacha Güell, Daniel Schwabe, and Patricia Vilain. Modeling interactions and navigation in web applications. In S.W. Liddle, H.C. Mayr, and B. Thalheim, editors, *Conceptual Modeling for E-Business and the Web: ER 2000 Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling*, volume 1921 of *Lecture notes in computer science*. Berlin:Springer-Verlag, 2000.

[3] G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26--49, 1988.

[4] Michael Norman and Peter Thomas. The very idea. In Paul Luff, Nigel Gilbert, and David Frohlich, editors, *Computers and Conversation*, pages 51--66. San Diego:Academic Press, 1990.

[5] S. Santini, A. Gupta, and R. Jain. Emergent semantics through interaction in image databases. *IEEE Transactions on Knowledge and Data Engineering*, (in press).

[6] Simone Santini. *Exploratory Image Databases; Content Based Retrieval*. San Diego:Academic Press, 2001.

[7] Simone Santini. Image semantics without annotation. In Siegfried Handschuh and Steffen Staab, editors, *Annotation for the Semantic Web*. Amsterdam:IOS Press, 2003.

[8] Matthias Veit and Stephan Herrmann. Model-view-controller and object teams: a perfect match of paradigms. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 140--149, New York, NY, USA, 2003. ACM Press.

[9] Patrcia Vilain, Daniel Schwabe, and Clarisse Sieckenius de Souza. A diagrammatic tool for representing user interaction in uml. In A. Evans, S. Kent, and B. Selic, editors, *UML*

2000; The Unified Modeling Language. Advancing the Standard: Third International Conference. Berlin:Springer-Verlag, 2000.

[10] Stephanie Williams. *Hongkong Bank: The Building of Norman Foster's Masterpiece*. Boston: Little, Brown, and Company, 1989.

San Diego and Madrid, March 2006