A Framework for The Semiotic Engineering of User Interface Languages

Jair C. Leite1,2

1 - Dept. of Informatics and Applied Mathematics
Federal University of Rio G. do Norte
Campus Universitário - Lagoa Nova
Natal RN 59072-970 Brazil
+55 84 215 3814
jair@dimap.ufrn.br

Clarisse S. de Souza2

2 - Dept. of Informatics
Catholic University of Rio de Janeiro
R. Marquês de S. Vicente, 225
Rio de Janeiro RJ 22453-900 Brazil
+55 21 529 9462
clarisse@inf.puc-rio.br

PUC-RioInf.MCC10/97

ABSTRACT
Semiotic Engineering is a theoretically based approach to user interface design in which the interface is seen as a message sent from designers to users carried by the computer medium and whose content is the software functionality and the interaction model. In this paper, we describe a framework founded in semiotic theory to this design process. It provides a Model and a User Interface Design Language which is a meta-language for the designer to express User Interface Messages and the Interaction Language. Our emphasis here is not in aesthetics aspects of user interfaces, but in the interactive and performing nature of the interface message as it is carried throughout the computational medium.

Keywords
User interface design, human-computer interaction, computer semiotics, design languages.

Resumo
A Engenharia Semiótica é uma abordagem de cunho teórico para o projeto de interfaces de usuários. Nela, a interface é vista como uma mensagem enviada de projetista para usuários, através do meio computacional, sendo seu conteúdo a funcionalidade e o modelo de interação da aplicação. Neste trabalho, descrevemos um quadro fundamentado em teoria semiótica para tal processo de *design*. Apresenta-se um modelo e uma linguagem de *design* para a interface de usuário, a qual funciona como meta-linguagem para o projetista expressar mensagens da interface de usuário, bem como a linguagem de interação. Nossa ênfase não recai sobre aspectos estéticos das interfaces, mas sobre a natureza interativa e atuante da mensagem de interface, à medida em que evolui pelo meio computacional.

Palavras-chave
projeto de interface de usuários, interação usuários-sistemas, semiótica computacional, linguagens de *design*.

INTRODUCTION

User Interface is that part of a software artifact which the user comes into contact with in doing his/her tasks in a domain of activity. It is composed of a collection of devices that allows user and system to exchange information through them. This information sent and received by them triggers computational processes in the system and mental ones in user's mind. *User interface design* is the process to create the devices and structures for user and system to produce these interaction messages. The devices and its structure constrain the user-systems messages in a way that only the correct ones can be interpreted.

In software industry practice, user interface design is conducted in an iterative way by the construction and evaluation of UI *prototypes* driven by empirical principles and guidelines, such as proposed in *The Windows Interface: Guidelines for Software Design* [15] and *Apple Macintosh* [4]. There are many practical design tools for prototype and implement the UI, but most of them lack a theoretical foundation. In a development process it is important to match practical design methodologies with sound theoretical foundations because it provides models to predict and explain human-computer interactions phenomena.

There are various approaches, perspectives and models to study human-computer interaction and to provide methods to user interface design [6,8]. In all of them information is a fundamental concept. Information processing is well understood in computer science. However, the crux problem in human-computer interaction is how the mind processes it. The cognitive approach is the one that tries to provide cognitive models to information processing [6,7,16,20]. Cognitive models abstract physiological characteristics of the brain considering the mind (in isolation) as a linguistic and information processor. With these models it should be possible to predict or analyze how users will perform activities with computers. Psychology and Cognitive Sciences are the major source of theoretical knowledge for this approach. The most important contribution for a practical process of design is the user-centered perspective and the formalisms for user's and task modeling and analysis.

However, there are still other problems that the cognitive approaches do not solve. We are interested in two of them: (1) how to make the user learn the necessary knowledge of application functionality and

interaction model - the ***Application's Usability Model*** *(AUM)* - and (2) how to match the UI to be designed with user's previous knowledge of the domain and other applications. The AUM covers the interaction language - the set of interpretive and generative rules applicable to this artificial language - and the application's functionality - the open set of things that can be done with the designed software application.

Kammersgaard has proposed that among other perspectives computer systems can be viewed as a communication *medium* [13]. There are various facets in which we can use this communicative aspects of computers. One of them is the one-way communication from designers to users which takes place when an application designed by one person is used by other. Andersen treats the designer as having the role of one sender in a collection of senders, who makes a contribution to each message sent through the medium [1].

This and other recent approaches to design bring up *semiotics* as another source of theoretical knowledge to HCI [3,13,18,19]. Semiotics is the science of communication and culture. Culture and knowledge are manifested and shared (communicated) in a society by the means of *signs*. Semiotics provides a pragmatic perspective to design as a communicative process.

This social (pragmatic) perspective is also manifested in [1,24,23] which introduce (respectively) particular meanings to the concepts of *usability, software design* and *design languages*. Our work follows these concepts, specially the latter. A design language denotes the visual and functional language of communication with the people who use an artifact. It is the basis for how we create and interact with things in the world [23]. Whenever a designer develop an artifact, he/she is designing to a community with a shared design language. We are complementing these concepts with the theoretical foundation of semiotics.

In this work we will propose a framework that help designers in the production of the user interface signs. Our framework tends to be a practical design method founded in Peirce's definition of sign, in the more recent Eco's *Theory of Sign Production* [22,12], and in our own proposal of Semiotic Engineering [13]. The main point is that the communicated usability knowledge is learned by users interpreting messages coded in a well-defined ***User Interface Design Language*** (UIDL). The design language drives designers to express computable messages and codes to users by the computational medium. In Semiotic Engineering the user interface is a meta-communication artifact, the user interface design language is a special kind of code - *a computable hypercode* - to be defined in the following.

We are limited to the Microsoft Windows and Apple Macintosh WIMP (Windows, Icons, Menus and Pointers) standards for Graphical User Interfaces (GUI) for "office" applications on personal computers. GUI is a virtual or "on-screen" interface. Its elements are graphical images produced by software. The GUI is a *display,* the basic medium where all UI messages are conveyed. The keyboard and other physical devices also convey messages from design to users. Of course they are simple messages such as "press the key with the letter **a** on its surface to produce an **a** on the screen". So the hard keyboard can be seen as part of the display, or it can also be reproduced virtually on the GUI. This simplification does not affect our results.

THE SEMIOTIC ENGINEERING

Semiotic Engineering is a theoretically-based approach to user interface design in which user interfaces are taken to be *meta-communication* artifacts. By this we mean that interfaces are messages sent form designers to users and that such messages can, in their turn, send and receive messages [13]. It should convey a message from designer to system user whose meaning is the answer to two fundamental questions: (I) "What kinds of problems is this application prepared to solve?" - the application functionality - and (II) "How can these problems be solved?" - the interaction model. Semiotic Engineering specializes the semiotic approach and centers on the structure and expressiveness of messages one can send and receive in computable codes.

The above-mentioned problems of HCI are handled in Semiotic Engineering by (1) communicating the AUM through the UI and (2) deriving user's knowledge from its culture - analyzing users' tasks, knowledge and domain languages.

This is firstly a change of paradigm and priorities that should be complemented by the cognitive and practical approaches. Whereas the cognitive approach provides for an individual perspective which focuses on learning, understanding and other mental processes - no matter the intrinsic qualities of the object somebody is trying to understand - the semiotic approach provides for an interpersonal perspective focusing on interpretation and expression of meanings by scrutinizing communicative processes - no matter the individual subjective characteristics of people involved. They are complementary approaches to each other.

In Semiotic Engineering the focus is on the interpersonal communicative setting where the AUM should be shared between design and users. Computer systems offer a means to create a performing artifact where the interaction language and application functionality should be taught to users. Manual, help systems, training sections, tutorials and the like are meant to support the teaching/learning processes. However, interaction language grammars and application functionality should be part of a one-shot message sent throughout the collection of interface signs. This process occurs in a demonstration-like fashion where the designer is saying: *"Watch what I mean!"*.

The semiotic paradigm

Nadin is one of the first authors to apply semiotic concepts to UI design [18] based in Peircean semiotics. Andersen almost at the same time, developed a theory of computer semiotics based on Hjelmslev's structuralism tradition for semiotics [1]. Andersen's work covers not only UI design but the also

application design and programming. Our approach, Semiotic Engineering, was first presented in [13] based on Eco's Theory of Sign Production [12].

Semiotics is the discipline that studies signs, sign systems, signification, communication and all cultural processes [12]. In this work we consider Peirce's tradition in Semiotics [22] and some extensions to it. In our approach it is fundamental to understand Peirce's notion of sign and the sign production framework proposed by Eco.

A sign is defined by Peirce "as something which stands to somebody for something in some respect or capacity". A second important concept is the notion of *unlimited semiosis*: "A sign is anything which determines something else (its interpretant) to refer to an object to which itself refers (its object) in the same way, the sign becomes in turn a sign, and so ad infinitum." [22].

The sign itself is the product of a three-way relation between *representamen* (what represents), the *object* (what is represented), and its mental *interpretant* (the mental capacity that in the context of the sign process performs the substitution of representamen for object), as in Figure 1. [18]. "When we interpret a sign, we become part of it for the time of that interpretation" [18].

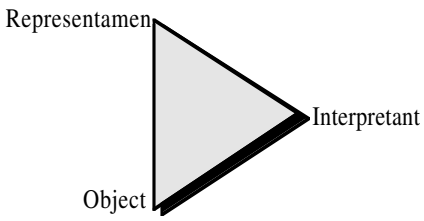


Figure 1: Peirce's Sign

Communication is the process that occur in a *context* where two or more agents (*a source* and a *receiver*) are interconnected by a *channel (medium)* to convey a *message* that denotes a *referent*. The message is a distinction or pattern of distinctions that is *produced* in the channel by a source and *consumed* by a receiver. A very important situation of communication is one where the message has a meaning, i.e., it can be interpreted by the agents as a sign. In order to warrant the interpretation of the produced sign, the communication process should be driven by a *semiotic system* or *code*.

Using Peirce's concept of sign [22], this model can be seen in another perspective. The message, the medium and context are the vehicle (*representamen*) of an abstract concept (*the interpretant*) which refers to another thing (*the object*). This triadic relation is a *sign* for someone (*the interpreter*). The code is the system that drives sign production and interpretation. The source is something or someone that produces this sign. This perspective is illustrated in Figure 2.

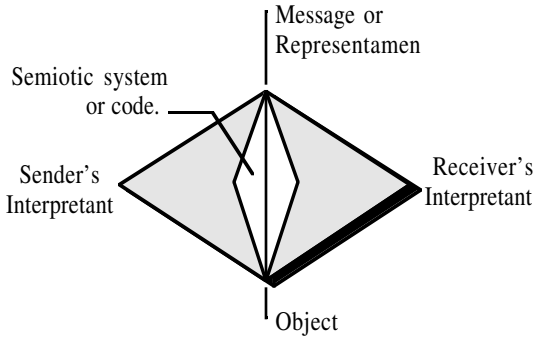


Figure 2: A communication perspective for Peirce's Sign.

Software Applications as Signs

Computers are electronic devices that gain functionality and usability by virtue of successive interpretations of patterns of electrical signals. A software application is an abstract or virtual machine that emerges from these patterns of electrical signals processed by an electronic computer. This *virtuality* is a **sign** whose meaning is interpreted by humans as its functionality. The software sign is composed by a collection of other signs. The electrical signals are the expression or *representamen* of the signs. In other words, signals should trigger interpretants in users. The electronic computer is the medium for it. The meaning or *interpretant* are the functionality someone (users or programmers) assigns to the executing program - the dynamic process of signals. The *object* of the sign is the range of interpretants someone can obtain from a software application.

Signs have their origin in programmers and users of computer application. Software is not interpreted by observing the electronic substance directly but by a cascade of interfaces that converts or de/codify signals into more intelligible expressions. Numeric keyboards, ASCII table and display, compilers and programming languages, and other are the interfaces that make these signs more easily interpretable. User interface devices are one subset of these converters to give a better usability of the machine. To program or to use of computers is an activity that involves the representation of several things. It is an activity that creates meanings by virtue of intensive sign production.

The Semiotic Engineering

In Semiotic Engineering the user interface is viewed as a meta-communication artifact (a meta-medium) that carries a message that itself is an expressive system for users and system to perform the interaction process. We call this message as ***User Interface Message***. The User Interface Message has four essential aspects [9]:

- The ***User Interface Message*** is a One-Shot Communicative Act from Designer to User
- The Designer's Intended Meaning and the User's Assigned Meaning for the User Interface Message are not the same, though they should be obviously consistent with each other.

- The Designer's Intended Meaning is an abstraction of the Application's Model - a ***Application's Latent Usability Model (ALUM)***
- The User's Assigned Meaning is a model of the ***Application's Achieved Usability Model (AAUM)***

The designer could send the *Application's Latent Usability Model* (ALUM) using several different media such as users manual, help systems ad so on. But as the user interface can be seen as a medium, designer should take it as the preferential one. The common problem of all these medium is that the designer normally can not send the User Interface Message directly to the users. It is a One-Shot Communicative Act. The designer just can send the message to the users by making the system present it during the interaction process. The whole interpretation process is triggered and determined by the interactions supported by the system's messages to users and corresponding reactions to mouse clicks, keyboard and voice input, data-glove motions, and other device signals. From the signs exchanged through I/O screens, users build a cascade of interpretants [22] that eventually crystallize into the user's assigned meaning for the user interface message - the Application's Achieved Usability Model [9].

In the UI meta-communicative setting, the designer is the source of the UI message and the user is the interpreter. UI lower-level messages, devices and the situation of use, are the representamen of the sign that should be produced. The user's interpretant is the AAUM. The object (the AUM) is the formally defined semantic model of the programmed application. Unlike what is proposed in more traditional semiotic approaches [22], in a computer environment the *object* of a sign is a formally defined tangible construct.

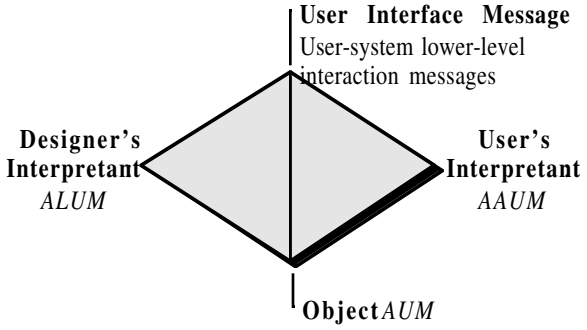
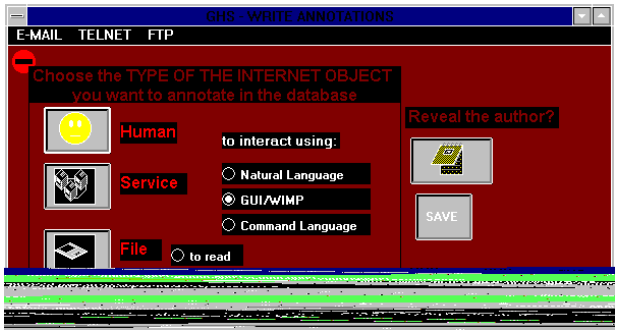


Figure 3: Semiotic Engineering meta-communicative setting.

A BRIEF CASE STUDY

In order to ground our new theoretical perspective in a design case study, we analyze two simple software applications. Our goal is to show how our framework can reveal UI design problems and how it guides the design of another prototype. These case studies also illustrates the very amount of abstract concepts with practical examples.



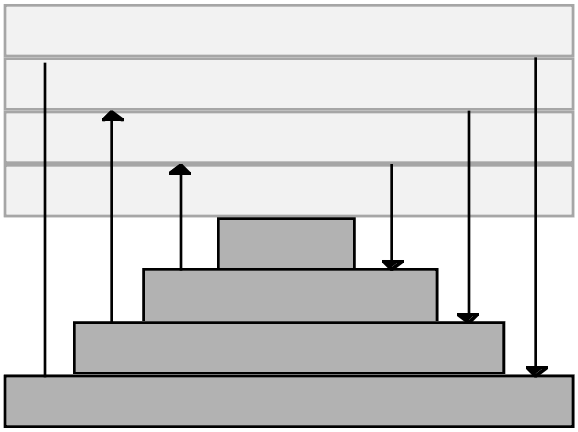
simplification. We know that it is necessary to have an expressive system for composing both designer's and application's message (computational results) that are sent to users. It will be more difficult to users if there are different semiotic systems in which these messages are encoded. A unique semiotic system for both designer's and application's messages is called for. The *design language* may perform this role. It should have meta-linguistic and inter-referencing capabilities to generate both messages in execution time. This is a meta-language that also defines the interaction system that users have to express their command when performing tasks. In this way, it determines both the output language and input language.

Eco's theory of codes explains a fundamental aspect to Semiotic Engineering. The interaction language is a well-defined (computable) code with interpretive and generative syntactic and semantic rules processed by the system. It is by using this code that the designer sends the message about the Application's Latent UM. In this case, this message is codified in a code that goes throughout the first. Since all the messages are presented to the user by the system it should be coded in a *hypercode* [12]. This hypercode is an articulation of codes and sub-codes. Our framework claims that we should define a ***User Interface Design Language*** (UIDL) as a *computable hypercode* - the one that is computable and allows the meta-communication function of user interface. It is necessary to define the computable sign-types of this semiotic system. In the next sub-section we present a model for user interfaces that provides a setting to the definition of the UIDL.

The UIDL is the semiotic system that allows the designer to compose a UI Message. This language should be specified over the layered model presented in the next sub-section. It defines a repertory of Interface Sign-types (IS-types) for messages production. The design language should be a tool to communicate the AUM, i.e., the knowledge users need to interact with the application. It structures the semantic of the User Interface Message (UIM) and correlates it with the expressions presented in interaction process.

What is proposed here is a more structured way to represent all the interface messages. The UIDL should evolve from a hypocode to a more well-defined code. GUI styles such as Apple Macintosh [4] and Windows [15] are not examples of UIDLs. They are not defined as having a vocabulary and a grammar. The UIDL should have a syntactic and a semantic system. The first defines all the UI messages and the latter, the application's semantic model, defines their meanings - the functionality and interaction model.

display medium. Using the executor, the user triggers the correspondent interaction of the IS. Users' arms and fingers, mouse pointers, light-pens are examples of executor mechanisms. Users' arms execute physical actions to navigate on the display medium and their fingers press the ISs that are in the display medium. It is important to note that the mouse pointer is not an IS and it is not part of display medium. It is like an extension of users' arm and fingers and it is part of executor mechanism. In GHS 2, the executor is the mouse pointer and users' fingers to type text to annotations.

The Performance Mechanism

The last element in our model is the ***Performance Mechanism***. This mechanism is the UI software that makes the interactions messages to be conveyed by ISs and where the languages - the UIDL and the consequent IL - are implemented. It should have a generator and a interpreter which have a defined set of