Semiotically Based User Interfaces Design Languages

Raquel Oliveira Prates+
Jair Cavalcanti Leite+*
Clarisse Sieckenius de Souza+

{raquel,jair,clarisse}@inf.puc-rio.br
+Departamento de Informática, PUC-Rio
*Departamento de Informática e Matemática Aplicada, UFRN

Abstract

Semiotic Engineering perceives interfaces as messages designers send to users. This message has an interaction model for syntax and the application's functionality for semantics. In order to create these messages, designers need models, tools and techniques that allow them to express themselves. In this article, we propose that User Interface Design Languages (UIDL's) in which designers can describe their message or part of it should be provided to them. We present two UIDL's that support the designer in distinct levels of the design process, one for single-user interfaces and the other for multi-user interfaces.

Keywords: Interfaces, semiotic engineering, design languages.

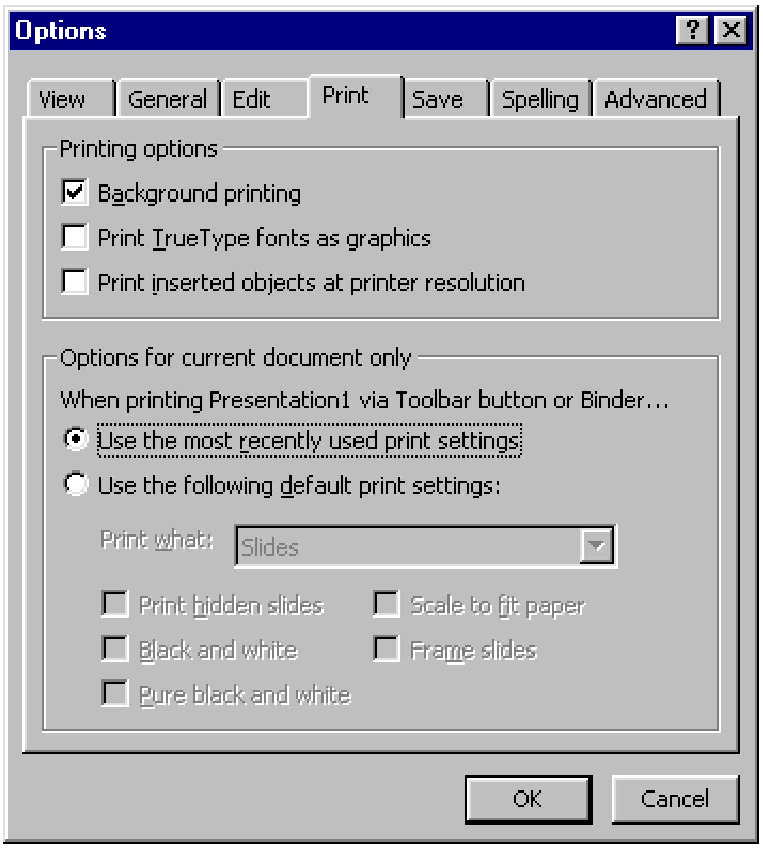
1. INTRODUCTION

User interfaces are the part of a computational system that users interact with. It is through this interaction that users grasp the functionality of the system. Therefore, the interface has to be carefully planned and designed in order to increase the chances that users understand it, and thus, that the application is successful. In a semiotic approach [14, 1] the interface is seen as a communication act between designer and users, using the computer as medium. In Semiotic Engineering [2, 3], particularly, it is perceived as a one-shot message being sent from designer to users. In this message the designer "tells" the users what he thinks the users' problem is and how to go about the application in order to solve it. This approach switches the design focus from "learnability" to "teachability" of interfaces.

As a consequence, one of the major goals of Semiotic Engineering at this point is to produce models and support the development of methods, techniques and tools that facilitate the design of interactive software. In order to achieve it, we are proposing two basic models: one for single-user and one for multi-user applications. In both cases, we associate specific Design Languages (DL's) and Heuristic Knowledge to the proposed models, so that together they can not only represent a certain design product, but also support decisions through the corresponding heuristic processing.

For instance, in single-user applications, a central part of the design task is to build the designer-to-user message to be conveyed through the user interface. As an example, let us consider the PowerPoint Print Option dialog [17] shown in Figure 1. This dialog is composed by 2 frames. The first frame has a title *Printing options* and contains 3 checkbox-and-label widgets. What the designer is telling the users is that (1) these widgets refer to features described by their labels, (2) they are related to each other, since they are spatially grouped within the same frame, which refers to a higher-order object (in this case, Printing Options), and (3) they are non-exclusive binary features, because check-boxes have only 2 states (checked and non-checked) and the state the user sets for each of them does not affect the others. The second frame, *Options for current document only*, holds two option-button with labels. Once again the labels refer to related features of a higher-order object. However, option-buttons are mutually exclusive, and the designer uses them to tell users that when printing they have to choose between the two options. Furthermore, below the second option-button there are some indented widgets that only become active when this option-button is chosen. This conveys to users the fact that the objects referred to by these widgets are only significant when the second option is selected.

In multi-user applications, if the designer affords to users a view of what a certain subgroup of people are doing, what he is telling all users is that there are (at least) two communities of users in the realm of this application, that one of them is more exposed than the other, and that the subgroup whose doings are not visible is at a higher level in the


Options

View | General | Edit | Print | Save | Spelling | Advanced

Printing options

[✓] Background printing

[ ] Print TrueType fonts as graphics

[ ] Print inserted objects at printer resolution

Options for current document only

When printing Presentation1 via Toolbar button or Binder...

[● Use the most recently used print settings]

○ Use the following default print settings:

Print what: Slides

[ ] Print hidden slides [ ] Scale to fit paper

[ ] Black and white [ ] Frame slides

[ ] Pure black and white

OK | Cancel

Figure 1: PowerPoint Print Option Dialog.

application's group structure than the other. For instance, in the Rio Internet TV [10] there are 3 kinds of users: the participants, the administrators and the super-administrators. We show the administrator's and the super-administrator's screens[1] in Figures 2(a) and 2(b), respectively. The pull-down menus make available to the administrators the actions they can take towards the participants. Notice that the super-administrator's list has one more option than the administrators. The Log option allows the super-administrators to see all the actions that have been taken by the administrators, therefore, providing them with information to supervise the administrators.

Models for single-user applications can benefit from a number of existing interface design patterns that are, de facto, a "code" in which the designer can express his message. Multi-user applications, however, lack an equivalent "code", and up to this date specialized widgets for groupware applications are still being proposed [7]. This explains why design models, design languages and heuristics, in both cases, are not applicable at the same stage of design.

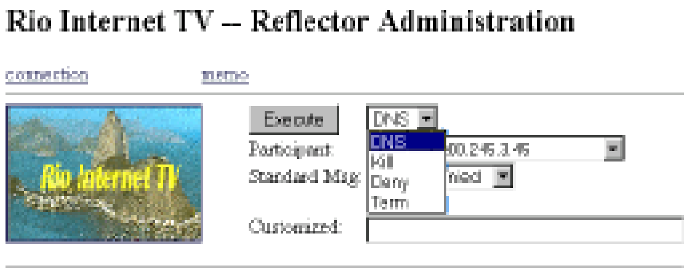
DL's and heuristic rules for single-user applications function almost analogously to stylistic and rhetorical principles over a language, a hypercode in semiotic terms [4]. But, because the "tradition" of groupware is still incipient, DL's and heuristic rules for multi-user applications function rather as a high-level grammar, organizing rationale over an undercoded candidate language (hypocode).

In the following we will illustrate features of models, heuristic rules and design languages for both types of interactive settings, and explain their semiotic underpinnings and expected effects.

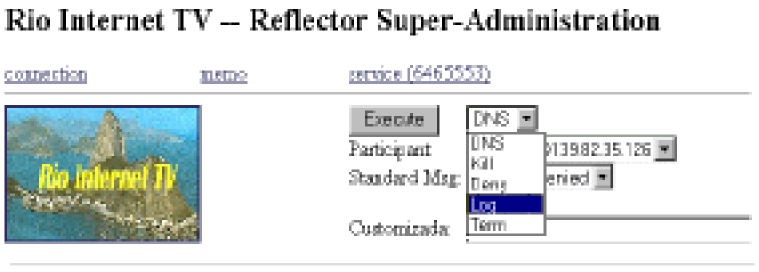
2. SEMIOTIC ENGINEERING

In Semiotic Engineering [2] the interface is perceived as a one-shot message from the designer to user. This message is actually a meta-communication act, since not only is the designer communicating with the users, but also the interface itself exchanges messages with the users.

[1]The screens of the Rio Internet TV have been translated to English by the authors with the consent and help of the owners.



(a) Administrator



(b) Super-Administrator

Figure 2: Rio Internet TV screens.

In this message the designer tells users his interpretation of the users problem and how they can interact with the interface in order to solve it. Notice that in the case of multi-user interfaces the designer must convey more information to the users, which are actually members of a group. The designer must include in the message the group's structure, the distribution of the tasks among members, the communication language and protocols among themselves and how their actions affect and are affected by the actions of other members [18].

For any communication process to take place (see Figure 3, the sender of the message must express his idea in a symbolic system (code) that both he and the receiver (or interpreter) know. The message being transmitted is formed by one or more signs - a sign being something that stands for something to someone [16]. As soon as the receiver gets the message, he generates an idea of what the sender meant, and starts making sense of it [9]. This idea that he creates is called an interpretant, and it can itself generate other interpretants in his own mind, in an indefinetly long chain of associations. This process is called unlimited semiosis [4] and goes on until either the interpreter believes he has a satisfying hypothesis of what the sender meant or he concludes that he is not able or willing to create a hypothesis (in which case he might or might not engage in further message exchanging with the sender).

In order to create his message to the users, that is the interface, the designer must first create a conceptual usability model of the application [3]. This model is the content of the message he is sending to the users. When the users receive this message, that is, start interacting with the interface, they start forming their interpretants, which are their acquired usability model of the application. The users assigned meanings for the interface will be different from each other's and from the designer's. However, the success of the communication depends on the users' meanings being consistent with the designers' and in multi-user interfaces with each other's. Figure 4 depicts the designer's communication to a single user through the interface.

3. THE DESIGN CHALLENGE

The designer is faced with the challenge of creating a conceptual usability model of the application and then building an interface that communicates to the users this model. To increase the users' chances of understanding the message being sent by the designer through the interface, it is important that designers express themselves clearly.

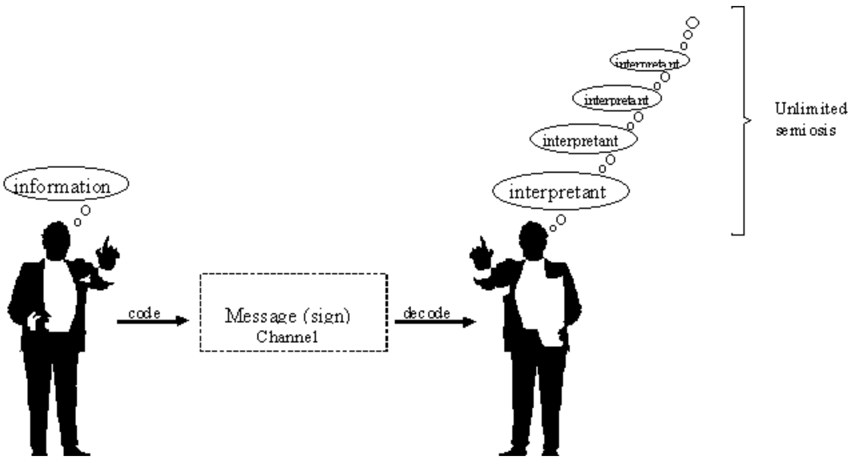
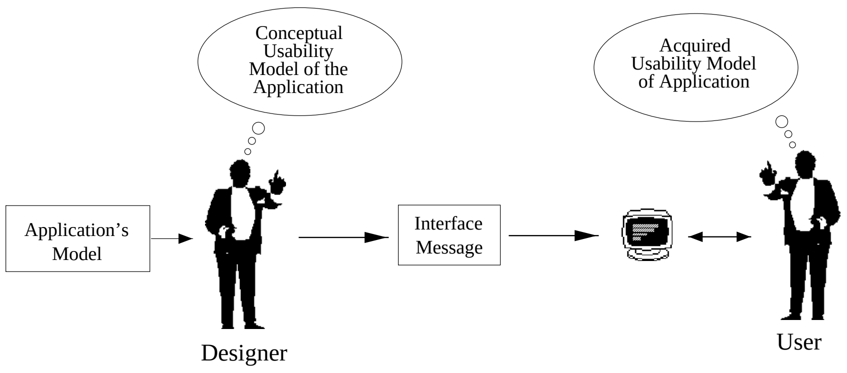


Figure 3: Communication process.



Conceptual Usability Model of the Application
Acquired Usability Model of Application
Application's Model
Interface Message
Designer
User

Figure 4: Designer's communication to a single user through the interface.

The generation of this message to the users has a planning step and a "phrasing" step[2]. In the planning step the designer defines what he is going to say, whereas in phrasing he defines how he is going to say it. The content of the message depends on the application's domain and results from the designer's conceptual model of the application. If the message is intended for a group, its content also includes the designer's model of the group.

When deciding how he is going to phrase his message to the users, the designer first defines which code he is going to use. A code is a system that has a well defined syntax and semantics. The syntax is composed by a vocabulary and a set of grammatical rules that define how the items of the vocabulary are combined and structured into phrases. Examples of codes the designer could use to express his message are command languages, WIMP (Windows, Icons, Menus Pointers) standards used for GUI's[3], among others. In this article we will adopt the WIMP standards as the basic code.

The choice of WIMP standards characterizes widgets as the vocabulary available to the designer. This means that in the phrasing step, the designer has to define which widgets to use, how to combine them creating interactive structures and their layout on the screen. By doing so, he creates the system's interaction model, which is the message's syntax. He also has to relate each one of these widgets and interactive structures to content units defined in the planning step. These content units compose the system's functionality, which is the message's semantics. It is this relation defined between the system's interface syntax and semantics that make it possible for users to grasp the functionality of the system by interacting with it.

Whereas in single-user interfaces the WIMP standards represent a code to be used by designers, in multi-user interfaces it is in fact a hypocode – a set of symbols that is not yet well defined and established as a code [4]. This is due to the many new forms of interactions required by group work and, thus, the creation of new types of widgets

[2]In text generation literature, these 2 steps are known as planning and realization steps [15].

[3]GUI stands for Graphical User Interfaces.

[6, 5]. We can regard the vocabulary for multi-user interfaces currently as being formed by single-user widgets, adapted widgets and new widgets [8].

In short, the designer's task is to define which are the content units he is going to express, which vocabulary and combinations he is going to use to express himself, and finally how to relate the syntactic and semantic units he has defined. Therefore, it is necessary to provide designers with models, tools and techniques that support each one of these activities. In the next section we present User Interface Design Languages for single-user interfaces and for multi-user interfaces that allow the designer to express himself about a specific part of the design process and guides him towards consistency.

4. USER INTERFACE DESIGN LANGUAGES

User Interface Design Languages (UIDL's) are languages in which interface design can be expressed. We can take UIDL's as meta-codes that define specializations over an existing code, that is, as hypercodes[4] [4]. Since there are many steps in the design process, the spectrum of possible languages that can be created to support the designer's task is wide.

Nowadays, the support designers have available amounts to toolkits, frameworks and guidelines. Graphical toolkits for single and multi-user interfaces provide the designer with widgets, means to combine them and to create an interface. Guidelines and frameworks direct the designer towards issues to be considered during interface design. Nonetheless, none of them help the designer plan and define his message, understand what is each widget's preferential interpretant, nor how to combine them and create interactive structures for an intended meaning.

In this section we present a single-user interface design language (SUIDL) and a multi-user interface design language (MUIDL) that allow the designer to express part of the steps of designing an interface. The SUIDL is to be used by the designer when creating the syntax of the message, that is, the interaction model, and its relation to the content units of the semantics. The SUIDL supplies the designer with a widget vocabulary and their preferential meaning, semantic categories for these widgets, heuristic rules to combine them to achieve an intended meaning, and a formalism to define their layout. Figure 5 shows how SUIDL assists the designer in the creation process of the interface. The SUIDL provides support at the phrasing level, whereas the MUIDL is at the planning level. The MUIDL allows

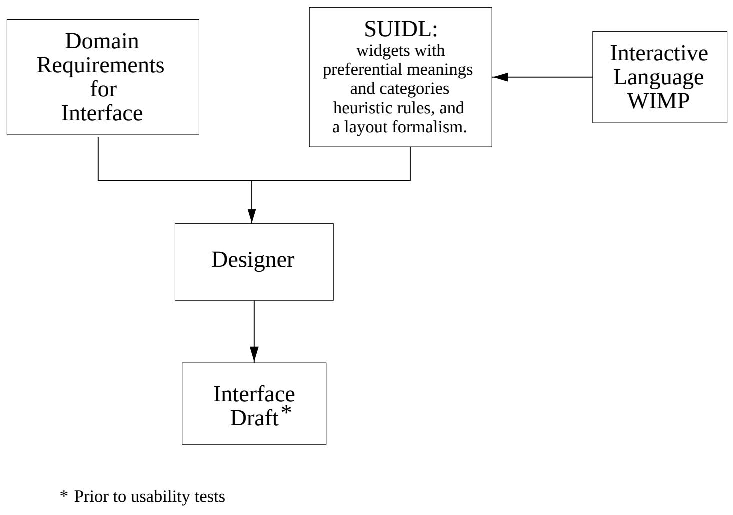

Domain Requirements for Interface
SUIDL: widgets with preferential meanings and categories heuristic rules, and a layout formalism.
Interactive Language WIMP
Designer
Interface Draft*
* Prior to usability tests

Figure 5: Design process assisted by SUIDL.

the designer to plan and conceive the collaboration and communication models of the group by assigning values to group features and combining them according to heuristic rules. It also provides some guidance into the phrasing level by giving some hints on the widgets to be chosen by the designer. Figure 6 depicts how MUIDL is integrated to the creation of a groupware interface. By comparing Figures 5 and 6 the different levels of expression in which they assist the designer become clear. Notice that in the multi-user interface design, the step that corresponds to SUIDL's level is

[4]As we will see in Section 4.2. hypercodes over undercoded systems are actually grammars.

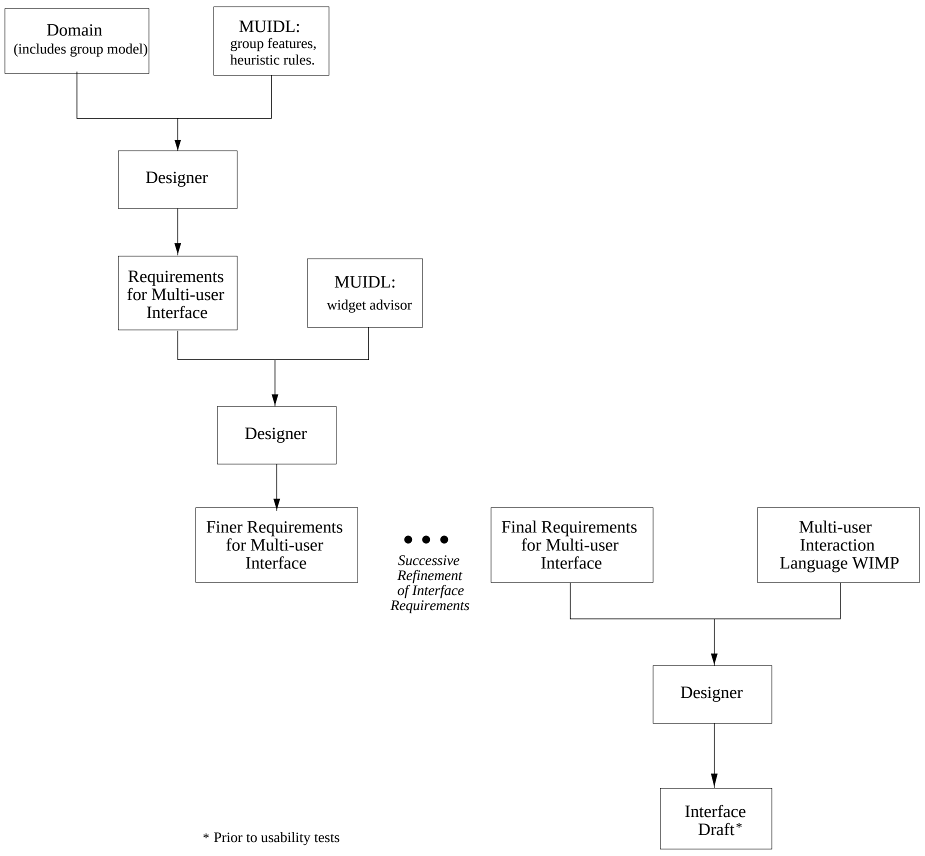

Domain
(includes group model)
MUIDL:
group features,
heuristic rules.
Designer
Requirements for Multi-user Interface
MUIDL:
widget advisor
Designer
Finer Requirements for Multi-user Interface
Successive Refinement of Interface Requirements
Final Requirements for Multi-user Interface
Multi-user Interaction Language WIMP
Designer
Interface Draft*
* Prior to usability tests

Figure 6: Design process assisted by MUIDL.

when the designer creates the interface draft, based on the final interface requirements and the multi-user interface interaction language. However, the difficulty of creating a MUIDL for this level comes from the fact that the vocabulary of the multi-user interface language is not yet an established or closed set.

4.1. Example 1: SUIDL

The goal of our SUIDL is to assist the designer in the creation of the interface interaction model. It supports the designer's expression by providing him with the "words" (widgets) and combination rules he can use to create the message, as well as with a layout formalism that guarantees the grammaticality of the interface.

The vocabulary provided to the designer is composed by a basic set of widgets that includes buttons, labels, text-boxes, check-boxes, radio-buttons, lists, menus, windows and so forth. For each widget the SUIDL provides expressive properties that include color, shape, size, and so on, and semantic properties that are expressed in terms of activation and revelation functions. The activation function tells the user how to interact with the widget in order to communicate with the system, whereas the revelation function defines how the messages from the interface relate to the content units. Orthogonally to these functions, the designer must also define the interaction and functionality represented by the widget. To illustrate this, a description of a button would be:

Interface sign type: BUTTON

Expressive properties:

shape: retangular
size: small and horizontal
color: gray
title: name of application function

Semantic properties:

Activation:

interaction: click on it

functionality: trigger action

Revelation:

interaction: light gray: active, dark gray: inactive

functionality: activates function described in title

The activation of the widgets define their preferential interpretant. For instance, when a button is clicked it triggers an action, and therefore, it should be used when the designer intends to give the user a choice of which action to trigger or when to do it. Our SUIDL classifies the widgets according to their semantic properties, allowing the designer to relate categories to the applications content units, before defining which specific widget to use. The widgets categories are: controls, objects, recipients and environments [11].

Control category is formed by widgets that allow the user to trigger an action or set new values, and includes buttons, radio-buttons, check-boxes and scroll-bars. Objects are widgets that can be used to represent the application's elements, such as icons and labels. Recipients are interactive structures that contain other objects or recipients, as is the case of text-boxes, lists, text-editors, canvas and tables. Finally there are the environments which are widgets that can contain control widgets and produce interactive structures, such as frames and windows.

Based on this vocabulary and categories we have created a set of heuristic rules. These rules relate widgets and interactive structures to their preferential meaning. Therefore, they "guide" the designer in his choice of "words" and "sentences" according to the message he wishes to tell the users. In order to give a flavor of these rules, we present some that guide the designer's choices of dialog boxes.

```
if you (designer) use a modal window then
   you are telling users to narrow their focus on a specific dialog.
end if
if you use a MDI[5] window with certain objects in it then
   you are telling users that all of these objects refer to the same theme.
end if
if you use non-modal window to hold different objects then
   you are telling users that objects in that window belong to the same context (objects in different windows belong
   to distinct contexts).
end if
if you use a message box without buttons then
   you are informing users of something.
end if
if you use a modal message box with one button then
   you are informing users of something and giving them control of when to execute the next step.
end if
if you use a modal message box with $n$ buttons then
   you are telling users to make a choice of which one of the $n$ possible steps is the next.
end if
```

Once the designer has defined the widgets he wants to use at each moment, he can then use the Display-based Recursive Transition Network (DRTN) to create the interface [12]. The DRTN is a formalism that allows the designer to define the layout of the chosen widgets and interactive structures on the screen and then automatically generates the grammar that will underly the interface's interaction model.

This SUIDL supports the designer through the whole phrasing step: choosing widgets and interactive structures, combining them based on their semantic properties, that is, the message they convey to users, and finally arranging them on the screen. Although the revelation x functionality property allows the designer to relate the widget to a content unit, our SUIDL does not contrain this relation. Only a domain dependent UIDL would support the establishment of such relation [13].

[5]MDI stands for Multiple Document Interface

4.2. Example 2: MUIDL

Our MUIDL aims at supporting the designer in the planning and specification of the communication and collaboration group models. It can be considered context-separable insofar as it defines group features and their combination intensionally and extensionally independently from the domain. The binding to the domain is made by the designer during the group modeling process (see Figure 6).

The solution space is structured by defining certain group features that have to be set by the designer. The first features that designers must define are the roles that will be assumed by members of the groups and the classes of objects that members will interact with. The designer must then define who are these members and their hierarchical structuring, as well as the actual objects and whom they belong to. The designer goes on to define the interdependency among members' tasks and who can see what and talk to whom. As the designer describes each one of these features for the group, he circumscribes the solution space and narrows down a specific collaboration and communication model.

Such group features are related to each other by heuristic rules, which assign meaning to featural combination. As the designers set values to the features, rule checking processes point out at potential inconsistencies. The rules can only identify **potential** inconsistencies, since domain dependent knowledge can always overrule generic heuristics. The set of rules comprises a high-level grammar over undercoded semiotic systems.

Coupled to the rules there is a heuristic knowledge base, which holds the design rationale of our proposed rules. Whenever a potential inconsistency is identified the knowledge base gives the designer an explanation about why it is so. The designer is allowed to override the inconsistency, and in this case he is prompted to enter an explanation for his decision.

As an example of the heuristic rules, we present the rules that relate the abilities of members to act upon, see and talk about objects. In order to define which members can interact with which objects, the designer declares the objects as being private (owned[6] by only one member), shared (owned by more than one member) and public (owned by all members).

```
if members can act upon, can see and can talk about an object then
  you (designer) have defined a highly collaborative setting for the group.
end if
if members can act upon, can see, but cannot talk about an object then
  if object is private then
    you have defined the object as being a confidential object.
  else if object is shared or public then
    you are refraining members from discussing and planning actions relative to the object.
  end if
end if
if members can see, but cannot act upon or talk about an object then
  you have not allowed members to use the information in a collaborative way, although it is OK if the information
  influences their decisions on their own work.
end if
if members can act upon, but cannot see or talk about an object then
  this does not make any sense, since users would have to act upon an object they could not see.
end if
if members cannot act upon, but can see and talk about an object then
  although members cannot act upon the object, you have allowed them to exert some influence on the actions upon
  that object.
end if
if members cannot act upon or see, but can talk about an object then
  although you have not provided members with knowledge about the object, you have allowed them to talk about
  it. This situation may foster the opportunity for members to be "nosy".
end if
if members can neither act upon, nor see or talk about an object then
  you have defined the object as being out of the scope of their work.
end if
```

[6]Owners of objects are the only ones allowed to act upon them.

To illustrate how the rules and knowledge base would work imagine that the designer defines object *O* as being shared by members *A*, *B* and *C*. Automatically these members can act upon and see the object. If the designer further defines that members *A* and *B* can talk about *O*, but *C* cannot, then the rule checking would identify the potential inconsistency and call the knowledge base for an explanation. The explanation the designer will receive tells him that although all members own *O*, only *A* and *B* can talk about it and therefore, *A* and *B* can plan their actions upon *O*, whereas C is left out. Furthermore, this could cause *C* to unknowingly interfere in the plans of *A* and *B*.

After using our MUIDL to model the group, the designer has as a result a set of consistent[7] requirements that defines the group, its members and their working (communication and collaboration) models. Moreover, he would have the design rationale for these requirements that would be formed by the explanations for the rules and any explanation he might have entered for an overriden rule. Once this set is defined, the designer can input it to the widget advisor module, which generates sets of characteristics that should be (or not) present in the widgets to be chosen to represent each object. This module is the first step towards the designer's phrasing of his message.

5. FINAL REMARKS

In this article we presented some of the major challenges faced by the designer during the creation of an interface. By using semiotic theory as framework, we were able to switch to a complementary stand among current HCI approaches and stress the designers' need for means to express themselves.

We propose that designers should be provided with user interface design languages. To illustrate our point we have presented both a single-user and a multi-user interface design language that support the designers planning and phrasing of his "text" in the interface language, the system's interface. However, it is easy to see that each supports different aspects of the design process. Whereas the SUIDL supports the designers expressive choices and layout, the MUIDL supports the designers specification of the group's working model to be expressed in the interface. They are both steps in the direction of developing an environment in which the designer may be able to plan, specify and express the interface message.

The UIDL's make available to the designer a set of heuristic rules that assign semantic properties to the designer's choices when creating the interface. We must point out that this set is a descriptive set, as opposed to a prescriptive one, inasmuch as they allow the designer to override inconsistent rules and ultimately to be in charge of the design process.

Finally, we would like to argue that semiotically-based UIDL's bring out the fact that designers are building communicative artifacts in a specifically designed linguistic environment. Because software artifacts, in their turn, are not but layers upon layers of linguistic representations and manipulations, we believe that the sort of coupling these UIDL's offer with software engineering itself is highly promising.

Acknowledgments

Clarisse S. de Souza and Raquel O. Prates are thankful to CNPq, and Jair C. de Leite is thankful to CAPES for the support we have received for this project. We would also like to thank Prof. Hugo Fuks and Carlos Laufer for their help with the Rio Internet TV example.

REFERENCES

[1] Andersen, P. B. Holmqvist, B. and Jensen, J. F. "The Computer as Medium". Cambridge Univ. Press, 1993.

[2] de Souza, C. S. "The Semiotic Engineering of User Interface Languages". International Journal of Man-Machine Studies (1993) 39, 753-773.

[3] de Souza, C. S. "The Semiotic Engineering of Concreteness and Abstractness: From User Interface Languages to End User Programming Languages". Dagstuhl Seminar on Informatics and Semiotics. Feb., 1996.

[4] Eco, U. "A Theory of Semiotics". Indiana Univ. Press, 1976.

[7]The MUIDL can only guarantee the consistency of the set of requirements when the designer complies to the rules.

[5] Greenberg, S. and Roseman, M. "Groupware Toolkits for Synchronous Work". Trends in CSCW, John Wiley Sons Ltd. Ed. M. Beaudouin-Lafon.

[6] Grudin, J."Groupware and Social Dynamics: Eight Challenges for Developers". CACM, Jan. 1994, Vol. 37, No.1, 93-58.

[7] Gutwin, C., Greenberg, S. and Roseman, M. "Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation". Proc. of HCI'96.

[8] Hazemi, R. and Macaulay, L. "Requirements for Graphical User Interface Development Environments for Groupware". Interacting with Computers, Vol. 8, No. 1, 1996, 69-88.

[9] Jackobson, R. "Linguística e Comunicação". Ed. Cultrix. São Paulo, 1970. Capítulo 4, 73-87.

[10] Laufer, C. and Fuks, H. "Distributed Administration - Service Integration on The World Wide Web". WebNet 97, Toronto, Canada, November 1997.

[11] Leite, J. and de Souza, C. "A Framework for the Semiotic Engineering of User Interface Languages". MCC10/97, Informatics Department of PUC-Rio. Rio de Janeiro, 1997.

[12] Leite, J. "Um Framework para a Engenharia Semiótica de Linguagens de Interfaces de Usuário". Forthcoming Ph.D. thesis, Departamento de Informática, PUC-Rio, 1998.

[13] Martins, I. "Um Instrumento de Análise Semiótica para Linguagens Visuais de Interfaces". Ph.D. thesis, Departamento de Informática, PUC-Rio, 1998.

[14] Nadin, M. "Interface Design: A Semiotic Paradigm". Semiotica 69 - 3/4 (1988), 269-302.

[15] Paris, C., Swartout, W., Mann, W. "Natural Language Generation in Artificial Intelligence and Computational Linguistics". Kluwer Academics, 1991.

[16] Peirce, C. S. "Collected Papers". Harvard University Press. 1931-1958.

[17] Perspection Inc. "Microsoft PowerPoint at a Glance". Microsoft Press, 1997.

[18] Prates, R.O., de SOUZA, C.S. and GARCIA, A.C.B. "A Semiotic Framework for Multi-User Interfaces". SIGCHI Bulletin, April 1997, 28-39.