

Designing and Evaluating Interaction as Conversation: a Modeling Language based on Semiotic Engineering

Simone Diniz Junqueira Barbosa, Máira Greco de Paula

Departamento de Informática, PUC-Rio
Marquês de São Vicente, 225 / 4^º andar RDC
Gávea, Rio de Janeiro, RJ
Brazil – 22453-900
{simone,mgreco}@inf.puc-rio.br

Abstract. A number of design models have been proposed in the area of Human-Computer Interaction (HCI) to support user-centered system design. High-level, abstract task models and detailed interface specification languages are among the most widely used. However, the need for designing applications to run in a number of different devices and platforms presents new issues that must be addressed from a platform-separable perspective. In this paper, we show how an interaction-as-conversation metaphor may face this challenge, and present an interaction modeling language that allows designers to build a blueprint of the range of interactions that will be able to take place in the application. Our goal is twofold: to motivate the designers to reflect upon the interactive solution they are creating, and at the same time provide a skeleton interaction specification that may be easily instantiated for different platforms or devices.

Keywords: interaction modelling, interaction-as-conversation, HCI design models, semiotic engineering

1 Introduction

A number of design models have been proposed in the area of Human-Computer Interaction (HCI) to support user-centered system design [14], among which scenarios and task models are the most widely used. The value of using scenarios in software development has been investigated for almost a decade [5]. In the area of human-computer interaction (HCI), it has been extensively used [6], especially in the preliminary stages of software design.

A few attempts have been made to use scenarios as a representation for bringing HCI aspects to software engineering processes [11]. However, as the collection of natural language scenarios gets larger for a single application, it becomes harder to make consistent decisions about what must be done. In addition, due to frequent ambiguities found in natural language texts, there are too many decisions to be made in

moving from scenarios to software specification. These decisions are seldom recorded for future reference or verification of the final product. Unfortunately, more often than not, what is developed is very different from what has been represented in the scenarios.

Task models attempt to bring some structure to HCI design and analysis. Typical goals of task modelling are: to ease the understanding of an application domain, to record the results of interdisciplinary discussions, to design new applications consistently with the conceptual model of users and to analyze the usability of interactive systems [15]. Some task models favor the understanding of a domain, such as GOMS [4], TKS [12], MAD [20] and GTA [22], while others focus on application design and specification, such as TAG [18], CLG [13] and CTT [15]. When analyzed with respect to their scope and level of detail [10], they seem inadequate to represent the interactive solution itself. Either their scope is too broad, in that they try to address in a single representation, dimensions we believe should belong to different models, such as everything from user goals to system specification, or their level of detail is too high and abstract to provide a concrete specification of the solution being built.

Another class of HCI models comprise what is typically called an interaction model, such as UAN [9]. Interaction models generally privilege certain interface styles and technologies over others, limiting their applicability across environments and devices. Also, when modeling an interactive solution too closely to the technological environment, designers lose sight of the “big picture”, and thus are prone to introduce inconsistencies in their design.

We argue that, in order to maximize the benefits brought about by using scenarios and task models, we need to reduce the gap between these representations and the actual software specification. For this purpose, we propose to create and use an interaction model that allows the representation of a design solution in such way as to give designers resources to reflect on it both globally and under certain perspectives that focus on few dimensions at a time. It must represent interaction at an adequate level of abstraction, so that it is neither too abstract to represent important design decisions (at an abstract task level, for example), nor too close to the device or software environment in which it will be actually implemented (at a concrete user interface specification level, for instance), which would hinder the design for multi-platform applications.

In order to provide this kind of support for designers to make appropriate decisions about interactive solutions, we have devised MoLIC, a “Modelling Language for Interaction as Conversation” [1, 16]. MoLIC is rooted in Semiotic Engineering [8], a theory of HCI which views the user interface as a metamessage sent from designers to users. This message is created in such a way as to be capable of exchanging messages with users, i.e., allowing human-system interaction. In Semiotic Engineering, interaction design is viewed as conversation design. This conversation is of a unique kind, however, because the designer is no longer there at the time it is carried out. This means that he/she must anticipate every possible conversation that may occur, and embed in the system his/her “deputy”: the designer’s spokesman in the system, with whom users will communicate during interaction. MoLIC supports the creation of this deputy, i.e., the design of the whole range of interactions that may take place

in the application. The idea underlying MoLIC is to support the designers' reflective and decision-making processes about their design throughout the design process [21].

The next section describes the role of scenarios in Semiotic Engineering, as providing the content that should be expressed by the designer's deputy in the user interface. Section 3 presents the three main representations comprising MoLIC, and which help designers' plan and envision the user experience: goal diagram, ontology of signs, and interaction diagram. Finally, in section 4, we discuss the role of MoLIC in supporting model-based design of human-computer interaction under a reflection-in-action perspective.

2 Scenario-Based Design and Semiotic Engineering

One way to support the creation of the designer-to-user message is to help designers express *what* they want to say, before going into *how* the designer's deputy will say it. This what-how coupling may be achieved by using scenarios [5] together with MoLIC.

Scenarios can be used throughout the development process, starting from the analysis of the domain and the users' tasks and characteristics. A major goal of using scenarios at this stage is to explore or confirm, together with the users, the designers' understanding of the goals and tasks to be supported. By means of scenarios, designers not only learn about the users' vocabulary and thought processes, but they have a chance to make this knowledge explicit in a language users fully understand, so it can be verified by users themselves. Scenarios are also very useful in uncovering and exploring typical and atypical courses of action in specific circumstances.

It is important to notice that designers should avoid including in the early scenarios references to concrete interface elements, such as texts and widgets. By avoiding an early commitment and raising user's expectations about the interface solution that will be adopted, both designers and users will be open to explore alternative solutions at later stages.

One of the major advantages of using scenarios in HCI is to provide a detailed setting for better understanding and conceiving contextualized pieces of user-system interaction. Taken from another perspective, however, the contextualized and detailed nature of scenarios may be, to some extent, a drawback in HCI design. When using scenarios alone, designers may have difficulty in grasping a global view of the application as a whole system, and also in understanding the interrelations between the tasks and goals it should support. This limitation hinders the design of a coherent communicative artifact, an essential requirement for the Semiotic Engineering of Human-Computer Interaction.

In order to fill this gap in HCI design, we propose to complement scenarios with MoLIC. MoLIC was conceived to be applied between the initial analysis and the interface specification phases. It has shown to be useful in helping designers grasp a global view of the conversations that comprise the application, and thus design a coherent designer-to-users message, keeping in mind the communicative role of the designer's deputy.

MoLIC may be viewed a first step towards user interface specification. It allows the representation of users' goals and steps to achieve them, as well as the information to be exchanged between users and the designer's deputy at each step. The focus is always on the communicative capacities of both user and designer's deputy: interaction occurs only as a result of the communication between these two agents.

When writing scenarios, the designer should have in mind what the purpose of each scenario is: to investigate certain aspects of a domain or socio-cultural environment, to propose a task structure or an interaction solution, to contrast alternative task structures or interaction solutions, and so on. These purposes should be made explicit when creating scenarios. The designer should annotate each scenario with the questions that should be answered by the users' feedback about the scenario, and also the design assumptions and possible interactive solutions related to a few envisaged answers to these questions.

Another benefit from asking questions based on scenarios is to uncover hidden information, so that the signs¹ (*what*) and conversations about signs (*how to*) are made explicit. In Semiotic Engineering, investigating "how to" means not only supporting users in how to *manipulate* signs, but also how to *understand* them in *context*. In addition, we propose to include a set of questions to investigate the motives underlying each decision the user will be making during interaction (*why*), in an attempt to capture the design rationale. The answers to these questions will provide valuable information for the designer to proceed. Some of these answers may generate further questions, which in turn may give rise to unanticipated scenarios. Viewed under this perspective, this approach is similar to systematic questioning as illustrated in [7].

As a running example, we will use a banking application, in which users may check their account balance and statement, as well as transfer money to another account from the same bank. A sample scenario would be the following:

Carol needs to transfer the rent money to Anna, her roommate. But first she needs to check if her salary has already been deposited. She enters the banking system [Q1], provides her account number and password [Q2], and receives the account balance [Q3]. She realizes there is more money than she expected, and decides to check her account statement about recent transactions by selecting the corresponding operation from the available options [Q4]. By the last few transactions presented [Q5], Carol realizes the check she wrote the day before hasn't yet been withdrawn, and her salary has been deposited. She then proceeds to transferring the money [Q4, Q6]: she chooses Anna's account from a preset list of frequent transfer recipients [Q2, Q8], provides the amount to be transferred, and confirms the operation with her password [Q2, Q7]. She gets the printed transfer confirmation to give to Anna [Q5], checks her balance once again [Q3] and leaves the system.

Some of the questions regarding this scenario excerpt (together with some of the expected answers) might be:

¹ The most widely used definition of signs in Semiotics is: "anything that stands for something to someone" [19]. We use signs here to stand for the whole range of representations used by designers and users, from conceptual information to user interface elements.

Q1: From where can the system be accessed? (ATM, website, phone, WAP)
Q2: What information is necessary to use the system? (account number, password, random personal information for security reasons)
Q3: How can one check his balance? (upon request, automatically upon entering the system, upon finishing a transaction, ...)
Q4: What are the available operations? Are all of them available at once? If so, how are they categorized? If not, what is the criteria for making each operation available?
Q5: How can users check the result of an operation? (output content, format & media)
Q6: Are there shortcuts from one operation to another?
Q7: The information provided in one operation needs to be provided again in another, during the same session?
Q8: Should the system “learn”, i.e., reuse information from one session to another? (personalization mechanisms)

From the scenarios, users' goals are identified and may be structured in a hierarchical goal diagram, as seen in the next section.

3 Using MoLIC in HCI Design

MoLIC comprises three interrelated representations: a diagram of users' goals, an ontology of the domain and application signs, and an interaction model. All of them are derived from the scenarios. In brief, the goals diagram represents the range of goals users may want to achieve using the application; the ontology of signs represent the concepts involved in achieving these goals (either by being manipulated by the user or provided as a resource for the user to make appropriate decisions during interaction); and the interaction model represents when and how the signs are actually manipulated to achieve the goals.

3.1 User Goals

The first step in using MoLIC is to extract, from the scenarios, the top-level goals users may have when using the application, i.e., goals that are explicitly supported by it. These goals are then organized in a hierarchical diagram, according to some classification the designer finds useful: what are the classes of users that will form this goal; how primary is the goal with respect to the system scope definition (or whether it is just a supporting goal); the frequency with which the goal is expected to be achieved, and so on.

Some user's goals extracted from the sample scenario are, for instance, “checking account balance”, “checking statement of recent transactions”, and “transfer between accounts”. These goals are part of a larger diagram, partially illustrated in Fig. 1:

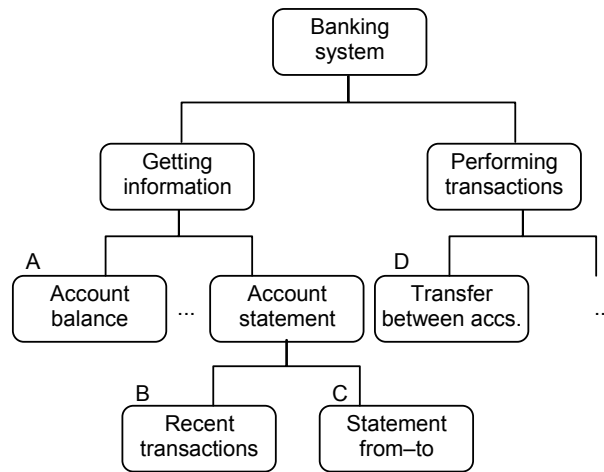


Fig. 1. Partial goals diagram.

As we will see in the next section, for each identified goal, a piece of interaction model is built in such a way as to be interrelated with the paths of interaction corresponding to other goals, aiming to form a coherent whole.

3.2 Domain and Application Signs

The next thing to extract from the scenarios are the domain and application signs that are meaningful to users. While these signs are usually treated as data, in Semiotic Engineering they acquire a new status, going further than establishing the vocabulary to be shared between designer's deputy and users. Defining domain and application *sign systems* help designers uncover users' expectations about how knowledge should be shaped and built. Signs allow designers to establish what the system is all about, and how users will be able to understand and use it.

A widespread example of a simple sign system is the use of ontologies to organize domain and application concepts [3]. A designer should be able to straightforwardly derive an ontology from usage scenarios. However, this is seldom the case. Scenarios are often incomplete and ambiguous, and it is hard to keep the whole set of scenarios consistent and coherent. Thus, in order to build ontologies that define the application signs, designers should explore as many dimensions or classifications of signs as necessary to grasp their full meaning.

As in typical data classification, signs may be grouped according to the kind of information they may have as a value. The most simple classification is probably the one that distinguishes only between textual and numeric values. In order to be useful for HCI design, this classification needs to be complemented with knowledge about the range of values these signs may assume. For instance, a user interface element used for providing unconstrained textual information is different from that for providing a piece of textual information that belongs to a known set of values.

An interesting classification is related to the degree of familiarity to a sign users are expected to have. In this classification, domain signs are those directly transported from the users' world, such as "full name". Application signs, on the other extreme of the scale, are those that were introduced by an application, and have no meaning outside it. Still in this classification, there is an intermediary kind of sign: a transformed sign is derived from an existing sign in the world, but has undergone some kind of transformation when transported to the application. This transformation is often related to an analogy or metaphor.

The reason for this classification to be interesting in HCI is that different kinds of signs may require different kinds of user interface elements to support users. In general, a domain sign would require an explanation only inasmuch there are constraints imposed by the application. For example, the concept of "full name" is clear to users, but a restriction about the number of characters allowed in the corresponding piece of information might not be, and thus need some clarification from the designer's deputy. A transformed sign would require an explanation about the boundaries of the analogy. For example, a folder in the desktop metaphor might require an explanation about its "never" getting full, except when the hard disk which contains it lacks space. At the end of the scale, an application sign may require a complete explanation about its purpose, utility and the operations that can manipulate it. An example might be a sign for "zooming" in a graphics application.

There are of course some signs that can be classified in either group. For example, a password may be thought of as a transported sign, derived from the existing domain sign: signature. In these cases, it is the designer's responsibility to decide, based on the analyzed characteristics of users and their tasks, the amount of support to provide in order to have users fully understand each sign.

It is important to note that users may become familiar with certain signs in one application and then transport this knowledge to another application. When this is done unsuspectingly, however, it may cause unpredictable distortions in the users' conceptual model of the latter application.

There is also the typical input/output classification, which establishes who will be responsible for manipulating the sign at a certain point during interaction: the user or the system (via the designer's deputy). This classification, however, changes during interaction, according to the user's current task, and thus may be considered a task-dependent property of the sign. Task-dependent signs will be explored in the next section, in which we describe MOLIC's interaction notation.

After having established these diverse sign classifications, designers can follow traditional ontology engineering techniques to represent the acquired knowledge.

3.3 Interaction Modeling

From the user-approved scenarios and their corresponding signs, HCI designers have enough elements to build an interaction model and thus shape the computational solution.

When interaction is viewed as conversation, an interaction model should represent the whole range of communicative exchanges that may take place between users and

the designer's deputy. In these conversations, designers establish *when* users can "talk about" the signs we extracted from the scenarios. The designer should clearly convey to users *when* they can talk about *what*, and what kinds of *response* to expect from the designer's deputy. Although designers attempt to meet users' needs and preferences as learned during user, task and contextual analysis, designing involves trade offs between solution strategies. As a consequence, users must be informed about the compromises that have been made. For instance, MoLIC allows the representation of different ways to achieve a certain result, criteria to choose one from among them, and of what happens when things go wrong.

MoLIC supports the view of interaction as conversation by promoting reflection about how the design decisions made at this step will be conveyed to users through the interface, i.e., how the designers' decisions will affect users in their perception of the interface, in building a usage model compatible with the designers', and in performing the desired actions at the interface. This model has a dual representation: both an abbreviated and an extended diagrammatic views. The goal of the diagrammatic interaction model is to represent all of the potential conversations that may take place between user and system, giving designers an overview of the interactive discourse as a whole.

The interaction model comprises scenes, system processes, and transitions between them. A scene represents a user-deputy conversation about a certain matter or topic, in which it is the user's "turn" to make a decision about where the conversation is going. This conversation may comprise one or more dialogues, and each dialogue is composed of one or more user/deputy utterances, organized in conversational pairs. In other words, a scene represents a certain stage during execution where user-system interaction may take place. In a GUI, for instance, it may be mapped onto a structured interface component, such as window or dialog box, or a page in HTML. In the diagrammatic representation, a scene is represented by a rounded rectangle, whose text describes the topics of the dialogues that may occur in it, from the users' point-of-view (for instance: *Identify account*). Fig. 2 illustrates the representation of a scene.

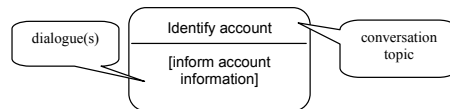


Fig. 2. Diagrammatic representation of scene "Identify account".

A system process is represented by a black rectangle, representing something users do not perceive directly. By doing so, we encourage the careful representation of the deputy's utterances about the result of system processing, as the only means to inform users about what occurs during interaction.

Transitions represent changes in topic. This may happen due to the user's choice or to a result in system processing. Transitions are represented by labeled arrows. An outgoing transition from a scene represents a user's utterance that causes the transition (represented by a bracketed label, such as u: [check account statement]), whereas an outgoing transition from a process represents the result of the processing as it will be "told" by the designer's deputy (represented by a simple

label, such as: `d: invalid account number or password`). In case there are pre-conditions for the transition to be made, they should come before the transition label, following the keyword `pre:`. Analogously, the `post:` keyword after the label is used to mark a post-condition, or a new situation or application state that affects interaction.

Some scenes may be accessed from any point in the application, i.e., from any other scene. The access to these scenes, named ubiquitous access, is represented by a transition from a grayed scene which contains a number following the letter U, for “ubiquitous”. Moreover, there are portions of the interaction that may be reused in various diagrams. Stereotypes are created to represent parameterized interaction diagrams, represented by a rounded rectangle with double borders (such as `View(account statement)`).

In Semiotic Engineering, error prevention and handling are an inherent part of the conversation between users and system, and not viewed as an *exception*-handling mechanism. The designer should convey to users not only how to perform their tasks under normal conditions, but also how to avoid or deal with mistaken or unsuccessful situations. Some of these situations may be detected or predicted during interaction modeling. When this is the case, we extend the diagrammatic representation with breakdown tags, classifying the interaction mechanisms for dealing with potential or actual breakdowns in one of the following categories:

- Passive prevention (PP). In this solution, the designer’s deputy tries to avoid user errors by providing online instructions or documentation. For instance, what is the nature of the information expected (and not just “format” of information), which users have access to the system.
- Active prevention (AP). Here the designer’s deputy will constrain users’ actions, actively preventing them from occurring. For instance, tasks that will be unavailable in certain situations, such as “transfer between accounts” when there is insufficient balance in the user’s current account. In the interface specification, this may be mapped to making widgets disabled depending on the application status or preventing the user to type in letters or symbols in numerical fields, and so on.
- Supported prevention (SP). These are situations which the designer’s deputy detects as being potential errors, but whose decision is left to the user. For instance, in the user interface, they may be realized as confirmation messages, such as “The remaining balance will be insufficient to pay for scheduled bills. Proceed with transfer?”)
- Error capture (EC). These are the errors that the deputy can identify and that notify to users, but for which there is no possible remedial action within the system. For instance, when there is insufficient disk space to save a certain information file.
- Supported error handling (EH). The most frequent situation: errors that should be corrected by the user, with support from the designer’s deputy. For instance, presenting an error message (such as “Invalid account number or password, please try again.”) and an opportunity for the user to correct the error (for example, taking the user directly to the interaction context in which he/she provided the incorrect information and is able to correct it).

The signs that make up the topic of dialogues and scenes may also be included in an extended representation. When this is the case, we adopt the following convention: when a sign is uttered by the deputy, i.e., presented to the user, it is represented by the sign name followed by an exclamation mark (e.g. *date!*); whereas a sign about which the user must say something about, i.e., manipulated by the user, is represented by a name followed by an interrogation mark (*account number?*, *password?*). Fig. 3 illustrates the extended representation of a scene².

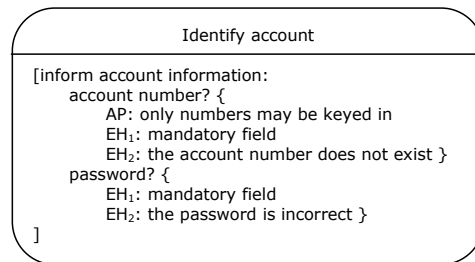


Fig. 3. Extended representation of scene “Identify account”, including signs and contextualized breakdown tags.

Fig. 4 presents a diagrammatic representation of the interaction model for the goals identified in the sample scenario.

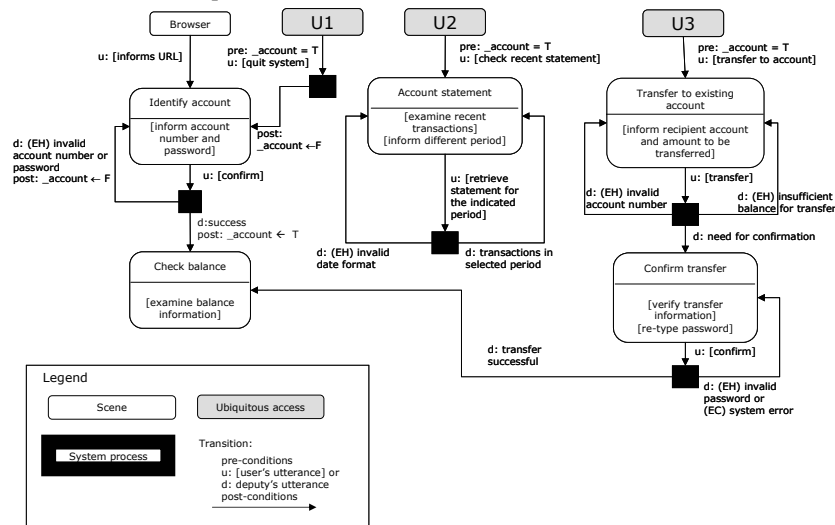


Fig. 4. Sample abbreviated interaction diagram.

² The detailed specification of the extended representation is found in [16].

In order to illustrate the benefits of using MoLIC, we now exemplify two kinds of decisions made based on the diagrammatic interaction model, in two distinct situations:

Example 1: balance upon request

In the first design solution, the user would not be presented with the account balance after identifying the account or finishing a transaction. In this case, he was required to explicitly request the account balance as any other operation (Fig. 5a). This solution, however, took no advantage of the fact that the initial and final account balances are the most common information sought after in banking systems. Therefore, the revised solution was to present the account balance after every user action, including the account identification (Fig. 5b).

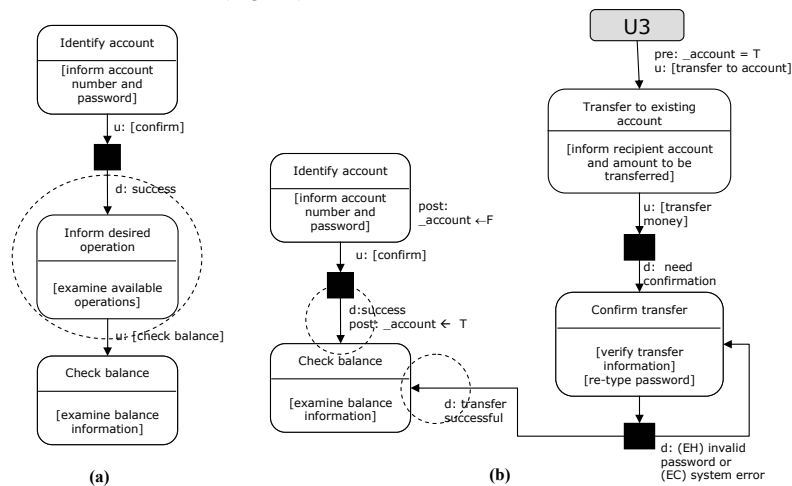


Fig. 5. Example 1 – Alternative interaction solutions for checking the account balance.

The interaction model does not present details about interface or technological environment. However, designers must take into account such factors when making design decisions, for the characteristics of the computational environment in which the system will execute may determine whether a certain interaction solution is adequate or not, as will be seen in the next example. In other words, the *design decisions* depend on the interface style and technology, but not the *interaction representation*.

Example 2: platform-dependent issues

ATMs have different security requirements from banking applications on the web. In particular, users may walk away from an ATM without closing their session. If the web application is assumed to be accessed from the user's private desktop, on the other hand, it is less likely that an open session will be left unattended. Therefore, for certain critical transactions, such as transferring money to different accounts, the interactive solution may differ from one platform to the other. In the sample scenario, we may consider it unnecessary to re-type the password in the confirmation scene in a desktop application (Fig. 6a), whereas it is needed in the ATM (Fig. 6b).

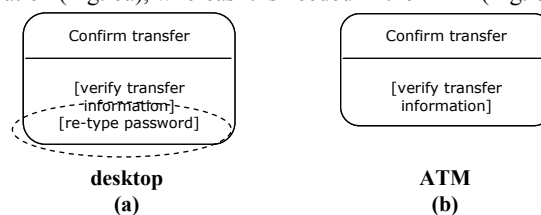


Fig. 6. Example 2 – Alternative interaction solutions indicating different strategies for confirming an operation, based on the platform in which the system will be instantiated.

The design decisions made during interaction modeling may be presented to users for their appreciation. It may be necessary to generate new scenarios that reflect the proposed solutions, exploring the distinctions between them.

4 Discussion

The goal of the work presented in this paper is to support design by using representations that favor the domain and task understanding (mostly scenarios), as well as the reflection about alternative solutions (mostly interaction model).

The use of HCI models in interactive systems design is essential for allowing designers to reflect about their solutions from the initial steps of the project on, and not just as a post-implementation evaluation. However, current approaches usually tend to use complex task models which —sometimes awkwardly— intermingle issues of a different nature without calling attention to the fact that many diverse factors are at play which could be better analyzed from different perspectives. This makes it hard to understand and predict the effects of a certain solution. We believe it is best to use simpler representations, each with a well-defined focus: scenarios focusing on establishing a common ground for mutual understanding designers and users, and MoLIC focusing on the discourse structures being designed. This way, both semiotically-enriched scenarios and the proposed interaction model are epistemic design tools. They support Schön's *reflection in action* epistemology [21] in that they explicitly represent dimensions of semiotic classification for communication-centered design. The centrality of communication and the importance of conversational affor-

dances are justified by the intellectual nature of software, be it in the traditional format of desktop applications or in contemporary applications for mobile devices. In both, *signs* (visual, textual, gestural, aural, etc.) must be used to convey ideas, create conventions and facilitate usage. The proposed epistemic tools are so formatted that the construction of storyboards and prototypes should be as straightforward as in the case of other popular notations.

As we have said, interaction design in the semiotic engineering perspective is concerned with building a coherent and cohesive message, focusing on the users' understanding of the designers' conception of the application. We believe that, if the designer's assumptions and design decisions are conveyed successfully, with few communicative breakdowns or misunderstandings, i.e., through a well-designed user interface, users will not only make better use of the application, but also have more resources to use it creatively when unanticipated needs arise.

Semiotically-enriched scenarios support communication with users, allowing for the identification, exploration and verification of the application's purposes as far as designers and users are concerned. Signs bring attention to what is known and unknown, what remains the same and what is changed, what new language(s) or conventions must be learned, and so on. And the interaction model expands a blueprint of the potential conversations that may be carried out between the user and the designer's deputy during interaction. They help designers gain a sense of what are *all* and *the only* conversations (or communicative exchanges, to incorporate non-textual interaction) that the designer's deputy is equipped to entertain.

A decisive factor in conceiving MoLIC was to keep it as independent of specific user interface issues and technological platform as possible. This consideration not only facilitates the reuse of models, but also avoids that decisions about the user interface are forced to be made prematurely, making it harder for designers to explore different alternative solutions.

MoLIC has been used in the design and redesign of interactive systems in different environments (mostly Web and Windows applications), by different teams of designers (from researchers to programmers) and in undergraduate class assignments. We have gathered some informal indications that they succeed in promoting the designer's reflection about alternative design solutions, offering better support for the design team's decision-making processes, before proceeding to the interface specification, storyboarding and implementation. Although reflection occurs when any design model is used, existing models generally allow the representation of solutions at too low an abstraction level, where important aspects of the solution are omitted, such as the representation of alternative or failure courses of action, and the communicative exchanges that take place between user and designer's deputy. In particular, this representation of "conversation" allows designers to perceive more clearly the effect of their design decisions, and thus makes it easier to determine which solution is adequate for the situation at hand.

We are currently investigating the use of MoLIC for building HCI patterns [17] and for modelling synchronous multi-user environments. Also a study about the integration of MoLIC with system specification models used in Software

Engineering in underway, aiming at augmenting HCI design quality without causing a negative impact on the software development cycle [2].

Acknowledgments

Simone D.J. Barbosa thanks CNPq for providing financial support to this work. Maira Greco de Paula thanks CAPES for the scholarship granted for her M.Sc. program. Both authors thank their colleagues at the Semiotic Engineering Research Group at PUC-Rio for their valuable comments on the ideas presented in this paper.

References

1. Barbosa, S.D.J.; de Souza, C.S. ; Paula, M.G. (2003) "The Semiotic Engineering Use of Models for Supporting Reflection-In-Action". *Proceedings of HCI International 2003*. Crete, Greece.
2. Barbosa, S.D.J.; Paula, M.G. (2003) "Interaction Modelling as a Binding Thread in the Software Development Process". Workshop "Bridging the Gaps Between Software Engineering and Human-Computer Interaction", at *ICSE 2003*. Oregon, USA.
3. CACM (2002) Ontology: different ways of representing the same concept. *Communications of the ACM*, Volume 45, Issue 2 (February 2002)
4. Card, S., Moran, T. e Newell, A. (1983) *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum.
5. Carroll, J. M. (ed) (1995). *Scenario-based design: envisioning work and technology in system development*, New York, Wiley.
6. Carroll, J. M. (ed) (2000) *Making use: Scenario-Based Design of Human-Computer Interactions*. The MIT Press. Cambridge, MA.
7. Carroll, J.M.; Mack, R.L.; Robertson, S.P.; Rosson, M.B. (1994). "Binding Objects to Scenarios of Use", *International Journal of Human-Computer Studies* 41:243-276.
8. de Souza, C.S., Barbosa, S.D.J., da Silva, S.R.P. (2001) "Semiotic Engineering Principles for Evaluating End-user Programming Environments", *Interacting with Computers*, 13-4, pp. 467-495. Elsevier.
9. Hix, D. and Hartson, H. (1993) *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons.
10. Hoover, S.P.; Rinderle, J. R.; Finger, S. (1991) "Models and abstractions in design", *Design Studies*, 12-4. October, 1991. pp. 237-245.
11. Imaz, M. & Benyon, D. (1999) How Stories Capture Interactions. *Proceedings of IFIP TC.13 International Conference on Human-Computer Interaction, Interact'99*. pp. 321-328.
12. Johnson, P., Johnson, H., Waddington, R., Shouls, A. (1988) "Task related Knowledge Structures: Analysis, Modelling, and applications", *Proceedings of HCI'88*, Cambridge University Press.
13. Moran, T. (1981) "The Command Language Grammars: a representation for the user interface of interactive computer systems". *International Journal of Man-Machine Studies* 15:3-50.
14. Norman, D. e Draper, S. (eds., 1986) *User Centered System Design*. Hillsdale, NJ. Lawrence Erlbaum.

15. Paternò, F. (2000) *Model-Based Design and Evaluation of Interactive Applications*, London, Springer-Verlag.
16. Paula, M.G. (2003) “Projeto da Interação Humano-Computador Baseado em Modelos Fundamentados na Engenharia Semiótica: Construção de um Modelo de Interação” (in Portuguese). Master dissertation. Informatics Department, Pontifícia Universidade Católica do Rio de Janeiro.
17. Paula, M.G.; Barbosa, S.D.J. (2003) “Bringing Interaction Specifications to HCI Design Patterns”. Workshop “Perspectives on HCI Patterns: Concepts and Tools”, at *CHI 2003*. Florida, USA.
18. Payne, S. e Green, T.R.G. (1989) “Task-action grammar: the model and its developments”. In D. Diaper (ed.) *Task Analysis for Human-Computer Interaction*. Chichester: Ellis Horwood.
19. Peirce, C.S. *Collected Papers*. Cambridge, Ma. Harvard University Press. (excerpted in Buchler, Justus, ed., *Philosophical Writings of Peirce*, New York: Dover, 1955) 1931.
20. Scapin, D. e Pierret-Golbreich, C. (1989) “Towards a method for task description”, *Proceedings of ‘Work with Display Units’ Conference*, Montreal, Canada, Elsevier.
21. Schön, D. (1983) *The Reflective Practitioner: How Professionals Think in Action*, New York, Basic Books.
22. van der Veer, G.C., Lenting, B.F. e Bergevoet, B.A.J. (1996) “GTA:Groupware Task Analysis - Modeling Complexity”, *Acta Psychologica*, 91, 1996, pp. 297–322.