Conveying Human-Computer Interaction Concerns to Software Engineers Through an Interaction Model

Maíra Greco de Paula, Simone Diniz Junqueira Barbosa, Carlos José P. de Lucena
Departamento de Informática, PUC-Rio
R. Marquês de São Vicente, 225
Rio de Janeiro, RJ, Brazil – 22451-900
{mgreco, simone, lucena}@inf.puc-rio.br

Abstract

This paper addresses the challenge of efficiently representing and communicating decisions about human-computer interaction to collaborate with software engineers. It describes and illustrates in a case study how an interaction modeling language based on the semiotic engineering of human-computer interaction may be used to derive a skeleton of certain UML diagrams, namely: use case, class, and sequence diagrams. Our goal is to provide a clear representation of the interactive exchanges that may take place, in order to prevent human-computer interaction decisions to be lost or inadvertently overruled when designing the system architecture and internal functional behavior.

Categories & Subject Descriptors: H.1.2 [Models and Principles]: User/Machine Systems

General Terms: Design, Documentation, Human Factors

Keywords: HCI models, interaction modeling, UML, HCI-SE integration

INTRODUCTION

It is well known that the development of interactive systems requires the involvement of professionals from distinct disciplinary backgrounds. They bring distinct perspectives on problem framing, on design issues, and on evaluating the impact of alternative solutions on the users' daily lives and work practices. Typical professionals involved in software design are: psychologists, requirements engineers, software engineers (SE), human-computer interaction (HCI) professionals, and graphic designers. The communication among them throughout the development process usually takes place mediated by textual reports and scenarios, model diagrams, visual sketches, and prototypes. These representations contain information that must be shared by all the involved professionals.

HCI professionals, who have the goal of designing systems with high quality of use, taking into account users' needs and preferences, need to convey information to the software engineers, whose main activity is to analyze and design the system's internal architecture and functionality. This information must describe how the application should behave, from the users' point of view, so that users may achieve their goals efficiently and with satisfaction. In this paper, we call this information the user-centered application semantics. Once this semantics is agreed by all design team members, software engineers should proceed with their share of design and specification, modeling the system functionality using their own representation tools, such as the Unified Modeling Language (UML) [15], making sure the resulting software respects the HCI concerns that were conveyed to them.

The representations used to convey information between HCI professionals and software engineers are usually scenarios, use cases, storyboards, and working prototypes. Paula and co-authors have argued that such representations are not adequate for bridging the two areas [10]: not all HCI concerns are directly conveyed through these artifacts, and some important HCI decisions about the user-centered application semantics are lost or need to be reconstructed by deeply (and costly) analyzing the artifacts. The main problem reported in their study is the difficulty to obtain an overall view of the application behavior. In other words, these representations give a fragmented and somewhat disconnected view of smaller portions of the system. For instance, each scenario represents a usage situation. To gain an understanding of how a scenario may interfere with another one, it is necessary to read all related scenarios and infer the interferences that may take place, making it difficult to get a global view of the application semantics and expected behavior. And if relevant HCI decisions are hidden in inadequate representations, software engineers may be unable to relate their own software design decisions to the HCI decisions made previously, and this lack of understanding may cause them to make inconsistent decisions which will lower the quality of use of the final product. For instance, interaction paths related to error correction or shortcuts for users to achieve a certain goal may be inadvertently overruled by architectural decisions, if software engineers are not aware of the rationale underlying the interaction design.

Some researchers are working to bridge the gap between HCI and software engineering research and practice [17, for instance]. In most of their work, they aim to better support the communication between HCI and SE by

proposing novel representations or extensions to existing representations, such as UML.

This paper also focuses on the communication between HCI professionals and software engineers. Our goal is to provide a means of communication that efficiently conveys the decisions made by HCI designers about the interaction to inform software engineers in their activities of software design and specification. We also need to allow software engineers to convey their decisions on matters that influence or constrain the interaction design. These decisions and constraints should, during the analysis and design stages, be negotiated by all the different professionals, in order to obtain a shared understanding and agreement on the final solution, via communication and collaboration (Figure 1).

In particular, this work considers that the main target audience for the HCI concerns and decisions are software engineers who will design and specify the software following an object-oriented approach, which is the most widely used paradigm in software engineering today.

In this work, the representation chosen to help foster communication among these professionals was MoLIC (acronym for "Modeling Language for Interaction as Conversation"). MoLIC is a language used to represent an interaction model devised to support designers in reflecting about the solution being conceived [12]. It was chosen as a boundary object because it provides a blueprint of the apparent behavior being designed, from the users' point-of-view [1]. We believe MoLIC makes it easier to establish relationships between the external (apparent) software behavior defined by HCI professionals and the internal software behavior and architecture defined by software engineers.

This paper is organized as follows: in the next section, the paper describes some related work and their limitations. In the third section, the paper describes MoLIC and the theory of HCI that underlies this work. The fourth section relates MoLIC to the UML notation for object-oriented design, highlighting possible derivations intended to make the HCI-SE communication more efficient, and describes a small case study. The paper is concluded by a discussion and some considerations about the proposed approach.

RELATED WORK

The research work that aims to integrate HCI aspects with OO modeling may be grouped in:

- extending UML to include user interface modeling;
- using models that represent HCI aspects and help guide OO modeling; and
- creating a method that includes OO and HCI models to specify the concrete user interface.

UML Extension

Silva (2002) proposed UMLi – Unified Modeling Language for Interactive Applications –, a UML extension to encompass user interface modeling [14]. He describes a survey of some Model-Based User Interface Development Environments (MB-UIDEs) to investigate which user interface models are described in these environments. He identified four kinds of models:

- application model: describes the application attributes that are relevant to the user interface, in terms of classes, attributes, and their relationships;
- task-dialogue model: describes the users' tasks and the relationships between them;
- abstract presentation model: describes the visual structure of the user interface in a conceptual way; and
- concrete presentation model: describes in details the visual aspects of the user interface.

From these kinds of models and a case study in which a library system was modeled using UML, the author

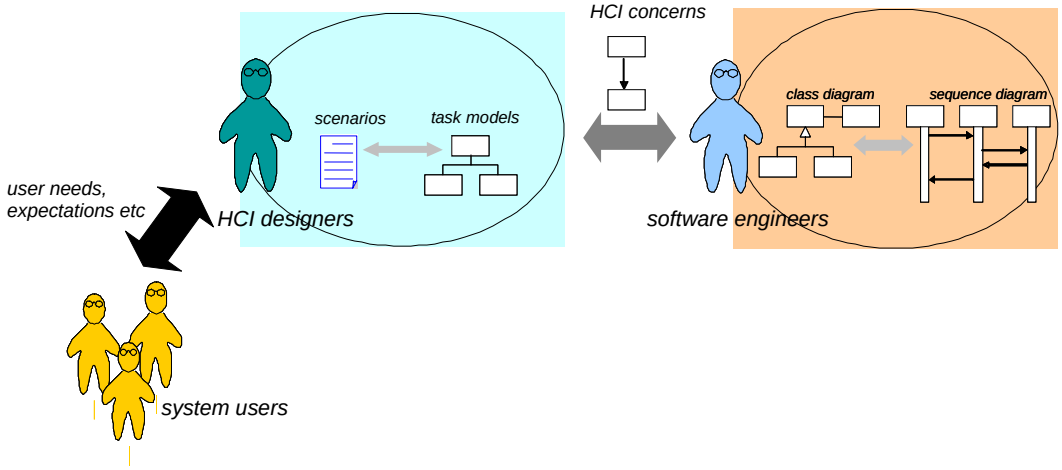

Figure 1. Envisaged communication between HCI designers and software engineers.

investigated the difficulties involved in modeling user-interface related aspects, and then proposed an extension to the UML meta-model and some of its models, creating thus the UMLi.

Nunes (2001) also extended the UML to include HCI aspects [9]. In his work, he intended to insert concepts of usability engineering [8] into software engineering. In order to do this, he proposed Wisdom – Whitewater Interactive System Development with Object Models –, a software development method that comprises a user-centered development process, an architecture that brings new models into UML to support HCI modeling (user, interaction, dialogue and presentation models), and a set of notations that extend UML based on the Wisdom method

Kruchten et al. [7] proposed to extend use cases to provide the necessary information for user interface design using UML. The authors proposed to use "use case storyboards", a conceptual and logical description of how a use case should be "realized" through the user interface. This storyboard is represented in UML through a <<use case storyboard>> collaboration. Each use case storyboard contains basically: a high-level textual description of the user-system interaction related to the use case; sequence and collaboration diagrams describing how the use case will be "realized" at the user interface in terms of the collaboration between objects and actors; a description of all of the usability requirements that need to be taken into consideration; and additional explanations about the creation of a user interface prototype. The authors described how the use case storyboards may contribute to the construction of both prototypes and the final user interface. They did so by defining some steps towards building a prototype from the elements in the use case storyboards.

There are a few problems in these proposals to integrate HCI and OO modeling by bringing HCI into UML. First, one of the goals of HCI design under many perspectives (user modeling, task modeling, interaction modeling, and so on) is to support the designers' reflection in conceiving the interactive system focusing on HCI concerns. When the language used in these models is targeted mostly at system specification – which is the case with UML – instead of its earlier conception ideas, the support to the HCI designers' reflection and decision-making processes about conceptual and practical HCI concerns is hindered. Second, HCI designers don't always know the OO paradigm. So, when HCI models turn into OO models, this may thwart these professionals, who will either have to learn the OO paradigm or will be unable to adequately use the languages to correctly represent their decisions in an OO-HCI model. Last, since UML is a language used by software engineers, its extensions to include HCI aspects may give the false impression that software engineers are fully qualified to handle HCI-related decisions. Thus, these kinds of work don't highlight the importance of having HCI professionals in the software development team.

Models that represent HCI as a guide for OO modeling

Constantine and Lockwood (2001) proposed the use of essential use cases to guide user interface design and traverse the gap between usability engineering and software engineering [4]. Essential use cases comprise an abstract description of the problem's essence, generalized and independent of technology. They don't contain user interface details nor internal software structures. They are based on the users' intentions, and visually separate these from the system responsibilities. The authors suggested to include tags in the use case narratives to indicate, for instance, objects, classes, and methods, in order to facilitate the later construction of OO models. Essential use cases represent the user-system interaction, and an interaction map relates the essential use cases through the following kinds of relationships: inclusion, specialization, extension, similarity, and equivalence.

Our work is in line with this kind of research, i.e., to keep HCI models and use the information contained therein to inform the construction of the OO models and diagrams. The major difference is that, when they create their interaction map, they stop focusing on the user-system interaction and instead highlight the relationships between use cases in terms of software architecture concerns. This shift in perspective hinders an overall understanding of the application's behavior, from the user's point of view.

As we will see in the next sections, in using MoLIC our work provides a map of the user-system interaction following the interaction-as-conversation metaphor. It always focuses on the possible interaction paths as experienced by the users, never from a system perspective.

Rosson and Carroll (2001) described how usage scenarios may contribute to the construction of object models, providing information such as: objects, their relationships and responsibilities [11]. According to them, while the system's usage scenarios are refined throughout the design, the object model may become more complex, and the object-oriented user interface design may take shape.

Scenarios are good artifacts for communication, because they provide context and are written in natural language. However, each scenario represents only part of the system. While we agree that scenarios may contribute to HCI-OO modeling integration, providing the kinds of information mentioned in [11], we believe they should be complemented by other HCI models that highlight the relationships between scenarios, and thus provide additional information that is relevant to OO systems design.

OO design method for creating the user interface

Van Harmelen (2001) presented the Idiom design method, which included techniques for user interface specification through object modeling [16]. The author presented a framework that described the use of HCI and OO models throughout the user interface design process. The models

he used were: scenarios, task models, domain models (representing the objects extracted from scenarios and their relationships), core model (representing only the objects and associations that were of interest to users), and view model (provided an abstract view of how the user would interact with the system).

After building all these models, the user interface was designed. This design contained the concrete representations of the core model objects. When the user interface design was ready, a prototype was built, and then the OO analysis and design of the internal software system could be done. All of the models plus the concrete user interface might be used as a resource for the remaining development stages.

Van Harmelen proposed the use of OO models to design the user interface. As with the proposals that extend the UML, HCI professionals need to learn OO to model the user interface using the Idiom method. A drawback of this approach is that the author didn't make it clear what the benefits from modeling the user interface following an OO perspective were. Also, he didn't describe in detail how the models built in Idiom might collaborate with or serve as concrete resources for building the software engineering models.

A SEMIOTIC ENGINEERING INTERACTION MODEL

This work is grounded on semiotic engineering, a theory of HCI that characterizes human-computer interaction as centrally communicative phenomena and provides an ontology from which HCI frameworks and models can be derived [5]. Semiotic engineering adopts a media perspective on the use of computer applications, investigating the designer-to-user metacommunication, i.e., how the signs engineered into the user interface tell the users how to communicate with the application to achieve the intended effects. The content of the designers' message is who they believe users are, what they have interpreted as being the users needs, values, and preferences, and how they implemented their vision in this interactive system to allow users fulfill their actual goals. It is important to underscore the interpretive nature of the designers' understanding of the material gathered and compiled during the analysis and requirements elicitation activities.

MOLIC: A Modeling Language for Interaction as Conversation

MoLIC is an interaction modeling language created within semiotic engineering. A MoLIC model represents all possible interactive conversations that users may have with the system, i.e., all the possible interaction paths, including alternative paths to achieve the same goal, and paths for recovering from interactive breakdowns or errors. We have used MoLIC models as a shared representation to achieve two distinct goals: to represent the designer-to-user communication (achieved at interaction time), and to foster communication among design team members (during design).

MoLIC models represent the user-system interaction as threads of conversation that users may (or must) have with the application in order to achieve their goals. MoLIC was not devised to replace existing representations, but to complement them. The event sequences depicted in scenarios are organized in a MoLIC diagram, which reveals the relationships and intersections between scenarios, from a user's point-of-view. It is important to note that MoLIC was devised for human usage; it is not meant for representing formal, machine-processable models.

The MoLIC diagram is complemented by an ontology of signs. Here, we use the term *sign* to denote any given element in the application domain or at the user interface, to which a user may attribute meaning with respect to his goal or task, or to the application itself. The goal of the sign ontology is to provide a shared knowledge of the user interface signs, which will be presented to or manipulated by users during their conversation with the system, and their relationships. In this ontology, the definition of each sign includes its inherent properties, i.e., the properties that remain unaltered during the user interaction with the system, and information about the origin of the sign (whether it exists in the domain or whether it only makes sense in the context of that particular application). For instance, a sign definition may include its default value (in absolute terms or relative to other signs in the ontology), the set of possible values it may assume, or even the abstract user interface widget associated to the sign (simple choice, free text, etc.).

As a running example, let us consider a hotel reservation system. A partial sign ontology for such a system is depicted in Figure 2. For clarity purposes, the figure includes only the signs' attributes and relationships, and ommits the details associated to each attribute.

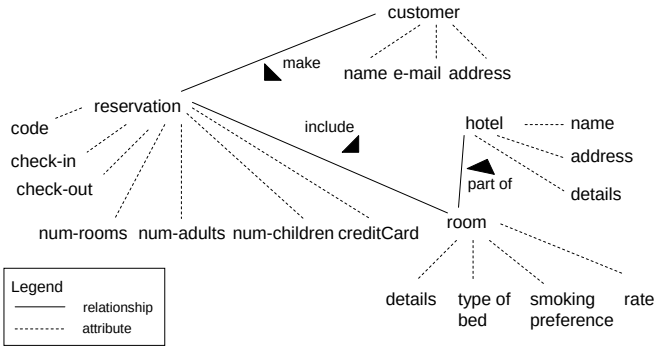

Figure 2: A partial sign ontology for a hotel reservation system.

The sign ontology is built *pari passu* the MoLIC diagram and the scenarios. The notation in which the ontology is represented is not prescribed by the authors. The only restriction is that it can represent the necessary sign

attributes and relationships. The design team may choose a representation that best suits their needs, such as a semantic network, for instance.

MoLIC's Diagrammatic Notation

A MoLIC diagram basically depicts the turn-taking between user and system, forming conversation threads. Although the graphical representation resembles a state-transition diagram, the coincidence is only superficial: we focus on the communication aspects of the interaction, such as turn-taking, topic and subtopic structures, and some mismatches between user' intentions and system behavior, encouraging a careful design for the recovery from interaction breakdowns.

In a MoLIC diagram, there are two different kinds of nodes, indicating the user's or the system's turn to "say something". We represent the user's turn to make a decision about how the conversation should proceed in a *scene*, represented by a rounded rectangle, containing a label describing the topic of the conversation at that moment, and a set of dialogs and signs related to that topic (to achieve a certain goal or subgoal). In order to facilitate the representation of scenes that can be accessed from anywhere within the application (e.g. from menu items), MoLIC diagrams contain *ubiquitous accesses* to these scenes, represented by gray, rounded rectangles.

The system's turn in the conversation is represented by a "black box", indicating that users cannot perceive what is going on inside the system processing nodes.

Linking scenes to system processes, and vice-versa, are *transition utterances*. Transition utterances stemming from scenes represent changes in focus or a conclusion of the conversation topic, as caused by a user's choice, indicated by the transition label. Those stemming from system processes represent the result of that processing, indicating whether the user's request was completed successfully or whether a breakdown or system error has occurred.

Interaction breakdowns in MoLIC

Repair utterances are an inherent part of human conversation, and so are breakdown prevention and handling in user-system interaction. Considering the interaction as conversation, designers are encouraged to represent not only how users should perform tasks under normal conditions, but also how to avoid or deal with mistaken or unsuccessful situations. When potential breakdown situations are detected or predicted during interaction modeling, they should be represented in the MoLIC diagram by breakdown tags. These tags are used to identify the interaction mechanisms designed to deal with potential or actual breakdowns, according to the following categories:

Passive prevention (PP): documentation or online instructions designed to prevent breakdowns from happening (e.g. the format of the data expected in a field).

Active prevention (AP): active mechanisms that will prevent breakdowns from occurring (e.g. forbidding the user to type in letters or symbols in numerical fields).

Supported prevention (SP): asking the user to decide if a situation is a breakdown or not (e.g., confirmation messages such as "File already exists. Overwrite?").

Error capture (EC): errors that are detected by the system and must be notified to users, but for which there is no remedial action (e.g., when a file is corrupted).

Supported repair (SR): informing the user about a detected breakdown and allowing him to correct it (e.g., presenting an error message and the previously filled fields for the user to correct the problem).

MoLIC diagrams may be represented in an abbreviated or an extended form. The abbreviated form includes the dialogs' topics, but not the individual signs or utterances composing the dialogs. The extended MoLIC diagram includes the signs associated to each dialog. For each sign, the attributes specific to that context of interaction need to be represented (i.e., attributes that are not represented in the sign ontology). For instance, one may represent attributes regarding the default values of that sign in a certain dialogue, or whether users need to provide a value for the sign (i.e. it is a mandatory sign), and associated breakdown tags, if necessary.

Figure 3 shows a MoLIC diagram for a hotel reservation application at an intermediary stage of design, in which signs for two scenes have already been defined. A typical interaction would proceed as follows: the user searches for hotels in a city, then indicates the desired hotel, informs the room and billing details, and then examines and confirms the reservation.

MoLIC encourages the representation of alternative interaction paths. For instance, Figure 3 shows that, in case the user searches for a city that exists in different states or regions, she will be prompted to indicate which one she was referring to (scene "Disambiguate city").

USING MOLIC TO SUPPORT THE CONSTRUCTION OF UML DIAGRAMS

In software development, software engineers use design and specification notations to help them understand, organize and represent the system's architecture and functionality. HCI professionals also use design notations, but in their case the notations help them understand, organize and reflect on *user-centered concerns*.

In object-oriented development, the UML (a *de facto* standard) provides us with a large set of models to represent complementary aspects of the system. Use cases, class diagrams, sequence diagrams are perhaps the most widely used UML models in the initial design stages. Although UML is process independent, the following steps are usually carried out. Use cases are created to represent a sequence of system actions to produce an observable result,

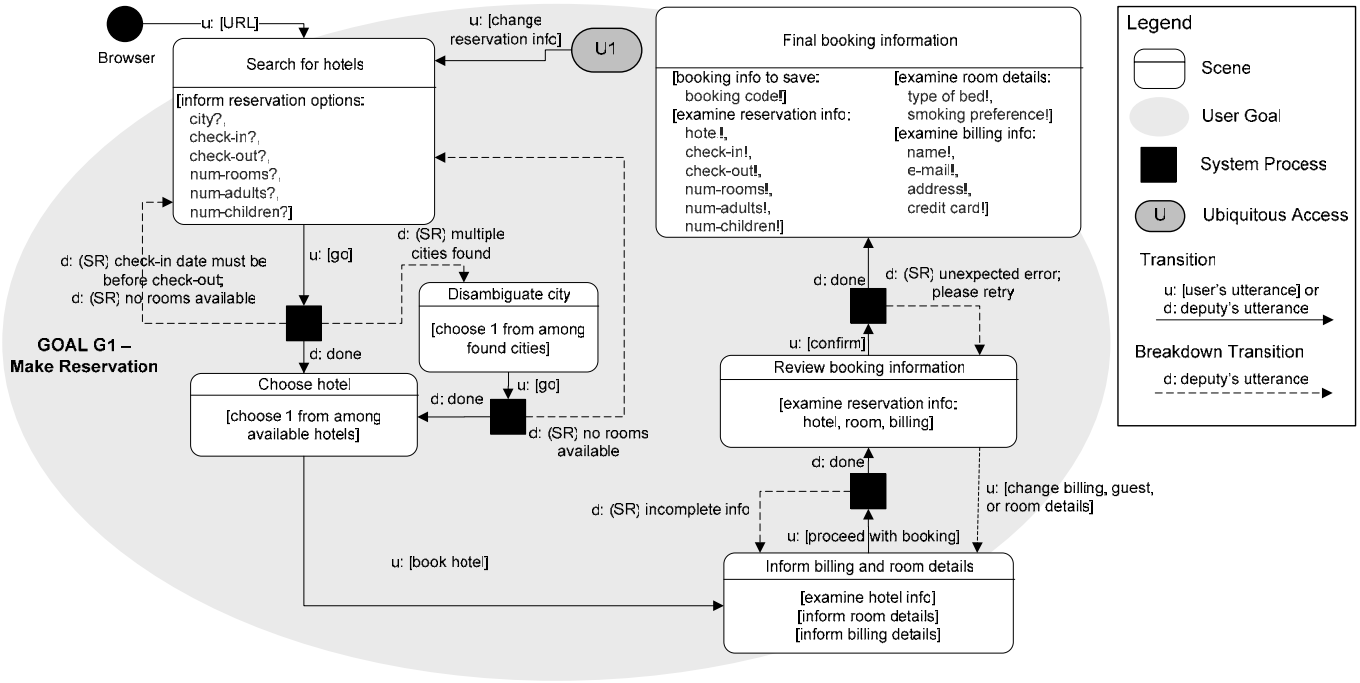

Figure 3: A sample MoLIC diagram for a hotel reservation system

relevant to at least one of the actors. They describe what the system does, and not how it does it. According to Booch and co-authors, a use case "names a single, identifiable, and reasonably atomic behavior of system or part of the system." [2, p.231]

From the set of use cases, several features of class diagrams are created, which represent the system's static structural model: the system classes, their attributes and methods, and the relationships between the classes.

To each use case, a sequence diagram is associated. It represents the possible interactions between instances, the messages that can be exchanged between them, and in which sequence (in time).

From an HCI perspective, one of the major drawbacks of UML is that most of its models concern the system only, leaving most decisions about the user interface to the later stages of development, or even to the implementation phase. The UML fails to properly model the human-computer interaction.

We claim that, in HCI, we need a modeling language that allows designers to build a blueprint of the application that will reveal its apparent (from a user's perspective) behavior. Such a blueprint could then be used as a reference point for global design decisions, and would be an additional resource for deriving both HCI and SE models. As such, this blueprint could act as a target for the application design to aim at. Figure 4 illustrates the relations between the UML models and this "blueprint".

We believe MoLIC can act as the blueprint illustrated in Figure 4, i.e., as a reference point to other design models. As such, MoLIC would be responsible for representing the user-centered application's semantics, i.e., the conceptual solution, from the user's point of view. This kind of approach has been proposed a long time ago by Frederick Brooks, when he stated that "the separation of architectural effort from implementation is a very powerful way of getting conceptual integrity on very large projects", and "by the *architecture* of a system, I mean the complete and detailed specification of the user interface" [3].

Besides representing the application's blueprint, MoLIC may directly contribute to the construction of UML diagrams. It is possible to define mappings between MoLIC and some UML diagrams.

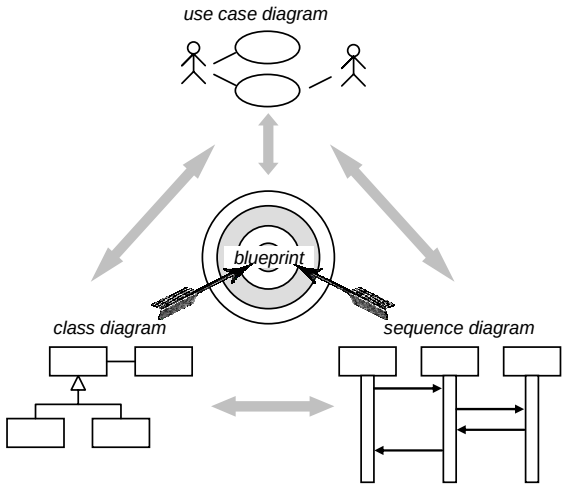

Figure 4: Proposal for an interaction blueprint as a reference point to UML models.

The sign ontology may be mapped onto a class diagram. The ontology provides the definition of the classes, attributes, some operations, and relationships that are directly accessible to users. Most of the signs present at the user interface have a counterpart in the data model. In a class diagram, these signs are usually mapped onto a class attribute, such as a person's name or birthdate. The value of a few signs is not directly stored, but calculated from one or more pieces of data. These may be mapped onto class methods, such as a person's age or whether he/she may obtain a driver's license (calculated based on his/her birthdate). Besides, by traversing the sign ontology, some of the relationships between the classes are also directly derived.

MoLIC diagrams also convey information that helps to create the class diagram. For each operation available to users at the user interface (as represented in the diagram), a method may be created in the corresponding class.

MoLIC transition utterances may derive messages in sequence diagrams: MoLIC provides the temporal ordering of the user-system interaction. For every sequence diagram in which one of the instances represent a user role, the messages incoming or outgoing from this instance may be retrieved from or verified against MoLIC's transition utterances. The use of MoLIC is quite important here: this information cannot be adequately derived from use cases.

MoLIC diagrams also provide information to build use cases: the goals and tasks users want to achieve through the system, and the user roles that need to be represented as actors in the use cases. While use cases are often used as an HCI model inside UML, they provide very little information about how the user-system interaction will take place, and instead focus on the description of system actions. MoLIC, on the other hand, always focuses on the user-system interaction as viewed by users. The actions that aren't visible to users are not directly represented in MoLIC. Their representation in use cases causes designers to shift from user- to system-centered design, which adds to the confusion. By using MoLIC and use cases, the former may focus on the user's point of view, whereas the latter may include the systems actions as the software engineers will need to understand and specify them.

Figure 5 illustrates the described mappings between MoLIC and some UML diagrams.

A distinctive characteristic of MoLIC is the emphasis in representing the breakdowns that may occur during user-system interaction. When a breakdown occurs, a careful HCI design is even more important than in normal courses of action. This is thus one of the main advantages that MoLIC brings to UML diagrams, since here the alternative paths for helping users in inefficient or invalid courses of action are not highlighted.

A Case Study for Deriving UML skeleton diagrams from MoLIC

In this section we present a small case study to illustrate the mappings we have described. The sign ontology (Figure 2) and the interaction diagram for the goal "Make reservation" (Figure 3) will be used as resources for the construction of skeleton UML diagrams for a simple hotel

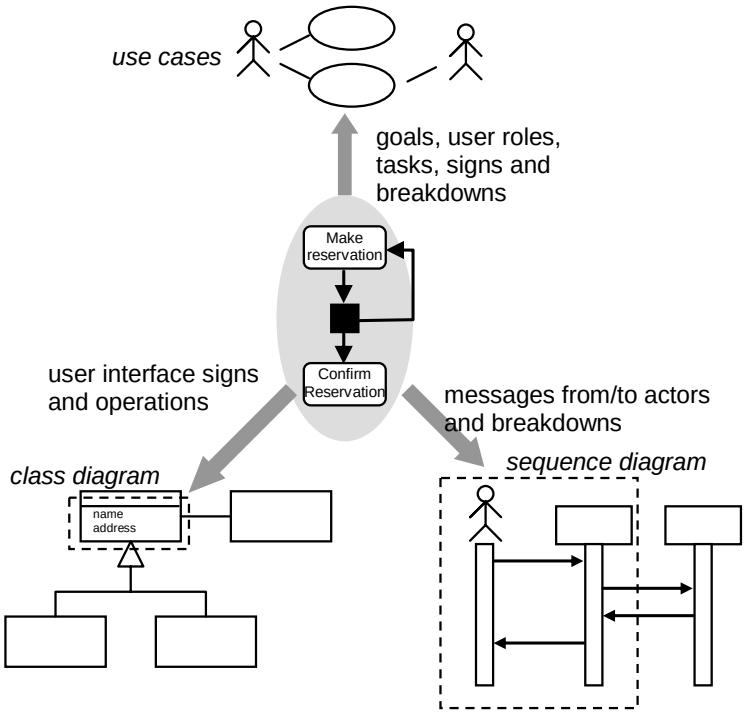

Figure 5: Sample mappings between MoLIC and UML diagrams.

reservation system. The UML diagrams built in the case study were: use case diagram, class diagram, and sequence diagram. It is important to highlight that our goal is to contribute to UML modeling by making derivations from MoLIC, but not to build the entire UML diagrams. We only map the elements that have a direct association with the user interface and the user-system interaction. The internal software decomposition is left to the software engineers.

Use case diagram

For the construction of use case diagrams, MoLIC provides information about the user's goals and tasks. Figure 6 presents a use case diagram that may be derived from the MoLIC diagram in Figure 3.

The following mappings were made:

- from the goal "G1 – Make reservation" to the use case "make hotel reservation", directly associated to the "hotel customer" actor;
- from the scene "Choose hotel" to the use case "Show available hotels";
- from the transition utterance "book hotel" to the use case "Get desired hotel";
- from the scene "Inform billing and room details" to the use case "Show available rooms"; and
- from the transition utterance "proceed with booking" to the use cases "Get room details" and "Get billing information".

One may notice that, while the use case that represents a user goal ("make reservation") is phrased from the user's point of view, the remaining use cases are represented from the system's point of view. This is made possible by the ambiguous and sometimes conflicting definitions of use cases.

We believe that, by having an interaction model as a starting point, not only do we facilitate the construction of use case diagrams, but also allow software designers to associate internal system operations to user actions at the user interface.

Class diagram

Some elements of a UML class diagram may be derived by MoLIC diagram and the sign ontology. While some researchers use class diagrams to represent ontologies, we argue that this separation is necessary because the sign ontology provides information that is relevant to HCI concerns (such as default values, for instance) that will not be directly represented in a class diagram.

Figure 7 illustrates a class diagram derived from MoLIC. Each sign and its attributes, as represented in the sign ontology (Figure 2) appear in the class diagram. In this case study, the mapping was straightforward; there were no transformations from the attributes in the sign ontology to the class diagram.

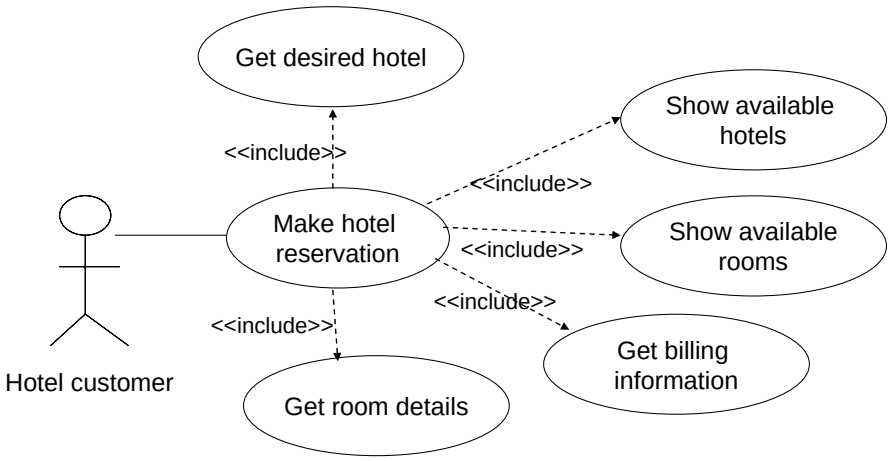

Figure 6: Use case diagram derived from MoLIC.

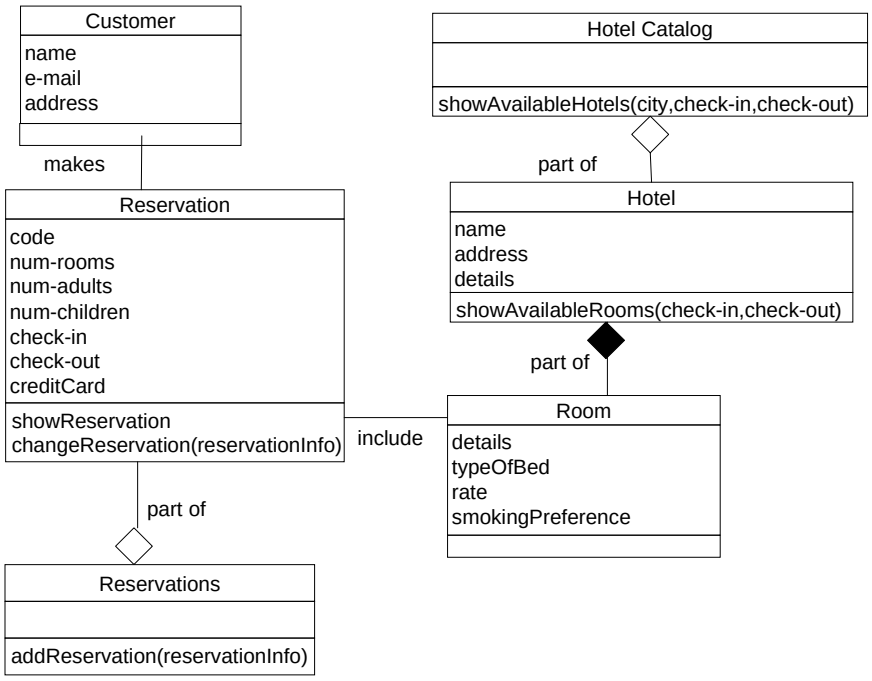

Figure 7: Sample class diagram derived from MoLIC.

In addition to the ontology, the interaction diagram provided information about the methods in each class. For instance, the following mappings were made:

- from the dialogue "[choose 1 from among available hotels]" in the scene "Choose hotel" to the method "showAvailableHotels" in the class "Hotel Catalog";
- from the dialogue "[examine reservation info]" in the scene "Final booking information" to the method "showReservation" in the class "Reservation"; and
- from the transition utterance "u:[change billing, guest or room details]", the method "changeReservation" in class "Reservation"

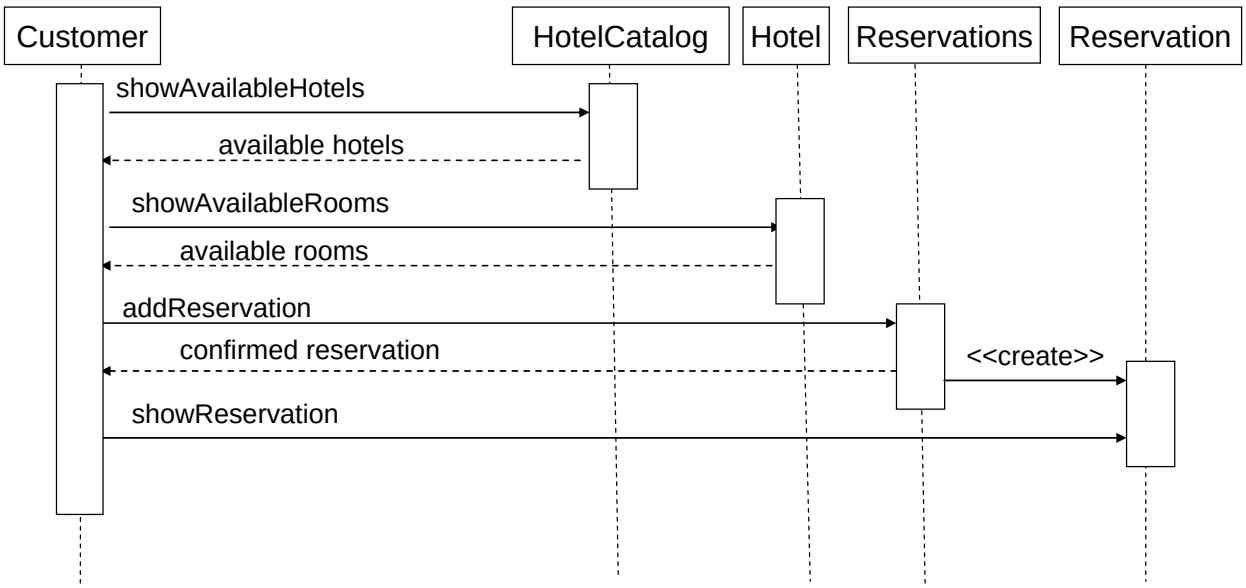

Figure 8: Simplified sequence diagram for the "Make reservation" use case.

In addition to these mappings, MoLIC may provide some of the information handled in each method. For instance, the method "showReservation" should present information about the signs depicted in MoLIC: hotel, check-in, check-out, num-rooms, num-adults, num-children, type of bed, smoking preference, name, e-mail, address, and credit card.

Sequence diagram

Figure 8 presents a sequence diagram from the use case "Make reservation" (Figure 6). The order in which the messages appear in the diagram was extracted from the MoLIC diagram presented in Figure 3. The other UML diagrams we have described did not represent any temporal aspects of the user-system interaction.

Using MoLIC as a resource for building the sequence diagram, some design decisions regarding the temporal sequencing of tasks was clear. For instance, it was defined that first the user would select the hotel, and then the type of room, and not the other way around, i.e., to first define the desired type of room, and then have the system search for matching hotels.

One of the main advantages advocated by the MoLIC designers is that MoLIC encourages the straightforward representation of interaction breakdowns. In contrast, representing alternative paths in a UML sequence diagram may prove to be confusing or difficult. In Figure 9, we present a more complete sequence diagram, including alternative courses of action anticipated in the MoLIC diagram.

CONCLUDING REMARKS

In this paper, we argued for the importance of supporting the communication between the areas of human-computer interaction and software engineering. Both areas deal with the quality of the final product, albeit from different perspectives and focusing on different concerns. HCI focuses on the interaction and user interface design, taking into account the users' needs, values and expectations, aiming at the quality of use of the designed solution. Software engineering, conversely, focuses on the system architecture and internal functionality design and specification, aiming at the structural quality of the final software product.

Since users view the user interface as "the" software [6], it is important that HCI design be awarded a first-class status in the design process, so that the apparent behavior of the application and the internal aspects of the application are not subordinate to one another. In order for this to work, it is imperative that HCI designers clearly convey their concerns and design choices to software engineers and vice-versa, so that together they can negotiate the final solution. Moreover, it is desirable that both professionals may rely on a shared representation about what the interactive system should be like.

In this paper, we proposed the use of MoLIC interaction models to serve as such a representation and thus bridge HCI and SE. In order to make the benefits of using MoLIC more evident to software engineers, the paper described how some elements of UML models may be directly derived from MoLIC diagram and sign ontology, and exemplified possible mappings in a small case study. By providing a clear representation of the interactive exchanges that may take place, we avoid that HCI decisions are lost or inadvertently overruled when designing the system architecture and internal functional behavior.

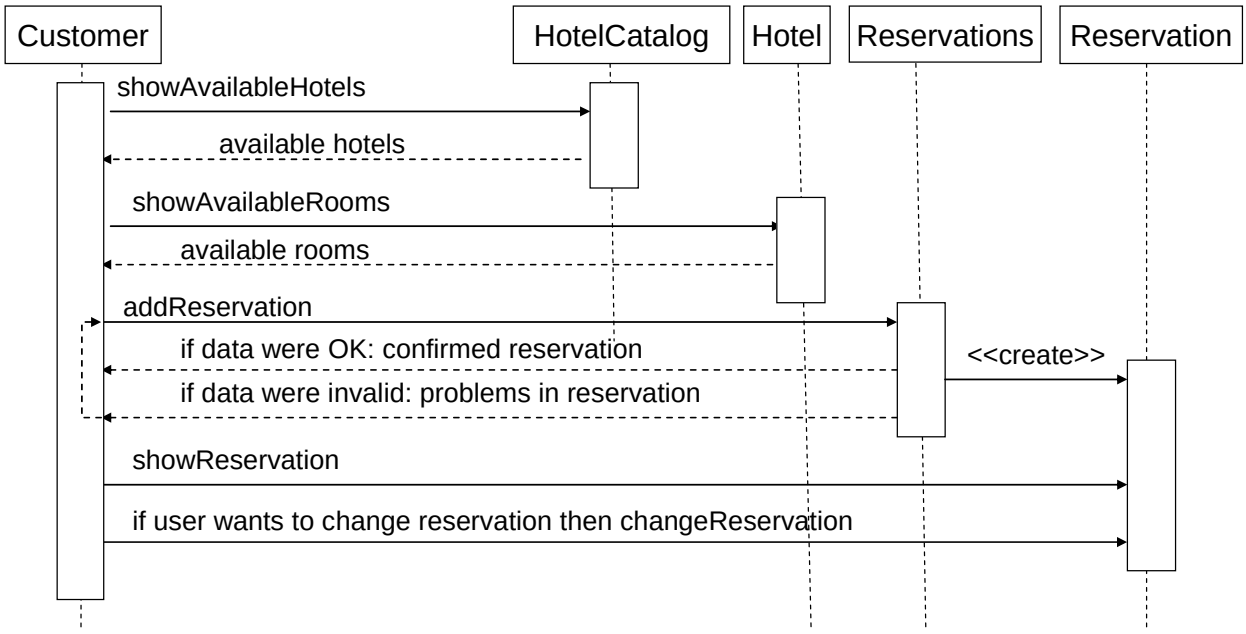

Figure 9: Sequence diagram for the "Make reservation" use case with alternative courses of interaction.

Our current work includes detailing the information that we can pass on to UML (or an extended version of UML), and defining a systematic procedure for the mappings. In addition, we are developing a number of case studies to evaluate the impact of our proposed approach in the software development process, and to analyze how MoLIC can further support the activities of the other members of the multidisciplinary design team. In these case studies, some extensions to MoLIC have been detected, and are being incorporated in the second edition of the model [13]. Such extensions include structuring dialogues (to represent sequential and mutually exclusive dialogues, for instance) and multi-user interactions (in which interconnected MoLIC diagrams are used to represent each user's interaction with the system). We are also investigating the use of MoLIC in representing interaction design patterns, which are also efficient communicative artifacts between design team members.

ACKNOWLEDGEMENTS

The authors thank CNPq for ongoing support to their work. They also thank their colleagues at the Semiotic Engineering Research Group and LES at PUC-Rio for precious contributions to their research.

REFERENCES

[1] Barbosa, S.D.J.; Paula, M.G. "Designing and Evaluating Interaction as Conversation: a Modeling Language based on Semiotic Engineering" In *10th International Workshop, DSV-IS 2003*, Madeira Island, Portugal, LNCS, vol. 2844, 2003, pp. 16–33.

[2] Booch, G.; Rumbaugh; Jacobson, I. *The Unified Modeling Language User Guide*. Addison Wesley. 1998.

[3] Brooks, F. P. *The Mythical Man-Month*. Addison-Wesley. 1975.

[4] Constantine, L.L. & Lockwood, L.A.D. "Structure and Style in Use Cases for User Interface Design". In: Mark van Harmelen (ed.) *Object Modeling and User Interface Design*, London: Addison Wesley Longman, 2001.

[5] de Souza, C. S., *The Semiotic Engineering of Human-Computer Interaction*, The MIT Press, 2005.

[6] Hix, D.; Hartson, H. R. *Developing User Interfaces: Ensuring Usability Through Product & Process*. New York NY: John Wiley & Sons. 1993.

[7] Kruchten, P.; Ahlqvist, S.; Bylund, S. "User Interface Design in the Rational Unified Process". In: Mark van Harmelen (ed.) *Object Modeling and User Interface Design*, London: Addison Wesley Longman, 2001, pp. 161-196.

[8] Nielsen, J. *Usability Engineering*. Academic Press, 1993.

[9] Nunes, N. J., *Object Modeling for User-centered Development and User-interface Design: The Wisdom Approach*, 301 pags., Tese de Doutoramento em Engenharia de Sistemas, Especialidade de Informática, UMa, Jul. 2001,

[10] Paula, M.G.; Silva, B.S.; Barbosa, S.D.J. Using an Interaction Model as a Resource for Communication in Design. Proceedings of CHI 2005, extended abstracts volume. Portland, Oregon, April 2005. pp.1713-1716.

[11] Rosson, M.B. & Carroll, J.M. "Scenarios, objects, and points-of-view in user interface design". In: Mark van Harmelen (ed.) *Object Modeling and User Interface Design*, London: Addison Wesley Longman, 2001, pp. 39-70.

[12] Schön, D. *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books. 1983.

[13] Silva, B. S.; Barbosa, S.D.J. "Modelando a interação do NiTA: um estudo de caso e extensões ao MoLIC". *Anais do VI Simpósio sobre Fatores Humanos em Sistemas Computacionais, IHC'2004*. Curitiba, PR. Outubro de 2004, pp.199-203.

[14] Silva, P.P. *Object Modeling of Interactive Systems: The UMLi Approach*. PhD thesis, Department of Computer Science, University of Manchester, United Kingdom, 2002.

[15] OMG. Unified Modeling Language 1.5 Specification. 2003. Available for download at http://www.omg.org/technology/documents/formal/uml.htm [last visited in April 2005].

[16] van Harmelen, M. "Designing with Idiom". In: Mark van Harmelen (ed.) *Object Modeling and User Interface Design*, London: Addison Wesley Longman, 2001, pp. 71-113.

[17] van Harmelen, M. (ed.) *Object Modeling and User Interface Design*. London: Addison Wesley Longman, 2001.