

Relating Human-Computer Interaction and Software Engineering Concerns: Towards Extending UML Through an Interaction Modeling Language

Maíra Greco de Paula, Simone D.J. Barbosa, Carlos José P. de Lucena

Departamento de Informática, PUC-Rio
R. Marquês de São Vicente, 225 – Gávea
Rio de Janeiro – RJ – Brasil – 22453-900
{mgreco, simone, lucena}@inf.puc-rio.br

Abstract: The UML suite of modeling languages fails to properly model the human-computer interaction. On the other hand, newly conceived HCI modeling languages need to foresee their role as members of the family of languages that constitute the UML representations for software design, due to their wide acceptance by both researchers and practitioners. MoLIC, our proposed HCI modeling language, seems to be a natural family member since many consistency checks seem to be possible between MoLIC and other software design notations used in UML. MoLIC is based on Semiotic Engineering and represents interaction as threads of conversation users may have with the system.

Keywords: interaction modeling language, UML, HCI and software design, HCI-SE integration

1 Introduction

Almost 20 years after Brooks' seminal "No Silver Bullet" (Brooks, 1986), we still face the challenges of traversing the gap between essence and accident in software construction, i.e. between "the mental crafting of the conceptual construct" and the implementation process.

Many methods, models and tools have been proposed since then to try and face this challenge. Currently, we find object-oriented design with the Unified Modeling Language (UML, 2003) amongst the most widely discussed in the academia and put in practical use in the software industry. In the realm of software engineering (SE), this approach has been working fine for most purposes. When we take into account human-computer interaction (HCI), however, we see that it does not provide designers with modeling languages and associated tools to explore the relevant HCI aspects of interactive software. Modeling languages that express the HCI concerns and that can be integrated with other notations that address software engineering concerns

are necessary to bridge both understanding and communication gaps between the areas of SE and HCI.

This paper proposes to use an interaction modeling language called MoLIC (Paula, 2003) as a tool to help bridge this gap. We illustrate how an interaction model can help define the system behavior from the user's point of view, or the "architecture of a system" as defined in (Brooks, 1975). We do not claim it is ready to be fully integrated to other SE notations (e.g. the UML suite of languages), but instead that it may be considered as a first step towards HCI-SE integration.

2 OO Design with UML

In software development, designers use models that help them understand, organize and represent the system's architecture and functionality. In object-oriented development, the UML provides us with a large set of models to represent complementary aspects of the system. Use cases, class diagrams, sequence diagrams and activity diagrams are

perhaps the most widely used UML models in the initial design stages.

Figure 1 summarizes the relationship between these models. 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, relevant to at least one of the actors. They describe what the system does, and not how it does it.

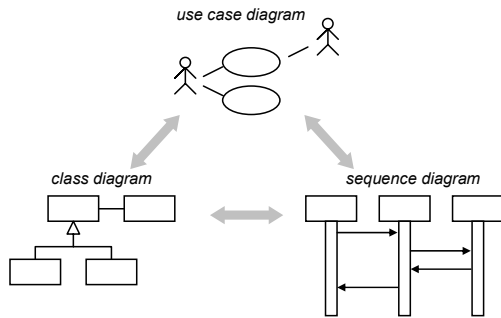


Figure 1: Typical UML models used in early stages of software design.

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).

Naturally, as with any design process, the construction of each model itself provides additional information that may cause refinements to the other models, hence the bidirectional arrows in the figure. 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.

In UML, use cases do concern users, but only in a fragmented way, in which each user goal is dealt with in isolation. As a result, designers lose the global view of the application as it will be perceived by the users. This makes it more difficult for HCI designers to envisage how the interaction will take place, in order to plan for and evaluate the quality of use of the final product. Thus, we claim that it is not enough to have use cases as a basis for the construction of the other diagrams from the user's perspective.

Moreover, the basic UML diagrams do not differentiate between what is internal to the system and what will surface at the user interface. If we examine the aforementioned diagrams with respect to the elements that are relevant to HCI designers, we find, in a use case diagram: the actors which represent classes or roles of users and the use cases directly related to them, as representing the users' goals that may be achieved by interacting with the application. In a class diagram, we cannot distinguish which attributes or methods will have a (direct or indirect) representation at the user interface, and which of them are internal to the system. In a sequence diagram, we may have instances that correspond to actors, but the interaction is dispersed between each actor and a number of different instances.

From the user's point of view, the user interface is a single unified artifact. In order to design the user interface, HCI designers adopt decomposition strategies that are different from those adopted in SE, deriving task models and storyboards, for instance. Before we start each decomposition, however, we need to carefully craft the solution at a conceptual level. At this level, we need representations that provide a global view of the application. In SE, we might accomplish this by representing the system architecture. 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 2 illustrates the relations between the UML models and this "blueprint":

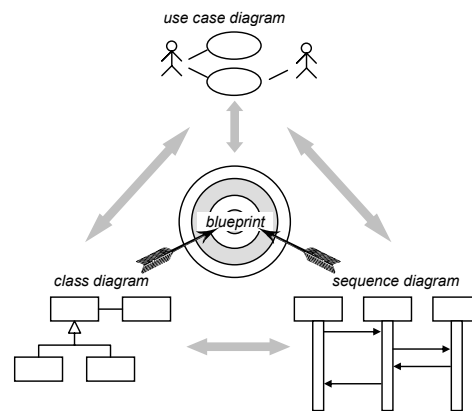


Figure 2: Proposal for an interaction blueprint as a reference point to UML models.

In the next section, we will describe an interaction model proposed in HCI to act as this application-as-the-user-will-experience-it blueprint. We represent in it how the user will be able to interact with the system, different interaction paths to accomplish his/her goals, and also how interaction breakdowns will be handled.

3 Interaction Modeling with MoLIC

HCI researchers and practitioners have proposed a number of methods and models to deal with the complexity of interactive systems design. Task models, user interface specifications and storyboards are the most widely used (see, for instance, (Paternò, 2000) and (Hix & Hartson, 1993)). However, none of these models gives designers a global view of the apparent behavior of the system, including alternative and breakdown interaction paths.

Paula proposed an interaction model named MoLIC, which stands for “Modeling Language for Interaction as Conversation” (Paula, 2003). Based on Semiotic Engineering (de Souza, 1993; de Souza, forthcoming), MoLIC represents interaction as threads of conversation users may have with the system¹, without yet specifying in detail the storyboards or the user interface itself. Each thread of conversation represents courses of action to accomplish a certain goal or to take remedial action when a communicative breakdown occurs. MoLIC was devised to explicitly represent all possible interactions, allowing designers to inspect the model for inconsistencies.

A MoLIC diagram may be seen as a graphical view of the whole set of scenarios or use cases, from the user’s perspective. In HCI design, we do not aim to substitute scenarios, but to organize the relevant information they contain. Although high-quality scenarios contain all the information represented in MoLIC, 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. Some of the difficulties are also due to frequent ambiguities found in natural language texts.

¹ In fact, the conversation takes place between the user and a part of the system’s user interface called “designer’s deputy”, as defined in the Semiotic Engineering theory of HCI. The semiotic engineering concepts used on the theoretical foundation for MoLIC and the complete MoLIC notation may be found at (Paula, 2003).

Many HCI designers generate task models from the scenarios, and then proceed to the user interface specification. However, task models usually assume the “correct” way of achieving a goal, without much consideration to breakdowns that may occur during interaction.

Finally, there are often too many decisions to be made in moving from scenarios and/or task models to software specification. And, unfortunately, these decisions are seldom recorded for future reference or verification of the final product.

3.1 MoLIC’s Notation

The interaction model basically comprises scenes, system processes, and user/system’s utterances — called transitions— that connect scenes and processes to form conversation threads. A scene represents a user–system 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/system utterances. 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 a window or dialog box, or a page in HTML.

In MoLIC’s 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.

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

Transitions represent changes in the conversation topic or focus. 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: [search document]`), whereas an outgoing transition from a process represents the result of the processing as it will be “told” by the system (represented by a simple label, such as `d: document(s) found`). In case there are pre-conditions for the transition to be made, they should come before the transition label, following the keyword `pre:.` Also, if there is a postcondition, it should be marked by the `post: keyword` after the

label. A postcondition is typically represented when there is a change in the application state that affects interaction.

A user goal is represented by a gray elliptical shape behind the thread of scenes and system processes with which the user will need to interact to accomplish it. User goals are extracted from scenarios (Carroll, 1995; Carroll, 2000) or use cases, and provide some level of traceability between MoLIC and them.

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”.

3.2 Interaction breakdowns in MoLIC

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): errors that are prevented by documentation or online instructions. For instance, about which users have access to the system, what is the nature of the information expected (and not just “format” of information).

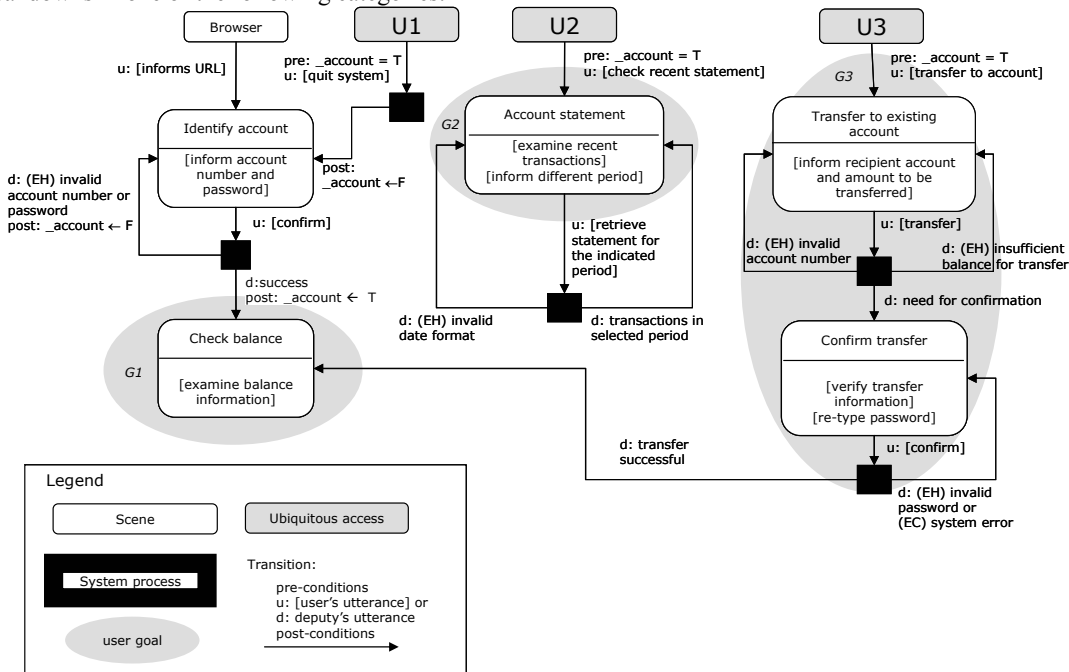
Active prevention (AP): errors that will be actively prevented by the system. For instance, tasks that will be unavailable in certain situations. In the interface specification, this may be mapped to making widgets disabled depending on the application status or preventing the user to type in letters or symbols in numerical fields, and so on.

Supported prevention (SP): situations which the system 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 “File already exists. Overwrite?”)

Error capture (EC): errors that are identified by the system and that should be notified to users, but for which there is no possible remedial action. For instance, when a file is corrupted.

Supported error handling (EH): errors that should be corrected by the user, with system support. For instance, presenting an error message and an opportunity for the user to correct the error.

Figure 3 illustrates a MoLIC diagram for a banking application which achieves the following goals: check balance, check account statement, and transfer between accounts.



MoLIC has both an abbreviated and an extended diagrammatic representation. The extended MoLIC diagram represents not only dialogues, but also what can be talked about in each dialogue. This is represented by the signs at the user interface that either the user or the system manipulates at each moment. Here, we use the term *sign* to denote any given element at the user interface to which a user may attribute meaning with respect to his/her goal or task, or to the application itself.

When a sign is uttered by the system, 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?`).

Besides the role responsible for manipulating a sign (user or system), the extended representation may also include additional information about each sign, such as: whether the user must provide a value for the sign, whether the system provides a default value for the sign (and how this default value is calculated), the abstract user interface widget associated to the sign (simple choice, free text, etc.), the degree of knowledge the user has about the sign, additional information to be conveyed to users to help them understand and/or manipulate the sign, and any additional annotations designers may want to represent. Figure 4 shows the extended representation for the “Identify account” scene.

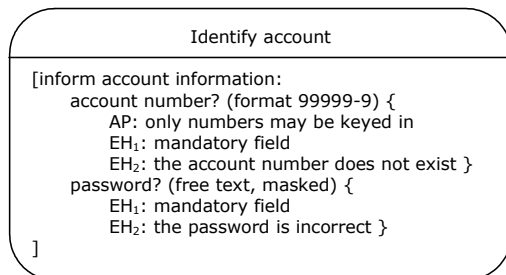


Figure 4: Extended representation of a scene.

3.3 MoLIC’s Contributions to Design

As seen in section 2, we needed a language to represent the apparent behavior of a system as a whole, and from the user’s point of view. It is necessary to identify the user–system turn-taking during interaction, but the system should be represented as a single entity, instead of already decomposed in internal software components. This way, HCI designers will be able to anticipate

interaction problems or inconsistencies, and make appropriate design decisions early on.

We believe MoLIC contributes to HCI design by overcoming some limitations of existing HCI models. In particular, MoLIC motivates the uncovering of interaction problems or breakdowns, allowing designers to reflect upon the remedial actions to be taken and additional signs to present to users in order to help them recover from challenging situations.

In the next section, we will illustrate the use of MoLIC together with UML diagrams, as an important resource for software design.

4 Interaction Modeling within Software Engineering

We believe MoLIC can act as the blueprint illustrated in Figure 2, i.e., as a reference point to other design models (Barbosa & Paula, 2003). As such, MoLIC would be responsible for representing the “essence” of the application, 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” (Brooks, 1975).

From an HCI perspective, MoLIC is an important resource for building the storyboards or rigorously specifying the user interface. The simplest way to proceed would be to map each scene to a screen or window in a storyboard, each sign to a widget, and each user-initiated transition to a button or menu item (for activating an operation, usually between a scene and a system process), or even to a hyperlink (for navigation only, usually between two scenes). More elaborate constructions can, of course, be derived. For instance, one scene with multiple incoming transitions may be mapped to multiple variations of a single storyboard. Each variation would provide information about the alternative course of action taken to reach the scene, such as the goal the user is trying to achieve or the error he/she should correct.

But how can MoLIC be integrated to or collaborate with the aforementioned UML models? A MoLIC model may be built from scenarios or use case descriptions. In building the MoLIC model, the

designers will be motivated to find and correct problems in these information sources, such as inconsistencies and incompleteness. The advantage of using MoLIC here is that no system decomposition needs to be made or revised before this step, and thus the cost of changes is not very high. In Figure 3, we may substitute the ellipses representing user goals by a reference to the use cases that are directly related to the user-actors in a use case diagram.

With respect to the construction of the class and sequence diagrams, we may cite additional points where MoLIC may contribute:

- mapping from MoLIC signs to class attributes or methods: 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).
- mapping from MoLIC transitions to messages in sequence diagrams: 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 transitions.

Figure 5 illustrates these mappings.

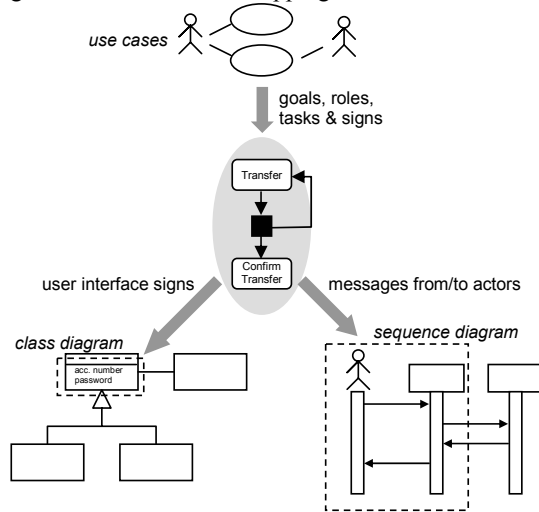


Figure 5: Sample mappings from use cases to MoLIC, and from MoLIC to UML class and sequence diagrams.

The UML suite of modeling languages (a de facto standard) fails to properly model the human-computer interaction. On the other hand, newly conceived HCI modeling languages need to foresee their role as a member of the family of languages that constitute the UML representations for software design. MoLIC appears to be a natural family member since many consistency checks seem to be possible between our proposed HCI modeling language and other software design notations used in UML. Incorporating MoLIC to the UML family of modeling languages will probably require changes in several existing UML diagrams to allow for its proper integration. Investigating process models based on an extended version of UML that encompasses HCI modeling appears to be a promising research area.

5 Final considerations

We have briefly illustrated an initial attempt to use an HCI design model together with SE design models. Our goal was to provide a common reference point for both HCI designers and software engineers upon which to base their solutions.

There have been some attempts to incorporate user interface concerns into UML diagrams, such as UMLi (Silva, 2002). The main difference between our approach and UMLi is that we adopt a specific theory of HCI —called Semiotic Engineering— which provides us with an ontology for describing and evaluating relevant HCI phenomena, always keeping the focus on the quality of use of the proposed solution.

Other approaches, UMLi included, typically follow a bottom-up approach, whose main resource is a collection of case studies and user interface aspects present in user interface development environments. These approaches adopt a system-centered decomposition of user interface representations, losing sight of important HCI concerns. As such, they do not adequately support the designer's reflection about the intended solution at a conceptual level, and that a proper representation of HCI at this level is necessary for bridging the gaps between HCI and SE.

We are currently investigating what kind of design decisions are facilitated by our approach, and what are the impacts (good and bad) in the HCI and SE design processes. We need to assess whether this approach effectively promotes collaboration between SE and HCI and, if so, how this collaboration may be further propagated to other models, both in HCI and in SE.

Acknowledgements

Simone Diniz Junqueira Barbosa and Carlos José Pereira de Lucena thank CNPq for providing financial support to their work.

References

- 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. May, 2003. Available online at <http://www.se-hci.org/bridging/icse/proceedings.html>.
- Brooks, F. P. (1975), *The Mythical Man-Month*. Addison-Wesley.
- Brooks, F. P. (1986), No Silver Bullet, *Proceedings of the IFIP Tenth World Computing Conference*, pp.1069–76.
- Carroll, J. M. (ed) (1995). *Scenario-based design: envisioning work and technology in system development*, New York, Wiley.
- Carroll, J. M. (ed) (2000) *Making use: Scenario-Based Design of Human-Computer Interactions*. The MIT Press. Cambridge, MA.
- de Souza, C.S. (1993) The Semiotic Engineering of User Interface Languages. *International Journal of Man-Machine Studies*. No. **39**. pp. 753-773. 1993.
- de Souza, C.S. (forthcoming) The Semiotic Engineering of Human-Computer Interaction.
- Hix, D. and Hartson, H. (1993) *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons.
- Paternò, F. (2000) *Model-Based Design and Evaluation of Interactive Applications*, London, Springer-Verlag.
- Paula, M.G. (2003), *Designing the Human-Computer Interaction Based on Semiotic Engineering Models: Building an Interaction Model* (in Portuguese). Master dissertation. Informatics Department, Pontificia Universidade Católica do Rio de Janeiro.
- Silva, P.P. (2002) *Object Modelling of Interactive Systems: The UMLi Approach*. PhD's thesis, Department of Computer Science, University of Manchester, United Kingdom.
- UML (2003) *OMG Unified Modeling Language Specification, v. 1.5*. March, 2003. Available for download at <http://www.uml.org>