Adopting a Communication-Centered Design Approach to Support Interdisciplinary Design Teams

Simone Diniz Junqueira Barbosa, Máira Greco de Paula, Carlos José Pereira de Lucena
Departamento de Informática, PUC-Rio
R. Marquês de São Vicente, 225
Rio de Janeiro, RJ, Brazil – 22453-900
{simone, mgreco, lucena}@inf.puc-rio.br

Abstract

UML has become a de facto standard in software industry, and many attempts to bridge the gap between software engineering (SE) and human-computer interaction (HCI) have extended UML to incorporate HCI elements and concerns into the software development process. However, by adopting a representation which springs mainly from a single disciplinary field, communication among the design team members is hindered and software engineers' values prevail.

We argue that, in order to truly bridge the communication gap among design team members, we need a more “neutral” common ground. We propose to use an interaction model to represent the designers' vision of how the system should behave. We describe MoLIC, a modeling language for interaction as conversation, as a language that may play the role of this common representation.

1. Introduction

In the realm of software engineering representations for design and specification, there is no doubt that UML [21] has become a *de facto* standard. In order for another design notation to be considered practical and useful, it needs to somehow relate to UML. Many attempts to bridge the gap between software engineering (SE) and human-computer interaction (HCI) have extended UML to incorporate HCI elements and concerns into the software development process (for instance, [12, 13, 20]). However, professionals from other disciplines have their own practices and may resist to adopt software engineering models and representations such as UML. They may be unable or unwilling to step out of their practice to learn a whole new representation that is comfortably situated inside another discipline, or they may simply need to consider specific elements that UML extensions do not or cannot contemplate. In this case, the communication among the design team's members is

hindered and software engineers' values prevail, perpetuating the madness eloquently described by Alan Cooper [7]. As a result, the end product does not take advantage of the multidisciplinary nature of the design team.

In this paper we argue that, in order to truly bridge the communication gap among design team members, we need a more “neutral” common ground. We also need a broader view of the question of design, defined by Winograd and Flores as “the interaction between understanding and creation” [22, p.4]. We argue that scenarios are not an adequate solution to the communication problems during design. We need to elaborate a more structured representation with the specific purpose of addressing communication among team members, i.e., to support their decision-making processes and to serve as a reference material to which every team member may refer during design and development. It is important that this representation comprise a minimal set of elements that are required to create a shared understanding of the solution being elaborated, and to motivate reflection of each professional in his own field [19]. Each professional would still work with his own set of tools and representations, but they would all share a single design vision that synthesizes what needs to be done. This way, we aim to address the needs of the many disciplines involved in software design without imposing the representations from any single discipline.

Inspired by Frederick Brooks [4] and borrowing Aristotle's terms [1], we want to represent the “essence” of an interactive software, leaving the “accidental” aspects of the software to the various design representations used specifically within each discipline.

2. Communication-centered design

Interaction design is concerned with “designing interactive products to support people in their everyday

and working lives” [17, p.6]. From a visual designer’s perspective, HCI is like graphics design, industrial design, and architecture, in that it is “concerned most directly with the user’s experience of a *form in the context of a specific task or problem*, as opposed to its functional or aesthetic qualities in isolation” [10, p.1].

Within HCI, Semiotic Engineering [8] has emerged as a semiotics-based theory that describes and explains human-computer interaction phenomena, adopting primarily a *media* perspective on the use of computer artifacts [9]. Semiotic Engineering focuses on the engineering of signs that convey what HCI designers and users have in mind and what effect they want to cause in the world of things, practices, ideas and experiences. The interface signs constitute a message sent from designers to users, representing the designers’ solution to what they believe is the users’ problems, what they have interpreted as being the users’ needs and preferences, what the answer for these needs is and how they implemented their vision as an interactive system.

Our work builds on semiotic engineering in the sense that, in order to increase the chances of engineering adequate signs at the user interface to convey the designers’ vision, we propose to engineer the designer’s message to be conveyed first to other designers. If designers are unable to convey their vision to each other and to every stakeholder, they will hardly succeed in conveying it to users. If, on the other hand, we succeed in promoting designer-designer communication via a representation as a communication artifact, we will be better equipped to communicate with users through the user interface. Our envisaged representation is not only a carrier of information. Instead, we want to emphasize its social role during the design process [22], to promote understanding of and communication among professionals from multiple disciplinary fields. This way, we aim to achieve a communication-centered approach to interactive software design and development.

So, we need to elaborate a representation or a modeling language that fosters interdisciplinary discussion, represents the designers’ vision of the essence of the system being designed and thus serves as a reference of *what* needs to be done, to which every professional must commit. The way each professional works when designing his own portion of the system (the “accidental” aspects of the system) may be done *however* he wishes, as long as the common vision is respected. Of course there will be breakdowns along the way, and the design unfoldings in one or more disciplines may require that changes in the common vision are made.

Many attempts to foster communication among team members and stakeholders are based on scenarios and use cases [6, 18]. However, scenarios frequently present

ambiguities or omit important information. Although scenarios are useful to convey a few concepts or elaborate specific usage situations, their usefulness in representing the design of a whole product is limited to small projects. In addition, 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 [2].

Therefore, we need a more structured representation language that describes the designers’ “vision” of the solution, and “speaks” to team members about what is of their concern: they need to be able to filter out what’s not immediately relevant to their current focus of attention. Also, in order to be of practical use, it must be possible to derive from it representations of each specific discipline, such as a draft version of some UML diagrams to be used by software engineers. This derivation or transformation may be such as to ensure a certain degree of traceability between each specific representation and the common vision.

3. Interaction modeling as the common ground

We have argued for the need to elaborate a common representation to describe the designers’ vision. But what exactly do we choose to represent? How should it be related to the software or to the user interface? We have made an attempt to represent the designers’ vision as an *interaction model* [2, 14].

The underlying assumption is that the designers’ vision of *what* needs to be developed (and not *how*) should reflect the application’s apparent behavior, i.e., the interactive behavior from the users’ point-of-view. This assumption is in line with Frederick Brooks’ suggestion that “the separation of architectural effort from implementation is a very powerful way of getting conceptual integrity on very large projects” [5]. Although this statement will be considered obvious now by any software engineer, Brooks’ elaboration of what an architecture is may not be as widely known: “by the architecture of a system, I mean the complete and detailed specification of the user interface” [5, p.45 (1995 edition)]. In Requirements Engineering, a similar consideration may be found in [3], which describes the advantages of employing a user’s manual as a requirements specification.

One question to be answered is: who may create this common representation? HCI designers, software engineers, graphical designers, or other professional? We

believe that the first draft representation of the common vision will emerge during the initial design sessions. We assume it will be collectively built, and not assigned to a single professional. As the designers' vision mature, each professional will then create or specify the representations concerning his own field.

In the next section, we describe a possible representation language for modeling interaction. It is called MoLIC, which stands for "Modeling Language for Interaction as Conversation" [2, 14].

4. Interaction modeling with MoLIC

Our goal when creating MoLIC was to give designers a global view of the apparent behavior of the system, including alternative and breakdown interaction paths. Based on Semiotic Engineering [8], 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 [15].

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. We do not aim to substitute scenarios, but to organize the relevant information they contain. As with scenarios, we emphasize the need to describe breakdown situations that may occur during interaction. In scenarios, however, there are often too many decisions to be made in moving to software specification. And, unfortunately, these decisions are seldom recorded for future reference or verification of the final product. We propose to use MoLIC to structure scenarios in such a way as to overcome the problems found in scenarios for defining the interaction.

4.1 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 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, as represented in the arrow label.

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".

4.2 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.

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 [8]. The semiotic engineering concepts used on the theoretical foundation for MoLIC and the complete MoLIC notation may be found at [15].

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

Figure 1: An abbreviated MoLIC diagram.

MoLIC has both an abbreviated and an extended diagrammatic representation. The extended MoLIC diagram represents not only dialogues. This is represented by the signs at the user interface that either the user or the system manipulates at each moment (represented by the marks ? and !, respectively). 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.

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 2 shows the extended representation for the “Identify account” scene.

Figure 2: Extended representation of a scene.

We propose to use MoLIC to represent the designers' vision, i.e., *what* needs to be the apparent behavior of the system. From this vision, each professional would then derive his own representations and models, to specify and refine *how* the system will be put together as envisioned.

Concluding remarks

In this paper, we have argued that, to address concerns from other disciplines such as HCI during design and development, we need to look for alternative solutions instead of focusing on extending UML to incorporate elements of HCI design.

We have presented MoLIC, an interaction modeling language that aims to play the role of the common representation to describe the designers' vision. We have been conducting studies of this approach involving HCI designers, graphics designers and programmers. In our preliminary findings, the usage of MoLIC as a specification tool for the surfacing behavior of the application was found to be very effective by all team members. Programmers, in particular, used MoLIC as a guide for developing the system functionality. However, they felt they had a lot of "avoidable" work in mapping or translating the signs represented in the interaction model into the system's data model. Graphics designers who were interviewed about the notation said they would like to "receive" from the HCI designers not only the interaction model and signs, but also the widgets they would need to deal with when creating the visual structure of the user interface.

We have described elsewhere some simple strategies for deriving partial UML class and sequence diagrams from a MoLIC representation [14]. An activity diagram [21] may be considered a UML diagram more closely related to MoLIC. A simple way to derive an activity diagram from MoLIC would be to represent scenes and system processes as action states in different swimlanes (user and system, respectively). Moreover, each system process (the black boxes) would be expanded to show decisions, along with the corresponding transitions.

We are now elaborating a more extensive study to investigate the benefits of using MoLIC from the standpoint of stakeholders and design team members from various disciplinary fields. In addition, we want to explore what kind of design decisions are facilitated by our approach, and what are the impacts (good and bad) in the design processes from the standpoint of the involved disciplines.

Acknowledgements

The authors would like to thank CNPq for ongoing support to their research. They are grateful for their colleagues at the Semiotic Engineering Research Group at PUC-Rio for interesting discussions about many of the issues described in this paper.

References

- [1] Aristotle, *Metaphysics, Book VII or ZETA*.
- [2] Barbosa, S.D.J.; Paula, M.G., "Interaction Modeling 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. pp. 84-91.
- [3] Berry, D.M., Daudjee, K., Dong, J., Nelson, M.A., and Nelson, T., "User's Manual as a Requirements Specification", Technical Report, CS 2001-17, University of Waterloo, Waterloo, ON, Canada, May, 2001. Available online at http://se.uwaterloo.ca/~dberry/FTP_SITE/tech.report/s/users.man.pdf.
- [4] Brooks, F. P., "No Silver Bullet—Essence and Accident in Software Engineering", Proceedings of the IFIP Tenth World Computing Conference, pp.1069–76. 1986.
- [5] Brooks, F. P., *The Mythical Man-Month*. Addison-Wesley, Reading, MA, 1975 (anniversary edition, 1995).
- [6] Carroll, J. M. (ed) (1995). Scenario-based design: envisioning work and technology in system development, New York, Wiley.
- [7] Cooper, A., *The Inmates are Running the Asylum*, SAMS Publishing, Indianapolis, IN, 1999.
- [8] de Souza, C. S., *The Semiotic Engineering of Human-Computer Interaction*, The MIT Press, in press.
- [9] Kammersgaard, J., "Four different perspectives on Human-Computer Interaction", *International Journal of Man-Machine Studies* 28:343-362, Academic Press, 1988.
- [10] Mullet, K., and Sano, D., *Designing Visual Interfaces: Communication-Oriented Techniques*, SunSoft Press, Mountain View, CA, 1995.
- [11] Norman, D. e Draper, S. (eds., 1986) User Centered System Design. Hillsdale, NJ. Lawrence Erlbaum.
- [12] Nunes, N.J., "Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach", Ph.D. Thesis, Universidade da Madeira, 2001.
- [13] Nunes, N.J., "What drives software development: issues integrating software engineering and human-

- computer Interaction” , in Closing the Gaps: Software Engineering and Human-Computer Interaction, Workshop at Interact 2003, Zurich, Switzerland, 2003, pp.89-95.
- [14] Paula, M.G., Barbosa, S.D.J., and Lucena, C.J.P. “Relating Human-Computer Interaction and Software Engineering Concerns: Towards Extending UML Through an Interaction Modeling Language”, in Closing the Gaps: Software Engineering and Human-Computer Interaction, Workshop at Interact 2003, Zurich, Switzerland, 2003, pp.40-46.
- [15] Paula, M.G., 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, 2003.
- [16] Peirce, C.S. *Collected Papers*, Harvard University Press, Cambridge, MA, 1931, excerpted in Buchler, Justus, ed., *Philosophical Writings of Peirce*, New York: Dover, 1955.
- [17] Preece, J., Rogers, Y., and Sharp, H., *Interaction design: beyond human-computer interaction*, John Wiley & Sons, New York, NY, 2002.
- [18] Rosson, M.B., and Carroll, J. M. *Usability Engineering: Scenario-Based Development of Human Computer Interaction*, Morgan Kaufmann, 2001.
- [19] Schön, D. *The Reflective Practitioner: How Professionals Think in Action*, New York, Basic Books, 1983
- [20] Silva, P.P. “Object Modelling of Interactive Systems: The UMLi Approach”. PhD's thesis, Department of Computer Science, University of Manchester, United Kingdom. 2002.
- [21] UML. OMG Unified Modeling Language Specification, v. 1.5. March, 2003. Available for download at <http://www.uml.org>. 2003
- [22] Winograd, T. and Flores, F., *Understanding Computers and Cognition: A New Foundation for Design*, Addison-Wesley, Reading, MA. 1986.