Designing for Change: Engineering Adaptable and Adaptive User Interaction by Focusing on User Goals

Bruno S. da Silva, Ariane M. Bueno, and Simone D.J. Barbosa

Departamento de Informática, PUC-Rio
R. Marquês de São Vicente, 225
Gávea, Rio de Janeiro, RJ, Brasil, 22451-900
{brunosantana,abueno,simone}@inf.puc-rio.br

Abstract. In the human-computer interaction area, research work in end-user programming, end-user development, and user or system-driven adaptation of interactive systems has attempted to cope with variations in users' intents, context changes and evolutions. In the field of requirements engineering, research that addresses similar issues has been called variability analysis. Most work in variability analysis, however, focuses on prioritizing one or few possible solutions to be implemented in the final product, whereas in human-computer interaction many researchers advocate that we should strive to enable users to adjust and adapt the product as needed. This paper presents an approach to bring the results obtained in requirements engineering to inform the choice of interaction design solutions to cope with variability.

Keywords: Variability analysis, interactive systems adaptation, bridging requirements engineering and interaction design.

1 Introduction

Despite the research effort to deal with differences and variations among users and devices in human-computer interaction (HCI), even today we lack a systematic approach to deal with variations that reflect differences in the context of use, in user goals, needs, preferences, and strategies to achieve them. Research focusing on requirements analysis calls our attention to the importance of variations among users, devices and contexts and which propose techniques to analyze them [10, 11, 12, 16, 20]. In HCI, on the other hand, many researchers have proposed strategies to deal with variations at interaction time, typically with flexible, adaptive and adaptable solutions [15, 17, 18, 19, 26]. Nevertheless, there is still a gap between the identification of variability in the problem space and the definition of adequate solutions for coping with variability in the solution space.

In requirements engineering, HCI-related concerns are typically treated as softgoals, which can be achieved by selecting, at design time, certain task decomposition, based on the user and business priorities identified by the requirements engineer [3, 13]. This means that, if users' priorities and softgoals change during system usage, the system will not be able to accommodate them, and may therefore become inefficient or even obsolete. The research area of intelligent

J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2009, LNCS 5610, pp. 715–724, 2009.

user interfaces, in particular on adaptable and adaptive systems [15], aims to accommodate changes in needs and priorities during interaction, thus rendering the system more efficient and useful for a longer period of time.

Most work in adaptable and adaptive user interfaces, however, focuses on the specification and implementation of the adaptation mechanisms. There is little work on how to decide which adaptation strategy to adopt to deal with certain identified variations, based on the analysis of requirements and user needs. More specifically, there is little work on designing adaptation based on contextualized user goals (identified during analysis), as opposed to specific ways of achieving them (which already result from design decisions that, if made too early, may unnecessarily restrict the possible adaptation strategies).

We have been working on a variability analysis approach based on the users' discourse that characterizes their goals [24, 25]. In those previous works, the focus was to separately explore the problem and the solution spaces of the HCI design activity related to differences and variations among HCI concerns. In this work, we relate the dimensions analyzed in the problem space and the corresponding strategies deemed appropriate to accommodate change in the solution space during interaction time.

This paper is organized as follows: the next section describes our account of the problem and solution spaces. The third section describes how variability is considered in each space. Next, the paper presents the proposed relations between the two spaces, and the fifth section presents some concluding remarks.

2 The Problem and Solution Spaces

The *problem space* may at first be characterized by the people who will use or benefit from the system, and, as in semiotic engineering, what they "want and need to do, in which preferred ways, and why" [4]. Moreover, the contexts [6] in which user activity occurs is also part of the problem space. Variations can be found in the heterogeneity of the user population, their goals or the context for their activities, as well as their evolution in time.

Regarding users, it is important to investigate the psychological characteristics (e.g. attitude, motivation, preferences), knowledge and experience (e.g. typing skill, task experience) and physical characteristics (e.g. color blindness, hearing difficulty). Regarding their goals, it is important to examine the user goals when they perform their activities, the frequency and importance of the activities, and the main artifacts and objects used in performing them. Concerning the context of the activity, it is important to study when and where the goals need to be/are achieved: the time divisions (e.g. hours and days) or intervals (e.g. seasonal intervals) which are relevant to user goals and related activities; the set of places (e.g. home and work) or a hierarchy of places (e.g. a room at a university, a campus, a city, and so on); the physical environment (e.g. open/close work areas, lighting, heat, noise level, distractions and interruptions); the social and cultural environment (e.g. in broad terms, morale, motivation, values and policy, or, in specific terms, the possibility of learning to use the system with colleagues, cases when users are pressured to go fast, and this culture works better with uncertainty than others) [11, 23].

To understand the *design solution space*, from an HCI perspective, we need to understand what goes on during the use of a system. The semiotic engineering theory of human-computer interaction [4] brings to our attention that, as intellectual artifacts, every software [5]:

- *linguistically* encodes both a particular understanding or interpretation of a problem situation and a particular set of corresponding solutions; and
- is designed with the goal that users will be able to formulate and express their intents within the linguistic system encoded in the artifact.

By *linguistic*, de Souza means that the artifact encoding is "based on a system of symbols—verbal, visual, aural, or other—that can be interpreted by consistent semantic rules" [5]. In semiotic engineering, Jakobson's communication model ([13], Fig. 1.) is used to illustrate human-computer interaction phenomena and as thus it provides a basis for defining the solution space.

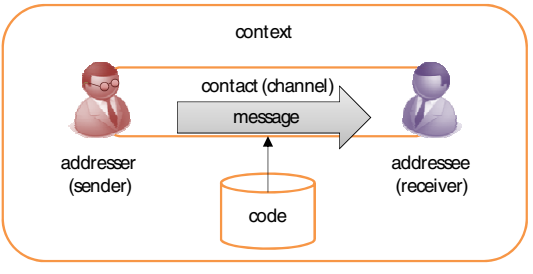

Fig. 1. Jakobson's communication model

As with other semiotic approaches, semiotic engineering views HCI as a particular kind of computer-mediated human interaction [4]. It views software as a meta-communication artifact, i.e., a (meta)message produced by the designer about the communication that may take place when users communicate with the message itself at interaction time. The user interface is said to be the "designer's deputy", in the sense that it encodes a range of meanings, meaning manipulations, and design principles that the designer chose to synthesize in the product.

The MoLIC language, Modeling Language for Interaction as Conversation, was devised to help designers elaborate the metamessage [1] at a dialogue level, allowing them to represent and reflect on: the user's goals or intents supported by the system; the conversations (i.e. sequences of illocutions and turn taking performed by the user and the designer's deputy) through which the users may achieve their goals; restrictions on the utterance of certain illocutions according to the context of the conversation (or the world); the perlocutions or effects of each (segment of) conversation; illocutions aimed at repairing communicative breakdowns that the designer is able to anticipate; and the signs[1] contained in the illocutions.

Thus, we may say that MoLIC further details the solution space with respect to human-computer *interaction*. MoLIC does not, however, represent the concrete user interface, which is also part of the solution space. In this paper, we do not deal

[1] Peirce defined sign as "anything that stands for something else, to somebody, in some respect or capacity" [22].

extensively with the user interface itself. When necessary, we only point to some forms of user interface adaptation, without detailing how it can or should be represented.

In this paper, we propose to classify the signs in the solution space in three groups:

- *object signs*, which represent concepts, entities or things;
- *task signs*, which represent actions that manipulate the object signs; and
- *user interface signs*, which represent the user interface elements that refer to objects and tasks.

As we will see in the fourth section, variations in each group of signs (as a result of the requirements elicitation and analysis) will point to different interactive solutions to cope with these variations.

3 Exploring Variability

Changes in the problem space may require changes in the solution space. In order to explore **variability in the problem space**, we follow the variability analysis process defined in [24] and refined in [25], which comprises the following steps:

1. Elicit information about domain, user goals, users, context of use and system (possible hardware and infrastructure).
2. Identify goal-directed user requests.
3. Identify and describe the signs present in the user discourse about their domain, goals, and tasks.
4. Rewrite user requests using cases.
5. Organize signs in an ontology.
6. Explore possible variations by expanding user requests.
7. Explore possible variations by substituting signs in user requests.

The data collected in (1) are typically answers to the general 5W2H questions: who, what, when, where, why, how, and how much. In addition, issues of time could combine when and how much to generate questions about how often and for how long.

Who participates in the interaction process? Examining the interaction process, one realizes that both *user* and *system* participate in it. Regarding the users, information about their skills and preferences [11, 12, 23] need to be elicited, as well as any constraints and special needs they may have. Regarding the system, information about the available hardware platforms (desktop, laptop, PDA, cell phone, etc.), input and output devices (mouse, keyboard, pen, etc.), and infrastructure (network access, disk space, etc.) are necessary. It is important to note that, at this stage, no design or implementation decisions are made. Instead, possibilities are being elicited that will help anticipate variations. Only later should these aspects drive design decisions.

What are the participants' goals? The users' goals (i.e. the expected results of their interaction with the system) are traditionally investigated during requirements engineering [3, 11, 1420, 20]. The "system's" goals, on the other hand, are a product of the designers' work to support the users' goals, and thus are designed in later stages of development process.

When, where, and in which contexts will goals be achieved? Besides common known time divisions, such as minutes, hours, days, months, and so on, the requirements engineer should investigate other relevant time divisions or intervals, such as seasonal intervals. The interaction can occur in a set or hierarchy of places, such as home and work, or university<campus<city<state<country. An analysis of the environments should investigate the physical (e.g. levels of light and noise), social (e.g. the possibility of learning to use the system with colleagues; pressure to be efficient) and cultural (e.g. ways to deal with uncertainty) aspects of the environment that can interfere in goal formulation and the user-system interaction [11, 23, 26]. Also, the more general question helps to uncover additional elements to be encoded in the problem space as part of the context, as discussed by Dourish in [6].

How can each goal be achieved? Possible strategies users follow to achieve each goal in their current context of activity. More recently, variability has also been taken into account [10, 12, 16, 20]. Later, during design, one or more selected strategies will be mapped onto interaction sequences at the user interface.

How often is each goal formulated? This is a variation of the *How much?* question that is very relevant for investigating recurring events and goals.

The analysis should also consider variations in time, with special attention to the frequency of change. For example, it is not sufficient to investigate that a specific user has such and such skills and preferences, because they can change in time, motivated, among other factors, by training or promotion. As the concerns involved in variability can change during system usage, it may not sufficient to deal with variability searching for "the best option" at design time for user X and context Y (privileging some softgoals over others) to design and develop the system according to this option. It is also important to consider strategies to deal with variability during system usage.

We identify the signs (words, pictures etc. that users employ to mean something) present in the users' discourse about their goals, and organize them in an ontology; and to identify goal-directed user requests and rewrite them using cases (e.g. Fillmore's [9]). These are then reviewed and expanded through systematic question asking and, finally, we analyze the system variability, also through systematic question asking and by traversing the sign ontology.

The focus on goal-directed user requests means that goals have not yet been decomposed into tasks. A user request may be viewed as a high-level "instruction" issued to a system for the user to achieve her goal, assuming the system could interpret it and take appropriate action(s). For instance, in a media player application, "play this playlist at 50% volume" is a user request.

The ontology allows us to explore if signs that are somehow related may fit in a user request case and thus represent a possible variation. For instance, the relation music–is-part-of–playlist gives us a clue to attempt to use a playlist in place of a music file. By representing user requests as verb+cases (Agentive, Dative, Objective, Factitive, Instrumental, Manner, Locational, and Temporal), one may restrict the ontology traversal to make the variability analysis more efficient.

Systematic question asking is used to review and expand the user requests that were previously identified. To systematically find out about the possible variations, diverse questions are asked (e.g.: What can vary? When? How often? How much?) As

a result of the analysis process, the answers associated to a structured user request and sign ontology, will help to make the upcoming design decisions.

For instance, a `play[O(media_file)]` user request may originate an expanded user request, like:

```
play[O(media_file),M(volume),M(speed),M(equalization),
O(current_playing_position),F(state=playing)]
```

Regarding **variability in the solution space**, literature on intelligent, adaptable and adaptive user interfaces (cf, for instance, [15, 17, 18]) addresses variations in the solution space according to multiple dimensions:

- What is adapted?What aspect of it is adapted?
- Who makes the adaptation?
- The adaptation is associated to what?
- And in the case of adaptive systems, The adaptation is computed as a function of what?

Regarding **what** is adapted (what kinds of signs are adapted), we have that *task signs*, (conceptual) *object signs* and *user interface (ui) signs* can be adapted. This adaptation can be of two kinds: *token selection* or configuration (attribute values or tokens), and *type selection* or composition (selection of a subset of types within a sign).

Regarding **who** makes the adaptation, we can have either the user (manual adaptation, as in adaptable systems) or the system (automatic adaptation, as in adaptive systems). If the adaptation is *manual*, we also need to decide: *To what is the adaptation associated?* An adaptation can be associated to any element of the domain model, the communication model or the context of activity that is encoded in the system, such as: a single user, a group of users or a user profile; the application, i.e., valid across users; a single device (e.g. my cell phone whose serial number is *VN2531*) or a set of devices, extensionally or intensionally defined (e.g. smartphones with 3G connection capability); a certain object (e.g. document, paragraph) or a set of objects extensionally or intensionally defined (e.g. all documents produced by the *WriteOn* text editor); certain periods of time (e.g. turn sound notifications off between 12am and 6am); certain locations (e.g. at home vs. at the workplace); and so on.

In the case of manual adaptation, there can be implicitly or explicitly setting parameters (e.g. in "preference" dialogue boxes), to creating macros, styles and templates. In this kind of adaptation, the user is in control of the application's behavior, but the necessary effort to adapt the application needs to be taken into careful consideration so as not to hinder the adaptation possibilities [18]. In importing a picture into a document, the path where the previously imported picture was located can be remembered in the next import operation, assuming that the user will keep most of their pictures in the same directory. This is a case of implicit parameter setting: it was meant by the user only to be part of defining a single operation, but designed to be remembered and later reused, as if the user had defined it as the preferred path for locating media files in a "preferences" dialog box.

If the adaptation is *automatic*, we need to decide: *How (As a function of what) will the adaptation computed?* In any case, care must be taken to allow users to easily regain control of the application behavior, either by configuring the rules of the adaptation mechanism "just-in-time" or by turning them off altogether.

Most adaptive systems adjust the content and formatting presented to users as a function of the user's (and other users') interaction history, device, network capabilities, and so on. In the text editor example, the documents could be listed with different formatting or in a different order depending on how often they were accessed, for instance. Adaptive hypermedia systems are among the most prominent examples of adaptive systems. In e-commerce applications, recommender system modules are a common example of this category of adaptation.

A few adaptive systems make permanent changes to the application or the object with which the user is working, such as "autocorrect" and "autocomplete" features in form fields, for instance. It is paramount that this kind of adaptation is adequately communicated to users, easily reversible, and possible to be turned off.

Thus, variations in the solution space can be encoded by the tuple `<what, type/token, who, f(X), g(Y)>`, where `f(X)` is to what the adaptations is associated, and `g(Y)` is the set of elements that drive the adaptation (in the case of adaptive systems, the actual rules that adapt the user interface).

4 From Analysis to Design: Make Decisions about Variability

When moving from analysis to design, there are at least four strategies to deal with variations:

- not to accommodate variation (single non-adaptable system);
- to accommodate variation in a family of products (software product line);
- to allow users to manually change some aspects of the system (adaptable system); and
- to embed rules and inference mechanisms for the system to automatically adapt itself (adaptive system).

Focusing on flexible, adaptable and adaptive systems, we can relate the problem space to the solution space, as follows:

Problem Space	**Solution Space (conceptual)**
regarding the focus of the adaptation (cases in the user request)	
agent, object, dative, factitive	object
manner, instrument, locational, temporal	task
regarding the structure of user requests	
goal-driven user request with fixed cases	token selection
goal-driven user request with varying cases	type composition
regarding the frequency of occurrence of a set of tokens or types	
varying set of tokens or types that cannot be associated to specific contexts of use	— (no defaults)
prominent set of tokens or types associated to most contexts of use	— (fixed defaults)

tokens or types associated to certain contexts that cannot be inferred	manual
tokens or types associated to contexts that can be inferred	automatic

Within the solution space, we can relate the possible adaptation solutions to different interaction design solutions, as follows:

Conceptual Solution	**Interaction Solution and Examples**
no adaptation, fixed solution	traditional HCI design solutions, with or without default values
no adaptation, flexible solution	alternative tasks and parameters for a certain task ▪ selection of printing parameters, character and paragraph formatting
<task, token selection>	setting or inferring application or task profiles, i.e., sets of configuration parameters related to the whole application, to a set of tasks, or to a single task, instead of being applied to specific objects ▪ configuration sets for different environments, so that when printing a certain document at home, the configuration would be different from when printing the same document at the workplace ▪ having the default printer configured differently in distinct environments, such as at home or at the workplace, or depending on the network to which the device is connected
<task, type composition>	manually or automatically composing sequences of tasks into a macro or script, especially by recording the user-system interaction to be reproduced later as a single interaction step. This kind of adaptation is especially useful for repetitive action sequences. ▪ manual: composing task sequences into a macro or script; macro recording, interactive macro editing and running, scripting ▪ automatic: programming by demonstration
<object, token selection>	setting or inferring attribute values of an object to establish a new default or a named configuration ▪ formatting styles in text editors, which can be applied to characters and paragraphs to facilitate consistent formatting ▪ certain patterns of printing for certain kinds of document, such as technical report, illustration, and so on. Each document could then be associated with a template so that it is printed with the corresponding configuration settings (also related to task adaptation)
<object, type composition>	manually or automatically creating new objects or templates composed of existing objects, either extensionally or intensionally ▪ special sets of documents, such as manually defined favorite documents or automatically inferred most accessed documents, most recently accessed documents, and so on
<ui, token selection>	setting or inferring attributes of user interface signs ▪ manually redefining colors, labels, images ▪ automatically highlighting user interface signs according to some context-dependent relevance criteria

<ui, type composition>	manually or automatically arranging user interface signs ▪ reconfiguring the layout, panels, toolbars, and menus ▪ putting frequently accessed operations first on a list, "remembering" the last user interface layout configuration

To increase the user's control of the interaction, a candidate design solution for adaptivity generally involves suggesting and adaptation and having the user approve or dismiss it or, for less risky adaptations, to make the adaptation but provide a clear and easy mechanism for reversing it.

It is important to note that the "associated to" and the "computed from" functions are closely related to the problem space, and their mapping onto the solution space depends on the domain and context elements the designer decides to encode in the software.

5 Concluding Remarks

Our preliminary evaluation of the variability analysis approach confirmed the benefits of avoiding early task decomposition, organizing signs in an ontology and systematically asking questions to analyze variability. The support for traversing from the problem to the solution space has been mainly inspired by existing literature involving both theoretical and technical work. Further work includes a more extensive evaluation of the proposed traversals, and also the design and evaluation of alternative interaction mechanisms to support the user goals in the identified range of variations.

References

1. Barbosa, S.D.J., de Paula, M.G.: Designing and evaluating interaction as conversation: A modeling language based on semiotic engineering. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 16–33. Springer, Heidelberg (2003)
2. Carroll, J.M., Mack, R.L., Robertson, S.P., Rosson, M.B.: Binding objects to scenarios of use. International Journal of Human-Computer Studies 41(1-2), 243–276 (1994)
3. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(1-2), 3–50 (1993)
4. de Souza, C.S.: The Semiotic Engineering of Human-Computer Interaction. MIT Press, Cambridge (2005)
5. de Souza, C.S.: Semiotic engineering: Bringing designers and users together at interaction time. Interacting with Computers 17(3), 317–341 (2005)
6. Dourish, P.: What We Talk About When We Talk About Context. Personal and Ubiquitous Computing 8, 19–30 (2004)
7. Eco, U.: Semiotics and the Philosophy of Language. Indiana University Press, Bloomington IN (1984)
8. Eco, U.: Theory of Semiotics. University Press, Bloomington (1979)
9. Fillmore, C.: The case for case. In: Bach, E., Harms, R.T. (eds.) Universals in Linguist Theory. Holt, New York (1968)

10. González-Baixauli, B., Laguna, M.A., Leite, J.C.S.P.: Aplicación de un Enfoque Intencional al Análisis de Variabilidad. In: Proceedings of the 8th Workshop on Requirements Engineering, Porto, Portugal, pp. 100–111 (2005)
11. Hackos, J.T., Redish, J.C.: User and task analysis for interface design. John Wiley & Sons, New York (1998)
12. Hui, B., Liaskos, S., Mylopoulos, J.: Requirements Analysis for Customizable Software: a Goals-Skills-Preferences Framework. In: Proceedings of the 11th IEEE International Requirements Engineering Conference, pp. 117–126 (2003)
13. Jakobson, R.: Linguistics and Poetics. In: Sebeok, T. (ed.) Style in Language, pp. 350–377. MIT Press, Cambridge (1960)
14. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. Wiley, Chichester (1998)
15. Kühme, T., Malinowski, U. (eds.): Adaptive User Interfaces: Principles and Practice. North-Holland, Elsevier (1993)
16. Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., Mylopoulos, J.: On Goal-based Variability Acquisition and Analysis. In: Proceedings of the 14th IEEE International Conference on Requirements Engineering, pp. 76–85 (2006)
17. Lieberman, H., Paternò, F., Wulf, V. (eds.): End User Development. Springer, Heidelberg (2006)
18. Mackay, W.E.: Triggers and barriers to customizing software. In: Proceedings of CHI 1991, New Orleans, USA, pp. 153–160 (1991)
19. McGrenere, J.: The design and evaluation of multiple interfaces - a solution for complex software. PhD thesis. Department of Computer Science, U. of Toronto, Canada (2002)
20. Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Exploring alternatives during requirements analysis. IEEE Software 18(1), 92–96 (2001)
21. Nielsen, J.: Heuristic Evaluation. In: Nielsen, J., Mack, R.L. (eds.) Usability Inspection Methods. John Wiley & Sons, New York (1994)
22. Peirce, C.S.: Collected Papers. Harvard University Press, Cambridge (1931-1955); excerpted in Buchler, Justus (ed.): Philosophical Writings of Peirce. Dover, NY (1955)
23. Preece, J., Rogers, Y., Sharp, E.: Interaction Design: Beyond Human-computer Interaction. John Wiley & Sons, New York, NY (2002)
24. Silva, B.S., Barbosa, S.D.J., Leite, J.C.: A Language-Based Approach to Variability Analysis. In: Proceedings of WER 2008 (2008)
25. Silva, B.S., Barbosa, S.D.J.: Variability Analysis: From requirements engineering towards interaction design. In: Proceedings of SEW-32, Kassandra, Greece (2008)
26. Sutcliffe, A., Fickasm, S., Sohlberg, M.M.: PC-RE: a method for personal and contextual requirements engineering with some experinence. Requirements Engineering 11, 157–173 (2006)