Anais do II IHC - Workshop sobre Fatores Humanos em Sistemas Computacionais (CD-ROM). Campinas, Brazil. 1999.

Users' Perceptions of Integrated GUI + EUP Communicative Environments

Alberto Barbosa Raposo
Léo Pini Magalhães
DCA – FEEC
Universidade Estadual de Campinas
C.P. 6101, Campinas, SP, Brasil, 13083-970
+55 19 788 3706
{alberto, leopini}@dca.fee.unicamp.br

Clarisse Sieckenius de Souza
Depto. de Informática – PUC-Rio
R. Marquês de São Vicente, 225
Rio de Janeiro, RJ, Brasil, 22453-900
+55 21 529 9462 ext. 4344
clarisse@inf.puc-rio.br

RESUMO
GUI (*Graphical User Interfaces*) e EUP (*End User Programming*) são tecnologias convergentes. Elas devem ser utilizadas de maneira complementar para melhorar não só a usabilidade da interface com o usuário, mas também o entendimento do comportamento do sistema. Este artigo introduz questões relacionadas a EUP e GUI, e apresenta um caso de estudo com um sistema de animação por computador que usa o paradigma de EUP. Os projetistas do sistema apresentam os resultados desta experiência. Sua principal contribuição é harmonizar os paradigmas de GUI e EUP para diminuir a distância e integrar as capacidades de programadores e artistas à procura de um melhor entendimento do comportamento do sistema sob a perspectiva do usuário final.

Palavras chave
Interfaces gráficas, programação para o usuário final, animação por computador, linguagens de programação.

ABSTRACT
GUI (Graphical User Interfaces) and EUP (End User Programming) are converging technologies. They should be used in a complementary way in order to enhance not only interface usability but also the way users understand system behavior. This paper introduces EUP- and GUI-related issues, and presents a case study carried out with a computer animation system using an EUP-like approach. The designers of the EUP language report the lessons learned from this experience. Its main contribution lies in harmonizing GUI and EUP approaches in order to shorten the gap between and integrate the skills of computer scientists and artists in search of a better understanding of systems behavior from an end user perspective.

Keywords
Graphical User Interfaces, End User Programming, Computer Animation, Programming Languages.

1. INTRODUCTION

Usability can be defined as a combination of the following factors (Shneiderman, 1992): ease of learning, high speed of user tasks' performance, low user error rate, user retention of constructs over time, and subjective user satisfaction. Moreover, usability is also related to the ability of users in applying known software to novel problem situations (Adler and Winograd, 1992). This requires not only that user interfaces support learning and understanding of computer application conceptual models, but also that they support creative usage encouraging analogies and generalizations motivated by interface design features.

End User Programming (EUP) introduced a new scenario in Human-Computer Interaction. User interfaces can now provide people not only with a means to fully explore the resources embedded in computer applications, but also with a means to customize, extend, and/or combine resources for other contexts of needs and usage. Furthermore, they can help users get closer to their application design environment, reducing the gap between system and interface. These ideas lead into the discussion about which kind of programming language is better suited for end users.

On the one side, some researchers claim that End User Programming Languages (EUPL) should spare users from dealing with the syntactic difficulties of typical programming languages, and offer a higher abstraction level for programming. The most appropriate EUPL would be natural languages. However, due to the unsurmountable complexities of these languages, there have not been encouraging results with experiments trying to make computers understand human languages.

Visual programming and programming by demonstration have emerged as promising solutions (Chang, 1990; Cypher, 1993; Myers, 1986). Their main advantage is claimed to be that it is easier for users to program in the same abstraction level of the application's GUI (Graphical User Interface). However, researchers recognize that, although powerful for some purposes, these tools have limitations. Visual programming, for example, may get unwieldy in handling some programming concepts, such as iterations and conditionals (Myers, 1992). Programming by demonstration is not adequate for non-repetitive tasks (e.g., conditionals and random user actions), besides posing problems related to wrong inferencing (i.e., the computer's interpretations of user's demonstrations) (Cypher, 1993; Nardi, 1993).

On the other hand, due to the limitations of more informal languages, there is a current belief that the EUPL can or should be real formal programming languages (Eisenberg, 1995; Gentner and Nielsen, 1996; Myers, 1992; Nardi, 1993; de Souza, 1996). The argument is that formal languages are "specifically designed to enable the kind of unambiguous, precise communication demanded by a machine" (Nardi, 1993).

However, two fundamental challenges need to be overcome in order to develop successful formal EUPL:

- How to combine GUI and EUP in a communicative environment?
- How to make the formal EUPL easy and attractive to the end user?

This paper presents an experience developed in the context of ProSIm (see <http://www.dca.fee.unicamp.br/projects/prosim/prosim.html>) where TOOKIMA 2.0 a *Too*l *Ki*t for scripting computer *M*odelled *A*nimation (Raposo, 1996; Raposo, 1997) reflects a design effort to harmonize GUI and EUP languages into a communicative environment aiming to introduce users into programming.

In the next section GUI and EUP will be briefly discussed. Then, the issues regarding GUI and EUP will be discussed in the context of a case study built in an animation environment based on TOOKIMA.

2. USER INTERFACE

We adopt the view that User Interfaces are metacommunications artifacts (de Souza, 1993). They are designed to convey a message from system designer to system user, and their intended meaning is the answer to two fundamental questions:

- What kinds of problems is this application prepared to solve?
- How can these problems be solved?

GUI and EUP offer different kinds of tools to help answering these questions.

Combining GUI and EUP

GUI style interfaces (Apple, 1992; Microsoft, 1995) offer pulldown menus and lists, buttons, icons, pointers, sliders, status and scroll bars, sometimes canvas to text and/or objects direct manipulations. This interaction style is based on some idealized metaphor related to the user environment (e.g. the desktop metaphor of the Macintosh), codified in the User Interface Language (UIL).

EUP languages, in their turn, offer a wealth of resources, normally based on application domain models, which frequently exceeds a user's typical knowledge about computers and computing. Common EUPL features are reference mechanisms for domain objects and conditional structures that can be used to express novel processing instructions, and this empowers users to design their own software. The design challenges faced by EUP can be summarized as that of designing a chain of intended high-level interpretations of computer constructs that can lead from algorithms to tasks, from data structures to domain objects, from applications to resources. In EUP, some concepts provide the interpretive guideline along which users perform extensions, analogies, and semantic transformations as they increase their insight about computation.

In order to help users understand the application's commands, the objects and operators provided by the EUPL should be consistent and continuous (i.e., have meanings that can be connected to each other in a continuous short chain of reasoning and thought) with those of the UIL (Barbosa, 1999). By interpreting EUPL commands and operators, the user should be able to create a mental model for the language and the machine that processes it. The way elements are presented in the language, their coherent bindings with UIL objects, and the user's own experience, all influence the construction of such model. The closer the model is to the language semantics, the faster will the user understand that language (Barbosa *et al*, 1997).

Several studies exploring the interaction between humans and language formalisms suggest that there should be a bridge from interface to programming (DiGiano, 1996; Eisenberg, 1995; Stenning and Gurr, 1997). Some of them take a semiotic approach, which adds a qualitative factor to analysis and design of user interfaces in general, since it can provide explanations and make predictions that are theoretically connected to each other (Barbosa *et al*, 1997; Barbosa, 1999; de Souza, 1996; de Souza, 1997).

Formal and Attractive EUPL

Eisenberg claims that well-designed applications should have graphical user interfaces, suitable for naive users, and should be so designed as to "keep an eye toward leading the user gently into programming" (Eisenberg, 1995). In other words, applications should contain both an extensive, learnable user interface, and an interpreter for a corresponding programming language.

This concept of "gently" introducing the user into the formal language is essential, specially because users of an EUP system expect to solve simple problems within a few hours of use. It is necessary to avoid the problem of "having to know everything to do anything", common in conventional programming languages (Nardi, 1993).

Another key issue for attractive EUPL is to make them task-specific. It is necessary to have in mind that the end user is not an underskilled user, but a specialist in a certain domain, who is interested in achieveing some specific computer tasks. The EUPL designer should be aware of the user's knowledge domain in order to develop a language that "looks familiar" to the user. A language using familiar primitives, well-known functions, and professional jargons will certainly increase the end user's will to learn and use it.

It is important to clarify that by "formal EUPL" we don't mean mathematical formal languages and the like, many times difficult even for experienced programmers. In this context, "formal" simply means precise and unambiguous, as is the case of musical notation, the alphabet, knitting instructions, and baseball scoring notation, all of them readily learned by their respective domain specialists (Nardi, 1993).

3. A CASE STUDY

This section explores the complementarity between GUI and EUP by means of a case study. It briefly surveys TOOKIMA 2.0 and then discusses its application's interface and embedded languages more extensively.

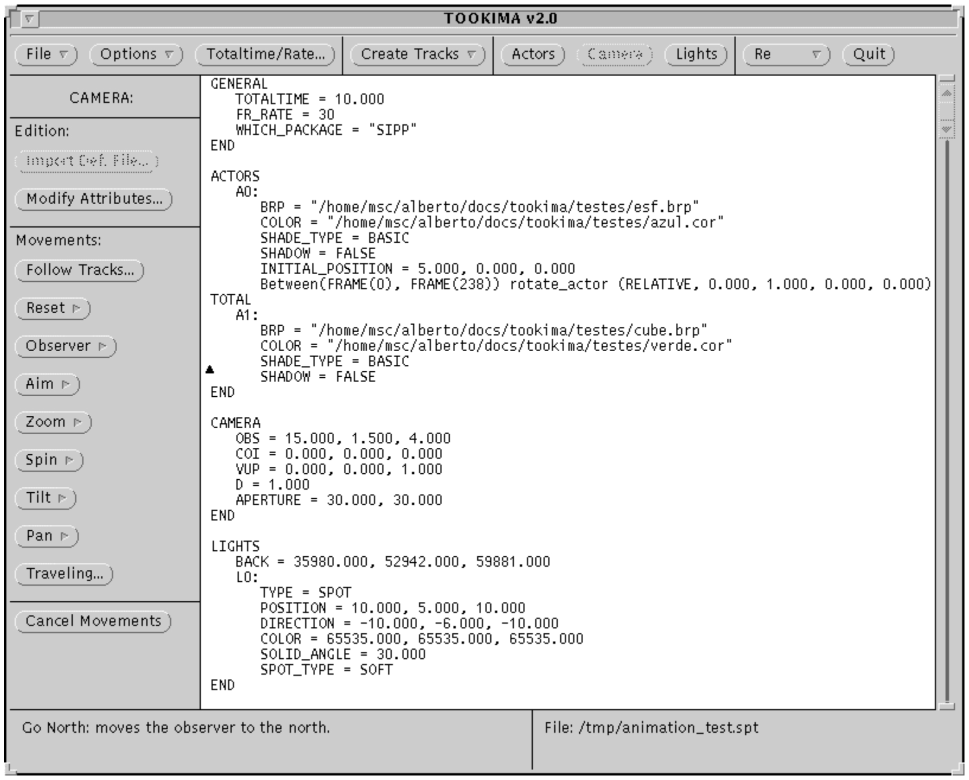
TOOKIMA 2.0

TOOKIMA is an academic computer animation environment designed to offer non-programmers (e.g., artists) a simple means of making relatively complex computer modeled animations.

The TOOKIMA design environment is "scene-oriented" i.e., oriented towards the creation of elements such as actors, cameras, lights, etc. It is divided into modules, each of them dealing with a specific object of the scene; for instance, everything related to the camera goes in the module CAMERA of the script, everything related to the illumination goes in the module LIGHTS, and so on.

This design environment offers two distinct levels of abstraction, the *interactive* and the *scripting* levels of one and the same "scene-oriented" paradigm.

At the *interactive level* TOOKIMA is built out of conventional graphic widgets like menus, buttons, etc, offering all the necessary functions for the construction of an animation. This enables naive users to create animations almost immediately, without having to know the scripting language. The work area of TOOKIMA's interface is shown in Figure 1.



```
GENERAL
   TOTALTIME = 10.000
   FR_RATE = 30
   WHICH_PACKAGE = "SIPP"
END

ACTORS
   A0:
      BRP = "/home/msc/alberto/docs/tookima/testes/esf.brp"
      COLOR = "/home/msc/alberto/docs/tookima/testes/azul.cor"
      SHADE_TYPE = BASIC
      SHADOW = FALSE
      INITIAL_POSITION = 5.000, 0.000, 0.000
      Between(FRAME(0), FRAME(238)) rotate_actor (RELATIVE, 0.000, 1.000, 0.000, 0.000)
TOTAL
   A1:
      BRP = "/home/msc/alberto/docs/tookima/testes/cube.brp"
      COLOR = "/home/msc/alberto/docs/tookima/testes/verde.cor"
      SHADE_TYPE = BASIC
      SHADOW = FALSE
END

CAMERA
   OBS = 15.000, 1.500, 4.000
   COI = 0.000, 0.000, 0.000
   VUP = 0.000, 0.000, 1.000
   D = 1.000
   APERTURE = 30.000, 30.000
END

LIGHTS
   BACK = 35980.000, 52942.000, 59881.000
   L0:
      TYPE = SPOT
      POSITION = 10.000, 5.000, 10.000
      DIRECTION = -10.000, -6.000, -10.000
      COLOR = 65535.000, 65535.000, 65535.000
      SOLID_ANGLE = 30.000
      SPOT_TYPE = SOFT
END
```

Go North: moves the observer to the north.

File: /tmp/animation_test.spt

Figure 1: the main window of TOOKIMA 2.0.

On top of the main window, there is a button bar with affordances for the construction of the animation script and the general control of the system (file manipulation, general parameters, etc). Some buttons, if pressed, cause a button column to appear on the left-hand side of the screen. This column functions as a submenu of the button in the top line and is divided into three parts: editing, creation of movements and their cancellation. In the editing part, the element can be created and altered. In the movement specification part, both the position of the element and its physical characteristics can be altered. In the cancellation area, the movements of the element can be deleted.

The text area, where the animation script is written (by the system or by the user), occupies most of the main window surface.

The third part of the main window is the message area, at the bottom. This area shows the name of the script being edited (on the right-hand side) and messages regarding the current command (on the left-hand side). These messages provide online help tips.

There are also two levels for movement control at the interface. On the upper level, actors, camera and light sources are associated with previously created trajectories (the interface also allows for the creation of these trajectories, using a "trajectory editor"). On the lower level, movements are directly defined, by the use of commands such as *translate* and *rotate* (for actors); *zoom*, *travelling* and *pan* (for the camera).

At the *scripting language* abstraction level, the users can actually write programs using the TOOKIMA scene-oriented scripting language. The ordered sequence of commands is interpreted unambiguously by the animation system in the way described above.

It is necessary to emphasize here the fact that the authors' attention is not geared towards good interface design from a complete integrated perspective that joins cognitive aesthetic and functional requirements, but rather geared towards getting the application's functionality and usability across to users.

The use of both interface alternatives is possible in TOOKIMA, a usual feature in many personal computing applications (e.g., the Homesite HTML Editor (Homesite, 1998) offers GUI widgets to compose the document's structure – HTML tags – and a text editor, where the content is written), but unusual in animation ones. TOOKIMA's scene-oriented scripting language is an essential feature of the system because it allows a full control of the animation (not only a visual control) and facilitates incremental design (i.e., consecutive parameters' adjustments to achieve the desired effects), which is very important in the animation development process.

TOOKIMA's languages will be considered in more detail in the following.

GUI and EUP in TOOKIMA 2.0

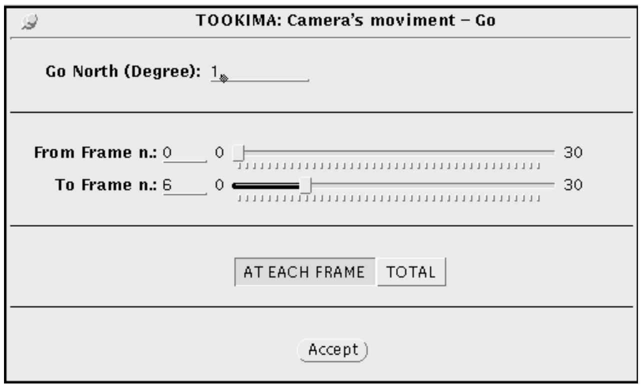
GUI (interactive) and EUP (scene-oriented scripting language) approaches are combined in the TOOKIMA, according to the idea that well-designed EUPL should provide a balanced combination of text and graphics. As stated by B. Nardi, "text and graphics each have their own special strengths and weaknesses, and the best strategy is to exploit each according to its particular characteristics" (Nardi, 1993).

The challenge was to design an environment where GUI and EUP were harmonized so as to lead users into progressively deeper understanding and familiarity with programming in TOOKIMA's embedded language.

As in any other interface, the focus of abstraction in both levels is different. At the GUI level, users are allowed to concentrate more closely on task components; required parameters and structures present in scripts can be either provided by default values or prompted for by unobtrusive interactive means (like dialogue boxes, buttons, and the like). At the scripting level, however, task-related focal signs present in the interface are repeated and integrated into extended linguistic constructs that give evidence of explicit control mechanisms and structural articulation of TOOKIMA objects.

The large text area of the interface (Figure 1) is the place where the script can be directly manipulated (similarly to text editing). While the script is being constructed by the graphic elements of the interface, it is automatically updated in the text area, "gently" introducing a naive user into programming, and not establishing a clear cut between naive and skilled users. Eventually, users are expected to be able to generate scripts directly, if they so wish. This "introduction to programming" is interesting because, as stated previously, certain tasks are better and more easily executed by the use of scripts.

In order to achieve harmony between GUI and EUP, TOOKIMA's interactive level maintains strict correspondence with the scripting level, so that interactions can be thought of as being interpreted into its scene-oriented scripting language. To better explore this relationship, a simple example showing how to move the camera in a TOOKIMA's animation is presented. In order to do that, the user has to click the button *Camera* in the top row of the main window (Figure 1) and then choose the desired movement. Figure 2 shows the dialogue box for the definition of a *go_north* movement (the camera moves to the North, considering a sphere centered in the camera's viewpoint).


TOOKIMA: Camera's moviment – Go

Go North (Degree): 1

From Frame n.: 0 0 … 30

To Frame n.: 6 0 … 30

Figure 2: the dialogue box to define a camera movement (*go_north*).

In the dialogue box in Figure 2, the user has to specify a number of parameters: the degrees to be moved, the duration of the movement (initial and final frames), and if the movement will be made at each frame (i.e., the camera will move the specified number of degrees per frame) or be made incrementally, to complete the specified number of degrees at the end of the interval. When the button *Accept* is pressed, the dialogue box is closed and the following code is added to the module CAMERA of the script (using values shown in the figure):

```
Between(FRAME(0),FRAME(6))
           go_north(1.) EACH_FRAME
```

The code above appears automatically in the text area of the interface, establishing a mapping between the GUI and the EUP (interface tasks generating script commands). This mapping is essential to the apprenticeship of the scripting language by a non-programmer, since it creates a continuous link between the UIL and EUPL metaphors. Nevertheless, the perfect mapping between the graphical interface and the scripting language is difficult to be achieved, since some features of a programming language cannot be easily represented in a traditional GUI (notably variables, conditionals, and loops, among others (Myers, 1992)). The result must be a trade-off between the flexibility of the language and how well it reflects the GUI metaphor.

As stated previously, another important characteristic of well-designed EUPL is their task-specificity. TOOKIMA's scripting language uses cinema's terminology for movement functions (e.g., *pan*, *zoom*, for camera movements) and, in order to be understood by those who are not familiar with these jargon, it also uses a more direct terminology (e.g., *go_north*, *aim_left*, *translate_actor_x*).

Testing the Environment

Members of the ProSIm project group, including artists, have tested the environment for two years. They have created some experimental animations that can be found on the Web (see <http://www.dca.fee.unicamp.br/projects/prosim/galeryPS.html>). The use of the tool by project members has been continuously highlighting the strengths and weaknesses of the tool and guiding the realization of improvements.

In order to verify if users would be able to make simple animations after a little of training, understand the link between UIL and EUPL, and use the scripting language, the authors decided to carry out a case study with two groups of end users: artists and computer scientists.

Eight users took part in the experiment. None of them had ever had any previous contact with the TOOKIMA. The experiments were conducted separately for each user and consisted of three parts. Initially one of the authors introduced TOOKIMA, showing its basic features (about thirty minutes). Then users were asked to work with the tool, trying to make some animations while being observed unobtrusively by the author (from thirty minutes to more than one hour, depending on the user's excitement). Finally, the user was interviewed about his/her subjective satisfaction with the tool.

In the scenario of the experiment, after the initial introduction to the tool, the user was asked to make a simple animation (a sphere rotating around a cube). Depending on the results, he/she was asked to make improvements to it (adding new actors, moving the camera, changing the velocity of the movement, etc). The user was free to use the GUI or the scripting language. The idea was to observe how comfortable the users feel using the tool and whether the approach used would fastly guide them into the scripting language.

The long-term experience with project members had already shown the validity of the approach, but it was necessary to observe more closely the effects of the first contact with the tool. The expectations of the authors were that artists would feel initially inhibited to use the scripting language, since they supposedly did not have any experience with programming languages (because of that, computer scientists were also chosen to take part in the experiments).

Lessons Learned

After the initial introduction, all users were able to make the first animation in less than twenty minutes. In this first attempt they used the GUI to construct the animation. This indicates that the tool avoids the problem of "having to know everything to do anything". After two or three modifications, invariably, the users started using the scripting language at least for small alterations in the animation (e.g., to change the degrees of a rotation). Of course, during this experiment, the users did not have time to completely understand the scripting language. But the results have confirmed that the continuity between UIL and EUPL metaphors efficiently introduced the users into programming and after an initial apprentice stage users tend to prefer the use of the scripting language for some purposes, in spite of the GUI.

Regarding the subjective satisfaction with the tool, the artists had a general opinion that it is easier to work with tools that allow a graphical control of the animation (like 3DStudio), but they understood the importance of the scripting paradigm and agreed that the approach used makes it more accessible. The computer scientists thought the tool is easy to use and the mapping between UIL and EUPL is well implemented. Another interesting opinion that appeared in the interviews is that the tool aims to shorten the gap between artists and computer scientists only in one direction: it introduces artists into programming. According to this opinion, in order to shorten the gap in both directions, it would be necessary to introduce computer scientists into the field of computer animation.

By comparing such outcomes with the initial expectations, we can perceive the edge of a misconception. The artists, despite their different background, had a similar performance to that of computer scientists in the first contact with the tool. The reason for that is, perhaps, the fact that artists who took part in experiment were not underskilled computer users; all of them had some experience with computer graphics (however, this is exactly the kind of user for whom TOOKIMA was designed).

The experience developed within this case study has shown that using domain-specific EUPL and a coherent mapping between EUPL and UIL the end user becomes able to explore both the convenience of a graphical interface and the power of a programming language. This confirms that the combination GUI + EUP is a valid approach in the search of an appropriate mechanism to shorten the gap between programming and non-programming communities, denying the traditional idea that graphical interfaces and programming are orthogonal approaches.

4. CONCLUDING REMARKS

EUP environments are present in many current systems, one of the most popular of them is the Microsoft Word for Windows (see e.g. (Barbosa *et al*, 1997)). This work used some of the EUP ideas in order to support users when working in an animation system environment. The experiences were related to the use of EUP principles to reduce the gap between system and end users metaphor.

In the presented context EUP has been shown to be a very powerful paradigm for incremental design and repetitive tasks, while GUI, a pleasant one for common use. The same is true in many other applications, as for example in the wordprocessors application.

Given our experience, the ideal solution for the interface seems to be a compromise between GUI and EUP. This compromise guided the presented case study, which attempts to harmonize and integrate both techniques under the same umbrella, and in a continuous way that permits users to switch between both paradigms.

Successful animation systems should combine the skills of computer scientists and artists shortening the gap between these two communities. Harmonizing GUI and EUP is a step toward this shortening.

ACKNOWLEDGEMENTS

The first author is sponsored by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo). The authors also wish to thank the Dept. of Computing Engineering and Industrial Automation (DCA) of the Faculty of Electrical and Computing Engineering (FEEC), Unicamp (State University Campinas) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) for the expressive support granted to this research. Thanks also to all the members of the ProSIm project (specially José Tarcísio de Camargo, Fernando Lamanna and Marcelo Malheiros) and the people that gently spent their time using the TOOKIMA for the case study.

REFERENCES

Adler, P. and Winograd, T. (1992). *Usability: Turning Technologies into Tools.* Oxford University Press.

Apple Computer (1992). *Macintosh Human Interface Guidelines*. Addison-Wesley, Reading, Mass.

Barbosa, S., Cara, M. P., Cereja, J. R., da Cunha, C. K. V. and de Souza, C. S. (1997). Interactive Aspects in Switching between User Interface Language and End-User Programming Environment: A Case Study. *Proc. of the III Workshop on Multimedia, Hypermedia and Human-Computer Interaction (WOMH'97)*, pp. 107-118.

Barbosa, S. D. J. (1999). *Programação via Interface*. PhD. Thesis. Dept. of Informatics, PUC-Rio, Rio de Janeiro.

Chang, S. K. (1990). *Visual Languages and Visual Programming*. New York, Plenum Press.

Cypher, A. (ed) (1993). *Watch What I Do: Programming by Demonstration*. Cambridge, MA. The MIT Press.

DiGiano, C. (1996). *A Vision of Highly-Learnable End-User Programming Languages*. Position Statements of Child's Play'96.

Eisenberg, M. (1995). Programmable Applications: Interpreter meets Interface. *SIGCHI Bulletin* 27(2): 68-93.

Gentner, D. and Nielsen, J. (1996). The Anti-Mac Interface. *Communications of the ACM*, 39(8): 70-82.

Homesite 3.0 HTML Editor (1998). Web page available at URL <http://www.homesite-now.com>.

Microsoft Corporation (1995). *The Windows Interface Guidelines for Software Design*. Redmont, Microsoft Press.

Myers, B. (1986). Visual Progamming, Programming by Example, and Program Visualization: A Taxonomy. *Proc. of ACM CHI'86*, pp. 59-66.

Myers, B. (ed) (1992). *Languages for Developing User Interfaces*. Boston, Ma. Jones and Bartlett.

Nardi, B. A. (1993). *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA. The MIT Press.

Raposo, A. B. (1996). *Um Sistema Interativo de Animação no Contexto ProSIm*. M.Sc. Thesis, DCA - FEEC - UNICAMP. Available at the URL <http://www.dca.fee.unicamp.br/projects/prosim/publiPS/tese_alb.zip>.

Raposo, A. B. (1997). *Uma Linguagem para Desenvolvimento de Roteiros de Animação*. Internal Report, 003/97 - DCA - FEEC - UNICAMP. Available at the URL <ftp://ftp.dca.fee.unicamp.br/pub/docs/techrep/1997/DCA97-003.ps.gz>.

Shneiderman, B. (1992). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 2nd Ed., Addison-Wesley.

de Souza, C. S. (1993). The Semiotic Engineering of User Interface Languages. *International Journal of Man-Machine Studies*, 39: 753-773.

de Souza, C. S. (1996). *The Semiotic Engineering of Concreteness and Abstractness: From User Interface Languages to End User Programming Languages*. Dagstuhl Seminar on Informatics and Semiotics, Schloss Dagstuhl.

de Souza, C. S. (1997). Supporting End-User Programming with Explanatory Discourse. *Proc. of ISAS'97 - Intelligent Systems and Semiotics - A Learning Perspective*, pp. 461-466. NIST, Gaithersburg, Md.

Stenning, K. and Gurr, C. (1997). Human-formalism interaction: Studies in communication through formalism. *Interacting with Computers*, 9: 111-128.