Making More Sense out of Users' Utterances

Simone Diniz Junqueira Barbosa
Departamento de Informática – PUC-Rio
R. Marquês de São Vicente, 225 – Gávea
Rio de Janeiro – RJ – Brasil – 22453-900
+55 (21) 512-2299 ext.3404
sim@les.inf.puc-rio.br

Clarisse Sieckenius de Souza
Departamento de Informática – PUC-Rio
R. Marquês de São Vicente, 225 – Gávea
Rio de Janeiro – RJ – Brasil – 22453-900
+55 (21) 512-2299 ext.4344
clarisse@inf.puc-rio.br

RESUMO
Este artigo trata de aplicações extensíveis sob uma perspectiva comunicativa. Nossa abordagem traz para a interface mecanismos poderosos de interpretação, que utilizam modelos do domínio e da aplicação para atribuir significado a enunciados não literais, para explicar como este significado foi alcançado e qual seria a expressão literal equivalente. Estes mecanismos geram, através de um processo de raciocínio abdutivo, metáforas e metonímias que podem explicar os enunciados dos usuários e ajudá-los a expressar suas intenções. Desta forma, também podem ser utilizados para ajudar os usuários a atingir dois objetivos: compreender os modelos subjacentes do domínio e da aplicação, se os mecanismos forem combinados com explicações; e obter uma forma mais eficiente de comunicação, com refinamentos retóricos, como destacar alguns aspectos de objetos e não outros.

Palavras chave
Engenharia Semiótica, Interação Humano–Computador, Metáfora, Metonímia, Raciocínio Abdutivo.

ABSTRACT
This article addresses extensible applications from a communicative perspective. Our approach brings to the user interface powerful interpretation mechanisms, that make use of domain and application models to assign meaning to non-literal user input, to explain to users how this meaning was reached and what is the corresponding literal expression. These mechanisms do so by an abductive process of generating volatile metaphors and metonymies that may explain users' input and help express users' intentions. In this way, they can also be used to help users achieve two goals. First, to understand the underlying domain and application models, if combined with explanations. The other is a more efficient way of communication that can serve rhetorical purposes such as focusing on some aspects of objects and not others.

Keywords
Semiotic Engineering, Human–Computer Interaction, Metaphor, Metonymy, Abductive Reasoning.

1. INTRODUCTION

Academia and industry alike have recognized the need to create extensible software in order to allow users to configure applications and make the best use of them for their particular purposes (Eisenberg, 1995; DiGiano and Eisenberg, 1995; Myers, 1992; Nardi, 1993; Barbosa et al., 1999). Nevertheless, most extension mechanisms proposed have focused primarily on automation of repetitive tasks using different techniques, such as macro recording, programming by demonstration (Cypher, 1993), or script and programming languages. Moreover, these approaches require that users understand how to get the task done in the first place.

We need to provide support for users to understand how to do a certain task, before trying to automate it. We do this by adopting a Semiotic Engineering perspective (de Souza, 1993; de Souza, 1999). According to de Souza, an application is a one-shot message to users, about how to send and receive other messages. It is a peculiar kind of message, because it is itself an artifact than can send and receive messages to users, and the messages it sends are exactly the same every time they are emitted under the same context and set of circumstances.

By following Semiotic Engineering principles (de Souza, 1999), we maximize the chances of effectively communicating the designer's interpretations and assumptions to users. In an extensible application, this is critical because users will assume the role of designers, albeit limited, and create new meanings in the application. Therefore, such an application must be designed for communicating knowledge relevant to the design task as well.

Our approach will not provide fully extensible applications, but instead a particular kind of volatile extension mechanism, that we call extension by interpretation (Barbosa, 1999). This mechanism allows users to utter metaphorical or metonymic expressions through the interface, then tries to make sense of this utterance and come up with a result by performing a task. This sense-making is in fact an abductive process (Peirce, 1931; Hintikka, 1997) of generating possible interpretations by means of metaphorical and metonymic operators.

This approach constitutes an extension in the sense of creating new meanings, but it does not involve a programming activity that would result in a permanent or persistent extension to the application. This raises an important issue: users' extensions to the application are generally persistent, and cannot be removed except by an explicit and sometimes obscure command triggered by the user. Since extensible applications may provide little or no disclosure about what the extensions actually mean and accomplish, persistence becomes a very dangerous feature, because it may turn a simple localized mistake into a permanent one.

One alternative would be to try to maximize users' understanding of the extensions they make, by progressively disclosing the commands and programming structures involved (DiGiano and Eisenberg, 1995; DiGiano, 1996). Our approach goes one step further and allows users to add new meanings through metonymic or metaphorical expressions that hold only in particular situations. But we also allow users to make more permanent extensions with the support of interactive dialogs, such as wizards. In this latter case, all the underlying models of the application are communicated to users in more detail.

2. ANALOGICAL REASONING

Many researchers in the field of Cognitive Science agree that we humans think and express ourselves in non-literal ways (Lakoff and Johnson, 1980; Lakoff, 1987; Ortony, 1993). In particular, we make use of metaphors and metonymies in order to understand or explain a concept in terms of others, by highlighting a concept's characteristics or relations, and hiding others. It is also known that, in order to effectively use this kind of figurative language in our communication, it is necessary for sender and receiver to share some knowledge, assumptions, and culture (Lakoff and Johnson, 1980).

When it comes to Computer Science, it is necessary to establish the designer's knowledge and assumptions and communicate them to users using some representation language and some user interface language (Barbosa et al., 1998; da Silva et al., 1997). A balance must be found between the interface language, which users must understand and in which they have to express themselves, and the application language(s) the machine is able to interpret and process.

It is impossible to represent in a software application all common sense that arises from our experience as humans interacting with the world, in order to provide mechanisms to interpret every known kind of metaphor. Nevertheless, users should not expect computer applications to behave like partners in a *natural* communicative process, where speaker and hearer can negotiate meaning until some convergence is reached. Instead, they should be aware that they are interacting with an artificial artifact created by a human designer, who represented in it some of this knowledge and assumptions about the application domain and users' needs (de Souza, 1996).

As Nardi points out, dealing with artificial languages per se is not the problem (Nardi, 1993). She states that the problem is with the languages people are expected to understand. We believe another great problem lies in the lack of knowledge about the application and domain models, which must be correctly understood in order to successfully interact with the application. However, when first interacting with an application, users typically do not have a complete and precise model of it (except those who participated throughout the application development process). One of the major usability challenges is to reduce the time necessary to learn to use an application (Nielsen, 1993; Preece, 1994). Our approach helps to follow this direction, not by describing some more user interface design principles, but by enhancing the representation of domain and application models, and providing abductive mechanisms for metaphorical and metonymic interpretation.

Why would users want to express themselves metaphorically or metonymically? Well, maybe they wouldn't, if they knew how not to! The point is, people are seldom aware of their use of metaphors, unless they are really going for a poetic effect (Jakobson, 1960). We sometimes make use of metaphorical expressions just because we understand a concept in terms of another that is familiar to us. By means of our interpretation mechanism, when a user has interacted with a portion of the application and made up a partial conceptual model of it, he or she will be able to refer to this known portion when trying to get things done somewhere else in the application.

When Halasz and Moran made their case against using analogies (Halasz and Moran, 1982), they were right about the dangers of "globalizing" a local analogy, i.e., take an analogy that holds for a particular situation and considering it valid throughout an application or a domain. The distinct feature of our model that avoids this problem is the volatility of extensions made during the interpretation of users' utterances. This volatility ensures that a metaphorical or metonymic expression will only be considered valid in the context where it was issued.

After several sessions of interaction with the application, patterns of occurrence of particular kinds of metaphorical and metonymic expressions may arise, and the application may offer users an option to make them persistent. This would be the computational equivalent of turning a live metaphor into a dead metaphor, such as the "leg of the table". In the next section we will describe the representations and calculations involved in metaphor and metonymy interpretation.

3. MODELLING FOR METAPHORS AND METONYMIES

Calculating metaphors and metonymies require some representations, namely of: static and dynamic domain and application models, enriched by classifications. In order to describe how the domain and application models must be represented, first we need to understand what is a user utterance, and what variations of literal utterances we want to be able to try to interpret.

A user utterance is a syntagmatic expression, a combination of elements in the user interface language in a sequence and following certain syntactic rules, or grammar (Saussure, 1916). The interaction paradigm determines the form of the expression: object+verb, verb+object, and so on. We consider an utterance *literal* when the application is able to provide a clear, direct and indubitable interpretation, i.e., whose form and meaning are completely defined within the underlying application models.

A metonymic utterance occurs when someone refers to an element by mentioning another element with which the first has a relation of part-whole, content-container, cause-effect, producer-product, and so on. For instance, when we say "He's got a *Picasso*", we mean he's got a work of art produced by Picasso. In a computer application, we might express "copy the *boldface*", to mean "copy the text formatted in boldface". A simple example of an existing application of metonymic reasoning occurs in some text editors: when there is nothing selected and the text cursor is inside a word, choosing *bold* formatting sets this word in boldface. This is an instance of a content–container metonymic chain (cursor location–word).

Still within computer applications, metonymies can also be used to generate iterations and recursions. For instance, in a graphical editor that allows users to group objects, if a user selects a group and chooses a different fill color, it iterates through all elements in the selected group and applies the chosen fill color to each element individually. Moreover, if an element is another group, the same operation is done throughout its elements, recursively. Nevertheless, these previously described usages of metonymies are ad hoc, and cannot be generalized throughout an application. This kind of metonymic reasoning generally occurs in isolated cases, usually in direct manipulation interfaces that allows for selection and grouping of diverse elements. Users must learn each situation where such facilities may be used, and cannot predict the behavior of seemingly analogous situations.

The need for a consistent approach to metonymic reasoning becomes more evident in command language interfaces where users must generally know the exact syntax of commands in order to get their job done. We need to make designers aware of potential metonymic chains and metaphorical expressions in their applications, so they can take advantage of this kind of reasoning and augment the language expressiveness, or rather, the application's abilities to generate a valid interpretation for more user utterances.

In order to be able to generate metonymic expressions, designers need to represent relations among elements, and to identify which relations may be part of a metonymic chain. Composition and aggregation relations, such as part-of, are natural candidates for metonymy. Other relations must be explicitly declared as having metonymic potential, such as relations representing location, ownership, possession, creation, and so on. From these representations, we generate metonymies using the following procedure:

For each element on a non-literal expression, traverse the metonymic chains in the static domain model, making a paradigmatic substitution and checking if the resulting expression has a literal interpretation. We will call the valid substitute *metonymic target*. The resulting command would consist of an iteration through every element in the original expression obtained by following the chain to the metonymic target, or every metonymic target reached from the original expression, depending on the direction traversed.

In a metonymy, the direction of traversal should go from "whole" or "producer" to "part" or "product", and then if the search fails it should go from "part" or "product" to "whole" or "producer". For instance, in a bibliographical domain, if we have a relation "Quincas Borba" written by "Machado de Assis" and we ask to copy "Machado de Assis", all references corresponding to literary works of "Machado de Assis" would be copied. On the other hand, if we ask to copy "Quincas Borba"'s biography, the result would be the copy of the author's biography (Machado de Assis's).

In order to be able to generate metaphorical expressions, designers need to represent similarity of domain elements, by some means of classification. For instance, let us define the classification "in volume" including "in book", "in periodical", and "in magazine", but not "book", if we ask for Smith's texts "in book" and there is none, a possible result would be the set of all of Smith's texts "in periodical" and "in magazine", but not his books.

Another use of metaphors arise when comparing the relations between pairs of elements. For example, there may be a relation "written by" linking a text to its author, and the instances "*O Cortiço* written-by Aluísio de Azevedo", and "*O Guarani* written-by José de Alencar". The expression "Aluísio de Azevedo's *O Guarani*" will result in looking for an instance of "Something written-by Aluísio de Azevedo". [Note: This interpretation is on the boundaries of poetic use. Our mechanism is not limited to non-poetic cases; it can interpret rhetorically sophisticated

utterances as well.] If there are many valid instances found, another criteria for disambiguation is called for, and the attributes of these instances will be compared to attributes of the original token "*O Guarani*". The final result could be "Aluísio de Azevedo's most famous book, *O Cortiço*".

Our mechanism for generating metaphors can also be used to create synonyms, that express the designer's idea of equivalence in particular contexts. For instance, in an academic institution there might be a "text" classification including "paper", "article", "report", allowing an interchangeable use of these terms.

Many researchers have investigated the computation of analogies (Furtado, 1992; Hoftstadter, 1995; French, 1995; Holyoak and Thagard, 1996). Our work is based on Holyoak and Thagard's criteria of similarity and structure. When a non-literal utterance is encountered, the application will first look for classifications in which the elements ocurring in the utterance may be substituted by a somewhat similar token, and check if the resulting utterance is valid. If the utterance involves two types A and B, we look for structural similarity that matches the classical analogy model A:X::Y:B. If many different alternatives are found, the mechanism may be designed to look for similarities among the attributes of the eligible types. The most similar candidate to the type occurring in the original utterance would then be the selected replacement.

We see from these representations, we may use classifications, relations, and attributes to generate and disambiguate metaphorical interpretations to user's non-literal expressions. When it is impossible to disambiguate or when there are many alternatives, the application may present these alternatives, together with an explanation about how they were generated, and the user will then be able to pick the one he or she intended, or discard all of them and try to utter another expression.

The abductive mechanisms described here for generating metaphors and metonymies are generic, but they are applied to domain-specific models. They may be used in a variety of domains, but the richness of representation will determine the potential for abductions.

Although this approach is domain-independent, it depends on the interaction style. In particular, on the level of articulation and expressiveness allowed by the interface language(s). For instance, a highly visual direct manipulation interface has a low level of articulation and expressiveness, while a command language may offer a detailed level of articulation and higher expressiveness. Researchers have shown that a combination of visual and textual language styles maximizes the benefits and help overcome the limitations of our approach (Maybury, 1993).

Next section will illustrate how these mechanisms can be use to augment the user interface language expressiveness, and to help users understand the underlying models.

4. MAKING SENSE: A SAMPLE CASE

Let us consider a simple toy application, inspired by Pattis' world inhabited by robots and beepers (Pattis, 1995). This world consists of 10 streets intersecting 10 avenues, making up a total of 100 corners. A robot named Karel can move from corner to corner, one step at a time, in the direction it is facing. It can also turn left, pick elements and put elements at its current corner. These elements can be beepers, toys, blocks, platforms, and connectors. In this world there are also walls, which the robot cannot traverse. Figure 1 illustrates the static domain model of this world.

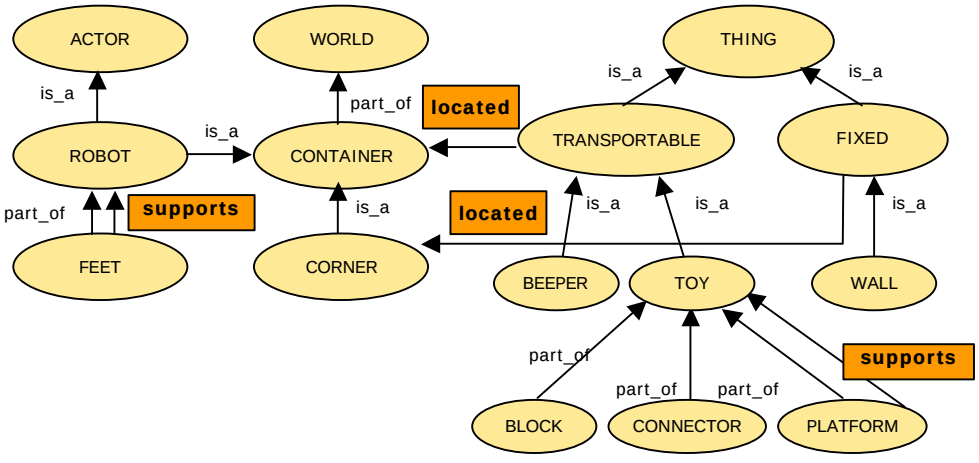

Figure 1 — Static domain model of Karel's world.

The static model presents not only inheritance (is_a) and composition (part_of) relations, but any static relations the

designer should choose to represent, in this case: supports and located. The represented types may also have attributes, which we chose not to represent in this drawing for clarity purposes.

The designer also defines which relations in the static model can be used within metonymic chains. Composition (part_of) relations are chosen by default. In our example, the designer also chose the relations located, and supports. Figure 2 shows a partial dynamic model of our domain, illustrating a few operations and their pre- and postconditions. Later in this section we will describe some metonymic and metaphorical expressions generated using these models.

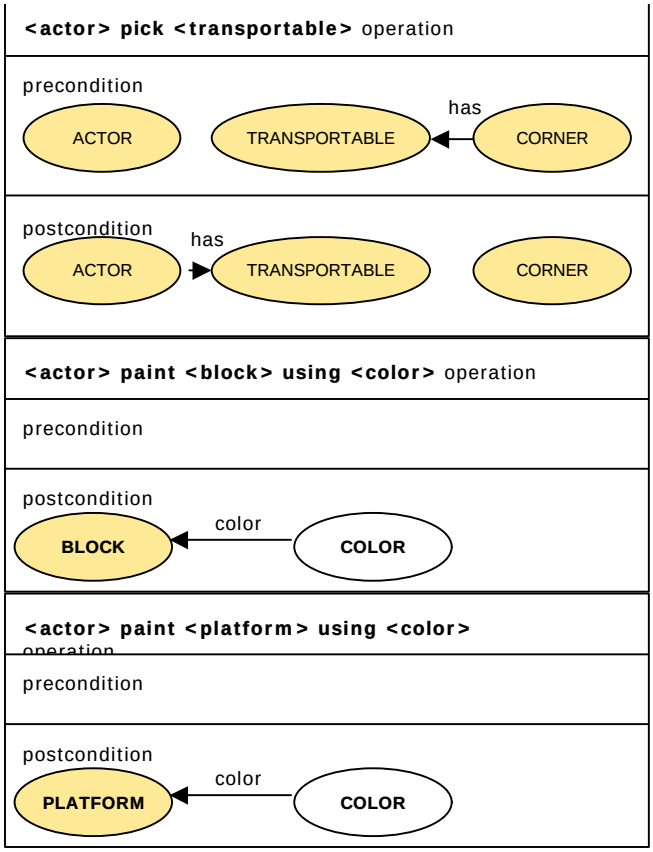

Figure 2 — Partial dynamic domain model of Karel's world.

According to Figure 2, the designer has defined three operations: <actor> pick <transportable>, <actor> paint <block> using <color>, and <actor> paint <platform> using <color>. Let us assume each type has two instances, called <type>-1 and <type>-2, and that corners are referenced as corner(X,Y).

Some classifications are implicitly defined in the static model, by inheritance relations (is_a). However, we often need additional classifications in order to obtain more sophisticated metaphorical utterances. Figure 3 presents one of the classifications from the sample domain.

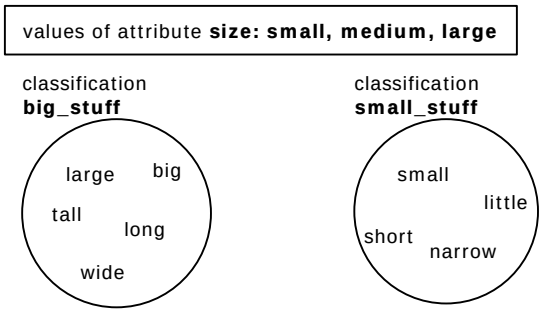

Figure 3 — Classifications to be used for generating metaphors.

From the static and dynamic models and the classifications, the following utterances exemplify metaphorical and metonymic expressions that can be generated:

user utterance	**corresponding sequences of action**	**reasoning**
robot-1 pick **platform-1**	robot pick <**toy**> where (platform-1 part_of <toy>)	metonymic
robot-1 pick **corner(1,3)**	for every <transportable> where (<transportable> located corner(1,3)), robot-1 pick <**transportable**>	metonymic
robot-1 paint **toy-1** using green	for every <block> in (<block> part_of toy-1), robot-1 paint <**block**> using green; for every <platform> in (<platform> part_of toy-1), robot-1 paint <**platform**> using green	metonymic
robot-1 paint block-1 using **block-2**	robot-1paint block-1 using **block-2.color**	metonymic
robot-1 paint toy-1.**feet**	robot-1 paint <**platform**> where (<platform> part_of toy-1)	metaphoric
robot-1 pick **big** blocks	for every <block> where <block>.size=large, robot-1 pick <**block**>	metaphoric (used here for synonym)

We have seen that there are two ways in which users may make use of metaphors and metonymies, both supported by our model. First, to understand the underlying domain and application models, if the interpretations are combined with explanations. This would be used primarily by naïve users that are familiarized to a fraction of the application. Second, expert users may use these mechanisms as a more efficient way of communication, that can serve rhetorical purposes such as focusing on some aspects of objects and not others.

An application following this model can disclose the generated interpretations, providing explanations than can

be triggered on demand or automatically. Such explanations could incorporate the models, the operations, and the utterances involved in such abductive processes, and could make use of textual languages and visual representations. The designer must be careful with the level of interference when designing for disclosure and explanations. A delicate balance between eagerness and obtrusiveness should be reached, in order to keep naïve users well informed about the interpretations, but not hinder expert users' efficient and more sophisticated usages of the mechanisms. We agree, however, that some feedback must always be produced in order to signal a non-literal interpretation of a user's input.

CONCLUSIONS

We have described a powerful mechanism to generate interpretations for a non-literal expression. An application allows users to express themselves inaccurately, by means of a metaphorical or metonymic utterance that makes reference to a known element of the domain. Following a process of abductive reasoning, our mechanism generates literal alternatives to the metaphorical or metonymic utterance. The resulting interpretation has a volatile nature, i.e., the substitution is only considered valid for the current situation and context.

Our main contribution is to bring to the interface a mechanism capable of abductively generating volatile extensions based on immediate user input. Together with metaphorical and metonymic operators, our approach widens considerably the range of users' valid utterances, and allows for users' more "natural" reasoning processes. Thus, an application that follows our approach augments the expressiveness allowed to users, and increases their chance of producing the desired result, even when they do not have a complete model of the application and underlying domain.

Another benefit arises from the fact that our mechanism can generate not only possible interpretations, but also explanations about these interpretations, so it will further disclose to users the underlying models. Since humans are said to learn and understand concepts metaphorically (Lakoff and Johnson, 1980; Lakoff, 1987; Ortony, 1993), this may also reduce an application's learning curve.

It is possible to generate multiple interpretations to a single non-literal utterance. Therefore, we need heuristics to disambiguate and give precedence to these interpretations. Our model should be applied to a variety of domains, in order to produce more refined heuristics. We believe that some universal heuristics will be found, which are valid across domains, and that more sophisticated heuristics will prove to be domain-dependent. The issue here is to decide whether these latter heuristics should be embedded in the mechanism, or disambiguation should be left to users at runtime. In our opinion, users play an important role in this disambiguation process, so applications should provide a means of interaction for selecting the intended interpretation.

This paper has described a volatile extension mechanism, used to generate situated interpretations, be it for increasing naïve users' understanding of the underlying model, or for allowing expert users' more efficient or rhetorical usages. Our model also allows for persistent extensions. In another publication (Barbosa, 1999), we propose an application that coaches users step by step through these extensions, disclosing the underlying models and processes in detail, so that users are made aware of the implications of the decisions made at each step, and of the effect of the resulting extension. Moreover, we also predict the occurrence of patterns of metaphorical and metonymic use. An application could keep a history of these usages, and allow for persistent extensions to emerge from these patterns of frequently used metaphors and metonymies.

A design tool should also be developed to guide designers through the representations needed for the abductive process of generating metaphors and metonymies. Such a tool should generate a set of metaphorical utterances corresponding to each literal utterance, and the designer would use this information to fine-tune the representation in order to correctly reflect his or her assumptions about the domain and the application.

ACKNOWLEDGMENTS

The authors would like to thank CNPq for providing financial support to this work. They would also like to thank the Semiotic Engineering Research Group at PUC-Rio for their contributions to ideas presented in this paper.

REFERENCES

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

Barbosa, S.D.J. (1999). *Programação via Interface*. Doctoral Thesis. Departamento de Informática, PUC-Rio. Rio de Janeiro.

Barbosa, S.D.J.; da Cunha, C.K.V.; da Silva, S.R.P. (1998). "Knowledge and Communication Perspectives in Extensible Applications". In *Proceedings of IHC'98*. Maringá, PR.

Barbosa, S.D.J.; da Silva, S.R.P.; de Souza, C.S. (1999). "Extensible Software Applications as a Semiotic Engineering Laboratory". To be published in *Working Papers in the Semiotic Sciences*. Legas, Ottawa, Canada.

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

da Silva, S.R.P.; Barbosa, S.D.J.; de Souza, C.S. (1997). "Communicating Different Perspectives on Extensible Software". In Lucena, C.J.P. (ed.) *Monografias em Ciência da Computação*. Departamento de Informática. PUC-RioInf MCC 23/97. Rio de Janeiro.

de Souza, C.S. (1993). “The Semiotic Engineering of User Interface Languages”. *International Journal of Man-Machine Studies*. No. 39. 753-773.

de Souza, C.S. (1996). “The Semiotic Engineering of Concreteness and Abstractness: from User Interface Languages to End-User Programming Languages”. In Andersen, P.; Nadin, M.; Nake, F. (eds.) *Informatics and Semiotics*. Dagstuhl Seminar Report No. 135, p. 11. Schloß Dagstuhl., Germany.

de Souza, C.S. (1999). “Semiotic Engineering Principles dor Evaluating End-User Programming Environments”. In Lucena, C.J.P. (ed.) *Monografias em Ciência da Computação*. Departamento de Informática. PUC-RioInf MCC 10/99. Rio de Janeiro.

DiGiano, C. (1996). “A vision of highly-learnable end-user programming languages”. *Child’s Play ’96 Position Paper*.

DiGiano, C. and Eisenberg, M. (1995). “Self-disclosing design tools: A gentle introduction to end-user programming”. In *Proceedings of DIS ’95*. Ann Arbor, Michigan. August 23-25, 1995. ACM Press.

Eisenberg, M. (1995). “Programmable Applications: Interpreter Meets Interface”. *SIGCHI Bulletin*. Apr. Vol. 27(2), ACM Press.

French, R. (1995). *The Subtlety of Sameness*. Cambridge, MA: The MIT Press.

Furtado, Antonio L. (1992). “Analogy by Generalization – and the Quest of the Grail”. *ACM SIGPLAN Notices*, Volume 27, No. 1, January 1992.

Halasz, F. and Moran, T.P. (1982). “Analogy Considered Harmful”. In *Human factors in computer systems, conference proceedings*. Gaithersubrg, Maryland. ACM Press.

Hintikka,J. (1997). *What is Abduction? The fundamental problem of contemporary epistemology*. Unpublished manuscript.

Hofstadter, D. (ed., 1995). *Fluid Concepts and Creative Analogies*. Basic Books, A Division of HarperCollins Publishers, Inc. New York NY.

Holyoak, K.J. and Thagard, P. (1996). *Mental Leaps: Analogy in Creative Thought*. Cambridge, MA. The MIT Press. 1996.

Jakobson, R. (1960). “Closing Statements: Linguistics and Poetics”. In Thomas A. Sebeok (ed.) *Style in Language*. Cambridge: The MIT Press.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things*. The University of Chicago Press. Chicago.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. The University of Chicago Press. Chicago.

Maybury, M.T. (ed., 1993). *Intelligent Multimedia Interfaces*. Menlo Park, CA: American Association for Artificial Intelligence.

Myers, B.A. (1992). *Languages for Developing User Interfaces*. London. Jones and Bartlett Publishers, Inc. Boston. 1992.

Nardi, B. (1993). *A Small Matter of Programming*. Cambridge, MA: The MIT Press.

Nielsen, J. (1993). *Usability Engineering*. Academic Press.

Ortony, A. (ed., 1993) *Metaphor and Thought*, 2nd Edition. Cambridge: Cambridge University Press.

Pattis, R.E.; Roberts, J.; Stehlik, M. (1995) *Karel the Robot: A Gentle Introduction to the Art of Programming*. New York, N.Y. John Wiley and Sons.

Peirce, C.S. (1931). *Collected Papers*. Cambridge, Ma. Harvard University Press. (extraído de Buchler, Justus, ed., *Philosophical Writings of Peirce*, New York: Dover, 1955).

Preece, J.; Rogers, Y.; Sharp, E.; Benyon, D.; Holland, S.; Carey, T. (1994). *Human-Computer Interaction*. Addison-Wesley.

Saussure, F. de. (1916). *Cours de Linguistique Générale*. Paris, Payot.