Extending software through metaphors and metonymies

Simone D.J. Barbosa
Computer Science Department
Pontifícia Universidade Católica
R. Marquês de São Vicente, 225 – Gávea
Rio de Janeiro, RJ 22453-900
Brazil
+ 55 21 521-8627
sim@les.inf.puc-rio.br

Clarisse Sieckenius de Souza
Computer Science Department
Pontifícia Universidade Católica
R. Marquês de São Vicente, 225 – Gávea
Rio de Janeiro, RJ 22453-900
Brazil
+ 55 21 529-9462 ext. 4344
clarisse@inf.puc-rio.br

ABSTRACT

This article is about applications that can be customized or extended through their own user interface. This is achieved by the interface's ability to interpret users' non-literal expressions, namely metaphorical and metonymic ones. Such increased interpretive intelligence depends on static and dynamic models of the domain and application, from which new figurative meanings are abducted automatically or semi-automatically. The system performs controlled modifications on the underlying models, based on its inferences about users' intentions as they produce figurative utterances.

Keywords

End-User Programming, Metaphor, Metonymy, Abductive Reasoning, Interfaces for Knowledge-Based Systems.

1. INTRODUCTION

Academia and industry alike have recognized the need to create extensible software in order to allow users to tailor applications to their particular needs and alleviate the programming effort involved in software upgrades [2, 7, 8, 19, 20]. Nevertheless, most extension mechanisms to-date have focused primarily on the automation of repetitive tasks (through different techniques, such as macro recording and programming by demonstration [3]), or on extensions that involve the use of script and programming languages.

Commercially available extensible applications have typically suffered from rigid and unhelpful interfaces. One of the major difficulties for extending software is the user's lack of knowledge about the underlying application model and about the role of familiar interactive patterns in turning interface events into function triggering protocols.

We propose mechanisms that help interfaces become more "intelligent" by assigning meaning to users' non-literal expressions, i.e. input that is not a primary well-formed syntactic and/or semantic interactive sentence. By interpreting metaphorical and metonymic expressions, our mechanisms recognize meaningful constructs beyond those explicitly modelled by the original designer. Thus, we are not only customizing or extending user interfaces, but actually extending software applications, through intelligent user interfaces.

In our approach, end-user programming (EUP) mechanisms are cast as abductive processes [14] that operate on figures of speech. Metaphors and metonymies have been chosen because they mirror a natural way of thinking about things we know little or nothing about [15, 16, 17, 21].

Compared to alternative approaches that try to meet major EUP cognitive challenges by progressively disclosing commands and programming structures [6, 7], ours allows users to achieve changes without even knowing that what they want to do is innovative in any sense. This feature is also distinctive in comparison to most programming by demonstration approaches [3].

However, not knowing about extensions may have undesirable side effects. Consequently, our approach distinguishes between vanishing and permanent extensions. Vanishing ones result from interpreting figurative utterances and performing the inferred actions on discardable extended copies of the original models. Permanent ones require that the users be made aware of extensions and carefully decide about the need or convenience of replacing the original models with the extended ones.

Permanent extensions can be also achieved with the support of interactive dialogs, such as wizards. In this case, all the underlying models of the application are communicated to users in more detail, and EUP is treated as an intentional user activity. However, this paper concentrates mainly on vanishing extensions.

Our approach has been partially implemented and tested for conceptual consistency in three separate modules: the intelligent UI knowledge base and reasoner, the graphic environment, and a preliminary knowledge acquisition interface. In the following, we will disclose and discuss aspects of our system's models and reasoning processes, and provide examples of interpreting vanishing non-literal expressions based on a toy application.

2. EXTENSIONS THROUGH ANALOGIES

Many researchers in the field of Cognitive Science agree that we humans think and express ourselves in non-literal ways [15, 17, 21]. 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 knowledge, assumptions, and cultural background [17].

Lakoff presents evidence of the existence of a real system of conventional conceptual metaphors. Our approach takes into consideration what he considers to be the most robust evidence [16]:

- Generalizations governing polysemy, that is, the use of words with a number of related meanings.
- Generalizations governing inference patterns, that is, cases where a pattern of inferences from one conceptual domain is used in another domain.
- Generalizations governing novel metaphorical language.

In computer applications, dealing with conceptual metaphors requires that the designer's knowledge and assumptions be communicated to users in a consistent interface language [1, 4]. For sake of computational efficiency, a balance must be found between interface language and application language, so that both are good enough for man and machine.

It is impossible to represent, computationally or not, all common sense knowledge that arises from our experience in the world. Thus, computer applications should not be expected to behave like partners in a *natural* communicative process, capable of negotiating meanings until they reach mutual understanding. Rather, computer users should be aware that they are interacting with a rational artifact created by a human designer, who consciously embedded in it only part of his or her knowledge and assumptions about the application domain and the users' profile [5].

One of the hardest EUP challenges is related to the users' lack of knowledge about such embedded application and domain models. When first interacting with an application, users typically have an incomplete and imprecise model of it. Our approach helps fill this knowledge gap with an enhanced representation of domain and application models, which are manipulated by abductive mechanisms that interpret metaphorical and metonymic utterances. These utterances often account for what is diagnosed as imprecise and incomplete knowledge in traditional approaches.

When a user is familiar with a portion of the application and has built a *partial* conceptual model of it, he or she is likely to make sense of unfamiliar portions in terms of known concepts, prone to overgeneralization or overspecification that give rise to misconceptions. So, instead of *correcting* users in an effort to adjust his or her models to the application, we try to adjust the application to meet the user's conception.

Underlying model adjustments can certainly have undesirable side effects. Our model handles this problem by supporting vanishing extensions made during the interpretation of users' utterances. This ensures that a metaphorical or metonymic expression will keep this status of figurative speech until it is explicitly integrated to the domain discourse with its corresponding syntactic and semantic specification.

One way to achieve explicit integration may occur after several sessions of interaction with the application, when patterns of metaphorical and metonymic expressions may arise. The application may offer users the option to make them persistent, as a computational equivalent of turning live metaphors into dead metaphors. In the next section we will describe representations and computations involved in interpreting metaphor and metonymy.

3. MODELLING FOR METAPHORS AND METONYMIES

Calculating metaphors and metonymies requires a special representation structure — static and dynamic domain and application models, enriched by metaphorical and metonymic chain classifications. In order to describe how the domain and application models must be represented, we first need to understand what is a user utterance, and what variations of literal utterances we want to interpret.

A user utterance is a syntagmatic expression [23], a linearized rule-based combination of elements in the user interface language. The interaction paradigm determines the form of the expression, like object+verb and verb+object, for instance. A literal utterance has a clear, direct and unambiguous interpretation, derived from the language grammar.

A metonymic utterance occurs when reference to an element is made by another element with which the first has a relation of part-whole, content-container, cause-effect, producer-product, among other possibilities. For example, 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". Still regarding 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 that can be filled, individually. Moreover, if an element is itself a group, the operation is performed on its elements too, recursively. Nevertheless, such usage of metonymies is typically *ad hoc*, or incidental, not to be consistently found elsewhere in the application. Users must learn where such metonymies may be used in isolation, for they cannot predict the behavior of seemingly analogous situations.

In order to be able to generate metonymic expressions, designers need to represent relations among elements and identify which ones 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 many others.

For each element in a non-literal expression, we traverse the metonymic chains in the static domain model, making a paradigmatic substitution and checking if the resulting expression has a literal interpretation. A valid substitute becomes the *metonymic target*. The utterance interpretation is the result 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.

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). The following pseudo-code illustrates our procedure for generating part-whole, contained-container, and other metonymies:

```
given an utterance of the form S-A-O (<subject> <action> <object>)

for each defined X-A-O,
    if there is a metonymic relation X-part-of-S or X-contained-in-S, consider an iteration, executing the utterance X-A-O for every instance of X in X-part-of-S or X-contained-in-S; then stop
    if there is a metonymic relation S-part-of-X or S-contained-in-X, consider the utterance X-A-O and stop

for each defined utterance S-A-X
    if there is a metonymic relation X-part-of-O or X-contained-in-O, consider an iteration, executing the utterance S-A-X for every instance of X in X-part-of-O, or X-contained-in-O; then stop
    if there is a metonymic relation O-part-of-X or O-contained-in-X, consider the utterance S-A-O and stop
```

Metaphors may 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 retrieving an instance of "Something written by Aluísio de Azevedo". If the underlying domain representation is rich enough to single out Alencar's *O Guarani* as his most famous novel, the metaphorical representation can qualify "something written by" with the attribute "most famous", and thus retrieve's Azevedo's novel that is, for him, as remarkable as *O Guarani* is for Alencar. We thus see that this interpretation is at the boundaries of poetic use. The appropriateness and sophistication of interpretations is directly proportional to the expressiveness of the underlying domain models.

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", and "report". A systematic interpretation of these words in terms of each other is equivalent to neutralizing the distinctions among them, and making them interchangeable in some or in all contexts.

Many researchers have investigated the computation of analogies [10, 11, 12, 13, 24]. 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 occurring in the utterance may be substituted by a 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::B:Y. If many different alternatives are found, the mechanism may look for similarities among the attributes of the eligible types. The most similar candidate to the type occurring in the original utterance will then be selected for the replacement.

We may use classifications, relations, and attributes to generate and disambiguate metaphorical interpretations for users' non-literal expressions. However, when it is impossible to disambiguate terms or when there are many alternatives for interpretation, the application may present choices to users, along with an explanation about how they were generated, and have users select the one they mean, or discard them all and try to use another form expression. The following pseudo-code represents a proposed procedure for interpreting command-related metaphors:

```
given an utterance of the form S-A-O (<subject> <action> <object>)

if there is only one construct C = X-A-O, where the substitution of S for X results in a valid operation, and there is at least one path of relations or one common classification between X and S, execute the corresponding operation and stop.
if there is only one construct C = S-A-X, where the substitution of O for X results in a valid operation, and there is at least one path of relations or one common classification between X and O, execute the corresponding operation and stop.

if there are many constructs C= Xi-A-O that constitute valid operations,
    verify for which Xi there are more classifications in common with S, execute the corresponding constructs, then stop
if there are many constructs C= S-A-Xi that constitute valid operations,
    verify for which Xi there are more classifications in common with O, execute the corresponding constructs, then stop

if there are many constructs C= Xi-A-O that constitute valid operations,
    verify for which Xi there are more relations in common with S, execute the corresponding constructs, then stop
if there are many constructs C= S-A-Xi that constitute valid operations,
    verify for which Xi there are more relations in common with O, execute the corresponding constructs, then stop

if there are many constructs C= Xi-A-O that constitute valid operations,
    verify for which Xi there are more attributes in common with S, execute the corresponding constructs, then stop
if there are many constructs C= S-A-Xi that constitute valid operations,
    verify for which Xi there are more attributes in common with O, execute the corresponding constructs, then stop
```

We acknowledge the existence of different approaches to analogy making and metaphor interpretation [9, 10, 11, 12, 13, 24] that make use of more sophisticated algorithms. Nevertheless, our main purpose here is not natural language interpretation. Instead, we want to provide users with more flexible yet artificial means of expressing instructions to an extensible application. We assume users will not need such complex mechanisms as used for natural language processing, since they will be aware that they are exchanging utterances with an artifact that has limited interpretive power.

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. However, they are not independent of interaction style. They are strongly dependent on the level of articulation (i.e. on how many parts constitute what sorts of wholes in the language) and expressiveness (i.e. on what contents can be conveyed by linguistic forms) of the interface language(s). For instance, a highly visual direct manipulation interface typically has a low level of articulation (because icons and images cannot be decomposed into many and varied meaningful constituents), whereas a command language may offer a finer level of articulation and higher expressiveness (because of natural language based regularities such as the morphology of number markers, the conjunction of phrasal constituents, and the like). Researchers have shown that a combination of visual and textual language styles maximizes the benefits and help overcome such limitations [18].

The next section will illustrate how these mechanisms can be used to augment user interface language expressiveness in the prototype application we have partially implemented.

4. UNDERSTANDING USERS: A SAMPLE CASE

We have built a simple toy application, inspired by Pattis' world inhabited by robots and beepers [22]. 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.

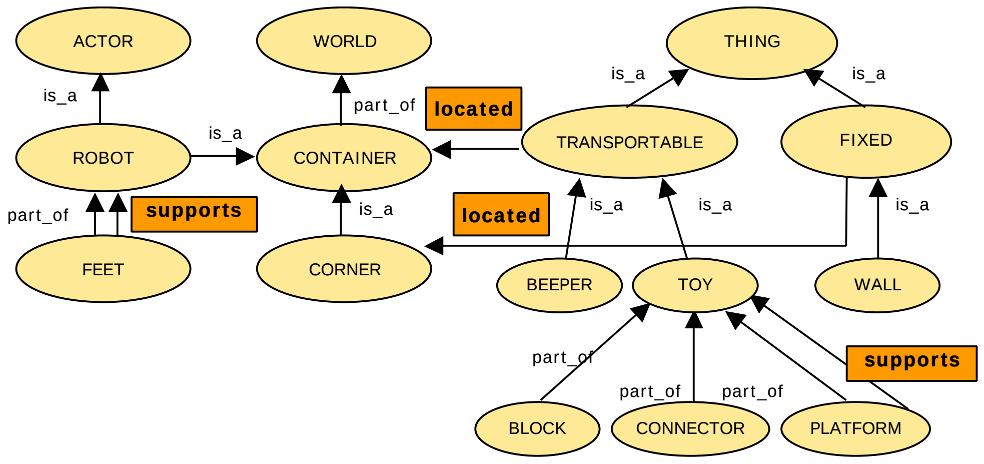
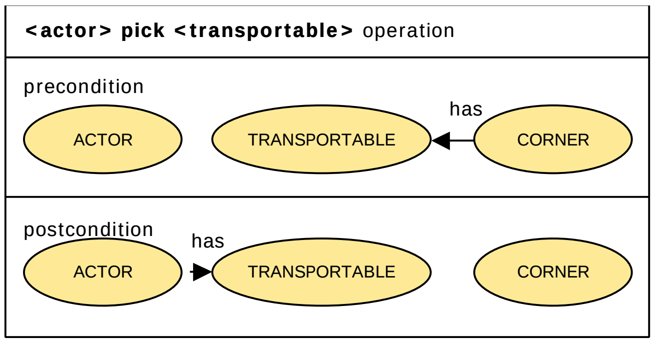
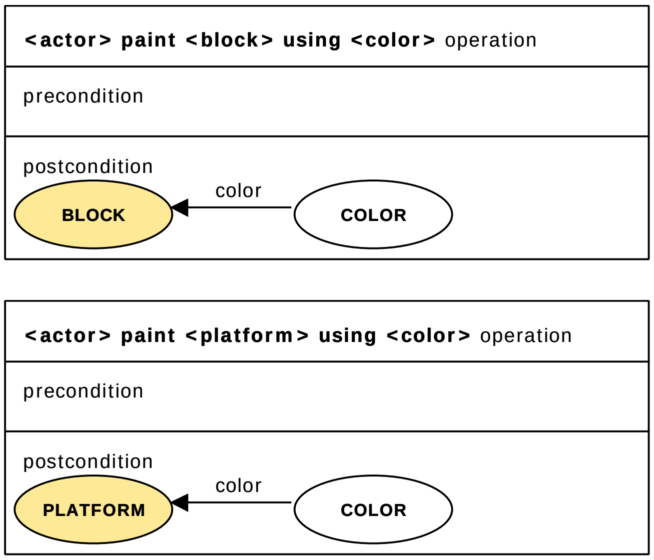


Figure 1 — Static domain model of Karel's world.

As we mentioned earlier, static and dynamic models support the interpretive processes. 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 Figure 1 we see that besides *is-a* and *part-of* relations, we have *supports* and *located*. The represented types may also have attributes, which we have chosen not to represent in this drawing for sake of clarity.

The designer must define which relations in the static model can be used within metonymic chains. Composition (part_of) relations are chosen by default. So, in this example, the designer chose the *located*, and *supports* relations. Figure 2 shows a partial dynamic model of our domain, illustrating a few operations and their *pre* and *post* conditions. Later in this section we will describe some metonymic and metaphorical expressions that are interpreted using these models.





<actor> paint <block> using <color> operation
precondition
postcondition
BLOCK
color
COLOR
<actor> paint <platform> using <color> operation
precondition
postcondition
PLATFORM
color
COLOR

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>. 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 involved in this example.

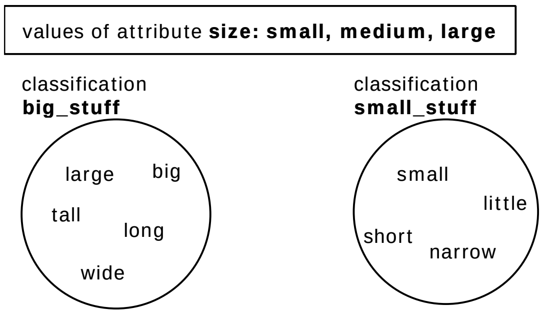


Figure 3 — Classifications to be used for generating metaphors.

Combining the static and dynamic models, on the one side, with the classifications provided by designers, on the other, the following utterances exemplify metaphorical and metonymic expressions that can be interpreted:

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>.large, robot-1 pick <**block**>	metaphoric (used here for synonym)

There are two ways in which users may benefit from metaphors and metonymies. One is that expert users may wield these mechanisms as a more efficient way of communication, which can serve rhetorical purposes such as focusing on some aspects of objects in detriment of others. For example, saying "paint corner(X,Y)" may serve to express the user's focus on the location of objects in detriment of their shape or size. The other is that they may gain more understanding about the underlying domain and application models, if interpretive chains are unfolded and combined with explanations for the system reasoning. This would be used primarily by naïve users that are unfamiliar with large portions of the application. 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 should however 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 without hindering efficiency as a whole.

5. EXTENSIBILITY

Our approach brings about some relevant changes relative to EUP techniques. Users may express themselves in a non-literal fashion, and achieve extensions without necessarily knowing they are doing so. In this section, we present exemples of literal and non-literal expressions, in order to show the inferencing capabilities of our mechanisms.

The constructs provided for creating subtypes, aggregates and relations are:

1. <A> is a <B> (used to create a subtype of B)
2. <A> is a collection of < B>(used to create an aggregate)
3. <A> <some-relation> <B> (used to create a relation between A and B)
4. <A> is like a <B> with <P> (inheritance in addition to some part or attribute)
5. <A> is like a <B> without <Q> (used to create a clone of type A, and then remove some part or attribute)
6. <A> is like a <B> with <P> instead of <Q> (used to create a clone of type A, and then substitute some part or attribute)

Therefore, we may have literal utterances like the following:

a toy is a transportable object (inheritance)

a toy is a collection of blocks (aggregation)
a platform is a toy block
a platform supports a toy (relation only)
a platform is a toy part that supports the toy (inheritance + relation)

In order for an utterance to be literal, syntactic and semantic rules must be followed. If there is any variation to these, the utterance is considered to be non-literal, and the metaphorical and metonymic interpretation mechanisms are triggered.

When a word or expression is found that cannot be understood in the provided context, the extension mechanism verifies if that word would be valid in any other context, and then maps from the valid context the appropriate elements and structure needed to make sense of it in the actual context. This consists of a mechanism to generate polysemies, or related meanings to a single word, one of the evidences of usage of metaphors presented by Lakoff [16].

Further evidence presented by Lakoff may also be accounted for. Previous abductive processes used to interpret metaphor and metonymy may be stored in order to be used as inference patterns that can be potentially generalized and used in mappings with a different domain. Moreover, our mechanisms offer ways to create types and meanings. Consider the following lines, with excerpts of interaction in a command language environment. The letter U indicates a user's utterance, and the letter A indicates an application response. In it, a morphological rule of many languages (namely that plurals are marked with the suffix *'-s')* is effective to facilitate representation.

Suppose that toy is a yet-to-be-defined type.

U: a toy is like a beeper, but with blocks and connectors
A: what is blocks (type or attribute)?
U: block is a type
A: "blocks" means one or more than one "block"?
U: yes
A: what is connectors (type or attribute)?
U: connector is a type
A: "connectors" means one or more than one "connector"?
U: yes

A more direct interface may be provided, with wizards and dialogs that present all the alternatives for a given concept at a glance, with one of them selected by the application as a default. This interactive style is more useful when we have multiple types referenced in an utterance, such as the following example:

U: a lamppost is like a wall and like a robot

In this case, the user must be prompted to select which attributes, parts, and relations are valid and which aren't, by taking as a starting point all elements related to walls and robots. If only one type is mentioned in the user's utterance, the application may provide an interpretation, tell the user it has done so and proceed, allowing the user to intervene if he finds it necessary. For instance:

U: a lamppost is like a wall, but with orientation and state
A: [I consider orientation an attribute, with one of the possible values: north, south, east, west]
A: what is state? (type or attribute)
U: attribute
A: what are the possible values of "state"?
U: on, off
A: are on, off disjoint values?
U: yes

Our approach brings about the need for a designer's tool, in which the designer can represent his model and run simulations of possible extensions to it. The tool should also provide for special tags and rules that govern how extensions are made, in which order the elements in the model are traversed when generating possible interpretations, incompatibility tags between pairs of elements, cardinality of attributes, and the like. A decision the designer must make is as to whether he should make this kind of control and rule customization available to users, and to what extent.

Let us now consider a non-literal operation extension. Given the operation defined as:

for a <robot> to pick a <beeper> from a <corner>,
before: <beeper> located <corner>,
after: <beeper> located <robot>

The following exchange illustrates the creation of a new operation based on an existing one:

U: to borrow is like for a robot to pick a beeper from another robot
A: for <robot-1> to borrow <beeper> from <robot-2>
A: before: <beeper> located <robot-2>
A: after: <beeper> located <robot-1>
U: change located to owned-by
A: for <robot-1> to borrow <beeper> from <robot-2>
A: before: <beeper> owned-by <robot-2>
A: after: <beeper> owned-by <robot-1>
U: owned-by is the inverse of owns
A: X owned-by Y <–> Y owns X
U: to rent is like for a robot to borrow a corner from another robot
A: for <robot-1> to borrow <corner> from <robot-2>
A: before: <corner> owned-by <robot-2>
A: after: <corner> owned-by <robot-1>

6. CONCLUSIONS

We have described a powerful interpretive mechanism that can handle non-literal expressions. When incorporated in an extensible application's interface, they allow users to express themselves inaccurately, by means of a metaphorical or metonymic utterance that refers to a known element of the domain. Following a process of abductive reasoning, our mechanism generates literal interpretations to the metaphorical or metonymic formulations. The resulting interpretation has a vanishing nature, i.e., the substitution is only considered valid for the current situation and context. Permanent changes must be explicitly intended by users.

Interfaces built according to this approach have enhanced interpretive intelligence, in that they are able to assign meanings by generating abductive interpretations. They also exhibit pragmatic intelligence, since they can actually act upon the application model and produce modifications equivalent to end user programmable ones, proposed by macro recording, scripting or programming by demonstration approaches.

Our main contribution for UI intelligence is that with metaphorical and metonymic operators, our approach considerably increases the users' communicative power, and allows them to follow a more "natural" pattern of thought and expression. Thus, it increases the users' chance of achieving their goals, even when they do not have a complete model of the application and underlying domain. Another related benefit arises from the fact that our mechanism can generate not only possible interpretations, but also explanations about these interpretations, if reasoning paths are unfolded. So, it can teach users about the underlying models and design assumptions. Additionally, given the power of metaphors and metonymies in our acquisition of new knowledge and information [15, 17, 21], this may potentially reduce an application's learning curve.

Because it is possible to generate multiple interpretations for a single non-literal utterance, we need heuristics to disambiguate and give precedence to interpretations. Our model should be applied to a variety of domains, in order to produce more refined heuristics. We believe that some universal heuristics may be found across domains, but that more sophisticated heuristics may prove to be domain-dependent. We should then decide whether the latter should be embedded in the interpretive mechanism, or disambiguation should be left to users at runtime.

Our future work involves testing the power and limitations of our approach in real-world applications. It is likely impractical to model a complex application in its entirety, so that its potential metaphors and metonymies can be consistently handled. Nevertheless, it may be possible to model a subset of the application in more detail, taking our interpretive and extension mechanisms into account. The designers would have to anticipate users' needs for enhanced interpretive abilities, or for extensibility, and calibrate the specificity of the model accordingly.

We acknowledge the overhead on the designers as more decisions and more unknown variables come into the picture. Nevertheless, we plan to provide them with tools to facilitate the modelling process, and to help anticipate potential distortions that may arise from the application of our mechanisms to the intended model. Such a design tool should not only guide designers through the representations, but also generate a set of metaphorical utterances corresponding to each literal utterance, so the designer would be able to use this information to fine-tune the representation and correctly reflect his or her assumptions about the domain and the application.

Finally, we believe the scalability of our approach is highly dependent of the degree of specificity of the application, due to the need for rich knowledge representation at a semantic level. Therefore, we reckon that it is more suitable for domain-specific applications, instead of general-purpose ones. Again, this assumption must now be empirically tested through a series of prototypes.

7. 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 some of the ideas presented in this paper.

8. REFERENCES

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] DiGiano, C. (1996). "A vision of highly-learnable end-user programming languages". *Child's Play '96 Position Paper*.

[7] 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.

[8] Eisenberg, M. (1995). "Programmable Applications: Interpreter Meets Interface". *SIGCHI Bulletin*. Apr. Vol. 27(2), ACM Press.

[9] Fauconnier, G. and Turner, M. "Conceptual Integration Networks". *Cognitive Science*. Volume 22, number 2 (April-June 1998), pages 133-187.

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

[11] Furtado, Antonio L. (1992). "Analogy by Generalization – and the Quest of the Grail". ACM SIGPLAN Notices, Volume 27, No. 1, January 1992.

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

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

[14] Josephson, J.R and Josephson, S.G. (eds.) (1996) *Abductive Inference: Computation, Philosophy, Technology*. Cambridge University Press.

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

[16] Lakoff, G. (1993) “Contemporary theory of metaphor”. In Ortony (ed.), *Metaphor and Thought*, 2nd Edition. Cambridge: Cambridge University Press.

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

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

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

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

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

[22] 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.

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

[24] Way, E.C. (1991). *Knowledge Representation and Metaphor*. Kluwer Academic Publishers. The Netherlands.