Working Paper Series
ISSN 1170-487X

A Teaching and Support Tool for Building Formal Models of Graphical User-Interfaces

by Steve Reeves

Working Paper 95/21
August 1995

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

A Teaching and Support Tool for Building Formal Models of Graphical User-Interfaces

Steve Reeves
Department of Computer Science
University of Waikato
HAMILTON
New Zealand
stever@waikato.ac.nz

Abstract

In this paper we propose the design of a tool that will allow the construction of a formal, textual description of a software system even if it has a graphical user-interface as a component. An important aspect of this design is that it can be used for two purposes—the teaching of predicate calculus and the formal specification of graphical user-interfaces. The design has been suggested by considering a system that has already been very successful for teaching predicate logic, namely Tarski's World.

1. Introduction

There are now many, well-documented uses of formal specification in the program development process—Diller, 1994; Dromey, 1989; Gries, 1981 are a few of the many texts in this area—that show that not only is formal specification of software desirable (and in many cases necessary, especially in the safety-critical field), for all the well-rehearsed reasons (efficiency of construction, demonstration of correctness and ease of maintenance), but it puts the design, construction and use of software on a basis that truly allows us to speak of software engineering as a discipline that is as principled, successful and well-founded as other branches of engineering such as civil and mechanical.

However, there is one area of great importance within software engineering that is particularly problematical. It is where the graphical meets the textual. Currently, the methods of formal specification which are used for describing the function of software (what it does, not how it does it) are textual and give us a basis for reasoning about the function of the software. When we describe the graphical user-interface part of some software system we currently have a problem, though. Although we can describe the interaction (via dialogues, say) in a way similar to the way in which we can describe the other parts of the system, we cannot similarly describe the way the system looks, i.e. we cannot say what it displays at each stage during a dialogue, at the same level of abstraction and formality.

We can, of course, say how it displays what it displays by providing the code to do it. We could draw pictures of what it displays, or describe what it displays in English, say. These are not, however, ways of describing what the display looks like which are at the same level of abstraction and formalization as the ways we have of describing the function of the software. Formally, the displays are second-class citizens.

Putting this another way, since we are talking about stages prior to implementation here, we do not want to have to describe the look of our systems by using any programming or other such low-level notation, since that says how the look of a display comes about, not what the look is. However, we do want the description of what the system looks like to be formal, since later we want to reason about it and, in particular, to prove that the implementation of the system really does look like what the designers and specifiers said it should look like.

Put simply, we have the problem of formally describing in words and symbols, i.e. textually, what something looks like.

This work is motivated by what appears to be a gap in current work on the use of formalization in interfaces. There has been much good work done in this area in recent years and particularly appealing is the work described in, for example, Harrison and Thimbleby (1990).

However, though most parts of a system might be formally described, the display, i.e. what the screen shows, never seems to be (apart from at the level of implementation, but we have already said that is not what we want when doing specifications). A good example of this is the very well-presented paper (Harrison and Dix, 1990) where the set 'D' is used within discussion of formal specifications for interactions, where D is the set of displays. Algebraic properties of D are discussed in various parts of the paper, but the elements of D themselves are never discussed, though we are told that a display, i.e. an element of D, is "a visual representation of some or all of the state...and might be, for example, an array of pixels (the details are not important)". This is very definitely relegating displays to a lower-level, and we have no hope of reasoning about them.

Another paper in the same collection (Alexander, 1990) goes some way towards overcoming the problem by allowing a designer to see how the interaction of a dialogue causes changes in a display, but it still leaves open the problem that what the display looks like cannot be reasoned about within the specification. A point from Lieberman, that it is difficult for designers to visualize a dialogue from a static description, is used in Alexander (1990); we would paraphrase this by saying that it is also difficult for a designer to visualize a display from a low-level description.

All this work is really focussed on formally describing dialogues, not with describing what each stage of a dialogue looks like, which must surely be important for the designer.

Such a gap in otherwise useful work is something that we hope to suggest a plug for in the rest of this paper. We aim to show one way that might help to bridge the current gap between the textual nature of designs and formal specifications and the graphical nature of the look of a system. The way we suggest looks, at first, very surprising. We consider a piece of software that has been developed over several years with the aim (which it achieves very well—see Goldson, Reeves and Bornat (1993) for a discussion) of supporting the teaching of formal logic to undergraduates (and senior school and college students).

This might seem to be far removed from the problem being addressed so in order to be clear this software will be illustrated in the next section.

Before we go on to that we need to explore the other reason for proposing this system, which is that it can be seen as a more computer science-oriented version of Tarski's World.

Rather than, as we shall see in the next section, using an application-neutral world of geometrically shaped blocks (cubes, dodecahedra and tetrahedra), we propose that computer science students would be more motivated by working in a world of graphical user-interface components. They would build pictures of graphical user-interfaces and then construct first-order logic descriptions of those pictures. So, rather than describing and reasoning about the blocks they would deal with buttons, menu items and windows, for example.

Tarski's World has proved to be very successful as a vehicle for teaching logic to first-year computer science undergraduate students—we would hope that making the world more relevant to computer science, and so providing more motivation, would mean that this tool was even more successful as a teaching tool for first-order logic.

Having looked at Tarski's World, as it stands, in the next section, in the third we will look at how some of the ideas behind the software described in section two can help with our problem of providing formal, textual descriptions of the look of systems.

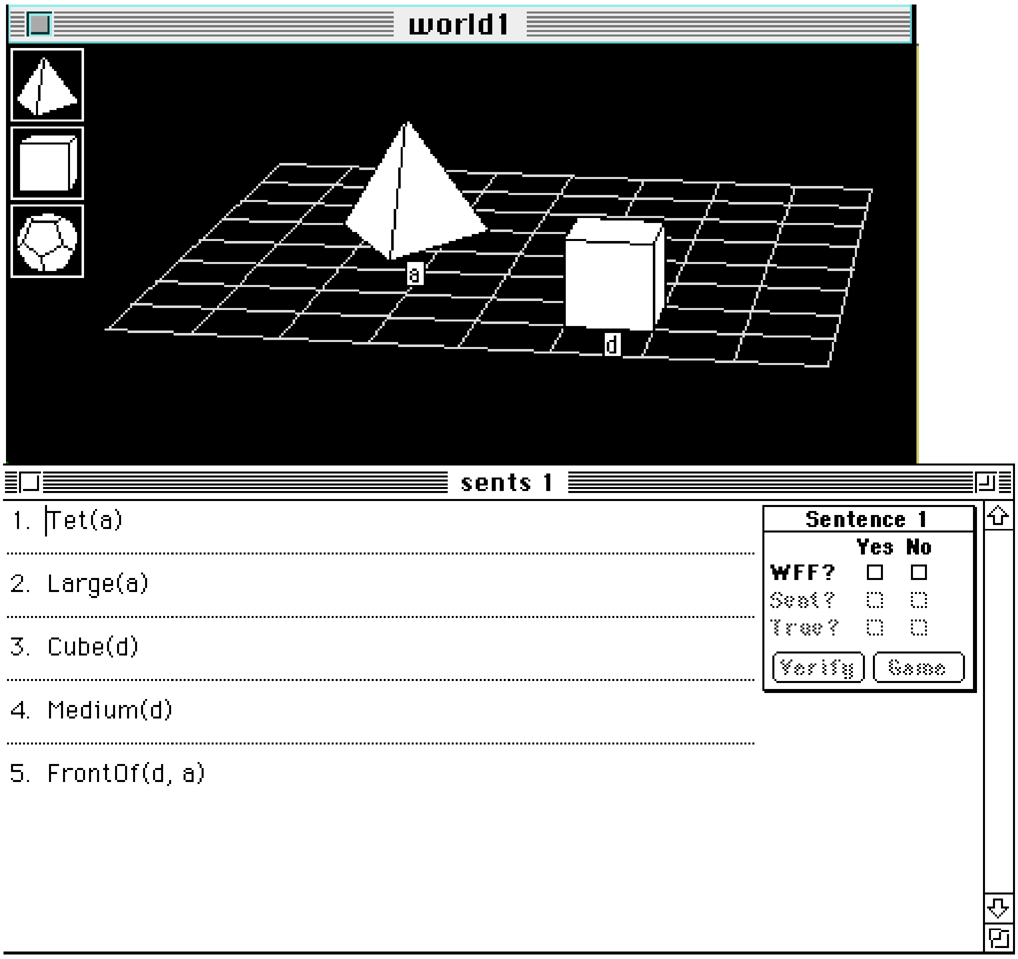

Figure 2.1

2. Tarski's World

Tarski's World, described in Barwise and Etchemendy (1991), was developed to support the teaching of (classical, first-order) logic. Some descriptions of other systems with the same aims, as well as Tarski's World, are given in Goldson and Reeves (1994); suffice to say that Tarski's World was one of the best. The author has had the pleasant experience of using it for teaching first-year undergraduates in computer science for a number of years. It is a robust, well-designed system and achieves its aims very well.

One of the points of this paper is to suggest an improvement that would make the program more relevant to computer science while, of course, retaining all of the other features which make it so successful.

As far as understanding Tarski's World and the rest of this paper is concerned, the important point to note is that the objects and relationships that exist in a certain picture, called a situation, are described by a set of sentences, called the description. The situation gives a meaning to each of the sentences in the description, so another way of thinking of the relationship between the situation and the description is to consider the situation as giving an interpretation of the sentences in the description.

Consider the situation, 'world1', and associated description, 'sents 1', in figure 2.1. Here each of the sentences in the description is true in the given situation. So, the description correctly describes the situation. At a certain level of precision we have reduced the graphical information in the situation to the formal, textual information in the description.

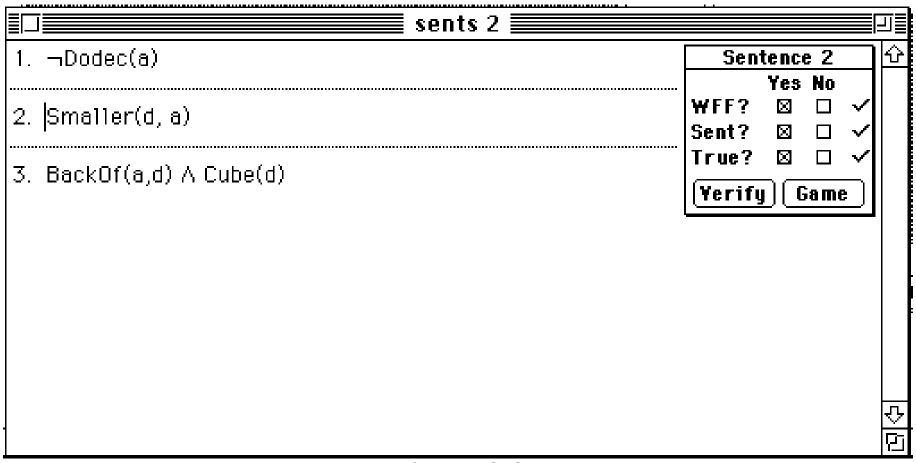

Figure 2.2

The phrase ‘at a certain level’ must occur in the previous sentence since other, different, descriptions may be given for the same situation; figure 2.2 gives an example where all the sentences are true in the situation in figure 2.1 too (note that the symbol '∧' means 'and'). So, there can be more than one description for some situations. However, descriptions do enjoy the property of being consistent: that is, all the sentences in any set of descriptions of some situation will all be true in that situation. No two descriptions of a situation can contradict one another and a bigger, more complete, description can always be made by collecting together all the sentences in a set of descriptions of some situation. Also, we often find that one description A is stronger than another B, which means that description A contains all the sentences that description B does, plus some more. The general case is where description A entails description B, which means that the sentences in description B follow logically from those in A, given some suitable logical definitions of the relations.

As a final part of this introduction we give a typical question that might be set as part of a laboratory exercise on a course in logic. The problem is to build a single world with as few blocks as possible in which each of the ten sentences given in the file Ockham’s Sentences is true.

The sentences and a solution are given in figures 2.3 and 2.4.

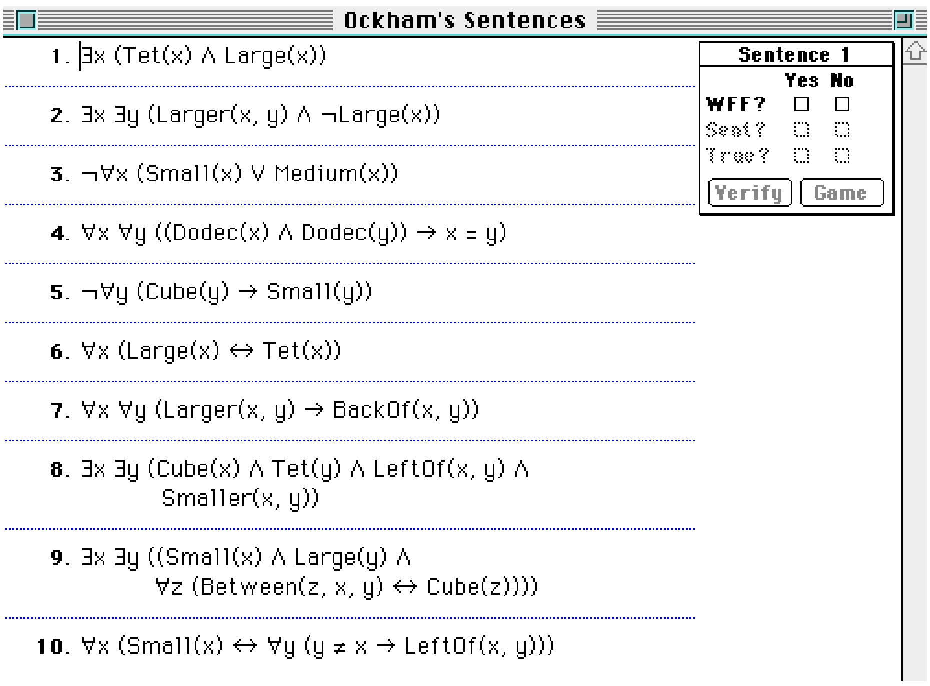

Figure 2.3

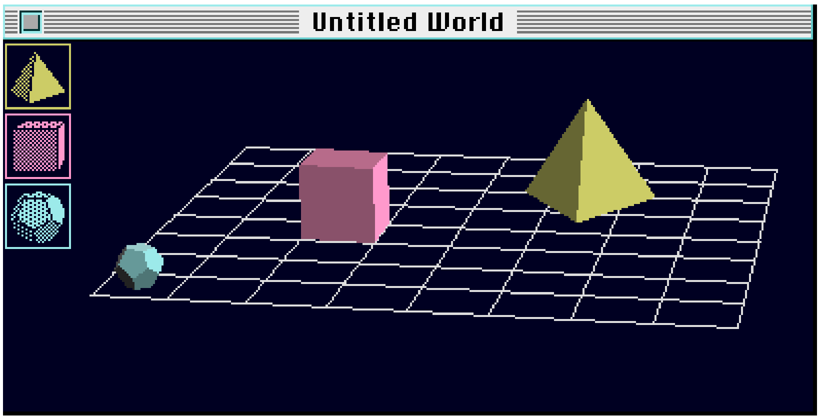

Figure 2.4

3. Modelling a graphical user-interface

The main idea taken from Tarski's World is that of describing, textually, a situation that is given graphically and doing this within a system that checks that what you have said is true and, if it is not, can guide you (by playing 'the game', whose rules encode the meaning of the quantifiers and connectives and which guides the user through the syntactic structure of a sentence until, at the atomic level, it is obvious why the sentence is not true in the given situation) to an explanation of why that sentence is not true.

The proposed system will allow a designer to experiment with the look of a display and then go on to develop a formal description of it. The system can be used to check at each stage that the description really is describing the display constructed by allowing the designer to check that all of the description's sentences are true.

In the end, the designer can be certain that the description of the display, in first-order logic, that they write is correct.

Allowing experimentation with the look and description of a display are an important part of this use of such a system - the fact that such an open-ended, relatively unconstrained exploration can result in a formal description (to some level of precision) is the main goal of this work.

Later in the process of building the interface in software, the existence of this situation and description will allow the software engineer to strengthen the specification if necessary. This might arise if the software engineer is not able to prove that a certain piece of code works correctly from their current specifications, but can for stronger ones, and they can go back to the situation and description and use the system to show that adding sentences to the description, to get a stronger one, still results in all the sentences being true, so that the stronger description can be used to further the specification process.

It also allows us to go the other way: given a textual description, within the system you can build up, checking correctness all along, a graphical representation of the sentences. This would be an approach used when, for example, animating a specification for a user or client who need not understand the formal language used.

This is also the point at which it becomes clear that the system could be used to support the teaching of logic. The system could be used just as Tarski's World is, except that the world contains not just simple solids but graphical user-interface elements.

Instead of the objects present in Tarski's World we would have windows, menus and buttons of various sizes and with various attributes. Their relative positions would be modelled just as in Tarski's World, as would, say, the text written on them, the procedures or methods that were connected to etc.

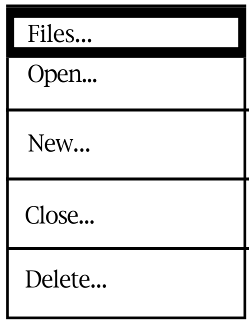

Figure 3.1

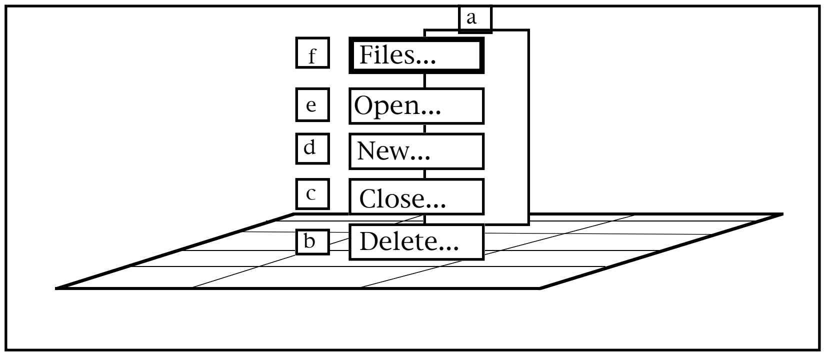

Figure 3.2

```
1. FrontOf(b,a) ∧ Text(b, "Delete...")

2. Above(c,b) ∧ Text(c, "Close...")

3. Above(d,c) ∧ Text(d, "New...")

4. Above(e,d) ∧ Text(e, "Open...")

5. Above(f,e) ∧ Text(f, "Files...") ∧ Dark(f)
```

Figure 3.3

Above is given an example of how such a description building enterprise might look. Figure 3.1 is the menu that we want to specify. In an analogy with Tarski's World, we can imagine the menu expressed in its component parts as in figure 3.2. By viewing the situation in 3.2 from the front we see the menu as required in 3.1.

We can then go on to build a description of the situation in 3.2, as given in figure 3.3. Notice that, as we want with specifications, the description is an abstract form of the menu in the sense that some things are left out; we do not need to give all of the detail (like exact positions) of the menu in order to usefully specify it. Later, of course, during the refinement process (the process that takes us from a specification to an implementation) a programmer will have to be specific about such things. However, the point of specifying is that we can leave out any unnecessary detail, i.e. perform abstraction, in order see clearly what is being asked for. In particular, we do not have to say how the menu is drawn, just what is drawn.

At each stage of building the description the system can be used, just as in Tarski's World, to make sure that the sentences added to the description are all true in the situation being described. If it ever turns out that this is not the case, then the game can be played to find out why.

In this way, the user can build-up as strong a description as they like for passing on, as a formal specification, to the software engineer. Later, the software engineer can strengthen the description, if necessary, since they will still have the situation being modelled. They can also take a description and, by building a situation which makes all the sentences in the description true, show a client what the specified graphical user-interface will look like.

As a second example, given the sentences (with approximate translations) in figure 3.4, considered as a specification, we might design a dialogue box that meets them as shown in figure 3.5. This would be the usual way, following the model of the Tarski's World exercises, that the tool would be used for teaching.

If it were being used to support formalization of a display then the opposite would happen. The designer would, in this example, construct a picture of the dialogue box and then start to write sentences, checking their truth as they went, and gradually build-up a correct description at whatever level of abstraction they thought appropriate for the problem in hand.

In the simple examples here we have confined ourselves to simple relations between objects in a situation (FrontOf, Text); clearly we will wish to have available other relations, such as those that for a button, say, describing not just its physical attributes but the way it is linked with the program being built.

What we have here, then, is a way of formally and textually, describing the look of a display at certain points during the dialogue, or other interaction, with a user. This is done at the same level of abstraction as we would usually want to work when specifying systems. The fact that it is textual allows us to use the sorts of methods discussed in section one. The fact that it is formal allows us to reason (either within or outside the sorts of systems mentioned in section one) about the look of the display. The display is now, formally, a first-class citizen.

$\exists x \exists y(\mathtt{Button}(x) \land \mathtt{Button}(y) \land x \neq y)$	There are atleast two buttons
$\exists x(\mathtt{Text}(x,\text{"Input the name of the file"}))$	There is a text box containing "Input the name of the file"
$\exists x \mathtt{EditBox}(x)$	There is an edit box
$\forall x \forall y((\mathtt{EditBox}(x) \land \exists z \mathtt{Text}(y,z)) \rightarrow \mathtt{Above}(y,x))$	The text is above the edit box
$\forall x \forall y((\mathtt{Button}(x) \land \neg \mathtt{Button}(y)) \rightarrow \mathtt{Above}(y,x))$	The buttons are at the bottom
$\exists x(\mathtt{Button}(x) \land \mathtt{Text}(x,\text{"OK"}))$	There is a button with "OK" on it
$\exists x(\mathtt{Button}(x) \land \mathtt{Text}(x,\text{"Cancel"}))$	There is a button with "Cancel" on it

Figure 3.4

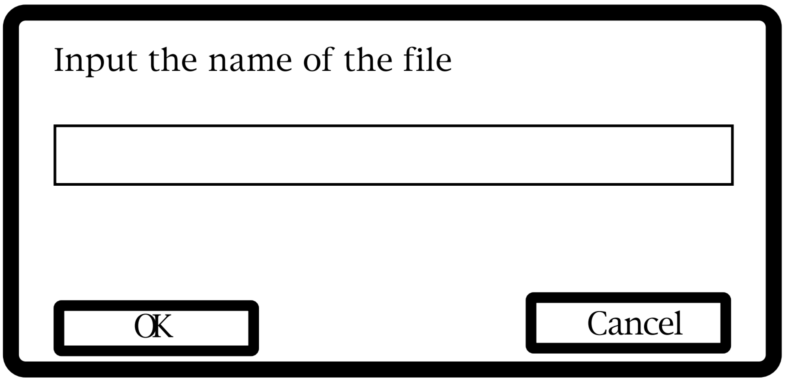

Figure 3.5

4. Conclusions

We have proposed a way of helping a designer build-up a formal, textual description of a display.

The argument that this really is a help is based on the fact that the design for the system has grown out of experience with Tarski's World, which has been designed and used for teaching first-order logic to students—a job which it does exceptionally well. All of its qualities will be expressed in the proposed system too.

We have also suggested that it can be used as a support for teaching logic which is more relevant to computer science because its 'subject matter' will be graphical user-interface components.

The next step is to implement a prototype of a system which performs this function in much the same way as Tarski's World does for designers of blocks worlds. This would allow us to build and experiment with, in a natural way, formal descriptions of displays for graphical user-interfaces.

Following our experience with tool building previously, we will be using the MacProlog32[1] environment on a Macintosh to construct the tool described in this paper. It should be noted that this environment already has a facility which allows the user to construct a display and then automatically produces the code that represents that display. So, with the addition of the tool described in this paper we will be able to generate both the low-level, code description of a display and also (utilizing the designer's intelligence) the high-level, abstract description too.

Finally, in order to check the truth of a sentence in some situation the tool will, of course, have to be able to do some theorem-proving. More precisely, the tool will gradually construct (as elements are added to the display) an atomic description (a description using just atomic sentences) of the display, i.e. a theory, T. Then, for each sentence s in the designer's description the tool will have to show that $T \models s$, i.e. that s is true in the theory T. The technology for the underlying theorem-prover is not too important—due to our previous experience we have chosen to use a semantic tableau-based method (Reeves and Clarke, 1990).

A final remark: because of the way in which many curricula are designed, which in itself is a reflection of the way in which the designers were taught, the 'specialisms' of user-interface design on the one hand and formal methods on the other do not often both feature amongst the skills of our students (or teachers). The tool described in this paper, we believe, will help to bridge this gap.

To have user-interface designers who appreciate the usefulness, precision and conciseness of formal descriptions and to have formal software engineers who have an appreciation of the requirements of users in their interactions with computers is surely a goal of those of us who educate tomorrow's computer scientists.

[1]MacProlog32 is a trademark of Logic Programming Associates, Studio 4, Royal Victoria Patriotic Building, Trinity Road, LONDON, SW18 3SX, England.

References

Alexander, H. *Structuring Dialogues using CSP*. In Formal Methods in Human-Computer Interaction, edited by M. Harrison and H. Thimbleby, Cambridge University Press, 1990.

Barwise, J. and Etchemendy, J. *The Language of First-Order Logic*. Center for the Study of Language and Information, 1991.

Diller, A. *Z An Introduction to Formal Methods*. Wiley. 2nd. edition, 1994.

Dix, A., Finlay, J., Abowd, G. and Beale, R. *Human-Computer Interaction*. Prentice Hall, 1993.

Dromey, G. *Program Derivation The Development of Programs from Specifications*. Addison-Wesley, 1989.

Goldson, D., Reeves, S. and Bornat, R. *A Review of several systems for the Support of Logics*. The Computer Journal, volume 36, number 4, 1993.

Goldson, D. and Reeves, S. *Review of "The Language of First-order Logic" by Barwise and Etchemendy*. The Philosophical Quarterly, Blackwells, Oxford, April 1994.

Gries, D. *The Science of Programming*. Springer-Verlag, 1981.

Harrison, M. and Thimbleby H. (editors). *Formal Methods in Human-Computer Interaction*. Cambridge University Press, 1990.

Harrison, M., Dix A. *A State Model of Direct manipulation in Interactive Systems*. In Formal Methods in Human-Computer Interaction, edited by M. Harrison and H. Thimbleby, Cambridge University Press, 1990.

Reeves, S. and Clarke, M. *Logic for Computer Science*, Addison-Wesley, 1990.