Names and Reference in User Interfaces

Harold Thimbleby
Swansea University
Wales
SA2 8PP
United Kingdom
+44 1792295393
harold@thimbleby.net

Michael Harrison
Informatics Research Institute,
Newcastle University
NE1 7RU,
United Kingdom
+44 191 246 4938
michael.harrison@ncl.ac.uk

ABSTRACT
This short paper argues that references in user interfaces, in particular names and the values they denote, are often designed in a way that is incomplete and inconsistent thereby causing problems for users. This paper explores names and values through illustrations in order to clear the way for a more systematic approach to the design of names and reference.

1. INTRODUCTION
People use computers to achieve things with greater ease, effectiveness, reliability, or enjoyment that they could not do, or could not do so well, without them. *Naming*, in particular, allows an activity, a specific set of features or a routine task to be exploited or invoked repeatedly with no more effort than it takes to use its name once. Names can also refer to ideas and objects that are not present in the "here-and-now"; they facilitate remembering, planning, explaining, and communicating. Names are frequently used for distinguishing objects that are otherwise indistinguishable. In practice, names are of course ubiquitous in computing: objects like computer servers are given names so that they can be distinguished by people and by internet name servers. Clarity, memorability, and consistency are key principles that apply to naming schemes.

An important aspect of interactive systems is how reference and naming is designed. Typically interactive systems are complex and are used in many ways, ways unforeseen from the early stages of design. This paper will argue that problems often occur in the use of interactive systems because of a lack of clarity about the mechanisms for naming and referencing.

Confusions arise for a variety of reasons. Some of these reasons have been explored relatively thoroughly: for example the issues associated with mode confusion in interactive systems. Others are less well understood in the context of interactive systems, even though they have been studied relatively thoroughly in other contexts. Naming issues become more important as mobile devices become more available as a platform for applications. This diversity leads to a richer set of mechanisms for referring to items and the requirement for consistency across a range of different interfaces for the user to the same device in different locations, different devices in the same location, and so on.

Published by the British Computer Society
Volume 2 Proceedings of the 21st BCS HCI Group Conference
HCI 2007, 3-7 September 2007, Lancaster University, UK
Devina Ramduny-Ellis & Dorothy Rachovides (Editors)

This paper will discuss how naming in particular and reference in general are being used in a number of designs and will reflect upon the problems that these create. The purpose of the paper is to explore what analysis is most appropriate for the design. In general it is important is to have theories or frameworks that can raise key design issues *before* systems are built, and, moreover, that raise issues that can be addressed analytically and systematically. Analytic insight into design is particularly important for safety- and mission-critical systems, where certain sorts of use experience may be too rare or too costly to evaluate by conventional UCD techniques.

Names and their meaning were explored in programming language design forty years ago [3,5,6], and concepts such as binding, assignment, environments, scope, encapsulation, and so forth are established and remain stable. While programming problems related to names (for example problems with aliasing) are well understood, similar problems in user interface design have attracted little attention and continue to be dealt with on an ad hoc basis. Most research activity has been in relation to naming schemes (for example [1, 2]), particularly concerned with psychological issues. This paper, in contrast, argues that there are also engineering issues relating to the structure and consistency of naming systems in interfaces that have an important impact on the usability of a particular design.

Names *bind* to objects and then refer to those objects. Hence, for example, "Thinkbridge" is a name bound to an object that happens to be a laptop computer. This is not the only name that is bound to this particular object. The computer is also bound with an IP number, a MAC number, plus other names that might be used by the software installed in the system. These different names will be used in different task contexts for the same object (the laptop); some names are known by the user, some are hidden. Furthermore, in the case of "Thinkbridge," the name binds to a computer with its *own* namespaces, which themselves are complex, partly hierarchical naming structures.

In user interfaces it is possible to refer to objects by name, as in the case of a programming language, but alternatively the interface may allow pointing at objects as a means of reference, or a combination of naming and pointing as in the case of "Put that there" [1]. Hence names like "that" and "there" are generic names that are made specific references when combined with other mechanisms (here, pointing) for reference. In user interfaces names can be organised hierarchically and systems can be moded as a result. Consider for example, an example from unix where invoking the command `cd papers/naming` followed by `emacs hci07.tex` refers to a file name that is part of a hierarchical naming structure. The effect of changing the directory using "`cd`" was to change the context or mode in which the file name is used.

The systematic analysis of reference and naming in HCI is a non-trivial, and probably long-term endeavor. This paper

aims to start the process by providing a number of illustrations to indicate the range of problems. Users can be confused by a number of aspects of bindings between reference and object. They can be confused because the extent and nature of the binding is not clear. They can be confused because actions that appear to be intuitive cannot be performed in the way that one would expect using the reference.

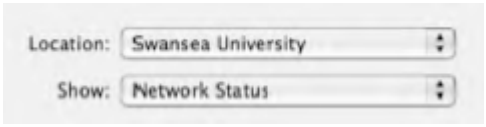
We describe four types of issue that are problematic in the design of interfaces. We do not intend to be complete, rather to indicate the kind of framework that would be of value in design. These issues are concerned with:

1. The binding between a name (or other reference) and its denotation
2. What the interface supports in terms of that binding, and how it is understood by the user.
3. Transparency of reference, and the extent to which denoted objects can be replaced by references.
4. Mode transparency and confusion.

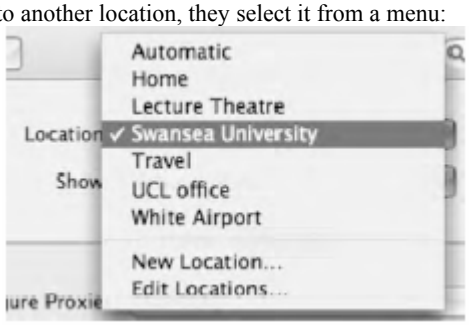
Note that the terms "binding" and "transparency" are standard [5].

2. REFERENCE AND DENOTATION

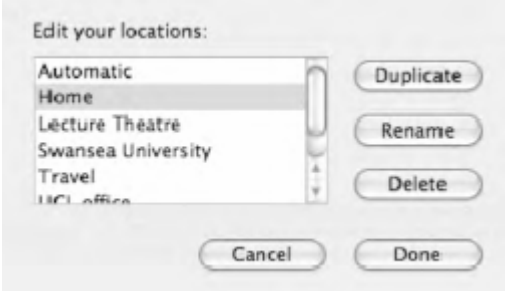
Mac OS X provides facilities for dealing with the mobility of a device. OS X allows the user to name the settings required to access the internet from a variety of locations the computer's owner might occupy. A pull-down menu selects a location as current. It is shown below selecting "Swansea University," which denotes a preset location and information for a particular office.



In other words, "Swansea University" is a name that *in this scope* is bound to various internet parameters. If the user moves to another location, they select it from a menu:



The system provides a menu of different locations so that the user can choose by name what settings are relevant to the current location. However, a setting may need changing, and this provides a simple illustration of a naming confusion.



Selecting "Edit Locations…" from this menu means editing the *names* of locations not editing the values of the names, that is the internet *settings* of locations, as might be intuitively expected by the user. Thus a user can duplicate an existing name; if "Duplicate" is selected, the user can obtain a new location name, "Home Copy," for instance, with the *same* settings as "Home" currently has. If the user wants to change "Home Copy" to some other name, such as "Holiday" then they now need to rename this location. Typically, the user would then change some or all of the internet settings of "Holiday"; the advantage being that "Holiday" has been initialized to have the same settings as "Home" has, and therefore this process is easier than entering all the holiday settings by hand.

The name "Swansea University" can be changed to be, say, "My office," but the IP number associated with it *cannot* be changed here. This interpretation of "Edit Locations" cannot be achieved at this level by deleting the existing location and then adding a new location with the intended name; IP numbers (the values of these names) are set or changed at a different level in the menu hierarchy.

As it happens, Mac OS provides an alternative view of these bindings, which enables a sophisticated user to shortcut this user interface. The network names are stored in an XML file called `preferences.plist`, which can be edited using the Property List Editor, so, in this case, the name `UserDefinedName` would be associated with `Home`:

```
<key>UserDefinedName</key>
<string>Home</string>
```

The file also defines what settings are associated with that name, and these settings can be edited directly. Therefore there is a means by which the intuitive meaning of "Edit locations" can be achieved, but it isn't a meaning supported by the normal interactive user interface. The point of mentioning XML in this paper is to emphasise that, technically, the user interface could have been different: the XML can represent "any" changes to names, bindings and values, so any constraints are purely user interface design choices.

Given the tasks that are being performed using these locations, it would seem natural to allow a user to refer to the names in other contexts, say to send an email to a technician to get help, or to email it to someone else who wants to edit the location to their own requirements. Unfortunately, location names only have meaning in the very restricted context of the menus described above. Emailing "Home" to anyone else sends nothing other than the word; the binding is lost (and it doesn't even surprise us that this is so). Indeed, the binding is lost even if the user cuts-and-pastes from any of the location dialog boxes. Worse, it is actually very tedious to determine what the name Home is bound to; its value (the IP numbers and so on) are spread over many windows and dialog boxes.

Eudora (a popular email client) illustrates another issue in relation to the binding between names and objects. For Eudora, `<Dominant>` is the name of the unique, default email account. Whereas all names can be edited — for instance, if a user wants to change the spelling of a name — the special name `<Dominant>` is fixed and cannot be edited. If a user changes their lifestyle and wants to change which account is dominant, they are *unable* to modify the dominant account, rename it to another account name, or rename an existing name as dominant. An obvious solution to this problem is that the dominant property should not be a property of the name spelling, but one of its values. Certainly the choice of name itself should not affect whether the user can change it. For example, each account name could have a check box "Dominant?" that the user can set to make it the dominant account. (Obviously, the program would ensure exactly one name had the associated dominant property.) In other words, dominance should not be a property of the name but of the denotation thereby making it possible to have a more uniform naming scheme.

3. NAMING THE RIGHT OBJECTS IN DESIGN

Our next example relates to the fact that interfaces may have concepts within them that are understood in the user's model, but which that may not be capable of being referenced directly. Issues of naming are often resolved using task analysis techniques (see [2] for a careful summary). Consider the following example. The Casio HS-8V is a basic calculator with a single memory. The calculator provides the keys MRC, M– and M+ to handle its memory. Formally, these are names denoting operations that have an effect on the memory. However this set of operations is not complete in terms of the user's mental model of how the calculator works. For example the user is quite likely to want to use the memory explicitly (why else are there buttons for it?) as a location in which a value is saved for future use. This cannot be done easily. If by chance, the memory is already zero, the user can press M+ to add the number to be saved to memory. If the memory is not zero, however, there is no obvious way to save any number.

The labeling of functions limits the means of use of the calculator and should therefore reflect typical mental models of the device, thereby making it easier to carry out actions that would seem to be intuitively obvious.

Again we see the use of names in a user interface that seem to be routine, but which conceal rather obscure issues — and not just technically obscure issues, but issues that affect users and the tasks they are able to achieve.

4. REFERENTIAL TRANSPARENCY

A number of issues in the design of interactive systems are associated with referential transparency and other issues that relate more specifically to mode confusion.

Key buttons are names that denote calculator functions. Consider calculating $10^{-\pi}$ (which should be about 0.00072) on the Sharp EL-531VHB calculator. Keying 10^x –π ± produces "Error" while 10^x 2 ± produces 0.01. So the names π and 2 behave differently.

Storing π in memory A, by pressing π STO A should make it possible to use A for the value of π: indeed, pressing π or RCL A both produce the same results. Yet 10^x RCL A± produces 0.00072, even though 10^x π ± is an error.

In these examples the key named π (which denotes 3.14159…) is not treated the same way as the key labeled 2 or A. There is no referential transparency. Different sorts of names, different sorts of numbers (i.e., Arabic names of numeric values) are apparently bound to very different things.

Consider now the following further example of a different lack of referential transparency. If a user enters just =, the last expression is re-evaluated; if a user enters a binary operator, ANS (a name representing the value of the last calculation) is automatically inserted as its left operand. These interface accelerations increase the power of the calculator: consider, say, the expression ANS+1 (or equivalently, +1, which gets the ANS name inserted automatically) which turns the calculator into a counter where every press of = calculates ANS := ANS+1. Unfortunately the abbreviation mechanism creates a feature interaction with names. If the user enters ANS+RCL A =, they as anticipated get the last answer added to the value of A. If they enter RCL A+ANS = different things happen. First, RCL A is treated as a query to find the value of A: immediately the calculator shows A's value. When the user presses the +, the calculator inserts ANS as its left operand, which in fact will be equal to the value just displayed, namely A. But when the user explicitly enters ANS, *that* ANS will be the last value displayed which is of course now A, rather than the intended answer.

The rules by which ANS works is as follows:

1. A missing operand defaults to the last answer. This makes writing + 2 a shorthand for ANS + 2.
2. Asking the value of a name straight after using = *immediately* gets the value. So RCL A immediately gets A's value (and hence changes ANS), saving the user writing RCL A = which would be the obvious way of finding A's value.

Each of these features makes some sense in isolation, but they interact with each other and names with the unfortunate consequence that they compromise the calculator's mathematics — spelt out more abstractly, the example is equivalent to the surprising $a+b \neq b+a$. The meanings of names such as ANS and A depend on exactly when they are used.

5. MODE CONFUSION

The final issue we explore in this paper is associated with modes, mode confusion and mode transition. Issues of mode confusion have been extensively studied (see [8] for review). The Canon EOS350D digital SLR camera has a mode selection dial so that the photographer can select how much control they have over photography or how much the camera performs automatically. A variety of parameters, such as aperture, shutter speed, exposure measurement, choice of object to focus on, whether to use flash, set the ISO speed, white balance, and others can either be set by the camera automatically or, depending on mode, different subsets of parameters can be specified by the photographer (the camera's computer sorts out unspecified parameters depending on prevailing conditions). By design, there are seven basic modes where the photographer chooses a type of photograph — such as portrait, scenery, macro or sport — and the camera selects all parameters fully automatically. There are also five so-called "creative" modes where the photographer overrides parameters. For example, in the creative mode named Av, the photographer can control the aperture, leaving the camera to automatically adjust the shutter speed to maintain the same exposure. In all creative modes, the photographer can also control the flash, ISO speed, focus sensors, exposure sensors, and the white balance.

The icons in the figure above are names that refer to the sets of photographic parameters, as well as the functions that are performed on them. The symbols therefore denote camera functions, in much the same way as the M+ button in the calculator example represented a calculator function. This time, however, the automatic processes of the camera use environmental conditions to fill in the remaining parameters. This makes sense, as often the user is more interested in changing or adjusting a value rather than specifying one outright; that is, instead of setting the aperture to f/5.6 (say) the user may prefer to set it to be so-many stops larger or smaller than whatever the camera suggests.

Suppose a photographer uses the basic portrait mode to photograph somebody's face. The camera will select all appropriate photographic settings. Now suppose the photographer wishes to do a portrait but wants to control the aperture; they must change to Av creative mode. Unfortunately, now the camera will use all the previously user-defined settings — such as ISO speed and white balance — and none of the automatically determined values that were being used a moment earlier in the portrait mode. Worse, the many settings are distributed around the menu hierarchy of the camera, and are not easy for a user to locate.

Hence in the mode transition, none of the parameters that were calculated automatically in the previous mode are carried over. The user is required to fill all the parameters explicitly, which are tedious to check or change. The more usual issues of mode confusion that arise because a user is unaware that controls mean something new because they have not observed the transition are not of concern here. Instead a new problem of mode is indicated: in the process of transition the previously-named state, and therefore potentially time-saving information, is lost. The underlying name/reference problem is similar to the internet setting problem we reviewed earlier: the name (an internet location; a mode of photography) is bound to settings which are inaccessible to the user.

An obvious solution would be something like the following. The mode selection knob can be pressed and, in so doing, the complete set of settings is saved. The creative modes then adjust with respect to this saved setting. Hence the user can opt to save the information in the transition. In conventional terms, pressing the knob performs an assignment from the current setting to the default setting, to be used in the creative modes. Of course, assignment is well-known in computing; one wonders why the interaction designer did not support this technically trivial solution.

Another solution is to separate modes from what can be changed. For example, as currently designed it is physically impossible to be both in portrait mode and to control the flash manually, because these choices are in different locations on the same knob. Alternatively, knobs may refer to function rather than a mode. The knobs themselves may compute automatically or be user-defined, and this choice could be selected by the photographer. There are many possibilities.

6. CONCLUSIONS AND AGENDA FOR FUTURE WORK

Reference and in particular names raise non-trivial user interface design issues. Designers need better support in understanding the naming implications of their design.

(1) A framework is required for describing reference mechanisms (including names and naming structures) so that designers can use it to consider design options and, in particular, a framework would enable appropriate schemes for linking an application to appropriate platforms.

(2) Principles for reference are required that would enable user interface consistency and ease of use. There are many in programming language design [5], but they have not been carried over and validated in the interactive case.

(3) The principles that are adopted should be clear and easy to explain or interpret to users.

(4) The semantics of names and reference is well-understood in programming, and this can be a creative (and consistent) source of user interface features.

In conventional programming, the concepts alias, binding, environment, scope, inheritance, extent and so on are well defined, and their combined use and interplay has been worked out thoroughly. The same concepts are not applied uniformly in user interfaces, and (at least as implemented) they interact in complex and non-intuitive ways. Studying these issues and knowing that corresponding abstract operations are possible in user interfaces will encourage interactive system implementers to make more consistent, more powerful, and more reliable systems — and ones where standard user interface design principles, such as undo and help, would be implemented correctly, consistently and generally. Thus a more thorough understanding of references, names and binding, much has already been developed and used for many years in relation to programming languages, will clarify many interaction design issues in user interfaces. An understanding would provide a clear and well-defined way to discuss user errors and confusions in relation to many user interface problems. However, user interfaces introduce many ideas that go beyond conventional programming languages, and this will be a substantial research project.

A further research project would be to determine appropriate models of reference, naming and scope specifically relevant to interactive usability that would be valid from the perspective of usability; this would make it possible for designers to reason about and conceptualize their user interface designs. With such a framework, for instance, it would become possible to redesign a camera and analyze experimentally the effect that it has had on the user's model of the system and their understanding and use of the design.

7. ACKNOWLEDGEMENTS

Tim Bell, Paul Cairns, Peter Mosses, and Will Thimbleby made many helpful suggestions.

8. REFERENCES

[1] Carroll, J. M., *What's in a Name? An Essay in the Psychology of Reference*, Freeman, 1985.

[2] Johnson, P., "Human Computer Interaction: psychology, task analysis and software engineering". McGraw Hill, 1992.

[3] Landin, P. J., "The Next 700 Programming Languages," *Communications of the ACM*, **9**(3):157–166, 1966.

[4] Nigay, L. & Coutaz, J., A design space for multimodal systems: concurrent processing and data fusion. Proceedings of the SIGCHI conference on Human Factors in Computing Systems. ACM Press pp. 172–178. 1993

[5] Strachey, C., "Fundamental Concepts in Programming Languages," *Higher-Order and Symbolic Computation*, **13**(1/2):11–49, 2000.

[6] Tennent, R. D., *Principles of Programming Languages*, Prentice-Hall, 1981.

[7] Gow, J., Thimbleby, H.W. & Cairns, P. "Automatic critiques of interface modes" In Gilroy, S. W. and Harrison, M.D. Interactive Systems: Design, specification and verification (DSVIS 2005). Springer Lecture Notes in Computer Science. No. 3941. 2006. pp. 201-212