A Unified Concept of Style and Its Place in User Interface Design

Philip Gray and Stephen Draper

GIST (Glasgow Interactive Systems cenTre)
University of Glasgow
Glasgow G12 8QQ

Tel: *+44 141 339 8855*
Fax: *+44 141 330 4913*
EMail: *pdg@dcs.gla.ac.uk, steve@psy.gla.ac.uk*

The term 'style' is used with great regularity in user interface design literature, yet it appears to refer to widely disparate phenomena. We present a notion of style which unifies these various uses. We then demonstrate how the notion may form the basis of a representation of style that can provide design assistance.

Keywords: user interface styles, UIDEs, user interface design

1 Introduction

Many word processors and text editors offer their users the facility to specify and apply sets of text properties to separated blocks of text, thus avoiding the tedious repetitive setting of these properties line by line or paragraph by paragraph. Such groups of settings are typically called styles. For example, in the text style facility of Microsoft Word, it is possible to define a set of hierarchic heading styles, such that each lower-level style uses a smaller font than its parent heading. Such styles, once defined, may be re-used in other documents. Often, however, one would like to be able to change some feature, such as text size, globally. For example, heading styles for conference papers might be appropriate for a memo, but only if the styles are globally reduced in size. Unfortunately, this is not possible. A number of other text editors and formatting systems (Andrew, LaTex) offer the facility to define one font size relative to another, but it is rare to find the ability to define a heading style in this relative way. What is offered in the way of style definition and application varies widely from system to system and, in general, matches user requirements in an ad hoc way. There is an apparent absence of any common understanding of text style from which required and desirable features might be derived.

Corporate identity is often represented by a similarity of appearance of documents, products, images, etc. Typically, such similarity is based on style rules which are applied to corporate artifacts. Often these rules are complex and interrelated. This can cause difficulties for a designer who is trying to determine their applicability. This is especially the case for conditional rules which may only be relevant in certain circumstances. Consider a rule that states that, for WWW pages, the name of the institution must appear immediately below the institution logo unless the institution name appears in the page title. It is quite likely that a user may forget the conditionality of the rule, especially if the institution name appears in the title as a result of a late modification, after the rest of the page has been designed.

Sometimes we apply the term 'style' to system properties which are more abstract than either text properties or corporate logos. In Macintosh™ applications, the File and View menus appear in the same relative position in the menu bar, possess a subset of choices which are always present, and associate these choices with similar functions in each application. Sometimes this style of menu representation is violated, usually resulting in disorientation of experienced users. A trivial example is the puzzle application found on early Macintosh computers. The Clear option in the Edit menu for this puzzle causes the puzzle pieces to toggle between a pictorial and a numeric representation, which is not the conventional semantics of the clear option as found in most other applications. Consistency, in the sense of maintaining the command name semantics of the Macintosh menu style, would have prevented this problem.

Each of these three examples, widely different though they are, capture ordinary uses of the notion of style. In this paper we will argue that they all represent variations of a single concept, a concept which is central to the process of design. We will present a general theory of style which explicates the advantages of the use of style, such as in our examples above, and we will suggest ways in which a proper understanding of style may lead to better assistance to the designers and users of interactive systems.

2 Notions of Style

References to style are encountered with great regularity in the literature of user interface design, in textbooks, (Shneiderman 1992), user interface guidelines (Smith & Mosier 1986), and style guides (AT&T 1990). However, the phenomena referred to by the term appear to be widely disparate. Consider the following iconic representation of a file, taken from the Macintosh Finder:

The icon exhibits, or belongs to, a number of styles:

- a visual representation style
  This representation of a file is primarily pictorial rather than textual, and more specifically, is the Finder's icon as opposed to its small icon view.
- a graphical style – one particular type of representational relationship
  The appearance of a piece of paper with a turned-over corner is using pictographic similarity, rather than an arbitrary symbol (cf., the international radioactivity symbol), to suggest meaning.
- a direct manipulation style

That is, in its home context you can click on the icon to operate on the object it refers to.

- a metaphorical style
  The icon's depiction of a piece of paper is prompting you to think of the computer file in terms of the printed paper it can be used to produce, which is one, but only one, of the functions and properties of the file.
- a "house" style for Macintosh applications
  We use "house style" to refer to a set of standards enforced within an organisation for functional reasons rather than to promote recognisability. Apple's User Interface Guidelines (1986) are in part an attempt to export a house style.
- a "corporate" style for Microsoft
  Corporate styles are designed for recognisability rather than functionality.
- a style of file naming.
  A personal practice to make file names more meaningful to the individual.

In addition, there are some types of style not obviously exemplified by this icon. These include:

- Architectural style, e.g., the Gothic style, in which various and seemingly unrelated decorative features can be demonstrated to derive from a unified design-generating strategy or set of principles.
- Fashion styles in clothing, e.g., haute couture.
- Style as the visual resemblance created by expert typographers, allowing one to recognise similarity between otherwise quite different glyphs, e.g., 'o' and 't', as belonging to the same font family.

The most obvious common thread amongst all these varieties of style is the fact that the categories they form are used by designers, and indeed, by users. 'Style' is a central element in both the language and practice of user interface design. However, how is it used? Is there some common thread of meaning which ties these uses together? Or are the stylistic ascriptions to the icon above ambiguous, having meanings in each of its occurrences so different from one another that only the use of an informal English word 'style' binds them together? If, on the other hand again, there is such a common semantic core, is it something of value to designers?

We propose that style is any set of decisions or constraints fixed before a particular stage of the design process, e.g., dictated to a designer as part of her brief. Furthermore, a family of designs sharing the same set of constraints are likely to look similar to observers – styles are often recognisable. We argue that there is such a common property, that it is relevant, indeed central, to design, and that it may be used as the basis for a representation of style in computerised design support tools.

3 A Definition of Style

3.1 Style as Categories of Fixed Design Decisions

We suggest that style refers to the fixedness of a collection of design choices or decisions. This notion of fixedness, the degree to which the choice is available for change (or the degree to which the designer is committed to the decision), is related to a number of aspects of the design decisions. Thus, one decision may lead to, or limit, other choices which may be made, i.e., a logical ordering. Often, the choices are viewed in terms of the temporal ordering of the acts of decision-making[1].

---

[1] These two forms of ordering form the basis of two views of design reasoning: justificatory and historical (Maclean et al. 1991).

Design has both a search space and a sequence, or at least a partial time ordering, of decisions. Design methods and interactive system architectures both have features intended to enforce or allow for the separability of design and specification decisions, but there are actually two aspects of this. One is being able to take decisions independently in any order; the other is being able to take later ones with no backtracking i.e., no interaction or repercussion on ones already taken.

Traditional top down design methods in computing are only viable if in fact there is an order that allows the latter: taking decisions with no backtracking; and that this order is from top to bottom, from high level specifications to low-level code. This actually seems plausible because or when a) programming is seen as converting functional specs to code b) computers are Turing equivalent, so there will always be an implementation of any function or component function found in the specification or derived during design.

However in real life, whether in commercial programming or in undergraduate Pascal exercises, the actual specification (i.e., including the implicit specifications as well as the written ones) almost always includes non-functional requirements. Principal among these is a specified machine, language, and often support software (e.g., the X window system and its widget toolkits). This latter type of specification means that most design is not building out from a single fixed point at the top (the function to be computed) but from two fixed points — the function and the implementation platform (top and bottom). In general there may be any number of fixed points, as becomes clear in cases where re-use of code is emphasised and programming is explicitly a matter of glueing together old code for new uses, e.g., using Rexx.

Reflecting on this draws attention to the temporal dimension in design. Some decisions are taken early on, e.g., in the requirements, and may not then be changed (more accurately, the costs of changing them are hugely greater than for changing other decisions). A given designer, then, is usually operating with what for her are fixed points that have been decided earlier. Even within an individual's design activity, there may be a more or less full ordering of decisions: top down methodologies imply a full ordering, while successes at separation, e.g., in UIMS and CAL authoring, allow a substantial degree of parallelism or independence in decision making (e.g., in deciding separately on screen appearance and on underlying functionality).

3.2 Application to UI style

The essence of the idea proposed here is that this early fixing or pre-empting of design decisions has in practice many varied forms. The word 'style' may without straining it be applied to all of these. It is a general phenomenon: to decide in advance either particular decisions, or particular conditional decisions (e.g., if there is an error message then it will have the following format). The key feature to note is that this restriction is not necessarily concerned with the functionality (the function to be computed); it may be function-neutral (always use insertion sort, always use one font style in menus — where the item chosen performs a vital function but is only one of a number of equivalent solutions), or function irrelevant (put the same logo on the screen, the casing, the shipping boxes, and the documentation — where the item chosen in no way supports the overt, overall function).
Styles, then, are sets of predetermined design solutions. As a result designers are working within a reduced, simplified, constrained design space. They have fewer alternatives to choose from, although possibly this may make it harder to achieve their goals, e.g., optimal performance. Furthermore, the space is additionally structured in that the types of choices available once the style is fixed are (partially) ordered. Thus, having chosen to use menus, the

designer is forced to decide upon how the menu will appear, how items will be selected, how selection will be indicated; these choices are highlighted as important, indeed, necessary, within the context of the style. Other choices are relegated to the background, although they may be organised with respect to other style choices.

Designs produced with the same style often have a family resemblance – they "look" or "feel" the same to people. They are recognisable. They have something in common that other designs, from a wider design space, do not have. This recognisability decouples style identification from a knowledge of its development history or its design-oriented justification. Styles not only serve designers, but their recognisability can also be used by users; this identifiability gives users cues as to the meaning or expected behaviour of artefacts which exhibit a style they know.

A style apparently cannot be defined without tacit contrast set(s): a larger set of alternatives of which it is one, from which it is chosen. But how then can we recognise a style bottom up? Only because, or when, we have seen other solutions to the same larger design space. It is of course possible to be familiar with a style, but not be aware of it as a style until one sees a contrasting example. For instance, pie menus are called 'pie menus' while vertical menus are just 'menus' and tiled windows are called 'tiled windows' while overlapping windows are just 'windows'.

The term 'style' as used here is neutral with respect to the particular type of design space structuring which it provides. That is, it may refer to choices which flow from a coherent principle or principles, like metaphorical representation, or it may refer to choices which are simply related due to common practice or convention, like visual style for menu items.

3.3 Style and Design Rationale

Newman (1988) has argued that what is important in the analysis of a style is the design rationale which underlies it. That is, the important aspect of style is its role as a structuring of the design space around which to organise design reasoning. There are some cases in which all the properties of a style are justified by a common design rationale, e.g., the features of a corporate image are justifiable solely with respect to making the corporate identity recognisable. However, there are other styles for which this is not the case, e.g., the style of UNIX command naming, which has arisen through practice rather than as the result of a conscious and well-defined design process.

Furthermore, concentrating on the relationship between design rationale and style obscures the other ways in which the concept of style influences design. Menus are devices for making choices from a set of items, just those where the choices are presented to user at the time the choice is to be made. Central questions then become: (i) how the choices are presented, and (ii) how the choice is made. Peripheral, though still important, are decisions about the length of time to display the choices, how to indicate selection, how to handle nested structures of choices. Equally, we may identify the "near neighbours" of menus: command languages, data-entry forms and scrollable lists but not video sequences or icon images. Scroll bars and editable text fields perhaps lie somewhere between forms and images in their relation to menus. The important point here is that the concept of a menu adds a structure onto the features of a design, a structure which partially orders those features with respect to potential design decisions. Such structuring, which may be captured via style definitions, need not include the reasons for the decisions. Although important for software engineering reasons, the justification is separate from, or rather exists on top of, the structures offered by styles.

The grouping of choices as a style coexists with, but is distinct from, design reasoning. The work of justifying the set of choices may be carried out independently of the rest of the design and indeed, prior to the particular design enterprise. Style guides and uses of style in guidelines capture this pre-justification by giving conditions in which the selection of a style is appropriate; the justification is often absent and assumed (or its existence flagged by references to relevant studies). In fact, the absence of facilities to incorporate styles by grouping named, and perhaps parameterised, design rationales for selected sets of choices is a notable failing of current work on design rationale notations.

We thus reject Newman's assertion. Design rationales relate to one important part of design reasoning, but "styles" refer to a distinct and broader class of structures that are important in design independently of whether their justification is recorded and used.

4 Current Tools and the Design Space

4.1 Implications for UIDEs

Support tools for designers should help them. If they are working in a restricted design space, then the tool should capture that. If it does not, in effect they will have to repeat work unnecessarily: (re-)specify to the tool decisions that are not real cognitive decisions because they are already in the brief dictated to the designer. Thus "styles," that is predetermined decisions, should be reflected in the tool once made. However since there is a very wide range of styles, as the above argument shows, this building-in is not something to be done in the factory, but more like defaults set at the site or per project. The appropriate tool to build therefore is one with a language for defining styles, which are then reflected in the design space presented to the actual designer.

Since changing a style is done much less often than other design decisions, it can be harder work. On the other hand since, as argued earlier, a given human designer may well have some time-ordering constraints on their design work; in this case a given individual may design a style and then work with it as part of a project. Thus the interface to design specification must be usable by the designers / programmers, not only by special experts.

4.2 Inadequacy of Widgets

Current user interface prototyping and construction tools are not particularly good at assisting a designer in discriminating among styles, largely because they use too simple a model, if any at all, of the specification as a collection of design choices. The standard X toolkits offer widgets, highly parameterised interaction objects. A designer's choices with widgets consist of (i) choosing a widget class (which determines the widget's parameters) and (ii) setting the parameters.

The class structure among widgets reflects an inheritance hierarchy and does not correspond, except by chance, to any other categorisation of interaction objects according to style or use. For example, MenuShell and ScrollingList, two of the widget classes in the Open Look™ widget set which support selection (a characterisation in terms of rationale) do not have a common parent class; their common class ancestor is the Composite widget class. Thus, the fact that they are both instances of a style of selection device is not reflected in the hierarchy.

Second, the widget's parameter space is entirely flat. There is nothing to indicate that setting one parameter is more or less important, from a design point of view, than another. As a convenience, most widget parameters have a default setting, so that a designer may safely ignore most of a widget's parameters. But which ones? There is no differentiation among the parameters to help.

The resource mechanism provided by X and similar window systems (Davison et al 1992) enables the values of some attributes of a widget to be determined, leaving other attributes to be set by default or by other resource specifications. Other systems take a similar approach to style, like HUMANOID's templates (Szekely et al 1992). Thus, styles which can be expressed as default parameter values to widgets are supported, but others are not, e.g., metaphorical styles and many aspects of house styles designed for usability. Also missing is any way of representing differences in degree of fixedness of choices.

4.3 Design Assistance

Several user interface management systems, such as Mike (Olsen 1992), Georgia Tech's UIDE (Foley et al 1989) and MASTERMIND (Szekely et al 1995) make a distinction between fixing the high-level decisions about an interactive system in a specification of its conceptual model, leaving decisions until later about the appearance and behaviour of the interaction objects to which the components of the conceptual model are related. Such systems employ sets of rules which putatively capture user interface styles and, when applied to the conceptual model, generate the user interface automatically.

Two types of fixed design decision can be identified in these systems. First, the conceptual model constitutes a set of fixed choices and, in terms of our definition of style, forms a style. This seems reasonable, since systems belong to recognisable categories with respect to the objects and operations which make up their conceptual model. However, this suffers from the problem identified in section 3.1, viz., that decisions about the conceptualisation of the application are fixed before, and independently of, decisions about the user interface's appearance and behaviour. This does not take into account the way decisions about the hardware/system software/ basic interaction style may restrict choices about the application's conceptual structure.

Second, the rules by which the user interface are generated are a fixed set of design choices, albeit ones (usually) without alternatives. This approach has been demonstrated to produce acceptable results when applied to limited domains of design choices, but its assistance fades as the range of design choices increases in scope; it is unlikely that, given current models of application, interface and style, the same technique will work for more sophisticated styles and application domains, such as metaphorical representation of a wide range of types of data and input techniques. Furthermore, the rule-based approach has so far focused solely on the generation of presentations based on the specification of the application and has ignored other types, such as "corporate" style.

5 Design-Time Constraints on Interface Specifications

5.1 Style as Design-Time Constraints

A style is any set of decisions taken before subsequent design proceeds, which will appear as constraints on or structuring of the remaining design space. A system supporting the general approach to styles which we have argued is actually required, must offer a language for specifying styles. This language will be, in effect, a language on the design specification; expressions in such a language specify the remaining design space and hence the allowed designs.

The solution which we propose to the problem of style-based support for design is to model design-time constraints on interface specifications: this is information about design significant properties which fixes the choices which a designer may make, and indeed sometimes forces additional design decisions to be made. The role of such constraints is to impose a structure, or indeed a variety of structures, on the various attributes which make up the interaction object's definition, ordering them with respect to the design considerations in terms of which the interaction object properties are specified. Facilities can then be included in design tools for defining and resolving these constraints.

Consider the Macintosh Finder icon discussed earlier. Design-time constraints one might place on such an object include:

- it must represent (i.e., carry information about) objects in the underlying file system, such as discs, directories and files,
- it must be one of a collection of such icons,
- it must possess a label which corresponds to the name of the represented object,
- it must be selectable,
- it should have an image which corresponds to the type of the represented object.

Some constraints, such as the choice of representational image, may have strong weightings, although they may be relaxed. The weighting of the choice is intended to make a decision regarding use of such an image a central decision in the context of icon design. That is, the designer should be forced to consider it early on in the design of the icon. Its absence may be compensated for by modifying other features, but if the decision is left too late, these other features may not have been modified appropriately.

Additionally, one may wish to include annotations to the constraints, such as the rationale for them. The constraint that the image be representational might include an annotation asserting that this property enables inferences about the type of the underlying object to be drawn from the nature of the image.

A corporate house style might be based on the incorporation of some features of the company's logo in the artefact being designed (Mahony & Gower 1991). Which features may depend upon a number of other design considerations. What is important here is to include a constraint on the properties of the object such that they satisfy the logo incorporation requirement. The means of doing so is not part of the constraint. It may be satisfied, perhaps, by using company colours, or by including the logo in the widget's area, or by including the logo in a component part of the object. Such a constraint is likely to be satisfiable only as the result of the activity of designers trying out alternatives, comparing them for adequacy against a variety of criteria and, finally, asserting a solution by including its specification in the design. In cases of this sort, the relationship of design rationale to style application becomes central.

5.2 The Role of Design-Time Constraints

Design-time constraints have a number of roles in the design process. First, they restrict, perhaps conditionally, the choices which a designer may make. Thus, given the choice to present information via a map, that information must be presented via a collection of appropriately located visual elements. This restriction is equivalent to the structure imposed by interaction object (e.g., widget) type or class as found in most user interface construction tools. Second, styles organise consequent choices, indicating the importance of specifying certain features of the chosen alternative interaction object type or class. Third, since styles are properties of specifications, they may be stored and retrieved for reuse in analogous situations (i.e., applied to new specifications).

Finally, as illustrated in the corporate logo example, some design constraints may force the designer to assert that certain design conditions or requirements are met. A central feature of these design constraints is that their satisfaction may require the intervention of a human designer. We cannot assume that (i) we can produce a constraint satisfaction engine sufficiently sophisticated to resolve constraints without such intervention and (ii) we can represent the knowledge to resolve the constraints in a design knowledge base. More importantly, the resolution of the constraints may be essentially unresolvable until a designer decision is taken, perhaps as the result of further reasoning or empirical investigation.

As the previous section suggested, sets of design-time constraints may be definitive of a set of solutions to a design problem. Thus, the notion of a map defines a class of solutions to the problem of displaying multi-dimensional data. A corporate "style" is a family of solutions to the problem of making artefacts identifiable as representing an institution. A representational or interaction object may satisfy more than one set of constraints at the same time and thus may belong to more than style.

5.3 Constraint Satisfaction

Constraints typically are defined as relationships among entities (Leler 1988); in a dynamic system these must be maintained true via a constraint satisfaction mechanism. Typically, constraint mechanisms have been employed to model dynamic systems specified declaratively.

This may appear an unlikely method of modelling style in a user interface design tool. However, if we conceive of a style as placing restrictions upon the degree of freedom which with certain features of a specification may be modified interactively by a designer, then the concept of a constraint appears more appropriate. A style, then, consists of a set of relationships, each a tuple of system property and conditions on that property, perhaps modified by a degree of fixedness (here represented by 'must' or 'should', cf. "glue" in Knuth's TEX box and glue model). In the Macintosh icon example, its style consists of:

Mac icon style =

1. source must be element of a collection
2. representation must be element of a collection
3. source must be filing system objects
4. representation must have a label representing the name of the source object
5. representation must be selectable
6. representation should have an image which pictorially represents the source object type

The first five constraints are easily satisfied if (i) collection is one of the legal values for the domain type, (ii) collection is one of the legal types or classes of representation (e.g., widget), (iii) the type of the source objects can be determined, (iv) the properties of the objects are known to the system and (v) selectability is one of the potential behaviours of interaction objects.

Determining if the sixth constraint is satisfied is more difficult. First, the constraint specifies a "should have" relationship between the attribute and its value. Second, it may be impossible, in the absence of a knowledge-base which holds information about the representational properties of images, for the system to tell if a given image is representative of a given file object type. Thus, the satisfaction of this constraint may well require input from a designer in the form of an assertion that the constraint has or has not been satisfied. This forces the designer to consider whether other factors influencing the design make necessary the inclusion of pictorial representation of type. The purpose of the constraint is to guarantee that the decision is made.

Constraint satisfaction occurs at different points in the process of specification. Thus, the constraint on representation type (i.e., element of a collection) will require satisfaction when an attempt is made to change the type of the representation, typically by restricting the choices available to the designer. The constraint on the label attribute of iconic elements must be satisfied when an instance of the specification is linked to a source domain and the name of source domain entities is mapped onto iconic elements.

The degree of fixedness of a style is modelled in the notion of style application permissions. When a designer applies a style to a specification, she may state whether that style may be removed from the specification or relaxed (temporarily made inactive). Styles which can be neither removed nor relaxed constitute the "givens" of a design, beyond the control of the designer to modify, perhaps because of project requirements which cannot be negotiated. Relaxable constraints represent those which are a central feature of the design, but may be "traded off" against other features. Removable constraints are those which are part of the current design hypothesis, available for retraction if evaluation shows they are not desirable.

5.4 The Structures of Styles and Constraint Satisfaction

Styles may be placed in an inheritance hierarchy, such that styles inherit and possibly override the constraints of their parent. While useful in providing a structure for styles themselves, there are problems with this approach.

As demonstrated earlier, one specification may exhibit, or possess, a number of different styles. If styles were structured in a single hierarchy, this would result in redundant constraints being applied. The constraint satisfaction mechanism must be able to remove such redundancies. More seriously, it is possible that inconsistent or mutually unsatisfiable constraints be applied to a specification. The constraint satisfaction mechanism must be able to handle such cases. Alternative techniques include disallowing more than one constraint applicable to the same attribute for the same relationship type (pre-emptive) or determining constraint conflicts and asking the designer to "relax" constraints until satisfaction is possible (interactive).

5.5 Assistance with Style Application

We suggested at the beginning of this paper that the application of styles to a computer-based artifact or design can be difficult. Without assistance, making document headings follow a consistent style requires wasteful repetition. Complex sets of style rules, such as in corporate styles, can be difficult to apply without inconsistencies and oversights. Finally, some styles are abstract and thus likely to be overlooked, as with the semantic consistency style of Macintosh menus.

There are three types of style-oriented assistance which can be offered by a computer based tool:

automatic style application
The style facilities in word processors provide automatic style application. Some UIMSs have automatic style application systems, such as DON in the UIDE (Kim & Foley 1990), and some automated visualisation systems (Mackinlay 1986, Casner 1991) use rules analogous to style rules. In all of these systems, the rules which can be specified are limited in terms of domain and in what can be specified within those domains. For example, none allow for applications or

their component structures to exhibit different styles. Also, the style rules which can be expressed are only those which can be dealt with automatically. There is no provision for style rules which go beyond computer-based semantics to include design constraints which require some designer intervention.

critiquing
If styles exist as a set of assertions expressed in a well-formed language, it is possible for a system to examine them for consistency and completeness. Although design critiquing systems currently exist (Malinowski & Nakakoji 1995), their rules do not incorporate the notion of style as elaborated here.

design space management
A tool which supports style as we have defined it would be able to hold design-significant information about those styles. For example, it would be possible to link styles which are related to one another by history (they have been used together in the past), design rationale (they share a design justification), domain of application (e.g., they have been previously applied to model geographic information). Furthermore, such a system could maintain a record of the design constraints which require designer input for resolution, reminding the designer of any such constraints not yet dealt with. No existing system offers such services.

6. Conclusions

This paper has analysed the apparently disparate uses of the word 'style', and combined this with considering the varying degrees of fixedness of design decisions to arrive at proposed requirements for a design support tool. Such a tool would be a general style definition facility, allowing designers to make and record decisions early on (as styles) that have a high degree of fixedness given the requirements (including implicit and general requirements such as "implement in X Windows," "make the corporate identity visible," "obey house style guidelines"). These decisions may be implemented early or late in existing design environments (e.g., corporate logos fitted in late on), but in fact are logically fixed from the outset and so should be dealt with early to avoid wasteful backtracking (e.g., having to redesign the page layout to make room for a logo whose presence was in fact predictable).

These requirements suggest significant limitations in what is currently available. Current facilities mainly revolve around what is easy to implement (e.g., word processor styles) or fixed UIDE architectures (e.g., X widgets) or support for design justifications. They do not, we have argued, correspond to the decisions designers actually have to fix, and therefore wish to record, at the early stage corresponding to designing a "style." However, by drawing on the mechanisms for constraint resolution including prompting the designer through structured judgements where these cannot be automated, a much more advanced facility could be implemented.

Such a design tool must benefit the designer by allowing and supporting a better ordering of design decisions that corresponds more nearly to the order of fixedness of decisions given the requirements, however heterogeneous those decisions are. Benefits to the end user will be limited inevitably by our understanding of what really matters to users. Styles for designers correspond to consistency for users; both refer to the issue of similarity across different designs, but not all similarities turn out to benefit user performance. While the work on Task Action Grammar (Payne & Green 1986) has demonstrated some of the issues that matter to users, Grudin (1989) shows that not all kinds of "consistency" do benefit users. However, as we learn more about the relationship of consistency to user performance, we will have an even greater need for tools such as we propose with which designers can directly and re-usably express these as constraints on design.

Acknowledgements

We wish to thank Dr. David England and Dr. Kevin Waite, who offered useful comments on an earlier draft of this paper.

References

AT&T Open Look Graphical User Interface Style Guide.(1990), AT&T.

Bertin, Jacques. (1987), The Semiology of Graphics, translated by William J. Berg, University of Wisconsin Press.

Casner, Stephen M. (1991), A Task-Analytic Approach to the Automated Design of Graphic Presentations, *ACM Transactions on Graphics*, 10(2), pp. 111-151.

Davison, A.; Drake, K.; Roberts, W. and Slater, M. (1992), Distributed Window Systems, Addison-Wesley.

Foley, James; Kim, Won Chul; Kovacevic, Srdjan; and Murray, Kevin (1989), Defining Interfaces at a High Level of Abstraction, *IEEE Computer*, pp. 25-32.

Grudin, J. (1989), The case against user interface consistency, *CACM* 32(10), pp. 1164-1173.

Human Interface Guidelines: The Apple Desktop Interface (1986), Apple Computer Inc.

Kim, Won Chul and Foley James D. (1990), DON: User Interface Presentation Design Assistant, *ACM SIGGRAPH Symposium on User Interface Software & Technology* (UIST '90), pp. 10-20.

Leler, William (1988), Constraint Programming Languages: Their Specification and Generation, Addison-Wesley.

Mackinlay, Jock. (1986), Automating the Design of Graphical Presentations of Relational Information, *ACM Transactions on Graphics*, 5(2), pp. 110-141.

Maclean, Allan, Young Richard; Bellotti, Victoria; and Moran Thomas (1991), Questions, Options, and Criteria: Elements of a Design Rationale for User Interfaces, *J. Human Computer Interaction*, 6(3-4), pp. 201-250.

Mahony, Karen and Gower, Andrew (1991), The Development of a Visual Style for a BT X Window System Toolkit., in D. Diaper & N. Hammond, eds, *People and Computers VI. Proceedings of HCI '91*. Cambridge University Press, pp. 265-279.

Malinowski, Uwe & Nakakoji, Kumiyo. (1995), Using Computational Critics to Facilitate Long-term Collaboration in User Interface Design, *CHI '95*, pp. 385-392.

Newman, William (1988), The Representation of User Interface Style, in D.M. Jones & R. Winder, eds., *People and Computers IV. Proceedings of HCI '88*. Cambridge University Press, pp. 123-137.

Olsen, Jr., Dan R. (1992), User Interface Management Systems: Models and Algorithms, Morgan Kaufmann, pp. 134-152.

Payne, S.J. and Green, T.R.G. (1986), Task-actions grammars: A model of the mental representation of task languages, *Human-Computer Interaction*, 2, pp. 93-133.

Shneiderman, Ben (1992), Designing the User Interface, 2nd edition, Addison-Wesley.

Smith, S and Mosier, J. (1986), Guidelines for Designing User Interface Software. Mitre.

Szekely, Pedro; Luo, Ping and Neches, Robert. (1992), Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design, in *Proceedings of CHI '92*, pp. 507-515.

Szekely, P.; Sukaviriya, P.; Castells, P.; Muthukumarasamy, J. & Salcher, E. (1995), Declarative interface models for user interface construction tools: the MASTERMIND approach, *EHCI '95*.