Direct Combination

Simon Holland
Department of Computer Science
Open University, Milton Keynes
MK 7 6AA, England
+44 1908 653148
s.holland@open.ac.uk

Daniel Oppenheim
Computer Music Center
IBM TJ Watson Research Center
PO Box 218, Yorktown Heights
NY10598 USA
+1 914 945 1989 music@watson.ibm.com

ABSTRACT
This paper reports on Direct Combination, a new user interaction technique. Direct Combination may be viewed variously as: a systematic extension to Direct Manipulation; a concise navigational framework to help users find the operations they need; and as a framework to make a greater range and variety of operations available to the user, without overburdening user or interface designer. While Direct Combination may be seen as an extension of Direct Manipulation, it may also be applied to a wide range of user interaction styles, including even command line interfaces. Examples from various hypothetical systems and from an implemented system are presented. This paper argues that Direct Combination is applicable not just to problem seeking or design oriented domains (where the technique originated) but is generally applicable. A variety of new interaction styles for Direct Combination are presented. The generalisation of Direct Combination to the n-dimensional case is presented.

Keywords
Interaction technique, interaction styles, interaction design, navigating large operator spaces, novel interaction objects, n-tuples, creating new operations, interaction theory.

INTRODUCTION
In direct manipulation and graphical user interfaces from the Xerox Star [5] onwards, the form of many, though not all, user interactions may be characterised loosely in terms of the following pattern.

interactionObject operator [arguments]

That is to say, user interactions often consist of the user selecting some interaction object, then using a menu, button, keystroke, mouse or similar means to specify an operation on that object. A dialog box or other mechanism may be used to allow the user to qualify the operation with one or more arguments. This interaction pattern can be re-stated metaphorically in the following way:

noun verb - with optional qualifying terms.

Direct Combination is a way of extending Direct Manipulation by focusing systematically not on single interaction objects but on *pairs* of interaction object. The essential requirement for Direct Combination is that for *every pair* of interaction objects in a system, there should be at least one or more operators defined and available to the user. To explore this idea, we will consider some examples. Direct Combination can be afforded using a variety of interaction techniques: we will begin by focusing on a style using a toolglass or magic lens [1]. (These terms are explained in the following section.)

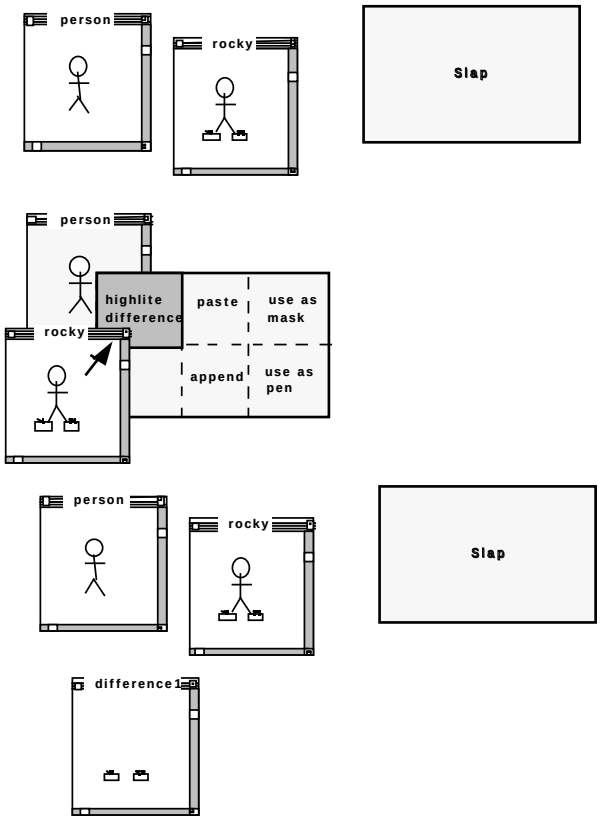

Figure 1: A Direct Combination of two bitmaps. One bitmap is dropped through a toolglass onto the other to produce a difference bitmap.

Introducing Direct Combination
Figure 1 shows two windows (labelled 'Rocky' and 'Person') in a hypothetical desktop environment. These

windows contain similar, but not quite identical bitmaps. If the user drags one window over, or partly over another window, nothing of interest happens; the first window is simply left lying over the other. But the situation can be transformed by using a toolglass (figure 1). A toolglass [1] is typically a transparent but shaded pane (here labelled 'Slap') which can be slid around independently over the other interaction objects. The user moves a toolglass by using a pointing device operated by the second (usually the left) hand. Tool glasses, and the closely related magic lenses, are generally used to modify the effect of some tool wielded by the other hand, or to provide specialised views of interaction objects underneath them. In Direct Combination, the toolglass can be used to modify the effects of drag and drop. For the following examples, we will introduce some terminology to improve clarity. In cases of drag and drop, we will refer to the item being dragged as the **'visitor'**, and the item it is dropped on as the **'target'.** We will refer to the tip of the arrow used to drag the visitor as the **probe.**

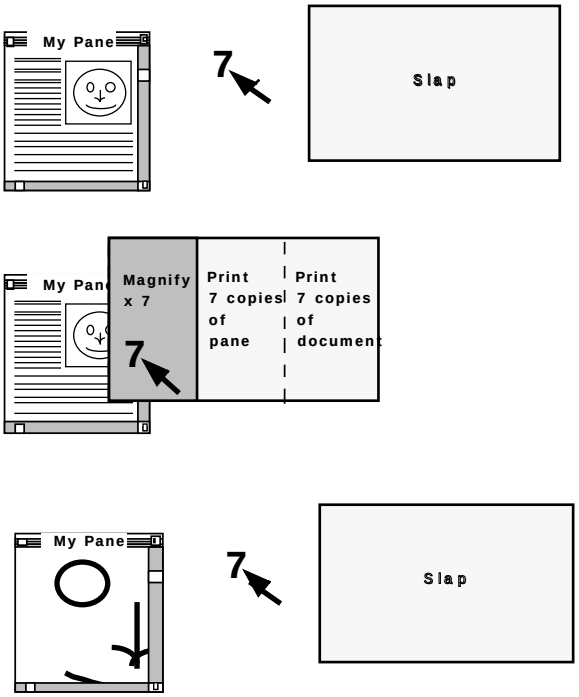

Figure 2: Direct Combination of the integer 7 with a bitmap.

To continue with our first example, when the tool glass by itself is slid partially or wholly over the window labelled 'Person', this has no particular effect. But if the user drags the visitor, 'Rocky', over the target, 'Person', with the tool glass sandwiched in-between, something different happens. As in conventional drag and drop, the target becomes highlighted when the visitor is held over the target. At the same time, the toolglass divides into labelled regions each representing a different operation that can be performed on the ordered pair of visitor and target object. In the present case, the visitor and target both happen to be of the same type, namely they are both windows open on a bitmap. Accordingly, the toolglass divides itself into regions corresponding to the following operations specific to two bitmap windows: *paste, append, highlight differences, use as mask, use as pen.* (Of course, many other operations could be defined on pairs of bitmap windows.) The region of the toolglass labelled with the default option (*'highlight differences')* is distinguished from the other regions by bold dividing lines. By sliding the toolglass around with the left hand between target and probe, the user can choose one operation from those on offer. Specifically, the tip of the arrow icon (the probe) used to drag the visitor object is taken to define the selection point. When a toolglass region is selected but before it is executed, it is highlighted (just like a menu choice). If the help system is switched on, a brief summary of each operation is displayed as the operation is selected. When the visitor is dropped, the highlighted operation is executed. In our first example, a third bitmap window is created containing just the differences between the two bitmaps (figure 1).

Diversity of Interaction Objects

Direct Combination (DC) is not limited to dragging and dropping windows or icons: it can be applied to any kind of interaction object. Indeed, the wider the range of interaction object available, the more expressive direct combination can be. Our second example (figure 2) deals with a hypothetical implementation of Direct Combination in Self [6], where all system objects, including numbers, are draggable interaction objects. In this second example, dragging the integer 7 onto the window labelled 'My Pane' has no effect, but placing the toolglass between visitor and target elicits three possibilities, *magnify by 7, print 7 copies of pane dump, or print 7 copies of full document.* Figure 2 shows the effect of dropping the number 7 through the 'magnify by 7' region of the tool glass. Note that all options differ from those in the first example. The options are determined neither by the integer alone nor by the window alone - they are determined by the context of the *ordered pair* of interaction objects *(number, window).*

An Alternative DC Interaction Style: Portals

The tool glass makes it possible to retain the ordinary functionality of drag and drop (e.g. moving files) while keeping direct combination interactions straightforward. But direct combination can be implemented without the complication of two pointing devices. One alternative interaction style that requires only a single pointing device uses *portals.* Figure 3 shows a desktop environment with a document and a folder. In this example, dragging the document icon over the folder icon causes the target icon to be highlighted and to expand, if legibility demands, revealing two portals labelled *move* and *other choices.* The default operation, **move** is shown in bold. If the user wishes to choose the default operation, the visitor object

can simply be dropped down the default portal as a continuation of the gesture used to highlight the target. Alternatively, if the user drops the visitor down the *other choices* portal, a menu of choices is offered. In this case, the choices specific to the file and folder ordered pair include the following: *copy the file to the folder; move the file to the folder (default); find all files in the folder of the same type as the sample file; find all files in the same folder created at the same time (minute, hour, day, week, month or year) as the sample file; find all duplicates of the file.* To cancel the operation, the user need only move the cursor away from the menu, which then vanishes, and the target flies back to its original position. In figure 3, the user has chosen the *find all files of the same type* option, which creates a new folder on the desktop containing the aliases of the found files.

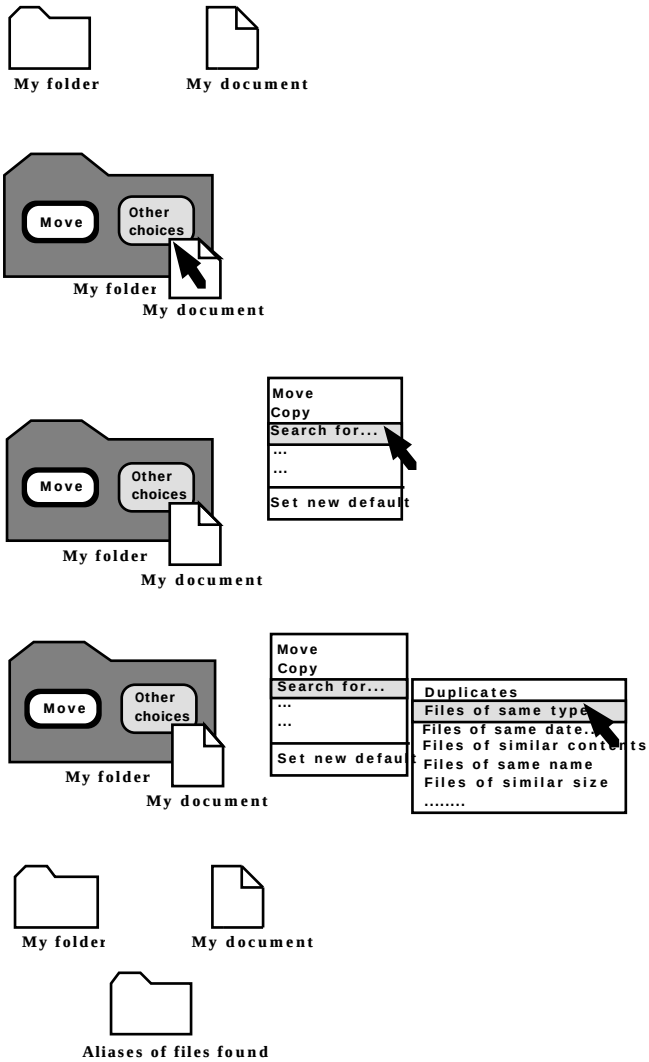

Figure 3: Direct Combination via a Portal.

DIRECT COMBINATION Vs DIRECT MANIPULATION

One way to view Direct Combination is as an extension of Direct Manipulation. We noted earlier that many (though not all) direct manipulation user interactions may be characterised loosely in terms of the following pattern.

interactionObject operator [arguments]

Direct Combination requires the following additional interaction pattern to be made available.

(visitorObject targetObject) operator [arguments]

In other words, Direct Combination requires (in the ideal case) that the system permit *any* pair of interaction objects to interact meaningfully. As noted earlier, this capacity should not be limited to icons. Interaction objects can include: graphic elements, diagrams, selections of text, items on a list, collections of cells, hotlinks, parts of a pane, numbers, characters, files, folders, or entire windows or panes. As the second example suggested, the scope of direct combination is greatest in systems with the widest range of interaction objects. An ideal example of such a system, noted earlier, is Self [6], which is both a programming language and a user interface construction environment. In Self, every internal object is potentially visible, and every visible item has the potential to take part in user interactions. In its drag and drop manifestation, Direct Combination requires that every object in the system can meaningfully be dragged and dropped on any other object. Direct Combination can be characterised by the following principles.

Principles of Direct Combination

• Every object of interest in the system should be visible (or more generally perceptible).

• Every object of interest in the system should be capable of treatment as an interaction object.

• Every interaction object should be capable of useful interaction, in one or more ways, with any other interaction object. The interactions available should be diverse, and should be well-suited to each *ordered pair* of object types.

Of course, there is nothing new about dragging and dropping, sometimes with a limited choice of operations (e.g. Windows non-default drag-and-drop). The key requirement in Direct Combination, that differentiates it from other interaction strategies is that *every pair* of objects of interest must have its own set of useful operations defined, well-suited to that particular pair. Conversely, Direct Combination is not limited to drag and drop. Some other interaction styles for direct combination will be examined later in this paper that do not involve drag and drop at all.

Treatment of Argument Objects

The objection might be raised that the 'new' interaction pattern is already implicitly present in the optional arguments of the existing pattern

interactionObject operator [arguments]

This may be true in some abstract mathematical sense, but not from a user interaction viewpoint, since the direct combination pattern gives rise to distinctly new affordances and new usability issues. However, focusing on the

treatment of *arguments* in the pattern does bring to light some interesting issues, as we shall now consider. In many direct manipulation systems implemented using the *noun verb* pattern, dialog boxes are used to specify any needed arguments. Such dialog boxes create a context which restricts the freedom of the user. Indeed, many dialog boxes are *modal*, requiring the user to specify certain information or to cancel the entire interaction before being permitted to take part in any other interaction. Conversely, dialog boxes (i.e. the parts of the system dealing with arguments to a command) are often *not accessible to the user* until the primary interaction object and the operator have been determined. Both of these restrictions violate the principle of direct manipulation that the items of interest should be visible and open to manipulation at all times. Thus, some styles of interaction characterised by the pattern *noun, verb, dialog-box* prevent users from manipulating primary objects (or operators) of interest while dealing with arguments, and vice versa. This can be irritating to users. For example, a dialog box may require the user to specify a pathname for a file, but provide no convenient means from within the dialog box to find it. Outside of the dialog box, the file may be directly selectable, but to no avail. Direct Combination provides a way around some of these violations of direct manipulation principles imposed by dialog boxes. This issue arises even more acutely in the n-dimensional case, treated later in the paper.

IMPLEMENTATION AND FEASIBILITY

To simplify discussion of implementation issues, it will help to introduce further terminology. Since direct combination is not limited to drag and drop, it is useful to have a common term for the *user action* of whatever kind used to interact two objects. This is referred to as a *slap* [4]. The resulting operation carried out on the two objects of interest in the underlying system is called the *mix* operation. If there is a choice of operation, then the mix operator incorporates a set of *submix* operations. At first sight, it might appear that direct combination must be arduous to implement, in terms of the amount of work required to specify the semantics of the mix operations for each *pair* of interaction object classes. This need not be so, as can be seen from the case where direct combination is implemented in a *single-rooted, uniformly object-oriented* operating system or environment such as Smalltalk, Omega or Self. In fact, Direct Combination can be applied irrespective of how the underlying system is implemented, but the method of implementation is particularly straightforward in such systems. The implementation strategy can be outlined as follows. Firstly, note that every visible (or more generally, every perceptible) object in the interface should correspond to a more or less well-defined computational object in the underlying object-based system. Secondly, note that in a single-inheritance class hierarchy, every object in the system will inherit its behaviour from one or more abstract classes (or prototypes in a prototype system) ultimately commonly rooted in the most general category *Object*. By exploiting inheritance, and assuming there are *n* classes of interaction object, it is clear that there is no need to define mix and submix operations explicitly for all n^2 possible pairs of classes. We need only define distinct mix operations for the much smaller number of appropriate abstract classes, and for some leaf classes. Provided these classes and their mix operations are suitably chosen, the mechanism of inheritance will insure that all other classes will inherit appropriate mix and submix operations. Note that for most pairs of interaction object, there will be several submix operators defined, with one operator marked as the default, subject to user customisation. Note that in a Smalltalk implementation, the mix operator will typically contain a method dictionary of submix operators, each of which can be implemented using double dispatch.

ORIGINS OF DIRECT COMBINATION

Direct Combination is a broader version of 'Slappability' [4]. Slappability is essentially the systematic exploitation of pair-wise interactions, together with an object-oriented architecture to facilitate this, as outlined in the previous section. The first system to support this idea systematically was DMIX [4], an object-oriented environment for music composition. See the next section for an example of pairwise interaction as implemented in DMIX. The original motivation behind slappability was to manage conversions between different musical representations. Holland extended slappability to the broader form presented here, Direct Combination[2]. In this paper we present these extensions in detail, including: new interaction techniques using toolglasses and portals; the generalisation to the n-dimensional case; analyses, definitions, principles, and characterisations suitable for HCI purposes; terminology such as 'direct combination', 'visitor', 'probe', 'portal', and 'command needle'; and example applications and scenarios outside of the musical domain, to illustrate the general usefulness of the technique.

USES OF DIRECT COMBINATION

Musical Uses

As mentioned in the previous section, Oppenheim's implemented system, DMIX [4] offers numerous examples of Direct Combination using a simple menu-based technique. As also mentioned, the original motivation for providing such a facility was to help users convert between the diverse representations encountered in music systems. A simple example of direct combination in DMIX is shown in figure 4. This involves slapping a mathematical function (here a sine function) onto a Bach prelude. The middle part of Figure 4 shows a sine function displayed in the standard graphical representation for function objects in DMIX. The top part of Figure 4 shows Bach's Prelude No. 1 as a DMIX music object displayed using piano roll notation.

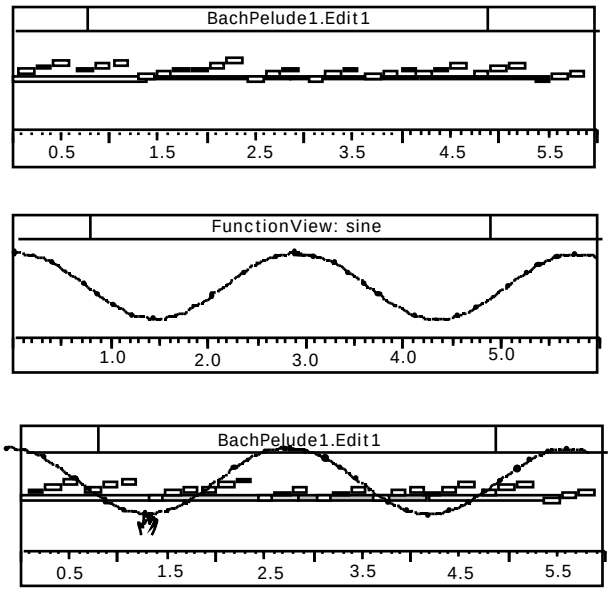

Figure 4: A musical use of Direct Combination. A sine function is slapped onto Bach's Prelude No. 1 to modify the pitches (or the dynamics, timbre or micro-timing).

In piano roll notation, the y-axis represents pitch and the x-axis represents time. The horizontal bars scattered on the diagram represent notes, each an object with various attributes in its own right. The duration of a note is indicated by its length. In DMIX, a slap is initiated by keying command-S when the cursor is over the visitor (in this case the sine function shown in figure 4). The cursor changes shape into a hand holding a sine curve (see bottom part of figure 4). The new cursor shape both indicates that a slap has been initiated, and identifies the visitor. By dragging the cursor in this state over the target (in this case the Prelude) and releasing the mouse button, the slap is executed. The default effect of a slap on an ordered pair consisting of a mathematical function and a music object in DMIX is that the music object takes new pitches from the function. If the user does not want the default mix operation, a menu of alternatives can be provided by initiating the slap using Shift-S instead of command-S, in which case a hierarchical menu of non-default operations is popped up (not shown). In principle, a wide range of alternatives operations could be provided for the interaction of a function and a music object. For example, the function might be applied not to the pitch, but to the rhythm, the dynamics, the timbre, or the micro-timing. Similarly, the function need not simply overwrite the previous values, it could be used to add to them, or to multiply them, or to vary them using any suitable mathematical relationship. To appreciate the power of Direct Combination in DMIX, it helps to realise that music objects can originate from a wide variety of sources; from a file, an algorithm, a music editor, or from a live performance. Direct Combination allows musicians to craft performance nuances (via successively applied functions) onto a piece of music (the music object) originating from whatever source. A key benefit is that users need know nothing about the originating format, or about the tools normally required to work with that specific format.

Uses In Problem-Seeking Domains

One interesting feature of Direct Combination discovered by Oppenheim in the musical domain [4] was that the inheritance of mix operations sometimes gave rise to useful operations between pairs of object that had not been explicitly foreseen by the system designer. In open-ended, problem-seeking [3] or design-oriented domains, this is a desirable property for fostering creative approaches. It suggests that Direct Combination has considerable potential as a design support framework for use in open-ended, problem-seeking or design-oriented domains. On the other hand, in safety-critical or enterprise-critical applications, the existence of interactions with unforeseen behaviours might be highly undesirable. However, designers who did not want to permit unforeseen interactions could simply enumerate and audit all possible combinations of object types and then explicitly block any undesired interactions. The Direct Combination principle demanding that all interaction objects should be able to interact with each other is of a heuristic and polemical nature rather than an absolute requirement. A wealth of other musical examples can be found in Oppenheim[4]. Oppenheim's DMIX and the papers written about it and the pieces composed with it furnish an existence proof that Direct Combination is implementable, usable, and produces useful results in at least one domain. Direct Combination clearly excels at managing conversions and interactions between different representations. Note that the user interaction gestures used in DMIX are control key, drag and drop, and menu mechanisms, rather than the toolglass or portal mechanisms introduced here.

Exploring Large Spaces Of Operators

One purpose of this paper is to demonstrate that Direct Combination is a widely applicable strategy, and not limited to uses in design-oriented domains. In particular, Direct Combination appears to be useful in more or less any application where there are very many operations defined on a variety of objects, and where the classification system for these operations may not be immediately apparent to the user. In such situations, users tend to experience navigation problems, and have difficulties in finding the relevant operations quickly. This is especially true when the name of the operation may be unknown, hard to guess, or when it is unclear even whether the operation exists. These problems can apply to more or less any large application program, such as well-known word processors. For example, when dealing with an unfamiliar word processor under an unaccustomed operating system, it may be unclear how to convert a document to HTML. Similarly, it may be hard to know how to itemise automatically the differences between two drafts of a document originating

from an unfamiliar word processor. In both cases, Direct Combination makes it possible to cut down the search space simply by considering pairs of relevant objects. A user wishing to carry out either operation might 'browse' the direct combination interactions available between two relevant document files, as shown in figure 5.

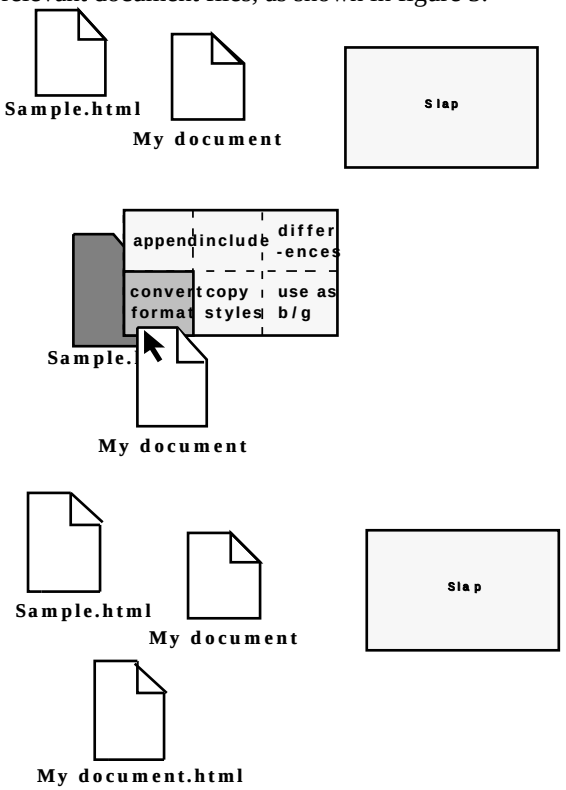

Figure 5: Converting the format of one document to the format of another by Direct Combination through a toolglass.

In the case of the format conversion, a sample document in the desired format could be employed by the user to indicate to the system which destination format is desired. In our example, to carry out the conversion, the user simply drops the document 'My document' through the 'convert format' region of the toolglass onto the sample target document Sample.html. A third document is thereby created, namely the first document converted to HTML format (figure 5).

ISSUES FOR DIRECT COMBINATION

Light Weight User Interfaces

Direct Combination can be implemented without the expense of a second pointing device or the graphical complication of the portals mechanism. For example, figure 6 revisits the interaction between two documents explored in figure 5 (the HTML conversion), but this time using a simple menu-enhanced drag-and-drop. This interaction style need not get in the way of ordinary drag and drop, since the control key can be used to elicit Direct Combination on dragging. Direct Combination can be implemented in even more rudimentary interaction styles. For example, a command line user interface parser could be designed to make use of the techniques outlined in the section on implementation to process commands with two initial 'nouns' such as the following.

> *> myDocument sample.html convertFormat*

Similarly, Direct Combination can, if desired, be implemented as a systematic extension of cut and paste, rather than as a variant of direct manipulation.

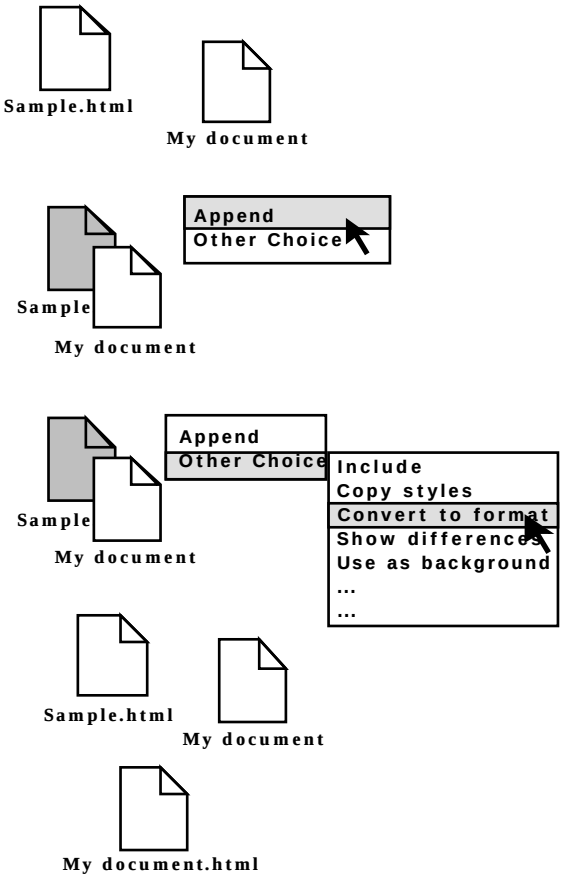

Figure 6: The same interaction as in figure 5, using an easily implemented, menu-based drag-and-drop version of Direct Combination.

Design Of Space Of Mix Operations

There are several approaches for system designers to determine which particular set of mix operations should be made available for a particular ordered pair of classes of interaction objects. One approach is to catalogue existing operations and make them available via additional routes as mix operations. This approach does not necessarily make any new operations available, except perhaps to some classes of interaction objects by inheritance, but it may make these operations accessible in new ways. This conservative approach is called *Operator Re-use.* By contrast, the more constructive *Operator Synthesis* involves systematically looking for new operations suitable for defining on ordered pairs of objects. This approach can make many new operations available. The search may be carried out informally, on an intuitive basis of what

operations appear to be useful, or more systematically. For example, one more systematic technique is to list the attributes of each object, and then to look for ways in which the state of one object can be used to alter the state of another object. Similarly, one can use attribute lists to look for ways in which new objects can be formed from two given objects. Note that operator sets may depend on the state of operands, as well as on their class. Where these processes produce too many candidate mix operators, the numbers can be reduced by expert pruning, or by conventional usability techniques such as task analysis and considerations of relative frequency of use.

N-DIMENSIONAL DIRECT COMBINATION

One of the key ideas of direct combination is to systematically extend the number of direct manipulation operations that are defined (and conveniently accessible) for *pairs* of interaction objects. But there is no reason to stop with pairs. In this section we show how Direct Combination can be extended to deal with arbitrary *n-tuples* of interaction objects simply and consistently. One important constraint on the interaction design is that we do not want any new elements of the interaction design to interfere with existing ordinary direct manipulation actions, or ordinary pairwise direct combination actions. So, for example, it might be confusing if the simple selection of a collection of interaction objects was taken as the invocation of an n-fold direct combination. Such a selection might be intended by the user simply as the first step in sending a single menu command to the selected collection of objects, which is a widely recognised direct manipulation idiom. A better candidate interaction design uses the *command needle*, described below (figure 7). The *command needle* is a new interaction object that looks like a spike on a base. When an interaction object is dropped on the needle, the object is highlighted to show that the object is *spiked*, and not merely placed close to the needle. The needle responds to the spiking by displaying a menu showing available commands. Of course, the commands displayed will vary depending on the number and class of objects spiked on the needle. This is best understood by means of an example. This example is intended to illustrate the interaction style and may not be an ideal example of a well-designed n-fold direct combination operator set. Figure 7 shows a cut and pasted area from a bitmap that has been pasted onto the needle. This elicits a needle menu with several relevant operations, including one shaded as a default (in the example, **Print...**). The user may ignore the available options, or may, in help mode, consult displayed information about any selected option, or may execute a selected option using the *do it* button. Note that in our example, the **Print** operation happens to be accessible via several routes from a variety of n-fold combinations using one, two, three or four objects. The user should not be expected to exhaustively catalogue all arguments to a potential operation just to be offered that operation. Much as conventional drag-and-drop operations may sometimes provoke a dialog box or other intervention to solicit additional parameters from a user, so it is with n-fold direct combination. To continue with our example (figure 7), the user chooses to spike a second object, namely a printer icon. This alters the choice of operations on the needle menu, any of which can be browsed for details, executed, or ignored, as before. When two or more objects are spiked, the *spatial* ordering of the objects top to bottom may, in some circumstances, make a difference to the operations evoked. This is because the designer may arrange for differently ordered n-tuples to evoke different sets of operations. However, the *temporal* order in which the items are spiked does not matter. One of the (hidden) needle preference controls allows the user to opt for the spatial order of spiked items to be ignored, in which case all relevant operations, irrespective of tuple order are made available. It was noted earlier that the simple selection of a collection of objects is not, by itself, an appropriate interaction design for invoking direct combination operations. Indeed, this observation led to the devising of the command needle. However, a collection of interaction objects may be selected together and then dropped as a group on a command stick. In such a case, the temporal order of selection *is* translated into spatial order of spiking, i.e. temporal order becomes tuple order. To make it easier for users to read the labels of items put on the needle, the needle is by default arranged vertically. This tends to aid legibility for labels in horizontally flowing languages (e.g. European languages). But the needle may be rotated through ninety degrees, if preferred. Continuing with our example, two more items are spiked next, the integers 7 and 2 (figure 7). As it happens, this does not alter the choice of operations offered. Selecting the 'print' operation now displays a dialog box for printing 7 copies of the graphics clip with a magnification of 2 at the specified printer. The dialog box and its contents provides feedback on how the selected operation is being interpreted. To alter this interpretation, the dialog box could be edited, or alternatively the order or identity of items on the spike could be altered by direct manipulation, in which case the changes would be reflected immediately in the dialog box. Note that this close coupling between the contents of the command needle and the dialog box avoids the restriction of the user's freedom identified in the earlier section on argument objects. Indeed, in some respects, a command needle is an open-ended direct manipulation version of a dialog box. To complete our example, in figure 7, the user finishes by changing some of the objects on the spike, ending up with the original graphics clip, an email address, a number, and a graphics document. In our example, just one operation is finally offered on the needle menu for this particular combination of objects. Selecting this operation discloses via the dialog box the interpretation - namely to

email the bitmap clip to the given email address, at half size, and converted to the format represented by the sample PICT file. The *do-it* button causes this command to be executed. When a command is executed, all icons fly back to their original locations. Note that the implementation of the n-fold case in languages like Smalltalk is much as in the two-fold case, but with double dispatching on multiple arguments. In CLOS, the implementation is even more direct using multi-methods. The n-fold version of Direct Combination may not have the immediacy of pairwise direct combination, but it does appear to have applications in helping users to constrain searches for operations in large and unfamiliar command sets by considering the objects involved.

CONCLUSIONS

We have presented a novel interaction strategy, Direct Combination, that builds on traditional direct manipulation techniques. Several Direct Combination styles using drag and drop through a toolglass, portals, and other techniques have been presented. We have shown how, in situations where it is hard to locate commands from a large command set, users may constrain their search space by an intuitive consideration of what kind of objects are involved. By focusing systematically on direct manipulation interactions between two or more objects, we have identified a framework that may have the potential to make a greater range and variety of operations available to the user, without overburdening user or interface designer. We have demonstrated a variety of interaction styles that make these strategies available without interfering with conventional drag and drop or cut and paste. Novel interaction objects such as the portal and command needle have been presented. We have demonstrated that the strategy has plausible applications much wider afield than design-oriented areas such as music composition, in which the basic idea was first devised and implemented. We have shown how the idea can be usefully generalised to the n-dimensional case.

ACKNOWLEDGEMENTS

We thank Bill Gaver, who provided deftly targeted advice and was a first-rate mentor. Mark Blurton-Jones and Rob Griffiths took part in useful discussions. Tom Carey created a much needed space to think about the problem.

REFERENCES

1. Bier E. A., Stone M.C., Pier K., Buxton,W., and DeRose T. D. (1993). Toolglass and Magic Lenses: The See Through Interface. Proceeding of SIGGRAPH 1993, Computer Graphics Annual Conference Series, ACM, 1993, Pages 73-80.

2. Holland, S. (1998). Direct Combination: novel user interaction strategies. Technical Report 98/20. Department of Computing, Open University, Milton Keynes, England.

3. Holland, S. (1999). Artificial Intelligence in Music Education: a Critical Review. In Miranda, E.R. (Ed.) Readings in Music and Artificial Intelligence. Contemporary Music Series, Vol. 20, Harwood Academic Publishers, Amsterdam, Netherlands.

4. Oppenheim, D. (1996). DMIX: A Multi Faceted Environment for Composing and Performing. Computers and Mathematics with Applications, Volume 32, Issue 1, pages 117-135, 1996.

5. Smith D., Irby C., Kimball R., Verplank B. and Harslem E. (1982). Designing the Star User Interface. Byte, 7(4), 242-82.

6. Ungar, D. And Smith, R.B. (1987). Self: The Power of Simplicity. ACM SIGPLAN Notices 22 (12), December 1987.

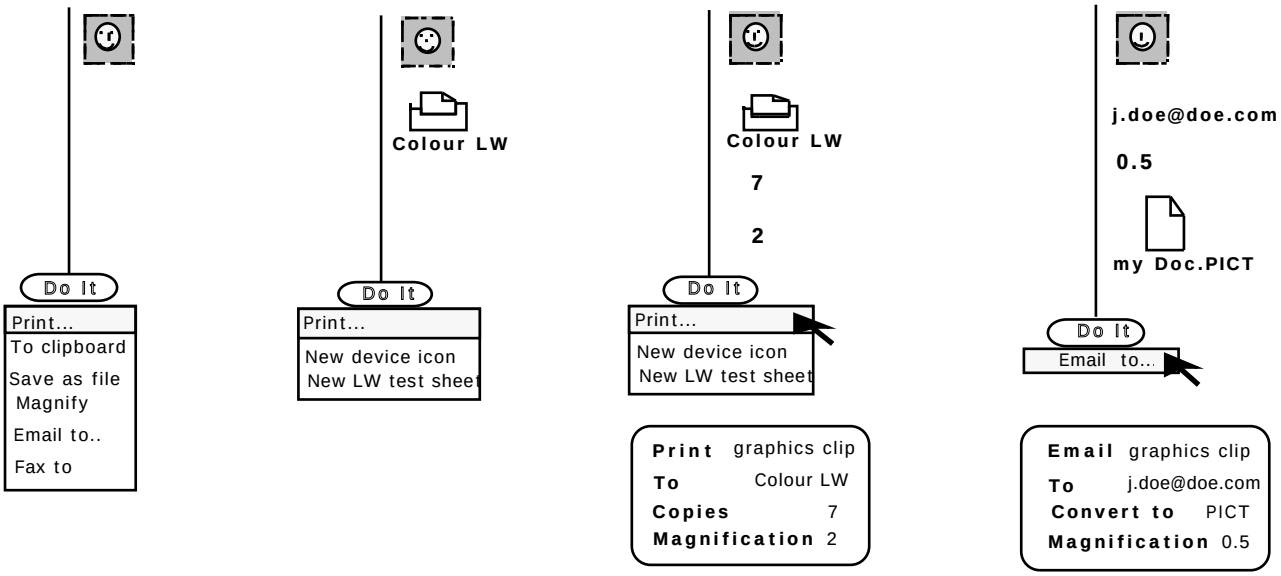

Figure 7: An example of n-dimensional Direct Combination using a Command Needle.