When is Direct Combination Useful?

Simon Holland

Computing Department, The Open University, Walton Hall,
Milton Keynes, MK7 6AA, United Kingdom
S.Holland@open.ac.uk

1 Introduction

When is Direct Combination useful? It would be valuable to be able to broadly characterise situations and domains in which DC is likely to be useful and those in which it is not. The question is open to empirical investigation, but pending such investigations, in this paper we apply theoretical arguments to seek preliminary insights. All of the arguments in this paper are of an a priori nature. They may offer a useful source of heuristic guidance to builders of prototype DC systems. The strategy of the paper is to seek to characterise domans and situations in which Direct Combination can usefully reduce search for users. and to try to identify the trade-offs involved in such reductions of search. We begin with some fairly straightforward observations.

2 **When is Direct Combination useful?**

2.1 DC may be useful even when applied only to some interactions, some of the time

DC does not have to be applicable all of the time in order to be useful. The principle of subsumption (TR2002/1) makes it easy for DC interactions to be used sporadically amongst a mixture of more conventional interactions. DC is not a straitjacket, it gives new freedoms: there is little or no cost to users to have it there for when needed. For designers of the system, of the other hand, comprehensive provision of DC requires effort, but DC can be applied iteratively and incrementally at low cost (Issues in n-dim DC).

2.2 *Sufficient variety of objects*

DC is unlikely to reduce search in domains with only one or two different kinds of object. However, the number of different kinds of objects in the domain does not have to be very large before the number of possible *pairs* becomes sufficiently large to have the potential to reduce the user's search space. This follows from simple arithmetic arguments. Given **n** differentiable object types in a domain, there are $\mathbf{n^2}$ different ordered pairs of object types. For example, in a domain with just five different object types, there are twenty five different object type ordered pairs. If we assume for the sake of argument that there are five operations uniquely associated with each ordered pair of object types, then specifying a pair of objects could reduce the search space for relevant commands from one hundred and twenty five operations down to a space of five operations. In practice, operations are unlikely to be so evenly distributed amongst pairs; not all operations will be unique to particular pairs; and the ordering of pairs will not always be important: however the general argument remains valid.

2.3 Sufficient variety of operations

DC is unlikely to reduce search in a domain with a very small number of distinct operations in total. However, note that whether or not a search space of a given size is large enough to pose search problems for a given user may depend on factors such as the user's familiarity with the domain, and the extent to which they are distracted at the time, for example in a 'minimal attention situation'. This makes it hard to characterise the size of the space of operations required for DC to be useful. At one hypothetical extreme, consider a domain with only a handful of operations to start with, but where the commands are unfamiliar, and where the user must focus on the environment as much as possible. If the user is able to select pairs of objects of interest in the environment by AC and so reduce even a small search space of applicable operations down to one or two operations, nevertheless given a sufficient degree of unfamiliarity and distraction, this might constitute a useful reduction in search. Hence, consideration of task, situation and user is necessary.

2.4 Distribution of operations among objects pairs

DC is unlikely to reduce search in a domain where all of the operations of interest are distributed amongst a very small proportion of the set of available object type pairs. Similarly, DC is unlikely to reduce search in a domain where all pairs of objects respond to more or less exactly the same set of operations. However, in many domains, opportunities can be found to better distribute operations amongst object pairs. Also, as noted previously, consideration should be given to task, situation and user.

2.4.1 Not too many operations associated with single pairs of objects

This is a special case of the previous consideration. Pairwise interaction may not be directly useful with pairs of objects that have too many operations associated with them. However, when two objects have a very large space of interactions, it often reflects the fact that the objects have many subparts, attributes or roles, whose potential interaction produces the multiplicity of operations. In such domains, if DC is applied *recursively* (Baroque DC) this typically cuts down the search space significantly, allowing DC to be useful again. However, a degree of navigation (among the subparts of an object) is typically required in recursive DC, so there are trade-offs on the amount of search required to use recursive DC. This may make recursive DC less attractive in some situations where minimal attention is at a premium, depending on the alternatives available

2.5 Sufficient visibility and findability of objects

For DC to be useful, objects of interest must be:

- visible to the user
- findable without undue search

This is partly a restatement of the principles of Direct Combination. The importance of the second stipulation is clear, since otherwise the problems of finding commands have merely been traded for the problems of finding objects of interest. Note that findability is not a function solely of the environment or display - it will depend on the user and the situation.

2.6 Cost of searching for commands is high
DC is worth considering when search for verbs is hard for reasons to do with the domain, user or situation. For example, when the user is distracted, or the task or objects involved are unfamiliar, or the user interface or environment is complex. The next item is a special case of this.

2.6.1 Unfamiliar situations, objects or tasks
In situations that are infrequently encountered, or unusual tasks, or in situations which involve unfamiliar objects, DC is likely to be helpful. There are some dynamic configuration situations which routinely involve the need for interaction between unfamiliar objects. DC is likely to be helpful in such domains.One converse of this consideration is that when users are highly familiar and skilled with existing ways of doing things, DC is less likely to be attractive to users.

2.7 Dynamically changing configurations
One class of problem is likely to arise more frequently as mobile and ubiquitous devices become more common. In situations with mobile users moving though resource rich environments, new devices and resources will continually be coming into range in novel combinations. The constantly changing *combinations* of resources have the potential to afford an abundance of unfamiliar and novel interactions. This will apply on a variety of temporal and spatial scales, as users walk around, drive around, enter particular buildings public spaces, or offices, collaborate in offices, visit homes and move to other rooms etc. As well as applying to individuals, this problem applies to groups. In many areas of life, groups of people, each with their own sets of varied equipment, information sources or other resources (e.g.laptops, projectors, printers, etc) must come together in ad-hoc groups to collaborate on tasks with non-routine elements. This can occurs in locations with their own combinations of resources which may be unfamiliar to participants. The consequent combinatorial explosion of interactions between resources presents problems for users and for user interface designers.
Ambient Combination is well adapted to cope with such situations by letting users counter the combinatorial explosions with their own combinatorial implosions: By explicitly selecting one or more object of interest, users can reduce the search space to manageable proportions.

2.8 *Tasks situations with an inherent pairwise focus*
DC is likely to be a particularly good candidate in domains or tasks where pairs of objects are the typical object of focus: for example in tasks where or domains where users need to take objects pair by pair and set up a particular kind of interaction between them.

2.9 Minimal Attention Situations and situations with limited feedback bandwidth
User tolerance for search tasks is greatly reduced in minimal attention situations and in situations where minimal feedback bandwidth is available. Given that other conditions are met, DC is a good candidate in such situations . DC is worth investigating wherever the user wishes or needs to focus on the environment, or must minimise the attention paid to feedback devices. DC is also promising where the user needs to minimize the need for

inputting information (for example when the user's favoured hand is required for other tasks) as it allows interactions to be specified as much as possible by simple pointing

2.10 Domains where rich, well abstracted object-oriented domain models alread y exist
In domains where a rich, well abstracted object-oriented domain model exists, a large proportion of the work required to create a DC broker has already been done. Hence in pragmatic terms, such domains are worth considering for the application of DC.

2.11 Data Translation
DC tends to be particularly appropriate when data must often be translated between diverse formats (indeed, this was the origin of DC). One class of special cases where this insight could be exploited is in any commercial and organisational situation where after a user has interacted with some system to provide information or express choices, information then needs to be transcribed into a standard form for the purposes of some other interaction. This is common where a customer interacts with a professional, who then needs to enter into a series of business to business interactions on their behalf, for example in the travel industry, legal professions, building professions etc.

3 Existing isolated uses of DC
In this section we consider some existing situations where DC is used. Dmix (H&O, 1999) is the first system where a version of DC was systematically and uniformly applied. However, existing isolated special cases of AC and DC can be found in many places. Here we note some examples.

3.1 Cut and Paste
Some applications routinely use a version of Direct Combination whenever cut and paste is used. The "special paste" in programs such as Microsoft Excel is an example of this.

3.2 Cash Tills
As noted in (Baroque) cash tills may be viewed as using an isolated example of Ambient Combination. Supermarket cash tills effectively use laser scanners to make an n-fold combination of items to be purchased, forming a resultant total cost . When a credit or debit card is swiped in the magnetic scanner, the card forms a pairwise combination with the collection of purchased items, and a single action is offered - purchase the goods. Some supermarkets allow another option: "cash back", where in addition to purchasing the goods, the customer may withdraw money.

3.3 Drag and Drop
VisualWorks is a Smalltalk programming environment, and the *Refactoring Browser* is a programming tool that allows various programming components to be listed and manipulated. Older versions of the browser required items or interest to be selected and then menus used to choose options. The refactoring Browser is designed so that as many drag and drop actions as possible between different pairs of items elicit appropriate actions. This is also true of recent Browsers in the Smalltalk programming environment

Squeak. Many Macintosh OS9 applications behave similarly to limited extents, for example Sherlock, Eudora the Finder etc.

4 The role of DC in future systems

We claim that AC/DC should play a role in any successful future framework for mobile, tangible and ubiquitous HCI. Some readers may be aware of a bygone era before dynamic menus allowed users to recognise and select existing path names from a list. During that epoch, user interfaces demanded that pathnames be recalled from memory and typed in character by character. Input boxes did not yet have built-in mini text-editors. Consequently, any subsequently discovered typing mistake at the beginning of a pathname meant deleting the whole string and starting over again via error-prone mental recall. At the time, and with the resources then available, the suggestion of creating dynamic menus on the fly and providing built in text editors for every input box might have seemed arcane, wasteful and extravagent. Today, these facilities are standard and pass unnoticed. Most users would be unable to name these facilities or to identify the principles on which they are based. However most of our user interfaces still demand that we specify most (though not all) of our commands in the rigid order order *noun verb*, and then use dialog box for providing any needed arguments. Sometimes this rigid ordering of assembling the command specification demands that the user recall a verb from a maze of possible verbs, even though the user could point to in the interface or environment the relevant nouns which would serve to constrain the verb search. Similarly, sometimes it is only on opening a modal dialog box and after going on to specify various arguments that one realises that one has the original noun or verb wrong - but the current rigid ordering of command assembly demands that this work be thrown away and the user start over from scratch. In line with the principle of subsumption, DC requires that the user to have the freedom to specify the command in any way they see fit, and to have immediate feedback while they do that. We hope that users will use DC in future without even noticing that it is there.
The first main cost of DC is a good, domain model, preferably object oriented and well factored. Increasingly, this is a basic requirement anyway. The second cost is the definition of the command space, but this need not hinder application, as this process is susceptible to an iterative, incremental approach.

References

Holland,S. Morse, D.R. & Gedenryd, H. (2002) Ambient Combination: a New User Interaction Principle for Mobile and Ubiquitous HCI. Submitted to Mobile HCI 2002 Fourth International Symposium on Human Computer Interaction with Mobile Devices, Pisa Italy.

Holland, S. and Oppenheim, D. (1999) Direct Combination. In *Proceedings of the ACM Conference on Human Factors and Computing Systems CHI 99*, Editors: Marion Williams, Mark Altom, Kate Ehrlich, William Newman, pp262-269. ACM Press/Addison Wesley, New York, ISBN: 0201485591.