Ambient Combination: Applying Direct Combination to Afford Spontaneity in Pervasive Computing

Simon Holland
Department of Computer Science, The Open University, Milton Keynes, UK.
s.holland@open.ac.uk

Abstract
In rapidly changing pervasive environments that are rich in resources, there will be numerous opportunities for end users to carry out tasks by causing *two or more devices or resources to interoperate together,* often in unplanned ways. Empirical studies have shown that users find such interoperation tasks hard to manage using current interaction techniques, particularly via devices with small, resource-poor user interfaces. In addition, existing software architectures make such situations difficult to address in a principled, scaleable way. We propose that a new user interaction principle called Direct Combination - and the user interaction techniques, software architecture, and analysis techniques that support it - have significant potential to reduce the time, attention and mental effort required by users to carry out such tasks in ubiquitous environments. When Direct Combination is applied to spontaneous interactions, the user interface can be made relatively simple, and the amount of search required by the user to specify desired actions can be greatly reduced. We present Direct Combination (DC) and the new interaction techniques it gives rise to for pervasive environments. We outline how Direct Combination can be further refined by the concept of *viewpoints,* which can be used by users and providers to personalise or focus the interactions available for different users, situations or roles. We outline how viewpoints might be automatically switched by context. We identify various issues that a Direct Combination system must address if it is to be usable, useful, maintainable, and deployable on a wide scale in rapidly changing pervasive environments. We argue the most recent architecture for Direct Combination, based on a software composition technique known as reflective composition (related to AOP and SOP) has particularly good properties for supporting the provision of viewpoints. The role-based architecture, which has been implemented in prototype, has good properties in rapidly changing pervasive environments for dealing with interactions that may cross-cut categorisation boundaries in ways unforseen by designers. The architecture is well-suited to highly distributed implementations. It is noted that the existing role-based analysis and design methodology OORAM is well suited to analysing environments in the way required for the application of Direct Combination. A simple way of formally modelling Direct Combination is outlined.

A shorter version of this paper was submitted to UBICOMP 2003. This longer version contains additional material on DC viewpoints.

1 Introduction

It has been noted that "As the vision of ubiquitous computing is fulfilled and computational services are spread throughout an environment, advances are needed to provide alternative interaction techniques" [28]. This paper presents a new interaction framework - *Direct Combination* - and explores the software support required to provide it. Over 30 years ago, Herbert Simon [16] suggested that the richer computing environments become, the more human attention becomes their most scarce resource. An important, open, general problem in ubiqitous computing is how ubiquitous environments can be made simple, powerful, flexible, usable and extendable for end-users**,** while requiring a minimum of attention to control. Pervasive environments have great potential to empower users, by offering access to diverse resources, 'anytime, anyplace, anywhere'. However, if they are difficult to use, or distract too much attention from tasks in hand, their use will be greatly impaired [4,13].

A characteristic feature of pervasive computing environments will be that normal use will involve multiple computing devices working in concert ([26,27] and multiple objects in the real and virtual worlds [6]. Another characteristic feature will be their dynamic nature. People and devices will move between different environments several times a day. Individual environments will have various people and devices flowing in and out of them continuously.

Today, configuring computers and peripherals to work together is a major problem of systems operation, and a primary chore for information systems departments. The interaction of these two factors (increasingly dynamic environments and the need for heterogeneous collections of resources to work collaboratively) will turn the problem of device configuration for pervasive technologies into a significant task. If configuration work is needed every time a user wants to use a different pair or set of devices in concert, this will present formidable disruptions to natural work flow. Even small individual configuration requirements would accumulate to a point where productive use is replaced by a constant tinkering to achieve interoperability.

Even if we assume that technical configuration is made completely invisible to the user, transparency of use would still remain a distant goal. Dynamicity and interoperability would still present significant usability problems, for example in determining what devices, actions and other options are available at any moment, and how to specify either of them. The problems aggravate when the possibilities are distributed over more than one device: how do you determine what capabilities arise from combining two or more devices, and how do you specify such combined actions and the entities that they involve?

Mobile devices tend to play important roles in ubiquitous environments, but most such devices tend to have resource-poor user interfaces [3,6] that are difficult and time-consuming to operate for any other than simple tasks. Mobile users in ubiquitous environments often have little attention to spare for user interfaces [24], since there are generally more pressing things to attend to, such as other people, physical tasks, and navigation of the physical environment [3,6]. It is generally agreed that new user interaction principles need to be devised to simplify the control of complex operations in ubiquitous environments, but there is no agreed solution. [2,4, 12,13,15].We propose

that a new user interaction principle, called Direct Combination [5,6], which may be viewed from one point of view as a generalisation of Direct Manipulation[6], can address a significant part of this problem. The principle is particularly useful whenever two or more objects (which may be physical or virtual - e.g. a remote display and a file) are involved. The principle affords systematic means for users to reduce their search space within a user interface when undertaking unfamiliar or complex tasks in a pervasive environment.

2 The Problem

This paper focuses on user interaction in general in ubiquitous environments, but particularly on the problem of s*pontaneous interaction* [1,17], a term we will now explain. In ubiquitous environments, end users must often control resources or access services in unfamiliar locations or ad-hoc situations [17,3]. Such interactions often require *two or more objects to be made to interoperate together* [1,3], for example one or more mobile devices , one or more fixed devices, and one or more virtual objects such as files or data streams. Spontaneous interaction, as characterised by the impromptu interoperation of two or more devices in ad-hoc circumstances has been found to be very difficult for end users to manage. [3, 4]. Because of the large number of possible combinations of object types, and the even higher number of potential interoperations, it is generally infeasible for user interface designers to provide quick and simple means for users to control arbitrary interoperations via any one device. This is the problem of *spontaneous interaction*.

Viewed from a user interaction perspective, the problem of spontaneous interaction is exacerbated in several ways. Firstly, pervasive environments will often be unplanned and fast changing, as both static and mobile elements change or are upgraded [3]. Secondly, end users will often be faced with the need for rapid interoperation and configuration of two or more unfamiliar devices/resources [1]. Thirdly there is an increasing need for users to improvise solutions without IT staff [4,1]. Most pervasive environments will tend to grow by accretion rather than by all-encompassing design, and such environments will not generally have a dedicated IT support team to assist end users. A third exacerbating factor is that current infrastructure and programming architectures poorly support spontaneous interaction [2].

We will note three approaches that may appear initially to solve the user interaction problem of spontaneous interaction, but on closer examination do not.

The first such proposed solution is the standardisation of protocols. While standardisation is of the essence, it cannot alone solve the problem of spontaneous interaction. In rich pervasive environments, rapid change will be the norm. Consequently, many devices will inevitably break protocols, introduce new ones, or mix partial or modified elements of diverse protocols [1,2]. In pervasive environments, for the forseeable future standardisation will always be a work in progress.

A second measure sometimes offered as a solution to the problem of spontaneous is the adoption of standards for universal remotes (e.g. the V2 standard) [1,17]. Again, this is a vital measure but it is inadequate to deal with the problem of spontaneous interaction, for reasons we will now explain. Universal remotes are well-suited to letting users deal with common tasks, or more complex tasks, provided the

user can spare sufficient time and attention. But when the user needs to get two devices co-configured, it is not much of an improvement for the user to be faced with two universal remotes, neither of which has explicit options for the desired interoperation. Because of the combinatorial explosion implicit in the need to manage *combinations of objects*, it is unreasonable to expect user interface designers to make provision for all useful combinations of objects, including future objects as yet unknown. A final approach sometimes proposed to deal with the problem of spontaneous interaction the use of context aware approaches. Again, this is a very useful technique, but not one able to dispose of the problem of spontaneous interaction. The problem is that to guess the user's intentions from context is in general an AI-complete problem or worse: what we might call a 'telepathy-complete' problem - consequently there is a tendency for such systems to guess wrongly [29], and furthermore such systems generally have no way to let the user apply leverage to incorrect guesses to satisfy actual intentions.

Thus, in summary, the problem of spontaneous interaction is hard because the mixture of objects in a given environment will tend to be mobile, diverse and changing in functionality. Users may frequently find new ad-hoc uses for combinations of object unforseen by the designers of any single resource [1,2,3]. The sheer number of possible combinations will overwhelm both user interface designers and end-users. Existing application-centric programming architectures make matters worse as they are badly suited to ubiquitous computing in general, and spontaneous interaction in particular, since it is impossible for the user interface designers of any one application to cater for all possible user-chosen interoperations [2]. Consequently, situations of this kind wil involve the user in time consuming navigation of the user interface, searching for needed functionality in a series of screens or menus, distracted from their primary task. From a human computer interaction point of view, we identify three key problems:

- to find a means for the user to specify complex actions as simply and directly as possible,
- to find new user interation techniques to support this,
- to find a programming architecture to support the required simplicity.

One common element in these problems can be identified: the problem of *search* when specifying and browsing for actions to effect intentions. The challenge is to find a way to reduce the users search space, without making heavy demands on their attention even when things go wrong, and to provide simple, uniform, parsimonious means of access to potentially vast arrays of functionality.

3 Proposed solution

We propose that a good theoretical basis for addressing all of these problems is the principle of *Direct Combination*
(DC) [5,6,22]. This principle was originally devised and applied to the desktop [5] , but it appears most aptly suited to ubiquitous environments [6]. We claim that when the Direct Combination principle is applied to ubiquitous environments, the user interface can be made more economical, and the amount of search in the user interface required by the user can be reduced. The principle of Direct Combination (DC) is perhaps best introduced by means of an scenario. See figure 1, which illustrates the scenario below. (Note: in this paper the term scenario is used in a loose, rather than a technical, sense.)

3.1 Scenario 1

Anne, on holiday in China, wants to display a file from her PDA on the public wall display in the shopping mall, for her companion, Bill. She searches the menus of her PDA to no avail. There is nothing relevant in any of the menus for the word processor, or in the various utilities for screen set up and communication that seems to control connecting with any kind of external display. Even if Anne managed to find a universal remote control program [7] for the wall display, things would not be much better, since nothing on the universal remote control for the display addresses this problem. Suddenly, Anne remembers that her new PDA is DC compliant. Anne will be able to narrow down what she wants by indicating some of the objects involved. Anne selects the particular **file** *on her PDA and then points her PDA at the* **wall display screen** *to select it (numerous technologies can support this kind of selection by pointing: such as optical, infrared or radio id-tags [rekimoto]). Now her PDA offers her the single action applicable to* this particular pair of objects*:*

Display the selected **file** on the **screen**

In the general case of Direct Combination, more than a single action would be offered, but for this particular pair *of objects, only one action happens to be applicable. Anne accepts the action and the wall display displays the file. Anne then uses a universal remote to adjust the display as she wants it. When she has finished showing the document to Bill, Anne uses the task monitor[1] on her PDA to find the task and break the connection.*

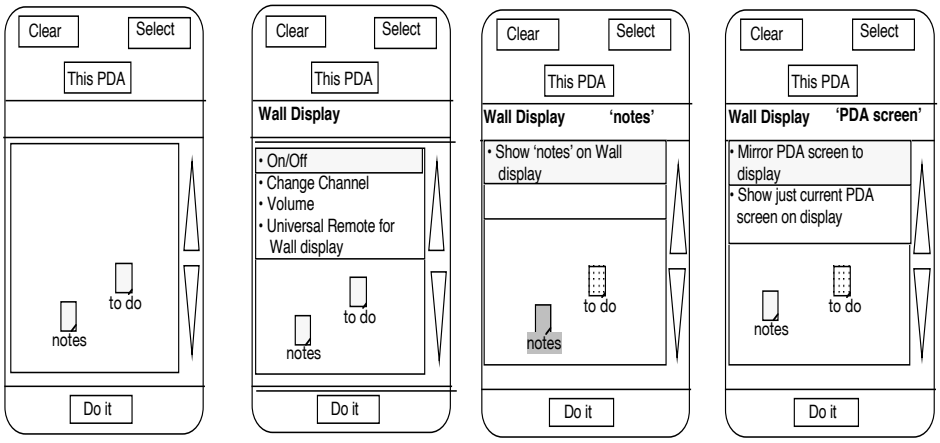

Figure 1. A possible user interface for the Direct Combination-enabled PDA featuring in the scenario described in sections 3 and 4. Figure 1a shows the case where no items are selected; Figure 1b shows the *wall display* alone selected (a *unary* interaction). (The terms unary, pairwise and n-fold are described in section 4.1.) Figure 1c shows *wall display* and *notes* selected (a *pairwise* interaction, discussed in section 4); Figure 1d shows the *wall display* and the *PDA screen* selected. The case where the *wall display* and *PDA itself* are selected (section 4) is not shown, but figures 1a-1d all show the button for selecting the PDA itself.

4 The principle of Direct Combination

A fundamental property of the Direct Combination framework, as illustrated by this scenario, is that if the user is allowed to indicate in advance *two or more interaction*

objects involved in an intended action, then, given the appropriate architecture, the system can use this information to narrow down the possible actions and offer commands relevant to that particular combination of objects. The user may then choose or refine actions from this targetted set of choices. Relevant options can be offered to the user instantly, so that if the desired action is absent, the user can browse further by selecting one or more different objects, or even s*ubparts* of objects (virtual or physical). For example, in the scenario, the pair of objects **{file, display}** elicited a single suggested action (figure 1c). But by selecting the *pda screen* instead of the *file*, yielding the pair **{pda screen, display}**, the options offered to the user might be refined as follows (figure 1d):

• *mirror the ongoing contents of the PDA screen, vs*

• *show the current contents of the screen on the display, ignoring subsequent contents.*

Yet again, if the pair of objects {**pda *itself,* display**} had been selected (Direct Combination enabled devices always provide a way to select themselves - see figures 1a-1d) , the options to be offered to the user would be likely to include the following:

• *get universal remote for display to run on PDA.*

Though in fact on a well designed PDA, this operation, among several others, might be elicited merely by the unary selection of the wall display (figure 1b).

4.1 Patterns of combination: nullary, unary, pairwise and n-fold combination

By selecting collections of parts and subparts of objects in this way, the user is able to focus more attention on the objects of interest, and less on navigating a user interface. The *pairwise* pattern of Direct Combination (DC), with exactly two interaction objects is particularly useful for constraining search with little effort, but *zero or more* objects are the general case. (The special cases of DC with: *exactly one object*, or *more than two* objects selected by the user, are known as the *unary* and *n-fold* cases respectively.) The unary case does little to constrain search, but its seamless inclusion as part of a uniform framework allows the easy mixing of a variety of different command styles. The ability to deal with any of these command patterns in quick succession makes it simple and fluid to browse for desired functionality. When Anne initially selects the wall display by itself to browse for relevant actions, this is an example of the *unary* interaction case of Direct Combination. When the file and the screen are selected this is a *pairwise* interaction. User interfaces built using the Direct Combination framework give the user the freedom to specify the parts of commands in any order desired, for example *noun noun* (to be followed later by a verb), as well as the more conventional *noun verb*, or even the simple *ver*b (the nullary case*),* any of which may be subsequently modified or followed by arguments. Crucially, at each stage, feedback is given on how choices made so far constrain further choices. One important aspect of these new interaction styles are that they are entirely compatible with pre-existing interaction styles (e.g. the predominant unary interaction *noun verb*, and the simple *verb*). [6] For a more formal statement of the principle of direct combination in general, see [5] and for a formal statement of the principle as applied to ubiquitous computing see [6]. In fact there is a concise term for the application of Direct Combination to Ubiquitous Computing, namely *Ambient Combination [6]* However, in this paper, in order to emphasise the continuity of Ambient Combination with other areas of application of Direct Combination [5], we will generally use the more general term Direct Combination.

5 The relationship of Direct Combination with search

Having introduced DC, let us now consider the relationship of Direct Combination with the abstract search space faced by a user in employing a user interface to specify an unfamiliar operation.Direct Combination analyses the space of user commands as constructed from elements such as *noun, verb* and *qualifier*. It identifies a strong constraint; imposed by most current user interfaces, that limits the order in which the elements may be assembled. Typically a user must specify a noun first, e.g. by selecting an object (or exceptionally a collection of nouns, all to be treated alike), followed by a verb, and finally a set of qualifiers. This restriction would be immaterial if it did not have profound implications on the *search strategies* afforded to the user when the necessary commands to carry out a task are not already known.

As a user incrementally specifies each element of a command pattern (e.g. by selecting an external entity, or selecting a virtual object or menu item), a good user interface should continually provide *feedback* on how the choices made so far afford or restrict further choices. However, where a selected entity affords numerous rich interactions (i.e. has many applicable operations), if the user has been forced to select a noun first and then inflexibly to follow it with a verb, the user may be left with a large space of possible operations (verbs) to search. This was precisely the problem faced by Anne initially in the scenario before she remembered that she could use Direct Combination. Where each individual resource is complex or unfamiliar, or the names or classifications of tasks are unknown to the user, searches of the user interface tend to disrupt flow and to impose unwelcome diversions on the user.

DC proposes that the restrictions on search outlined above should be dropped, and that users should be given the freedom to assemble commands patterns in any order they like. This freedom is of particular importance for spontaneous interaction, since by indicating the relevant objects of interest, the search space for relevant commands can be greatly constrained, as illustrated above. In sufficiently rich environments, application of the DC framework has great potential to reduce the user's search space. In the further work section we outline how formal models could be used to quantify the reduction search afforded.

Another possible benefit of Direct Combination is that in environments rich in objects of interest, (whether physical or virtual, local or remote - see section 6) users often know, or can recognize, or can be given means to select, objects they want to use (a screen, a wallet, a car, a room, a document, etc). However, operations, apart from the most commonly used operations, tend to be relatively more abstract, and harder for people to recall by the name used in a system. The freedom given by Direct Combination allows users to take advantage of *recognition* (easy) vs. *recall* (hard). We claim that, in pervasive environments, direct combination can:

- offer a cheap, fast way for users to browse and constrain the searc hspace of possible interactions,
- increase, rather than restricting user freedom,
- distribute the user interface around the environment,
- allow users to use recognition rather than recall,
- allow the environment to act as a systematic distributed source of affordances for action,

However, in order to realise, and empirically test these claims, a supporting software infrastructure is required. We have so far implemented three supporting infrastructures

in prototype. In the remainder of the paper we consider what properties such an infrastructure must have, how these properties can be realised, report on what we have learned from designing and building our prototypes, and outline the work that remains to be done.

6 Requirements for a Direct Combination Architecture

Without explicit architectural provision, in a ubiquitous environment there is generally no way of computing on the fly what interactions are possible within a collection of two or more selected objects (physical or virtual). Indeed this is not usually possible to determine even for virtual objects in a stable, isolated desktop system, far less for objects in a dynamic, distributed, ubiquitous environment. We require an architecture that can systematically provide this information instantly and which is maintainable and extensible, allowing new and unanticipated devices to be added in a robust manner. The principal requirements for any Direct Combination Architecture are outlined below.

- It should be relatively easy for analysts, designers, programmers and providers to cope with continual change, in the kinds of object, their functionality and interactions.
- The system should be maintainable, flexible and robust under rapid change.
- It should be cheap and easy to publish, refine, and amend suites of interaction (roles, role actions and role models - see below).
- It should be possible, quick and cheap in terms of attention for end-users to tune the distributed system on the fly to provide interactions appropriate to their current role or situations.
- Inter-object communication and computation demands should be minimised when information is highly distributed.

We claim that ways of supporting all of these requirements are demonstrated by our second generation architecture This is achieved by the use of techniques [11] similar to those used in role-based [41] and aspect oriented [32] programming to keep changing concerns separated, and by appropriate computational support for representing alternative (and possibly conflicting) viewpoints [11, 34,36]. As an illustration of these issues, we will contrast the first and second generation architectures used to implement prototype Direct Combination systems.

6.1 DC architectures: degrees of distribution

A simplified way to build a Direct Combination system is to have a single central server (the combination server) that compiles and coordinates information about the available entities. It involves the following principal functional units. The third element listed below forms part of client 'wands' (like the PDA of section 3) used by users to select items and effect operations, rather than forming part the server.

1. An environment database: this is where devices register their presence, and where object tags are converted into full representations of the entity in question.
2. A combiner: which computes combinations from reference to, or reflective descriptions of, the combined objects.
3. An option selector: essentially a simple user interafce component which presents the resulting options to the user, and collects a response and returns it.

4. A command performer: a unit responsible for carrying out the chosen option.

In most realistic settings, it is desirable to have a greater flexibility than such a central server can provide. In an infrastructure rich environment, typically in office environments, combination clients may profitably be thin, i.e. small, cheap, and depending on others for information, computation and effecting operations, whereas in roaming environments (i.e. outdoors) it would be desirable to have a 'thicker' combination 'wand' that may carry the object models and databases as well as perform the combination operation, and thereby does not rely on other devices for its function. An even more desirable option is the ability to draw on multiple distributed information sources as available. Under realistic circumstances, it is likely that the representation of the environment will be distributed, so that some sources will hold information about available entities, and others serve as sources for deeper information about the capabilities of entities that may appear, for example in manufacturers' databases. For these reasons a generalized, more freely distributable and reconfigurable architecture is desirable, having minimal ties between the various functions that must be performed and the specific machines or setups that perform them. To make the units easily redistributable, they should be separable and simple to link into different configurations, e.g. across different machines. It is also desirable to allow the combination operation to be performed in a distributed manner. Our second generation architecture can meet these requirements - but the first generation architecture simply used a central server, whose properties we will now outline

6.2 An inheritance-based architecture for Direct Combination

Our first architecture, which was inheritance-based, can generally be retrofitted, by relatively simple modification, to an existing central object-oriented representation (if one exists) of the classes (or individual objects, where available) in a pervasive environment. As noted in section 6.1, this kind of architecture, based on a central server, constitutes a severe restriction, but it allows several issues to be illustrated. This was also the architecture used in earlier work to apply DC to desktop computing [5]. The details of this scheme are described in [5] and it will only be briefly outlined here. The inheritance-based architecture exploits the economy of expression inherent in object-oriented inheritance hierarchies to make information about possible interactions between resources relatively easy for the designer to specify. Essentially, each pairwise interaction is encoded as a method belonging to one of the classes involved in the interaction, or one of its superclasses. Double dispatch techniques are used to implement the detail of interactions between classes. Complete sets of interactions relevant to a given collection of objects can be gathered by reflection techniques applied to the available methods (which can be named following a scheme to make this easy). Code describing interactions between classes, defining what interactions are possible and how they are to be carried out, is added at points as high as possible in existing abstraction hierarchies. This information may be overridden in leaf classes where necessary. One significant problem of this kind of approach is that with any kind of inheritance-based hierarchy, it can be hard to keep the system well-factored when new kinds of resources are being continually introduced whose functionality crosscuts previous class boundaries. This kind of rapid change tends to be a characteristic of pervasive environments. When using an inheritance-based solution in such situations, there tends to be a need for continual refactoring, and the build-up of code redundancy. Hence maintainability under rapid change is compromised.

6.3 A role-based architecture for Direct Combination

In order to address the above problem, we developed a second generation architecture for Direct Combination using a software composition technique known as Reflective Composition, also known as universal composition [11,8]. This allowed us to represent domain objects and their properties more flexibly. Reflective Composition is closely related to Aspect-Oriented Programming and other related approaches to software composition [32, 31,30], but it is also closely related to role-based programming [41] and to the prototype/delegation system used in Self [35]. These in turn are related to earlier work on representing perspectives in object-oriented systems [36,34]. Our second generation architecture provides the flexibility and extensibility needed to address the open-ended nature of the requirements outlined above. It is well suited to coping with situations of rapid change and has good properties for enabling the architectural building blocks to be flexibly distributed across different machine configurations in highly distributed pervasive environments in an uncomplicated and robust manner. Explaining the details of the role-based architecture is beyond the scope of this paper, but we will briefly outline its key features, and summarize the implications of the resulting emergent properties. One of the key properties of the architecture turns out to be the fact that that it can represent directly what are known as *role-based* descriptions of entities, as we will now explain.

6.3.1 Roles, role models and role actions

As in any computing system, an architecture that is to support Direct Combination generally needs to draw on some kind of representation or model of the capabilities of the various entities of interest. The most popular modelling techniques for computing systems in general are currently object-oriented, in which each entity is modelled as belonging to some *class* of object. In established *role-based* design methodologies such as OORAM [10], objects are still used, but by contrast, instead of trying to create classes that exhaustively model all of the relevant properties of the various different kinds of objects, design is typically focused on just one scenario, or even one interaction, at a time. This contrast in approach leads to consequences which turn out to be important for our purposes: the relevant issues are best illustrated by a simple example. Consider a situation involving support of audio in ubiquitous environments. Instead of having to model this by designing a detailed class hierarchy for the many kinds of objects potentially involved (microphones, telephones, loudspeakers, mixers, CD players, tape recorders, etc) a role-based designer might begin by identifying just one or two relevant roles that different objects might be called on play in some particular scenario of interest. Consider the following scenario (again the term is used loosely).

6.3.1.1 Scenario 2

> A driver on a long car journey might wish she could use her car radio to listen to a spoken version of a text document located on the computer in her office.

Two relevant roles here might be that digital documents containing text might be considered as *vocalizable*, given a suitable program or device that can act a *vocaliser*. Two other relevant roles might be that diverse devices can play the role of *sound sources*, and many devices include, or can play the role of a *loudspeaker*. Roles can be represented in diagrams very similar to object interaction diagrams (figure 2), but with vertical lines representing *roles* rather than specific objects or classes.

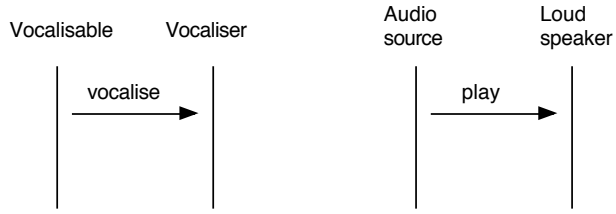

Figure 2. This figure shows two *role interaction diagrams*, used to support scenario 2 in which a driver on a long car journey wishes to use her car radio to listen to a spoken version of a text document located on the computer in her office. Because of the simplicity of the examples, these diagrams may also be taken to represent the resulting *role models*. This terminology and approach is of the kind used in the OORAM role-based design methodology [10].

Each role typically encapsulates a very narrow set of actions among related roles, or sometimes, as in the present example, a single action. Each interaction diagram becomes the basis of a *Role Model*. This consists effectively of a set of named *roles* and named interactions (*role actions*) between pairs of roles, with any information about the temporal ordering of actions discarded. As it happens, such information was not present in the first place in our simple examples. Now something similar to inheritance-based classes can be assembled role-by-role, by assigning roles to specific objects, following what is known in OORAM as the 'hat stand model' [10] (figure 3). In effect, objects need not be defined by their class, but can be defined by aggregating roles (such as *sound source*, *vocaliser* etc). In an implementation directly reflecting this design, such as provided by the second generation architecture, each role will be implemented directly, and will act something like a mixin [40] or small sliver of a conventional abstract class. Under such a scheme, by assigning roles to objects or classes, it is possible to describe new combinations of functionality without having to refactor classes. Furthermore, it becomes possible for objects to change their roles dynamically at run-time.

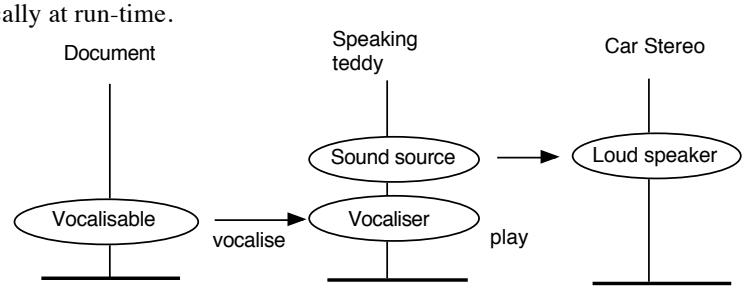

Figure 3. Using the 'hat stand model' [10] roles may be assigned to 'classes' or individual objects as a way of describing their capabilities, and showing possible collaborations with objects that play related roles. The second generation architecture allows roles to be added and subtracted from objects at run-time to reflect the changing roles of objects.

Role-based analysis does not classify objects immutably. Objects may play different roles at different times, and an object may typically support many roles simultaneously. Each role typically encapsulates a very narrow set of actions, or even a single action. Actions typically relate an object to one or more other objects playing complementary roles, e.g. *vocalisable* and *vocaliser*. Roles may also aggregate and modify other roles.

Crucially, reflective composition provides *direct support for role based programming*. As we will see below this turns out to be one of the keys to supporting Direct Combination in a flexible manner.

6.3.2 How the role-based architecture implements DC

In order to further illuminate the issues involved in an architecture supporting Direct Combination, it is useful to consider the steps involved a direct combination interaction, and see what the supporting archicture has to do and how it is done. There are three main steps involved.

Step 1. When the user uses a wand to selects an object, a 'class' id or object id is retrieved (figure 4)

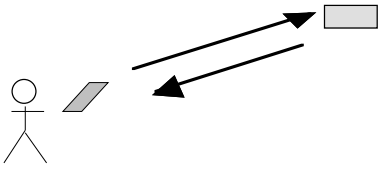

Figure 4. The user selects an object using a wand (or by other means) and a 'class' or object id is retreived.

Step 2. This tag may be used to retrieve a list of the names of roles currently associated with that object (which may be held on a manufacturers class server, or in a more distributed fashion e.g. associated with a particular building, room, individual device, or personal role).

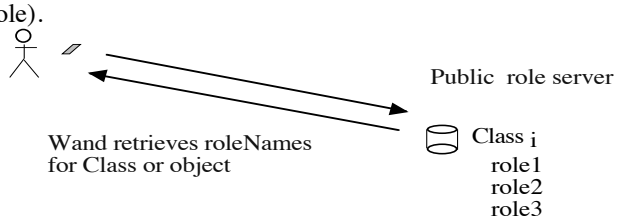

Figure 5. The tag is used to retrieve the names of roles currently associated with that object. A variety of servers might be used for the purpose, as described in the text.

Step 3. Every time a new object is selected, the wand checks all of the names of roles associated with that object against all of the roles associated with every other object for potential matches between roles as defined by role models in a role model server (see figure 6, and figures 2 and 3).

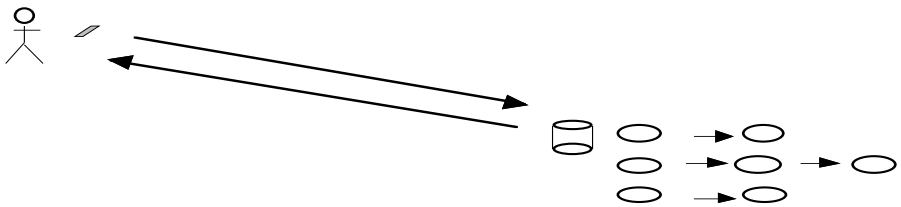

Figure 6. The wand retrieves the role models relevant to the roles played by the selected objects from a role model server.

Let us see how these steps would be applied to realise scenario 2 in the case of the driver who wishes to have her remotely held text document read to her in an ill-

equipped hire car. The driver in our example might use her videophone as a direct combination wand to selecting two relevant objects initially as follows:

- make a video call from her car to a camera in her house to remotely select the talking teddy in her bedroom,
- use the videophone to connect to her computer at home and select the relevant document.

In this way, the driver can directly combine the teddy with the document. The wand will retrieve the 'class' types of the two objects and use the manufacturers server (or a house server, or other relevant server, as described in the next section - or the objects themselves) to retrieve the currently relevant roles. In this case the text document supports the role *vocalisable* and the talking teddy supports the role *vocaliser*. A role model server is then examined to find all of the role models that match the roles. In this case (figure 2) the *vocalise something vocalisable* role model will match. Consequently, the user will be offered at least the option:

- Use talking teddy to vocalise document

In a second step, the driver can now cross the *talking teddy* which may also play the role of a *sound source,* with the car stereo, which supports a *loudspeaker* role. Consequently the role model *play sound source through loudspeaker* will be matched. In the simplest case, the role model will specify a way of invoking the relevant operation directly from the objects in question. In other cases , the operation might be executed by the wand or by a relevant server. In yet further cases, a manufacturers database might need to be consulted to find out how to effect the role action. Note that given certain assumptions, the interaction could have been carried out in a single step, though this is beyond the scope of the present paper.

6.3.3 Advantages of role-based architecture for applying DC to pervasive environments

We are now in a position to see why direct support for role-based programming confers several advantages for our purposes.

First of all, a role-based architecture makes it relatively easy to *analyse* and *represent* the changing properties of resources found in a pervasive environment, by means outlined above: the user-accessible interactions of most objects can be fully defined merely by specifying and updating a passive, easily transported list of roles. Resources can be represented simply by listing the roles that they play. New role models can be identified by domain analysts and implemented as needed on a role-by-role basis. Once the interactions of individual roles have been identified, the interactions that an entity can enter into will follow automatically from its constituent roles. To analyse an entity for DC purposes, the domain analyst need only identify the roles that that a particular entity can play. This is relatively quick and simple to analyse and code. For each set of roles that interact in a role model (e.g. *vocaliser and vocalisable*), the analyst must specify the following: a relevant operation (e.g. *vocalise*), and a human-readable description of the operation for user interface purposes. For easy interoperability, the executable operation associated with the role action should ideally be expressed in some universal protocol such as the future V2 standard [7].

Secondly, a role-based approach is highly suitable for *distributed* computation, since in order to fully specify and reason about the functionality of an entity, it is only necessary to have the relevant list of roles.

Thirdly, computing the possible interactions between two entities at run time is computationally simple and cheap. For each role in turn, corresponding role

combinants is sought in the other entity. Every time a reacting pair is found, the descriptions of the corresponding operations are collected. These are then filtered to avoid duplication, and the options presented to the user.

A fourth, perhaps unexpected advantage is the strong support given by this approach to perspective-based computing [8], which allows users to leverage the power of Direct Combination by making use of *viewpoints*, as explained in the next section.

7 The concept of Viewpoints

When applying Direct Combination to a given environment, there is clearly a room for a great deal of disagreement in deciding precisely what set of operations should most aptly apply to any collection of selected objects (i.e. what roles they should play, and what role models should apply). We will now outline a mechanism called the *viewpoints* mechanism which allows such potential disagreements, far from being treated as a drawback, to be pro-actively exploited as a major advantage. Differing viewpoints are commonplace among, for example, different social, national, professional and interest groups, and among those playing different temporary roles as part of work or leisure. For example a fire-fighter, a policeman, a paramedic, an office worker, a tourist, and a child might identify the same object in different ways at different times, depending on the current personal role(s) they were playing: they might view the object as a potential fire-risk, a possible security breach, a health risk, a tourist attraction, an information source, a toy, etc. Informally, we would say that they have different *viewpoints*. This informal notion of viewpoints can be very useful to the individual user, since it provides a convenient way of filtering and focusing options in very rich environments. We will now describe how, in a more technical sense, the term *viewpoint*s also applies to an advanced feature of the Direct Combination framework that mirrors the informal sense just outlined. To give some simple examples of how direct combination viewpoints work:

Scenario 7.1
"How do I reprogram this central heating system in the spare room/switch off the power to this wing of the house/ switch off the water to the second floor? Better set the Readers Digest Home Maintenance view on the wand."

Scenario 7.2
" How do I, a traveller, buy a ticket at Ulan Bator railway station/ change money/ find accomodation? Better set the Rough Guide view on my wand."

Scenario 7.3
"How do I fix my car? I hope the subscription for the automobile association viewpoint is up to date"

Scenario 7.4
"How do I get this door open? I wish I had a security guard's wand (and the biometric profile to make it work)."

7.2 Viewpoints in Direct Combination

The term *viewpoints* in Direct Combination is used to refer to a mechanism by which users can adjust the perspective from which they view an environment, allowing different functionality to be elicited for different users, purposes, or situations. The

term *viewpoint* also refers to any one such view. Viewpoints allow both designers and end users to manage the diversity of potentially conflicting requirements of what interactions should be associated with given sets of object. Such a mechanism offers the possibility for users to subscribe to one or more viewpoints, which they may choose to vary at home, at work, or when focused on particular tasks or roles, etc.

7.3 Requirements for the viewpoint mechanism to be useful

For the support of diverse viewpoints within Direct Combination, as informally illustrated above, to be practically useful, viewpoints must be simple for end-users to select at the user interface. Furthermore it must be easy for users to define and amend personal viewpoints. Also the framework must make it widely worth peoples' while to maintain and publish the necessary substrate of information -in the same way that the simplicity and accessibility of the World Wide Web makes widespread authoring of webpages worthwhile. We will now briefly consider how these various requirements may be met, using the second generation architecture. Note that within the context of the role-based approach outlined above, a viewpoint is a way to permit different users at different times to see different

- sets of mappings from classes (and individual objects) to roles, and
- different sets of role models,

7.4 Viewpoints at the user interface

The application of viewpoints at the user interface is relatively straightforward for users, as the above scenarios suggested. The user need only specify a named viewpoint to filter, focus or afford interactions as desired (e.g. tourism, consumer association, electricians' union etc). It should be clear that some viewpoints might be available only to those with the right professional credentials, passwords, paid up subscriptions or biometric data - in other words the user interface can easily police a range of security models and business models. Prima facie, this suggests that the framework offers a lot of scope for commercial and other organisations to use viewpoints to cheaply leverage their specialised databases and services, making it worthwhile for useful viewpoints of this kind to be professionally defined and maintained.

7.5 The perspectives mechanism

Viewpoints within Direct Combination can be implemented without using the second generation architecture, but the architecture makes viewpoints relatively straighforward to define and maintain. This is due to a mechanism, that is part of Reflective Composition (the composition technique used) known as *perspectives*, which is itself closely related to the informal notion of viewpoints outlined above. Perspectives [34,36] relate closely to Subject-oriented programming [42]. The mechanism of perspectives allows an object to be regarded from several different perspectives (in the informal sense), *as though* it were different objects with different implementations. Conceptually, this elaborates the role-based approach of decomposing an object into different subcomponents implementing different functionalities, in that it allows even potentially *conflicting* functionalities or purposes to be reconciled within a single object. (There may or may not be sharing between viewpoints, similarly there may or may not be conflicts between viewpoints.) Viewpoints are useful to draw on multiple sources for direct combination information, for example to support multiple competing standards for similar functionality. Viewpoints also useful for implementing different security clearances, sharing of resources, etc. Most importantly, for our purposes, they are useful for realizing a central usability objective noted above, namely providing

viewpoints which offer differentiated access, for example based on personal preferences, current personal role or situation. (Note that the term *personal role* is used in a different, though related sense to the term *role* as used so far.)

7.6 Who defines viewpoints?

Given that a mechanism is available for defining viewpoints, a key question is who should define viewpoints? Our claim is that the framework is simple enough for the answer to be everyone. In other words, the framework is sufficiently simple to allow viewpoints to be defined by much the same population of people as those who publish webpages: in other words, manufacturers, special interest groups, professional groups, news groups, magazines, service providers, teachers, private individuals and school children. Of course, in some cases, as already noted, subscriptions, passwords, specialised wands, or the correct biometric data may be required for individuals to gain access to certain viewpoints. We further claim that the level of granularity of viewpoints can be easily adjusted at the level of individual objects and primitive behaviours. We will defend these claims through the hypothetical case study of the *burp-vocalising teddy*, which follows.

7.7 Creating, modifying, and maintaining interactions: a hypothetical case study

In this section we will use the hypothetical case study of the burp-vocalising teddy (sic) to demonstrate that Direct Combination allows easy publishing, refinement, amendation and basic security protection of interactions.

In our hypothetical case study, it is discovered by six year olds that a popular brand of talking teddy has a little known feature - when this feature is accessed, the bear's next utterance will be made in the highly popular (with six year olds) style of vocalisation where (if performed by a person) every syllable is effected not by simple exhalation from the lungs, but by a burp from the stomach. We will refer to this style of vocalisation as *burp-vocalisation* [39]. The older siblings of the six-year olds are intrigued, and further discover that feature can be also be accessed programmatically via the teddy's wireless net connection using an undocumented procedure call. They post the relevant information on a news group. Seeing this news item, their yet-older siblings studying at university decide that it would be amusing to have the teddy burp-vocalise compiler error messages in the university computer labs, and for convenience decide to make the new feature Direct Combination accessible. In order to effect this, the students simply create two new roles *burp-vocalizable* and *burp-vocalizer* (figure 7) and a single new role-action *burp-vocalize,* together with simple meta-data including a template for describing the role-action to users in menus of available operations. Together, these items constitute a new role model.

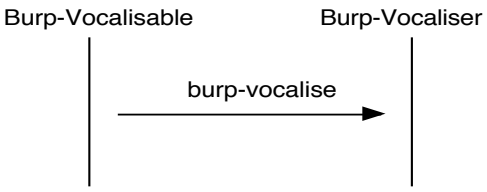

Figure 7. The *burp-vocalise* role model - part of an illustration of the simplicity of creating, modifying, and maintaining interactions (see text).

The new role model is then published on various role servers to which the students have access (belonging to the computing lab, the university, student organisations, etc) and submitted to relevant national and international organisations (the manufacturer, Viz, Slashdot, New Musical Express, etc). On accessible local class servers (or if none are accessible, on individual local teddies), the *burp-vocalize* role is then added to the class descriptions of the teddies. The students also send a request to the manufacturer of the teddy to amend their own class server to add the *burp-vocalize* role to the official teddy class description. Finally, the student amend their favourite viewpoints so that any objects with the role *vocalisable* (such as text) automatically acquire the role *burp-vocalize*.

The teddies manufacturer accedes to the students request, and adds the newly defined role to the class definition for the teddy. For reasons of respecting cultural diversity, this is done at the national, rather than international level.

At breakfast the next day, children use their direct combination wands to make the teddy burp-vocalise the ingredients printed on cornflakes packets. Appalled, some parents immediately ask the household god (i.e. house server) to block this role and role model within the house. Other parents simply ask parents' organisations to which they belong to censor this role model on the house servers that subscribe to the organisation's *role-nannying* service.

This scenario assumes that, on purchase, an appliance may be constrained by the purchaser via a wand, or instruction book etc, to route all combinations in the first instance through the household god (i.e. in the manner of a WWW proxy server). This allows various services in the house to be security protected or managed by a role-nanny as previously indicated.

8 Implemented prototypes

So far we have implemented one prototype version of the inheritance based architecture described in section 6.2, and two prototype version of the role-based architecture described in section 6.3 9 (both in Smalltalk, one using the Squeak dialect, and one using the Visualwork dialect). As indicated in section 6.1, either architecture can in principle be applied in at least three different ways, as follows.

- **Thin client:** thin, small cheap direct combination clients supported by servers that know about objects and their capabilities.
- **Thick client** thicker combination clients that carry their own object models, databases and the functionality required to request, and in some cases carry out, operations between other objects.
- **Fully distributed:** more realistic pervasive environments of the future, where the representation of the environment is more distributed.

The Squeak version of the role-base architecture has been implemented both as thick and thin client, but we have not yet attempted to implement a fully distributed version. Tagging of objects in the environment was achieved using barcodes and USB hand-held bar-code scanners, including a 30-foot range scanner. The prototypes reported here were designed for purposes of proof of concept and for refining the DC mechanism. Since our emphasis is on usability issues rather than on the technology of pervasive computing, technically sophisticated solutions were not ends in themselves. Our initial thin client solution uses HTTP for the transport, and encodes the reflective

descriptions in XML. The latter is in general a poor choice for pervasive solutions, in that mere text and particularly the verbose format of XML consume more bandwidth resources than necessary. However, the ability to use something as simple as XML to encode the reflective descriptions does serve well to illustrate our point: This illustrates that a very low degree of technical sophistication is required of a distributed, reflection-based combination architecture. The use of Smalltalk, unlike for example Java and C++, means that powerful reflective capabilities are part and parcel of the language as well as the environment and tools, which makes the reflective solution very straightforward to implement. The extreme portability of Squeak, and its platform-independence down to the bit level, means that all parts of the system (including the server functionality) can be directly transferred to iPaq-class devices with no porting effort. The server uses the core of the Comanche web server framework , hooking our object model directly into the HTTP request mechanism.

9 Related work

Rekimoto has previously noted the importance of inter-device operations for pervasive computing [Rekimoto, 1997 #982]. However, Pick-and-Drop [Rekimoto, 1997 #982;, 1998 #984], the InfoStick [Kohtake, 1999 #983] and DataTiles [Rekimoto, 2001 #991] do not have the open-endedness that Direct Combination allows, they only deal with a restricted range of built-in inter-device operations. Consisely put, Ambient Combination (i.e. the application of Direct Combination to pervasive computing) is to Pick-and-Drop what Direct Combination is to Drag-and-Drop on the desktop. The iRoom project [Fox, 2000 #994] involves a media-augmented room and multi-device interactions. However, they focus on the technological infrastructure and do not aim to achieve transparency of use, or to push the technology into the background. For example, their room is operated via a PC-based web browser where you control the technology by clicking on links on specially prepared webpages, with links such as *Turn on all projectors* and *Switch Smartboard 1 to laptop drop*. Abowd [,1999 #990] pointed to the need for separation of concerns and software engineering techniques for pervasive architectures, and described how the CyberDesk project encountered problems by not being able to properly disentangle context gathering from context-triggered behavior. Previous work on architectures for ad-hoc settings include the Proem project, which aimed to provide infrastructure for building special-purpose ad-hoc collaboration applications [Kortuem, 2000 #986]. Our aim is to support *strongly* ad-hoc conditions; specifically, if there is specially collaboration software that has been prepared in advance, then the collaboration is arguably less ad-hoc already. The Aura software architecture [Sousa, 2002 #995] also addresses dynamically changing resources, but it does this by explicitly representing the user's task, and it centers on the technical challenges, in effect on the level of infrastructure, whereas our concern is on the level of user interaction. DataTiles [6] is an attractive tangible computing system using tagged transparent tiles placed on a flat display. Interactions between any two tiles are effected by physical adjacency, or by a pen gesture. The kind of interaction is determined by the tile types, although a pen gesture may be used to make this continuous or discrete in nature. DataTiles, like many tangible computing system, has the potential to be given greater flexibility and power, without loss of elegance, by applying Direct Combination in a variety of ways. For example: DC options could be

visibly offered when two tiles were placed together or connected by pen. Alternatively, DC interactions could be invoked via a single additional pen gesture to invoke Direct Combination between two tiles. Or as a third alternative, a special magic lens tile could be laid on top of inter-tile borders, to reveal and allow the selection of available interactions. From one perspective, DC may be viewed as a novel way of exploiting a *relational* approach [10] to Tangible User Interface (TUI) design, *systematically* allowing the selection of multiple objects to determine dynamically bindings between objects and computational operations. However, although DC is a generally useful technique for TUIs, DC works beyond TUIs to allow mixed interactions between virtual and real objects [6]. DC relates strongly to Direct Manipulation, and as discussed in [3], parts of DC may be variously viewed as generalisations or specialisations of DM.

10 Limitations

Viewpoints have not yet been implemented. N-fold combination has not yet been implemented. Several other limitations are noted in the section on implemented prototypes.

11 Other issues and further work

In this section, topics for further work, and topics discussed elsewhere are identified.

For an early discussion of possible extensions to DC provide context-aware interactions, or context-aware switching of viewpoints see [6].

The OORAM methodology [10] for role-based design appears well suited for analysing pervasive environments for the application of Direct Combination. We plan to test an adapted process using a role-based variant of the CRC game [10] drawing on the OORAM analysis techniques [10].

The notion of entity *selection* in Direct Combination appears capable of considerable elaboration. For example, particular user interfaces could harness physical pointing, tangible selection, remote pointing, and various kinds of selection by speech.

The basic interaction styles of DC could and should be extended and tested to accommodate various general issues [13] in ubiquitous user interaction such as resource discovery, feedback, monitoring of tasks in progress, cancelling, and remote interactions.

Although it is hard to test claims about a general principle, in the case of particular systems in which the principle is applied to different interaction styles, environments, users, tasks and situations, we plan to empirically investigate the usability, and capacity of DC interactions to reduce: search space, attention, mental load.

Formal models could be developed tested and refined to model how DC affects search in various environments and circumstances. Given an enumeration of the discrete options offered to a user by a given interface in a given environment and circumstances, a tree-like representation of the space of user commands available can be constructed. In practice, many factors will affect the users' search space, but, given

appropriate assumptions about the user, task, and situation, such a tree could be used as the basis of a formal model of the user's search space. Such models could be used to estimate the extent to which Direct Combination might reduce the users' search space compared with conventional command patterns, under various assumptions. Such models, if successful would allow designers to predict the extent to which Direct Combination is likely to be useful in different circumstances.

12 Conclusions

In rapidly changing pervasive environments that are rich in resources, there will be numerous opportunities for end users to carry out tasks by causing *two or more devices or resources to interoperate together,* often in unplanned ways. Empirical studies have shown that users find such interoperation tasks hard to manage using current interaction techniques. Existing software architectures make such situations difficult to address in principled, scaleable ways. We have demonstrated that a good theoretical basis for addressing an essential aspect of all of these problems is the new user interaction principle of *Direct Combination*. We have outlined the new user interaction techniques, software architecture, and analysis techniques - together constituting a new framework - required to support the principle. We have argued that the framework has significant potential to reduce the time, attention and mental effort required by users to carry out such tasks in ubiquitous environments. When Direct Combination is applied, the user interface can be made relatively simple, and the amount of search required by the user to specify desired actions can be greatly reduced. Poetically speaking, Direct Combination may be said to allow users to combat the combinatorial explosion of functionality in pervasive environments by surgically precise combinatorial implosions. These 'implosions' are effected by multiple object selection. We have outlined how Direct Combination can be further refined by the concept of *viewpoints,* which can be used by users and providers to personalise or focus the interactions available for different users, situations or roles. We have identified various issues that a Direct Combination architecture must address if it is to be usable, useful, maintainable, and deployable on a wide scale in rapidly changing pervasive environments. We have argued that argue the second generation role-based architecture for Direct Combination, based on the software composition technique of Reflective Composition has particularly good properties for addressing these issues, and for the provision of viewpoints.

Acknowledgements

This paper is dedicated to the late Henrik Gedenryd, with thanks for his insights into the relationship between Reflective Composition and Ambient Combination. Thanks to attendees at the 2002 UBICOMP Spontaneous Interaction workshop for useful discussions. Thanks to Alan Dix and Jen Rode for useful comments. Thanks to Lindsay Court for giving me an opportunity to test for clarity in front of an attentive audience. Thanks to Rob Griffiths, Mike Newton Robin Laney, Leonor Barroca and Janet Van der Linden for listening patiently and demanding yet more clarity. Thanks to Michael J. Holland for deftly honed burp vocalisation expertise and vigorous demonstrations on

demand. Thanks to Marian Petre & Bashar Nuseibeh for gently encouraging the shipping of product in timely fashion. Thanks to David Morse for many useful discussions and for helping to establish the conditions in which this work could be developed.

References

1. Newman, M.W., J.Z. Sedivy, W.K. Edwards, T. Smith, K. Marcelo, C.M. Neuwirth, J.I. Hong, and S. Izadi. (2002) Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments. In Proceedings of *Designing Interactive Systems (DIS2002)*, June 2, 2002. London, UK.
2. Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J.B., Zukowski, D. (2000) Challenges: an application model for pervasive computing. In Proceedings of the sixth annual international conference on Mobile computing and networking MOBICOM 2000 pp 266-274. ACM Press.
3. Keith Edwards and Rebecca Grinter. (2001) At Home with Ubiquitous Computing: Seven Challenges, Ubiquitous Computing 2001, Atlanta, GA, September 2001.
4. Kristoffersent, S. and Ljungberg, F., (2000) Representing Modalities in Mobile Computing: A Model of IT-use in Mobile Settings. White Papers, Norwegian Computing Center, Oslo.
5. 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.
6. Holland, S., Morse, D.R., Gedenryd, H. ((2002) Direct Combination: a New User Interaction Principle for Mobile and Ubiquitous HCI. In Proceedings Mobile HCI 2002, LNCS, Springer Verlag.
7. Zimmermann, G. Vanderheiden, G, Gilman, A. (2002) Prototype Implementations for a Universal Remote Console Specification. CHI 2002 Extended Abstracts , pp 510 - 511. ISBN:1-58113-454-1
8. Gedenryd, H., Holland,S. Morse, D.R. (2002) Meeting the software engineering challenges of interacting with dynamic and ad-hoc computing environments. TR 2002/08, Computing Dept, Open University, Milton Keynes, MK76AA, UK.
9. G. Gottlob, M. Schrefl, and B. Rock. (1996) Extending object-oriented systems with roles. ACM Transactions on Information Systems, 14(3):268-296, 1996.
10. Reenskaug, T, Wold P. and Lenhe O.A. (2001) "Working with Objects, The OOram Software Engineering Method". By Trygve Reenskaug, with Per Wold and Odd Arild Lehne. ISBN 1-884777-10-4.
11. Gedenryd, H. (2002) Beyond Inheritance, Aspects and Roles: a Unified Scheme for Object and Program Composition. TR2002/09, Department of Computing, The Open University, Milton Keynes, MK76AA, UK.
12. Weiser, M. “The Computer For the 21st-Century,” Scientific American, vol. 265, pp. 66-75, 1991.
13. Victoria Bellotti, Maribeth Back, W. Keith Edwards, Rebecca E. Grinter, Austin Henderson, Cristina Lopes (2002) Ubiquity: Making sense of sensing systems Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves April 2002.
14. Ramsay, M and Nielsen,J. (2000). WAP Usability Report, Nielsen Norman Group, Fremont, California.
15. Winograd, T. (2002) Towards a Human-Centered Interaction Architecture. In Carroll, J.M. (Ed.) (2002) Human-Computer Interaction in the New Millennium Addison Wesley.

16. Simon, H.A. (1971). Designing organizations for an information-rich world. In M. Greenberger (Ed.), Computers, communications, and the public interest (pp. 37-72). Baltimore: MD: The Johns Hopkins Press.
17. Holland,S., Gedenryd,H., and Morse,D. (2002) Applying Direct Combination to afford spontaneity in Pervasive Computing. UBICOMP 2002 Workshop on Supporting Spontaneous Interaction in Ubiquitous Computing Settings, Gothenburg, Sweden.
18. Holland, S, Morse, D.R. (2002) Audio GPS: Spatial Audio Navigation in a minimal attention interface. Personal and Ubiquitous Computing, Vol 6, 4, Pages 253-259.
19. Morse, D.R., H. Gedenryd, and S. Holland, (2002) A simple, technology-neutral lingua franca for location systems, applied to combined indoor-outdoor navigation, TR 2002/10, Computing Department, The Open University, UK.
20. Monk, A., Wright, P., Haber, J. & Davenport, L. (1993). *Improving your human-computer interface: A practical approach*. New York: Prentice-Hall.01
21. Brewster, S.A., Wright, P.C. and Edwards, A.D.N. The design and evaluation of an auditory-enhanced scrollbar. In Proceedings of ACM CHI'94 (Boston, MA) ACM Press, Addison-Wesley, 1994, pp. 173-179.
23. Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. and Carey, T. (1994) *Human Computer Interaction*, Addison Wesley, New York.
24. 1. Pascoe, J. Ryan, N. and Morse, D. "Using While Moving: HCI Issues in Fieldwork Environments," ACM Transactions on Computer Human Interaction, vol. 7, pp. 417-437, 2000.
25. J. Pascoe, N. Ryan, and D. Morse, "Issues in developing context-aware computing," in *First International Symposium on Handheld and Ubiquitous Computing (HUC 99)*, vol. 1707, *Lecture Notes in Computer Science*, H.-W. Gellersen, Ed. Karlsruhe, Germany: Springer-Verlag, 1999, pp. 208-221.
26. Rekimoto, J., Pick and Drop: A Direct Manipulation technique for multiple computer environments, In proceeedings of UIST '97 p.31-39.
27. Kohtake, N., Rekimoto, J. and Anzai, Y. InfoStick: an interaction device for inter-appliance computing, Handheld and Ubiquitous Computing (HUC '99) 1999.
28 Abowd, G.D. Software Engineering Issues for Ubiquitous Computing. in Proceedings of the 21st International Conference on Software Engineering (ICSE-99).
29. Erickson, Thomas (2002) Some Problems with the Notion of Context-Aware Computing. Communications of the ACM Feb 2002/Vol 45 No.2 pp102-104.
30. Czarnecki, K., Aspect-Oriented Decomposition and Composition, in Generative Programming:Methods, Techniques, and Applications, K. Czarnecki and U. Eisenecker, Editors. 1999, Addison-Wesley: Reading, MA.
31. Elrad, T., R.E. Filman, and A. Bader, (Eds.), Special issue on Aspect-Oriented Programming. Communications of the ACM, 2001.
32 Kiczales, G., et al. Aspect-Oriented Programming. in Proceedings of the European Conference on Object-Oriented Programming ECOOP'97. 1997: Springer Verlag.
33 Kiczales, G., J.d. Riviéres, and D.G. Bobrow, The Art of the Metaobject Protocol. 1991,Cambridge, MA: MIT Press
34 Kay, A., The Early History of Smalltalk, in History of Programming Languages–II, T.J. Bergin and R.G. Gibson, Editors. 1996, ACM Press: New York. p. 511-578.
35 Ungar, D. and R.B. Smith, Self: the power of simplicity. Lisp and Symbolic Computation: An International Journal, 1991. 4(3): p. 45-55.
36 Bobrow, D. and T. Winograd, An overview of KRL, a knowledge representation language. Cognitive Science, 1977. 1(1): p. 3-46.
37. Rekimoto, J., Ulmer B. and Oba, H. DataTiles: A modular platform for mixed physical and graphical interactions. In Proceedings of CHI 2001. 2001 p 269-276.
38 Ullmer, B., Ishii, H., (2000) Emerging Frameworks for Tangible User Interfaces. IBM Systems Journal 39 (3&4), 915-931.

39. Indiana University School of Medicine (2001), Burp. Sound Medicine,Indiana University School of Medicine, 1110 W. Michigan St., LO 401. www.soundmedicine.iu.edu/archive/2001/mystery/burp.html. Accessed 22 April 2003.
40. Gilad Bracha and William Cook. (1990) Mixin-based inheritance. In Proc. of the Joint ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications and the European Conference on Object-Oriented Programming, October 1990.
41. Gottlob, G, Schrefl, M. and Röck, B. (1996) Extending Object-Oriented Systems with Roles. TOIS 14(3): 268-296 (1996)
42. William Harrison, Harold Ossher. (1993) Subject-Oriented Programming (A Critique of Pure Objects). Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM, Washington, DC, September 1993, pp. 411-428