Places to stay on the move

Software architectures for mobile user interfaces

Alan Dix

aQtive limited
Birmingham Research Park
Vincent Drive
Birmingham, B15 2SQ, UK
and Lancaster University
alan@hcibook.com

Devina Ramduny, Tom Rodden, Nigel Davies.

Department of Computing
Lancaster University
Lancaster, LA1 4YR, UK.
ramduny@comp.lancs.ac.uk
tam@comp.lancs.ac.uk
nigel@comp.lancs.ac.uk

http://www.hiraeth.com/alan/topics/mobile/

Abstract

Architectural design has an important effect on usability, most notably on temporal properties. This paper investigates software architecture options for mobile user-interfaces, in particular those for collaborative systems. One of the new features of mobile systems as compared with fixed networks is the connection point to the physical network, the point of presence (PoP), which forms an additional location for code and data. This allows architectures that bring computation closer to the users hence reducing feedback and feedthrough delays. A consequence of using PoPs is that code and data have to be mobile within the network leading to potential security problems.

Keywords: mobile computing, collaborative work, CSCW, software architecture, active networks, client-server

1 Introduction

At first sight, it seems that software architectures are about the internals of system design and not a necessary concern for the user interface. However, internal details have a way of showing themselves at the surface.

One aspect of this is the changing view that users have of systems due to the merging of computing and communication systems and the maturing of distributed computing techniques. With the growing prominence of an almost universally accessible information infrastructure, we are increasingly seeing our interaction focus on the communication infrastructure rather than the devices that access it.

Nowhere is this more evident than in the World Wide Web, which represents the most dramatic example of this shift in how we view our interaction with computer systems. The popular acceptance of massively interconnected computer systems has seen computer and communication systems being seamlessly interwoven within the everyday life of the general public. Interaction is now routinely through the web (rather than the machine used to access it) and our everyday experience of this interaction makes inherent assumptions about the architecture of the infrastructure.

As the opportunity to access the underlying infrastructure increases with the development of mobile devices, the link to the underlying architecture is likely to become even more significant and the need to consider the architecture of the infrastructure underpinning user interaction will grow.

Arguably, even with the web, users see a virtual architecture – is that ".co.jp" domain really served from Japan or from some other place? However, consideration of physical architecture becomes unavoidable when we look at temporal aspects. Again, the obvious delays on the web due to network latency and bandwidth have raised the profile of these issues. For interactive networked applications, the need for rapid *feedback* – seeing the results of one's own actions – is essential, leading to various forms of client-server architecture. For collaborative applications *feedthrough* – seeing the effect of other users' actions – is equally important, but even more intractable due to the physical distance between the users.

The importance of feedback and feedthrough can be demonstrated by considering the way in which a selection button behaves in most interfaces. When a user selects a button this is reflected through a change in the appearance of that button. This feedback is central to the interaction. But when we consider shared interfaces a new situation arises in that the effects of the actions need to be made available to more than one user. This implies that in addition to providing feedback by changing the state of the selection button, the application also needs to feedthrough the effects of the action by altering the state of the selection button on each users' interfaces. This feedthrough of effect is important in providing an awareness of the actions of others and in allowing users to co-ordinate their actions. Delays in the feedthrough of action will obviously make co-ordination of activities more difficult.

Mobility brings yet more issues with unreliable and often low-bandwidth communications, small screens, restricted input devices. In some previous work, we have looked at the general nature of mobility and context awareness [11]. In this paper, we will analyse the architectural options available in developing mobile interfaces, especially collaborative systems. Our particular concern lies in the dynamic nature of the infrastructure required to support mobile and context sensitive applications and the impact on the relationship between interaction and architecture.

We will begin by reviewing the current state of software architectures in Human Computer Interaction (section 2) and discuss how these are realised in single-user, multi-user and web-based applications over fixed networks (section 3). In section 4, we will start the main work of this paper by looking at the additional issues which emerge for mobile single-user architectures and in particular, the role of the *Point of Presence* (PoP) as an additional potential site for computation. Sections 5 and 6 are the heart of this work as we will describe how it is possible to support different architectures, by making use of PoPs, in collaborative mobile applications and how these can be dynamically reconfigured to optimise feedback, feedthrough and resource usage. Finally, in section 7, we will look at some of the issues this raises for code mobility and security.

This paper does not present a particular system or single solution, instead it is based on our own experience in producing fixed and mobile collaborative applications and that of others. However, the analytic framework takes us beyond existing systems design and is intended to inform and guide our own and other researchers' ongoing development in the area.

2 Software Architectures in HCI

Concerns about the architecture of interactive systems are not new to HCI. Experience has shown that the internal structure of a system has a dramatic effect on its external behaviour [15]. Because of this, user interface architecture has been a concern for many years and has seen the development of models such as Seeheim, MVC, PAC and Arch/Slinky as well as the whole stream of UIMS systems [7, 24, 27, 29]. The problem involved in developing appropriate interactive behaviour has focused on uncovering an architecture that offers the most appropriate set of dynamic behaviours for the activity being supported.

In collaborative systems research, the general interest in software architectures has continued with CSCW architectures and toolkits including Rendezvous, MEAD, Suite and Groupkit [3, 10, 16, 18]. These collaborative systems almost always imply some form of networked solution, and increasingly single user systems also involve access to central information giving rise to a whole industry in client-sever and n-tier applications.

The emergence of distributed architectures to support interaction across geographically disparate communities of users has seen a series of debates about the nature of these architectures and their impact on interaction. This debate has tended to centre on the propagation of the effects of the actions of users to others involved in the activity being supported. Essentially, the critical issue here is the means by which feedthrough of actions and the interface can best be supported. A core element in this discussion has been the tension between the responsive nature of replicated architectures, which allow rapid feedback to be provided locally and the need for some centralised component to make users aware of the action of others by providing feedthrough of actions.

The majority of these arrangements tend to assume 'control' over the entire system, with bespoke software running at the users' own workstations and at various central servers. The implicit assumption in many of these systems is that the machines are connected to a single local area network and the properties of this network are stable. However, two developments have recently challenged this assumption and hence the whole basis for network-based user interfaces.

The first is the World Wide Web, which has promoted alternative architectural arrangements for applications. A Web 'application' may include code running at a web-server (via CGI scripts or other server side technology), web pages displayed on browsers of many different kinds, and applets or similar downloaded code [12]. In previous work we and others have investigated the ways in which CSCW architectures can be married to the web infrastructure [2, 6, 26, 28].

The second development is the massive growth in mobile communications and mobile computing. Although the end points here may be well understood (but in the case of small mobile devices difficult to design for), the network itself is much more dynamic than even the Internet, with limited bandwidth, temporary disconnection and an ever changing network topology. The design of appropriate user interfaces for this environment is becoming an increasingly important topic [4, 9, 13, 21, 22 , 23, 25, 30]. The challenge here is to support interactive arrangements that are sensitive to the properties of limited portable devices and to provide appropriate access to the underlying distributed infrastructure.

The need to consider the dynamic nature of this infrastructure and its effects on interaction places new demands on the software architecture and the overall role of the architecture. Essentially, software architecture is about 'what goes where'. In stationary networks, the 'where's tend to be fairly obvious and are normally characterised as either clients or servers. Even this can lead to a rich set of architectural alternatives. In mobile systems however, the changing network topology suggests a much richer set of alternative possibilities.

These new and emerging arrangements provides the focus for this paper. We wish to investigate what possible arrangements exist for mobile and dynamic infrastructures and the implications for future networks. In order to achieve this, we will first revisit the topology of interactive single-user and collaborative systems on fixed networks. We will then proceed to see how this changes when we consider mobility.

3 Architectures for static networks

3.1 Single-user architectures

In the traditional single-user arrangement for networked interaction, a single user client usually interacts with a single server (Figure 1). The user interface sits at the client-end while the data is on the server. The choice between opting for a thin or thick client affects the performance of the application and depends on many factors, not least the volume and the rate of change of central information – which is itself a weak form of collaboration.

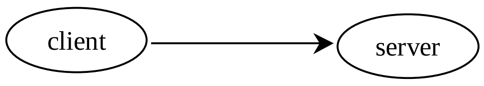

Figure 1. Single-user interaction

The separation of concerns inherent to this arrangement is exploited by a number of windowing systems. For example, when a button is pressed, its local display effect is managed by the client. The action of pressing the button sends a message to the server which causes some effect in the application.

3.2 Multi-user architectures

In multi-user collaborative systems, the arrangement becomes more complex as we may have one of more clients handling the demands of a community of users. These clients may in turn interact with one of more servers via some inter-process communication mechanism. For the sake of simplicity, we will assume that interaction is taking place through a single server as shown in figure 2.

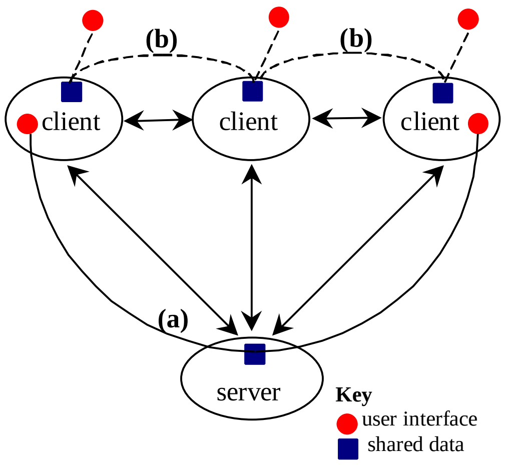

Figure 2. Collaborative networked system

This simplification is due to presentation reasons, however, the single central server in figure 2 and in subsequent figures is effectively a shorthand for a potentially replicated /distributed server system. The important point is that the centralised server functionality is associated with fixed network locations as opposed to the clients, which may potentially roam.

The two traditional architectures that emerge from the above are **(a)** a centralised architecture and **(b)** a peer-peer architecture.

In a centralised architecture, each user's client manages the screen layout and accepts inputs. The server holds the shared data and receives all users input events. Output events are broadcast by routing them through local client programs to all the users. As all the data is held centrally, access management, concurrency control and data consistency is simplified.

When a user selects an option on the screen in the centralised arrangement, a message is first sent to the server and this causes a change in the application state on the server. The server then sends a message to all other displays to reflect this change. Obviously, this arrangement introduces a bottleneck as the server becomes responsible for updating all of the displays in the system.

A server failure due to a network breakdown or a delayed feedback, especially in a distributed setting, can also give rise to a deadlock state. Client-server architectures have been adopted in conferencing systems, such as MMConf [8] and shared window systems like shared X [17]. Centralised architectures can easily support WYSIWIS or presentation level sharing as the server broadcast the output to all the clients. View level sharing can also be supported as demonstrated by the Rendezvous system [18].

In a replicated or peer-peer architecture, a separate copy (or replica) of the application runs on each workstation, which executes the application code and send output to the local user. To ensure synchronisation among the different replicas, input from each workstation is sent to each replica. The copies then communicate with each other to maintain data and interface consistency.

Each replica handles its own screen management and user's feedback locally and is also responsible for updating the screen in response to any changes in application data from other replicas. When a user selects an option button under this arrangement the local application state changes immediately. This selected option is broadcast to all other applications and they in turn amend their state to reflect this selection. This independence provides significant performance advantages but also introduces considerable complexity. Essentially, there is no control over the other displays and multiple users can simultaneously select different and often conflicting selections.

The replicated approach offers the advantages of a centralised architecture with the added benefits of performance as the output of a workstation is produced by the local workstation itself. Because the clients can be managed locally, alternate views are supported and it is relatively easy to provide end-user interface tailoring [3]. However, the major difficulties with replicated architectures lie with synchronising and maintaining data consistency. For example, if a user deletes a selected object in a WYSIWIS group drawing program while another user is changing the selection to a different object, inconsistent interfaces can result due to events arriving in a different order at each workstation.

The focus of a typical conceptual architecture for collaborative systems (as shown in Figure 2) is on the user interface and the shared data components. In the peer-peer case, both reside on the client whereas in the centralised case, only the user interface resides on the client.

In a wireless environment, the location of each component is critical as it has a direct impact on the usability of the system. This is partly due to the impact on feedback – centralised systems rely on a complete round-trip for feedback thus leading to interface delays. Various forms of peer-peer or local caching alleviate this [28] but at the cost of more complex code. Traditionally this is mainly a barrier to the construction and maintenance of code, but in a mobile setting, the size and computational ability of the devices is limited. Client-based peer-peer architectures may be impossible due to these limitations. Even if it is possible to implement such distributed solutions, the extra computational time and storage requirements will reduce the available resources for other user-interface functionalities.

3.3 WWW-based collaboration

The web has had a significant impact in increasing the prominence of the underlying infrastructure but in many ways the impact on the architecture has been less dramatic. However, applet security mechanisms make true peer-peer architectures difficult except via some form of 'post office' server as found in various chat programs [31, 32] or a client-end plug-in [19].

The use of applets on the web also opens up the possibility that code may be executing on a client, but its permanent home is on a server. The interactions between applet-based code mobility and the movement of data via caching have been used to classify the different modes of web-based architectures [28].

The movement of code and functionality inherent within applets also starts to alter our consideration of architectures. If architectures are about deciding where the location of component are, then we need to consider what happens when the supporting mechanisms allow them to move easily?

4 Mobile Architectures

On the web, although applet code may be mobile, it usually runs on static computers. With truly mobile computing, the devices themselves move. In a previous paper we investigated various kinds of 'mobility' [11], but for the purposes of this paper our focus is on the physical mobility of devices. However, we will find that like the web, this also tends to require computational mobility.

Computational mobility means that computation may start at one network site, but then move and continue to execute at another network site. Mobility may involve:

- *code mobility*: very useful as demonstrated by the increasing use of Java applets.
- *control mobility*: moving a thread of control from one network point to another which then returns to its originating point, for example Remote Procedure Call (RPC).
- *data mobility*: data is exchanged over the network in the form of parameters.
- *link mobility*: the endpoint of one network connection is sent to another network connection to allow the receiving party to connect to it.

Computation is not limited to code. Rather, computation is the combination of the code and the context of its execution. When code is moved from one network point to another, the current state of the execution is lost and the connections that the computation had at its original site no longer exist. The code can only execute at the remote end if state and connectivity is re-established at the receiving site. Therefore control must move through some form of dynamic binding and data must also migrate in order to preserve the state of the computation. As network links form part of the state information they must move as well.

Although there are many complicated issues for dynamic code mobility, there is ample work in this area in the distributed systems community. We will instead look at what impact computational mobility has on potential architectures. In particular, the location of computation is crucial in determining the effectiveness of mobile applications and computational mobility provides us with a more flexible choice of code location.

4.1 Points of Presence

To understand the need to consider location and mobility within the architecture, let us introduce the notion of a *Point of Presence* (PoP). Again, let us begin with the case of a single-user application over a mobile network. In the static case this simply involved the user's client machine and the central server. In the mobile case, in addition to the client and server, there is some form of *Point of Presence* (PoP) where the client machine has its connection to the physical network (Figure. 3).

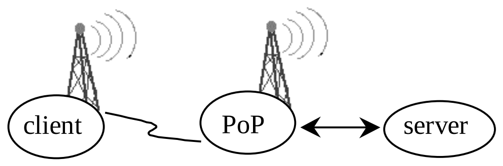

Figure 3. Points of Presence

In the case of cellular mobile-phone-based connectivity, the PoP may be the local cell's base station accessed via radio, in the case of a small hand-held PDA, this may be a desktop computer accessed by an infra-red port or a fixed cradle.

The PoP is not necessarily the very first point of contact; the PoP must also satisfy some functionality criteria. For example, the PoP may be the closest point with sufficient computational ability, the closest point within a corporate network, or the closest point with suitable security. Because of this, there may be different PoPs for different applications, just as in client–server applications, there may be servers for different databases or different web-servers on the web.

The crucial point is that the location of the PoP is governed by the location of the users relative to some geographical point or network topology, rather than a predetermined static position – the PoP is where it is because you are where you are.

The PoP effectively forms a third place where computation and data may reside. For mobile devices, a PoP has both better network connectivity and potentially greater computational power than a hand-held or body-adorning device. In addition, the PoP is closer to the user and will thus be able to engage in a faster pace of interaction than a server-based interaction. It is therefore a natural place for part of a user-interface.

There are already several examples of using computation at a PoP in web applications, for instance, the use of proxies to transform normal web pages into small-screen-friendly versions for hand-held computers [14]. In fact, in this work, the location of the translation server is fixed, it is 'mobility aware' as compared with the web server itself. Note however, that in principle this translation could be performed on a suitably fast client, but because of computational limitations of the clients (and to reduce network bandwidth) the computation is at a third-party location.

Another example is Active Caches [5] which allow servlet-style applications to run on a proxy server in order to improve caching algorithms and to perform some server-side computation on the proxy, and hence closer to the user. The current work is aimed primarily at fixed user-proxy configurations, but the mechanisms would translate well into more mobile situations.

Other areas where code on web proxies have been applied include their use as computational foci for collaborative application and also some early work carried out at aQtive on local (client end) proxies for uniform local–remote web-based interfaces [1].

The standard protocols for web proxies combined with the web's global user base make PoP options both easier and more essential than in traditional LAN-based office systems. This accounts for the growing interest in proxy-based computation. Mobile systems pose additional technical challenges, but improving technology for code mobility and economic necessity will make PoP-based approaches more common.

For example, there are already a variety of web-based diary/contact management services. If, as the providers hope, these substitute for client-end applications, there will be increasing economic pressure to dynamically move data and processing towards the users. However, such applications will need to work closely with PDAs if they are to become the primary corporate or personal contact management systems. This will inevitably mean some form of PoP approach.

4.2 Collaborative Points of Presence

In considering Points of Presence let us now turn to collaborative systems (Figure 4). Again we may have many users' client computers (hand-held devices, wearables etc.) and one (or more) central servers. However, for each client we now have a PoP and these clients become present in the network through these Points of Presence. Some clients may be close enough to have a shared PoP others may have completely different PoPs. Furthermore, very close clients may even be able to communicate directly rather than through the fixed network, for example, Palm Pilots that can communicate via iRDA.

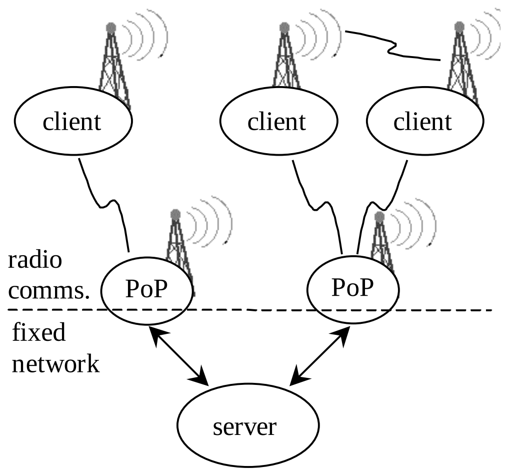

Figure 4. Mobile collaborative network

In this arrangement, there are three possible places where data and programs may sit: client, server and PoP; and also a variety of communication paths. In the following section, we will investigate the possibilities for collaborative architectures based on these locations.

5 Options for collaboration

In order to clarify the different cases we wish to consider we will draw upon a running example. Imagine two people, Alison and Brian, using some form of digital paper. The target application is a shared drawing tool – as Alison draws on her pad with a stylus, marks appear on her own digital paper and also (with some feedthrough delay) on Brian's pad. The question then is what forms of software architecture may best support the interaction needed for our shared digital paper.

5.1 Static network PoPs

The first pair of architectures are those when we effectively ignore the mobile nature of the network and treat it exactly like the static network. For fast and reliable networks, such as the roving radio-based ethernets used within offices this arrangement makes considerable sense.

Two sensible application configurations are possible under this arrangement (Figure 5).

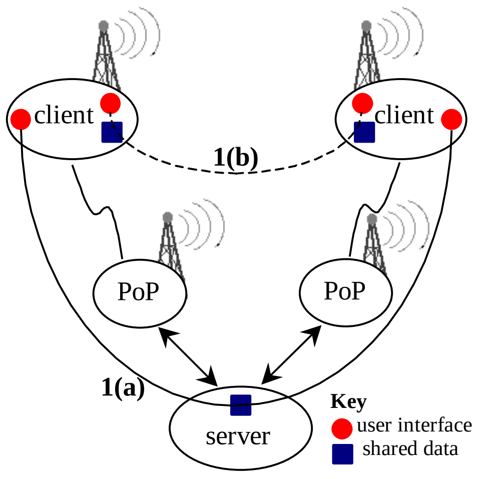

Figure 5. The static network arrangement

1(a) *server-based centralised architecture*

Shared data is held at the server end.

1(b) *peer-peer architecture*

Communication is from the client down to the network and back to the other client, just as two users communicate with each other via a mobile phone.

The PoP has no computational role in either case, being merely a router or post office passing on communication. This is effectively the same as **1(a)** and **1(b)** in figure 2 except that the network has both mobile and fixed links. The issues are similar to fixed networks, but the delays involved may be longer. In case **1(a)** Alison may experience some delay in getting feedback for her actions (disconcerting on digital paper). In case **1(b)** both Alison and Brian's digital pads contain all the shared data and they have to communicate to maintain a consistent replicated state.

5.2 Power in the PoP

Now consider a similar case, but where the client computation is spread between itself and the PoP.

This may be useful if the drawing pads supporting the digital paper have only a small amount of on-board memory or limited computational power.

In figure 6, we see versions of **1(a)** and **1(b)** where the PoP has a greater role. Two potential configurations emerge.

2(a) *centralised data, PoP as intermediary*

The data sits at a central server, but the PoP takes an active role as a proxy client/server. The PoP runs part of the application and also communicates with the client for I/O.

2(b) *decentralised replicated solution*

Replicated data sits at the PoP (and possibly part of the application) – the PoP acts as a virtual server. Although the data resides at the PoP, it is likely to be there only temporarily, so long as there are local users(the exception being ubiquitous data such as telephone directories.)

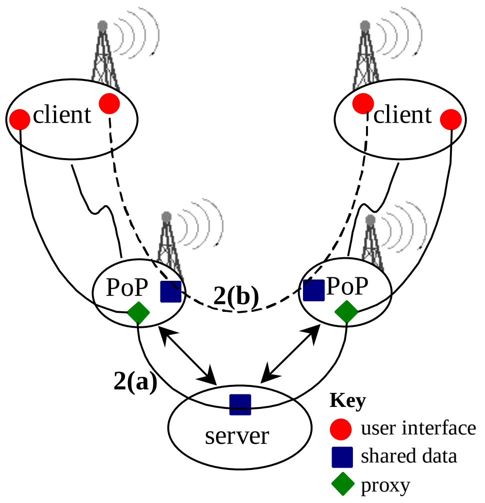

Figure 6. The use of computational PoPs

One example of **2(a)** would occur if the digital pads were configured as X servers (this in fact is the arrangement used in the early versions of the PARC ubicom environments). The shared drawing program could then consist of X clients at each PoP with each accessing a shared server. As in case **1(a)**, Alison will experience feedback delays, unless there is additional caching.

Although we described case **2(b)** as a virtual server, it will have a far greater pace of feedback than a centralised solution – Alison's stylus strokes only have to register on the replica at the local PoP before being echoed back to her pad.

5.3 Moving PoPs together

Finally, there are two options that are only possible when devices are physically close together. For example, imagine that Alison and Brian have brought their digital pads with them to a meeting. As they talk they start to sketch on their pads. If any of the previous architectures are used, the feedthrough of Alison's actions on Brian's pad will experience full network delays, little different than as if they were hundreds of miles apart.

However, being so close they might reasonably expect virtually instantaneous response. Two arrangements exist that might enable the level of response anticipated by Alison and Brian.

3(a) *distributed centralised solution*

The server gives some of the functionality to the PoP to allow the clients to communicate with the PoP – the PoP therefore assumes the role of server.

3(b) *peer-peer communication*

Direct communication using local communication (e.g. infrared). The shared data is at the client end.

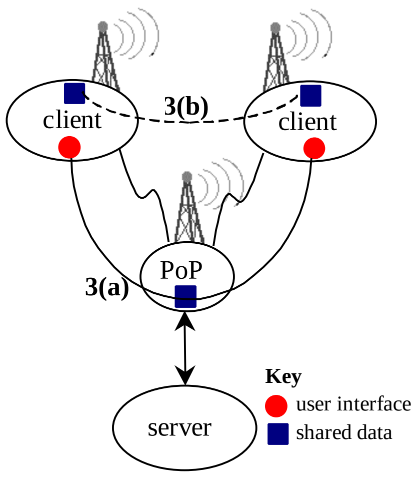

Figure 7. Colocated PoPs

3(a) and **3(b)** are similar to **1(a)** and **1(b)** in figure 2 however, the communication takes place entirely in a mobile environment, thus ensuring faster feedback and feedthrough.

Of the two options **3(b)** is very simple in that it is only a local network version of **1(b)**, the only difference being a greater pace of feedthrough for Brian and the problems of hand over within the network.

Although in some ways **3(a)** is the same as **1(a)** it has the crucial difference in that the machine that acts as the server for the shared application, the PoP, is configured dynamically rather than at a fixed location. For example, if Alison and Brian are in a reactive meeting room, the PoP may be in a device embedded within the wall that senses the presence of the two pads and downloads relevant software from a remote server. Thereafter Alison will experience almost instant feedback (just a roundtrip to the wall and back) and Brian equally fast feedthrough. In addition, being a centralised application it has all the advantages of software simplicity that are associated with server-based solutions.

6 Dynamic reconfiguration

The development of the notion of Points of Presence (PoPs) within our study enables us to think, in a systematic manner, about the various architectural alternatives which may arise in a collaborative mobile environment. Cases **2(a)** and **2(b)** show that the introduction of a PoP enables users to have better feedback. Similarly, cases **2(b)** and **3(b)** highlight how locality can be maximised to improve feedthrough.

The latter is particularly important as it is difficult to improve feedthrough in non-locally networked applications. Although ever-increasing network bandwidth may help to alleviate temporal problems to a certain extent, only an effective architectural framework can enable us to have a better understanding of these temporal problems and only the movement of 'server' functionality closer to the users can help to improve feedback and feedthrough significantly.

In presenting the six cases, we grouped them in pairs which had some architectural similarities. However, we have also pointed out other types of similarities between cases. In cases **1(a)**, **2(a)** and **3(a)**, we have different versions of a centralised architecture, whereas **1(b)**, **2(b)** and **3(b)** are forms of peer-peer architecture. If instead we focus on the work the PoP is doing we find that in cases **2(a)**, **2(b)** and **3(a)** the PoP is heavily involved in the computation whereas in cases **1(a)**, **1(b)** and **3(b)** the PoP acts at most as a router. These similarities are summarised in figure 8.

	PoP as computational entity	PoP as router
Centralised	**2(a)**, **3(a)**	**1(a)**
Peer-Peer	**2(b)**	**1(b)**, **3(b)**

Figure 8. 2X2 matrix of similarities

Some of the cells in figure 8 show only one option while others show two possibilities The difference lies in the number of PoPs involved. Both options **2(a)** and **3(a)** are based on a centralised arrangement, but being distributed, option **2(a)** has more than one PoP. Similarly, options **1(b)** and **3(b)** share a peer-peer arrangement but unlike option **3(b),** which consists of only one PoP, option **1(b)** involves more than one PoP.

In cases **1(a)** and **2(b)** we have several PoPs involved. This obviously suggests looking at whether there are variants of these, **1(a)'** and **2(b)'**, where there is only one PoP involved. In both cases, these are degenerate variants where all the relevant users happen to be physically close to one another, but where the computational infrastructure is treating them as distant (see figure 9).

	PoP as computational entity	PoP as router
Centralised	**2(a)**, **3(a)**	**1(a)**, **1(a)'**
Peer-Peer	**2(b)**, **2(b)'**	**1(b)**, **3(b)**

Figure 9. Degenerate variants

Looking first at case **1(a)**, the variant with a single PoP, **1(a)'**, is a degenerate case where all communications go back and forth to the server via a single PoP that acts as a router. This suggests an obvious choice for dynamic architecture reconfiguration. If the underlying platform supports running code mobility and the PoP is powerful enough then it would be possible to shift the server-end code from the server to the PoP (becoming case **3(a)**).

This dynamic reconfiguration from **1(a)'** to **3(a)** requires no major changes or worries about replicating or synchronising the shared data (as in option **2(a)**). The migration from option **3(a)** back to **1(a)'** is also possible if the users move apart again and simply involves code migration without any new client/server synchronisation. This change is equivalent to deliberately not routing a mobile phone call through a central switch when the callers happen to be in the same cell. The overall effect is to reduce network resources and improve both feedback and feedthrough.

The degenerate variant of **2(b)** is when the peer-peer replicas all reside on the same machine. This suggests two potential optimisations to improve performance. First, naïve implementations may actually involve several processes acting as a local peer for each client. Whenever all or simply some of the users are working through the same PoP there are clear performance gains if co-resident peers can merge and then split again when the users move apart. Furthermore, there are additional opportunities for optimisation when there is effectively just one peer left – a form of case **3(a)**. We know of no current system, which manages this form of process merging/division and this is perhaps a new distributed system support requirement.

This analysis has highlighted opportunities for dynamic reconfiguration offering the possibility of improved feedthrough in a mobile environment, a requirement which is difficult to achieve in a fixed distributed collaborative environment due to the latency of the network. Furthermore, the analysis gives a framework that enables us to understand the options and the implications of such reconfigurations.

The explicit identification of Points of Presence also offers a place within these network arrangements to allow access to management and reconfiguration information. Users can select and reconfigure the architectural arrangement to reflect their particular needs and constraints. For example, a user may wish to reconfigure the architecture in such a manner that it is not optimal but offers lower cost communication tariffs.

7 Code mobility issues

As we expected, the mobility of physical devices means a constantly shifting topology within the infrastructure and just like web-based solutions, issues about data and code mobility resurfaces in mobile settings.

In cases **2(b)** and **3(a)**, some of the functionality of the server has migrated to the PoP. In case **3(b)**, some of the client functionality is in the PoP. So how does it get there? Unless the application is ubiquitous it will not be sensible to have copies of the code sitting at every PoP, so it must either download from the client or the server. In cases **2(b)** and **3(a)** shared data must also migrate to the PoPs.

The resulting security and management implications are somewhat daunting:

i. will public carriers allow foreign code to run *inside* their network?

ii. will users trust devices embedded in the environment to run parts of their (perhaps private) applications.

Current sandbox technology where the computational context is bounded within a protected region of execution (despite its occasional flaws), for example as found in Java 1.2, which offers the potential for flexible schemes that go some way to satisfying question (i). Question (ii) – running code on 'untrusted' platforms is however more problematic. To date we know of no other work on this topic with the exception of very conservative approaches that trust the results of such computation only as much as the weakest link. Cryptographic techniques have allowed far better results for communication over untrusted channels and we may hope to see some cross-over into this harder problem area.

8 Discussion

The usability (and indeed infrastructure efficiency) gains of moving computation close to users is obvious. To some extent much of the work on 'active networks' has appeared to be solutions looking for a problem. The demands to move running code and reconfiguring networks in the ways we suggest certainly show that there are real and ongoing issues to be solved. However, current active network research is largely focused on computation for low-level communications purposes (for example, the recent IEEE Computer special issue on active networks. [20]). These will need some modification in order to be useful for application-level computation.

One of the most exciting aspects of execution on PoPs is the potential in reducing the dependence on *feedthrough* – the ability of users to see the results of one another's ongoing work. The irreducible delays due to network latency have meant that this is an intractable problem for distributed users and the solution in such cases is to design applications which in various ways 'cope' with these inevitable delays. However, it seems intolerable to make users, who can see one another across a room, suffer the same delays as those across continents. The options suggested in this paper and the framework for considering the static and dynamic choices between those options make it possible to design systems which are feasible in development terms and which offer improved collaborative interaction.

9 References

1. aQtive limited. *Web Research* http://www.aqtive.com/community/research
2. Bentley R, Busbach U, Kerr D, Sikkel K. (eds) Groupware and the World Wide Web. Kluwer, Dordrecht 1997.
3. Bentley R, Rodden T, Sawyer P, Sommerville, I. Architectural support for cooperative multi-user interfaces. IEEE COMPUTER special issue on CSCW, 1994; 27(5). pp 37-46.
4. Borovoy R, Martin F, Vemuri S, Resnick M, Silverman B, Hancock C. Meme tags and community mirrors: moving from conferences to collaboration. In: Proceedings of the ACM 1998 conference on Computer supported cooperative work, 1998. pp 159-168.
5. Cao P, Zhang J, Beach K. Active Cache: Caching Dynamic Contents on the Web. Univ. Wisconsin-Madison, 1999.
6. Clarke D, Dix A. Proceedings of The Workshop on the Active Web, 20th January 1999.
7. Coutaz J. PAC, an object oriented model for dialogue design. In: Bullinger H-J, Shackel, B. (eds) Human–Computer Interaction – INTERACT'87. Elsevier (North-Holland), 1987. pp 431-436.
8. Crowley T., Millazzo, P., Baker, E., Fordsdick, H., Tomlinson, R:. MMConf: An Infrastructure for Building Shared Multimedia Applications. In: Proceedings of CSCW'90. ACM Press, 1990. pp 329-342.
9. Davies N, Blair G, Cheverst K, Friday A. Supporting Adaptive Services in a Heterogeneous Mobile Environment. In: Cabrera L-F, Mahadev Satyanarayanan. (eds) Proc. Workshop on Mobile Computing Systems and Applications (MCSA), Santa Cruz, CA, US. IEEE Computer Society Press, December 1994. pp 153-157.
10. Dewan P. A tour of the Suite user interface software. In: Proceedings of UIST'90. ACM Press, 1990. pp 57-65.
11. Dix A, Rodden T, Davies N, Trevor J, Friday A, Palfreyman K. Exploiting space and location as a design framework for interactive mobile systems. ACM Trans. on Computer-Human Interaction, 1999 (in press).
12. Dix A. The Active Web - Parts 1 & 2. Interfaces, 1998; 38 pp 18-21, 39 pp 22-25.
13. Dix AJ. Cooperation without (reliable) Communication: Interfaces for Mobile Applications. Distributed Systems Engineering, 1995; 2(3). pp 171-181.
14. Fox A, Gribble SD, Brewer EA, Amir E. Adapting to Network and Client Variation via On-Demand, Dynamic Distillation. In: Proc. ASPLOS-VII. Boston, MA, US 1996.
15. Gram C, Cockton G. (eds) Design Principles for Interactive Software. Chapman and Hall, UK 1996.
16. Greenberg S, Marwood D. Real Time Groupware as a Distributed System; Concurrency Control and its effect on the Interface. In: Proceedings of CSCW'94, North Carolina, Oct 22-26. ACM Press, 1994. pp 207-217.
17. Gust P. SharedX: X in a distributed group work environment. 2nd Annual X Conference, MIT, 1988.
18. Hill RD, Brinck T, Rohall SL, Patterson JF, Wilner W. The Rendezvous

architecture and language for constructing multi-user applications. ACM Transactions on Computer-Human Interaction, 1994; 1(2). pp 81-125.

19. ICQ ("I Seek You"). http://www.icq.com/
20. IEEE Cover Feature – Active Networks, April 1999; 32(4). pp 32-56.
21. Johnson C. (ed.) Proceedings of the First Workshop on Human Computer Interaction with Mobile Devices. University of Glasgow, 21-23 May 1998, GIST Technical Report G98-1, 1998.
22. Johnson CW. The impact of time and place on the operation of mobile computing devices. In: Proceedings of HCI'97: People and Computers XII. Bristol, UK, 1997. pp 175-190.
23. Joseph A, deLespinasse A, Tauber J, Gifford D, Kaashoek MF. Rover: A Toolkit for Mobile Information Access. In: Proc. 15th ACM Symposium on Operating System Principles (SOSP), Copper Mountain Resort, Colorado, US. ACM Press, 3–6 December 1995; 29. pp 156-171.
24. Lewis The Art and Science of Smalltalk. Prentice Hall 1995.
25. Long S, Kooper R, Abowd GD, Atkeson CG. Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study. In: Proc. 2nd ACM International Conference on Mobile Computing (MOBICOM'96), Rye, New York, US. ACM Press, 1996.
26. Palfreyman K, Rodden T. A Protocol for User Awareness on the World Wide Web. In: Proceedings of CSCW'96, Boston, Massachusetts, Nov. 1996. ACM Press, 1996. pp 130-139.
27. Pfaff G, Hagen PJW. (eds) Seeheim Workshop on User Interface Management Systems. Springer-Verlag, Berlin 1985.
28. Ramduny D, Dix A. Why, What, Where, When: Architectures for Co-operative work on the WWW. In: Proceedings of HCI'97, Bristol, UK. Springer, 1997. pp 283-301.
29. UIMS. The UIMS tool developers workshop: A metamodel for the runtime architecture of an interactive system. SIGCHI Bulletin, 1992; 24(1). pp 32-37.
30. Want R, Schilit BN, Adams NI, Gold R, Petersen K, Goldberg D, Ellis JR, Weiser M. An Overview of the ParcTab Ubiquitous Computing Experiment. IEEE Personal Communications, December 1995. pp 28-43.
31. Welie VM, Eliëns A. Chatting on the Web. ERCIM workshop on CSCW and the Web (Sankt Augustin, Germany). GMD/FIT 1996.
32. Yahoo! Chat. http://chat.yahoo.com/