Middleware Support for Mobile Multimedia Applications

John Bates, David Halls and Jean Bacon
University of Cambridge Computer Laboratory
Pembroke Street, Cambridge UK

This paper describes a system we have developed to enable applications to follow mobile users as they move. If an application built using our system has made connections to distributed information resources, then these connections are re-established seamlessly after movement. It is thus possible to support mobility within applications as complex as multi-user distributed multimedia applications. The support system to achieve this is provided at the middleware level, i.e. between the environment (OS, network and system services) and the application levels. The interface can thus be system-independent and the system more easily extended than if it was implemented within the operating system. One system requirement is tracking the location of users and equipment and to this end a location-awareness service has been developed. Another requirement is the availability of components for constructing mobile applications and, for this, two types of mobile object are available. Firstly, mobile agents which allow complex state migration. Secondly, mobile media endpoints which provide lightweight migration for objects with a minimal amount of state and which benefit from minimum implementation overheads. A scheme for locating and binding to mobile objects is also outlined. We have implemented and experimented with the system described. The applications we have built include a mobile multimedia conferencing system in which the sessions of a mobile user can pop up on the nearest workstation so he/she can continue to communicate with colleagues whilst on-the-move.

1 Introduction

1.1 Motivation

The motivation for this work is to provide support for user mobility within distributed multimedia applications. Mobile telephony has indicated the attraction of transportable media endpoints. Mobile professionals who rely on computing resources can benefit from multimedia applications which are aware of user location and are themselves able to move within and between networks, remapping endpoints, such as cameras, microphones and user interfaces, onto current user locations. Mobile networking technology is not required for users to participate; computer deployment is increasing and users without networked portable computers will be able to make use of computers available in most sites they visit.

Some example applications which we can support are as follows:

- *Sending running programs to other users* – David can prepare and send a multimedia presentation about his latest project to John's current location, with an enclosed questionnaire. John can examine it, annotate it, fill in the questionnaire and send it back.
- *Location-triggered media presentations* – An application can be configured to monitor for when

Jean and Ken are in the same room, go to that location and play a video clip from John.

- *Mobile multimedia* – Distributed media endpoints, such as software abstractions of cameras and microphones as well as user interface components such as video display windows, can be part of a mobile application. A simple example is tracking a user with video. When the user moves, the image switches to the nearest camera. Another example is for music to be relayed to different speakers as a person moves around a home.
- *Mobile cooperation* – An enhancement of mobile multimedia is to support mobility for a cooperating set of users. In a multimedia conference, if a user moves then the camera, microphone, speaker and user interface endpoints must be moved to the new location. Connections to the other users involved must then be re-established.

We envisage the most common requirement will be supporting user mobility within a single computing domain. Each domain provides a set of services to support this mobility. More infrequently, a user may move to another domain. Supporting this involves a handover from one set of services to another.

1.2 Related Work

In this section we discuss three projects which have targeted mobility support for application users. ORL's *Teleporting* project redirects the user interface of a mobile user to the user's current location. The *Total Mobility* environment provides support for users who move between machines and work over low bandwidth or disconnected network links. The work on *Migratory Applications* using Obliq employs a mobile agent system which supports a mobile user interface.

1.2.1 Teleporting

Teleporting [RBMH94] is a technique to support user mobility by the user interface of an application following a mobile user. In order to *teleport*, a user clicks a button on their active badge (a type of electronic tag – see section 2.5) while in proximity to a workstation. Their current session then pops up on that workstation.

The teleporting system is based on the technology of the X window system. A proxy X server is able to forward X protocol requests to another server thus providing a level of indirection. The display of X applications can thus move with a mobile user. A simple database provides a mapping between current location and workstation display.

Teleporting is a powerful technique, the main advantage being that it makes any existing X application mobile. This approach also has some disadvantages in that

- It is tied to the X system. Only the display connections follow a user; there is no dynamic reconfiguration at an application level. If it is used with any medium that does not go through X, e.g. audio, then the approach fails.
- It is potentially inefficient as control messages have to go to the original site and then be forwarded. The system performance varies in accordance with the bandwidth of the network connection between the home site and the teleport site.
- The applications do not themselves move, are unaware of their user interface moving and are not built around the premise of mobile users. This means that factors explicitly related to mobility cannot be capitalised on, e.g. the application behaving differently depending on location.

The user mobility framework described in this paper allows applications as well as their user interfaces to migrate between computers, thus avoiding these disadvantages.

1.2.2 Total Mobility Environment

The Total Mobility environment (TM) [WH96] views computers as "replaceable tools, which can be hired, used and given up once no longer needed". Like the work described in this paper, it aims to support the use of locally available computing resources whenever a user moves, to eliminate the need to carry equipment when it is not useful. It addresses the problem of providing users with access to their data even when using mobile devices that are not connected to a network.

TM provides location information about users and computer equipment and a data management system based on a disconnected file system architecture. Users register and de-register for use of a mobile computer in disconnected operation. Their files are replicated at the mobile devices when they register. A logging process is used to identify modifications that users make while they are on the move and disconnected. When the user de-registers with the mobile device, or network connectivity is available again, the main copies of the files are updated with the modifications. The logging process also helps to cope with conflicting modifications made by multiple users sharing the same set of files.

Our work differs from TM in that it concentrates on support for making (multimedia) applications mobile over broadly uniform networks rather than providing generic access to user files over low-bandwidth links. TM does not address migration of applications to follow users. However, combining the two would provide migratory, follow-me applications with support for working in low-bandwidth and disconnected environments.

1.2.3 Migratory Applications

A large number of mobile agent systems have been and continue to be developed. The applications of these systems include active documents [AG96], managing hypermedia [DHDD96], network management [HR97] and information gathering [RGK96]. One mobile agent system which has specifically linked user and application mobility is the Migratory Applications work using Obliq [BC95]. This work employs the Obliq mobile agent system so that applications can move from one user to another or can be configured to follow users from machine to machine. Support for state saving in Visual Obliq allows applications to take their user interfaces with them when they move.

The Migratory Applications work is similar to that described in this paper in that it employs mobile agents and knowledge of user location. However, it produces monolithic applications for single users in which the mobile agent contains all control functionality and the user interface. The following sections describe a framework for user mobility which subsumes this functionality but also has the advantage of being able to support mobility for distributed multimedia applications involving more than one user.

1.2.4 Conclusions

The main limitation of these projects is the lack of mobility support for distributed, multimedia and cooperative applications. Teleporting moves only the X user interface and does not consider reconfiguration of multimedia streams. TM is concerned with uniformity of access regardless of location but does not address real-time issues. With Migratory Applications, only what is encoded within the application moves; there is no notion of re-establishing existing communication channels after movement.

1.3 Overview

We elected to build the framework described in this paper at the middleware level, i.e. between the environment (operating system, network and system services) and the application. This is so that our system is not dependent on any particular operating system. Also, since it is not within the operating system, new components can more easily be added over time. Figure 1 outlines the main components in our framework.

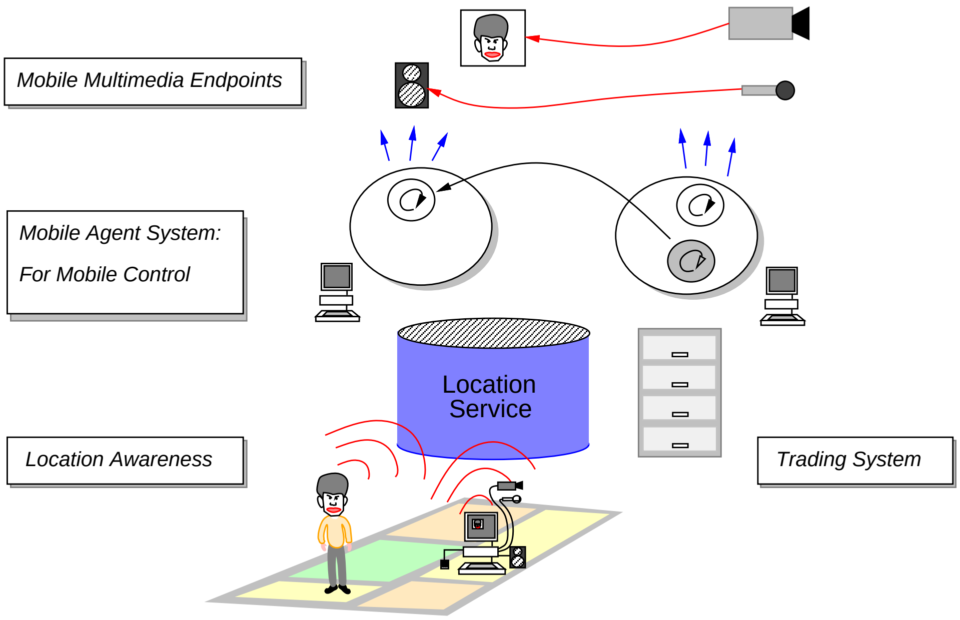

Figure 1: Components in our mobile applications framework

The first requirement of our system is to allow applications to monitor the locations of users. To this end we have designed and built a location-awareness system, described in section 2.

It is necessary to move applications or components of applications in response to changes in user location. One technology we deployed to this end is a *location-aware mobile agent system*, described in section 3. Agents can detect user movements and respond by moving to the user's nearest computer or taking other action.

Our experiments showed that mobile agents were too heavyweight for use as media components in mobile multimedia applications, e.g. as video sources, processors or sinks. To address this problem we developed lightweight moveable media objects, described in section 4.

It must be possible to locate and bind to objects, such as mobile agents or multimedia endpoints, before communication is possible. In section 5 we describe how trading technology can be adapted for naming and locating mobile objects.

The integration of location-awareness and mobile objects into a mobile application support framework is described in section 6. The deployment of and experimentation with this framework is described in section 7. Section 8 summarises and concludes the paper.

2 Supporting Location-Awareness

2.1 Requirements

The basic requirement of a location-awareness system is to allow applications to find out about the location of users. To enhance support for location-awareness, it is useful to be able to find out other factors about the environments which users inhabit, for example the deployment of computing equipment.

There are various technologies which can be used to support user tracking, for example monitoring the location of user logins or users wearing electronic tags. A location-awareness paradigm should be independent of underlying technology.

It is beneficial for clients of a location-awareness system if the system can perform some filtering on their behalf. The location system can do this more cheaply than clients by generalising the filtering process for multiple users. This also cuts down on the amount of traffic sent to clients by the location system.

Run-time location events can be viewed in the same way as parametrised run-time occurrences from other systems. In selecting an event filtering and propagation system, it is beneficial to use technology which is applicable generally. This enhances the interchangeability of events between different systems, encouraging interoperability.

2.2 A Model for Active Location

We have developed a mechanism of event-based programming which we use to support location-awareness [BBHM95]. The motivation behind event-based programming is that many applications, including those associated with mobility, are *active* in nature, i.e. run-time actions are triggered in response to asynchronous occurrences (events). This situation is complicated within an open distributed context since applications can be composed of components running on separate machines, each of which is a potential source of events. It is desirable to avoid providing interfacing code to handle occurrences from every potential source.

To address these circumstances we have developed a generalised model in which events are transmitted as parametrised run-time occurrences. An event service is defined as any network service which publishes a specification of classes of occurrence it can inform a client of. Clients register interest in the set of occurrences they wish to be informed of. Services notify clients asynchronously if an occurrence matching the registration criteria is detected. The registration paradigm enables scalable and flexible construction of active applications and avoids communication saturation by filtering at source and/or intermediate nodes.

The event approach aims to augment rather than replace current distributed programming methods (such as CORBA [Gro91]). We have therefore extended an existing Interface Definition Language (IDL) to handle events. This enables a server to declare, in a strongly-typed way, the events it can notify. It also allows a client to see the server's specification of events and to select those of interest.

In the case of location-awareness, applications can register interest with a location service (see figure 2). The service can advertise location monitoring facilities, declared in an IDL, for example:

```
LocEvent: EVENTCLASS [ person : USER,
                       location : PLACE ];
```

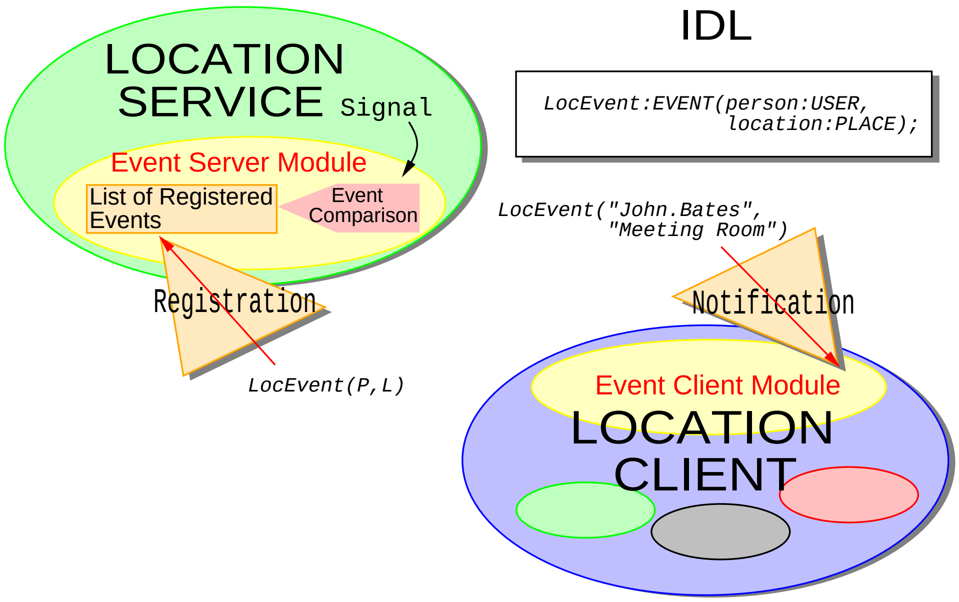

Figure 2: Event-based location-awareness

A location service inherits the properties of an event service, i.e. generic support for handling and storing registered events, for comparing signalled instances with those registered and for notifying any matches. A mechanism for querying the specific location technology is contained within the location service. Related work is currently underway which involves the correlated use of multiple location technologies to increase the accuracy and flexibility of a location service [Nel97].

An event client inherits facilities for registration and for handling notifications. Structural information generated by processing an event interface can be used by both client and server for the transmission of event instances. A client provides an *event template*; the parameters of the event are given as values which must be matched, or variables which are wildcards. Examples of location event templates are:

- `LocEvent("John.Bates", L)` – Report wherever John is seen
- `LocEvent(P, "Meeting Room")` – Report when anyone is seen in the meeting room
- `LocEvent(P, L)` – Report when anyone is seen anywhere

When event instances are notified to clients, all parameters are given, e.g. the instance `LocEvent("John.Bates", "Meeting Room")` means John has been seen in the meeting room.

In the above example we have simplified the event class for illustrative purposes. In the example we are assuming there is one location service per computing domain, so a client is aware of the computing domain implicitly. An alternative approach is to federate location services so that a registration is propagated to location services in other domains or to intermediate event brokers. The event class would include a `domain` field to inform the client where the event was generated and to allow clients to select domains of interest, using event templates. In such a scheme, users must be uniquely identifiable worldwide.

2.3 Monitoring for Complex Occurrences

Often, activity within applications is triggered not just by a single event but by a complex pattern of events. Such composite detection can involve parameter checking of event occurrences, state lookup and monitoring for particular event orderings. An example query is "tell me everyone who is still in the building 20 seconds after the fire alarm goes off". This form of event composition is simplified by using our general event architecture, since events, despite coming from different systems, all inherit generic event features. We have developed a mechanism for efficient composite event monitoring based on non-deterministic finite state machines. For further information on the techniques involved see [BBHM95].

2.4 Environment Location Information

It is beneficial if other information is available to enhance location-awareness, for example the following types:

- The names of rooms and their geography, i.e. how they interconnect. This is a useful measure of proximity. For example, monitoring can be extended to detect when two or more users are on the same floor of the building.
- The equipment stored in each room. This information can either be static or equipment can be electronically tagged. Using this information it is possible to determine whether a specific room or part of a room offers a certain facility, e.g. live video.
- The names and capabilities of pieces of equipment. For example, the architecture, operating system, hardware facilities, attached peripherals and software configuration of a workstation.

It is envisaged that each computing domain would offer such a database, detailing its geography and facilities. Such a service is a useful component of adaptive location-aware multimedia applications. For example, if an application is following a user or is dispatched to a new location it can check that the required facilities are available there. If the application requires video and audio and these facilities are not supported in a particular location then it can adapt and use text-only conferencing instead.

2.5 Our Experimental System

We have implemented our event architecture as part of an ODP-ANSA compliant distributed programming platform [Bat96] and are working on integrating it with a CORBA platform. Within this work we have enhanced IDL processing tools to generate stubs for event code. We have also developed server and client libraries to provide generic event operations, such as registration and notification.

Using this technology we have built a location service and a location information database. The location technology employed by the location service is the active badge system [HH94] deployed in our laboratory. Each user wears a badge which periodically transmits a unique infra-red signal. A network of detectors allows the location of the user to be pinpointed. Our location service collects badge events and then passes them to the event server library using the operation `Signal`. If an event matches any registered templates, the event system will notify the appropriate client(s).

The location information database contains data about the organisation of rooms and the names and capabilities of equipment stored in each room. Our implementation was based around an entity-relationship data model, which we implemented in Prolog. An example from the model is shown

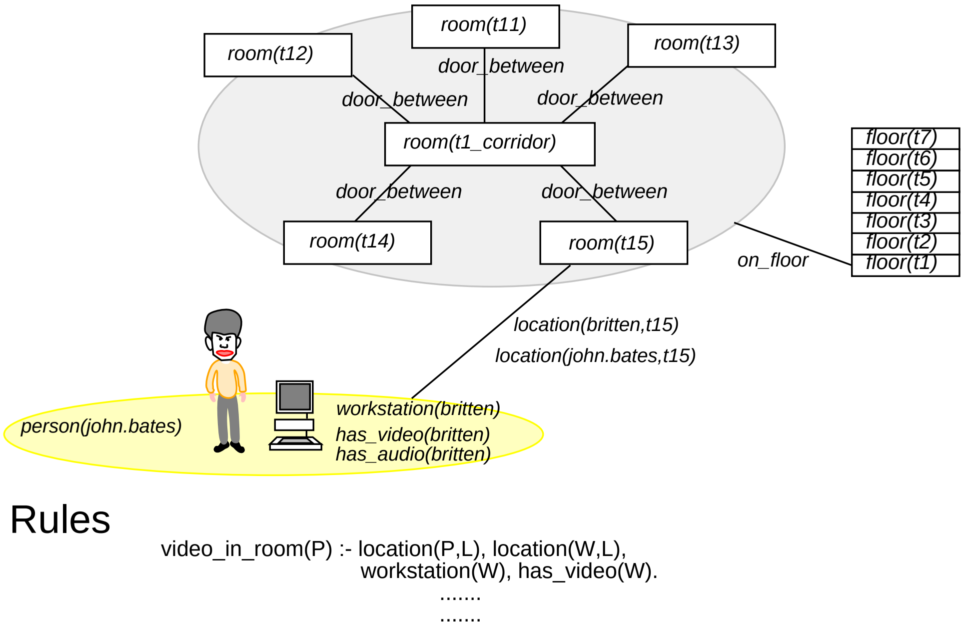

Figure 3: Entity-relationship model from location database

graphically in figure 3. This illustrates how the interconnected geography of a building, containment of items in rooms and item properties can be described. The database is given an initial set of data and is updated using events describing the latest locations of people and equipment. It is possible to add new rules as well as to submit queries to the database. Examples are "is there a workstation with video capability at John's current location?" (`video_in_room(john.bates)` – see figure 3) and "is there a route out of the building avoiding my supervisor?!"

3 Location-Aware Mobile Agents

3.1 Mobile Agents

Mobile agent systems facilitate mobile applications. Application code and execution state, sometimes including the user interface, can be migrated from one host to another. Each host within the system must run a service which can send, receive and execute agents. Mobile agent systems differ from process migration systems in that they are implemented in user space, allowing system independence. Many useful applications of such systems have been documented (see section 1.2.3).

In this work we introduce a new range of applications of mobile agents. By making location information available to mobile agents, the integration of user mobility monitoring and mobile applications is facilitated. This enables mobile agents to be sent to users rather than to a named host. By using location information, the host nearest to the user's current location can be determined. Many existing applications involve mobile agents traversing a number of hosts. Location information enables agents to traverse a number of user locations. Most interestingly, it is possible for mobile agents to *follow* users by triggering movement in response to location events. This is beneficial since much state is set up in the course of using an application and it is a waste of time and resources if this state has to be recreated manually at various hosts. If the user becomes mobile for an extended period of time, it is possible to persistently store agents on disk or transportable smart card.

The alternative to using mobile agents is for an application to create and control an object remotely. This method involves factories and protocols. Factories are distributed services which can create objects of certain classes. Protocols which are understood by both client and server are used to manage the process of object creation and control. Mobile agents have various advantages over using such an arrangement. Firstly, it is quick to prototype a new application, since no compilation or service setup is required. Creating and controlling an application remotely involves network traffic, whereas a mobile agent for this purpose can be sent to an execution point near to the resource required. Also, mobile agent platforms are system-independent, so there is no requirement to comprehend an system-dependent API.

There are two main ways in which mobile agents can be used for location-oriented applications. Firstly, the mobile agent can be the entire application, containing all functionality including the user interface. Secondly, the mobile agent can be used for high-level mobile control. In this case the agent is responsible for setting up and controlling distributed connections using other services. It is possible that these services are distributed around the environment, however, if they are not location-specific they can be linked dynamically to a mobile agent. This enables a wider range of applications than coding the entire application into one monolithic agent. For example, as will be described later, distributed multimedia applications involving multiple users can be supported. Using mobile agents for control also aids the management of heterogeneity. By binding dynamically, mobile agents can access domain-specific functions. Mobile agents are appropriate for this high-level management role since it can be prototyped rapidly and moved flexibly.

3.2 The Tube: Our Mobile Agent System

We have gained experience from building and deploying our own mobile agent system, the Tube [Hal97]. This section describes features of the Tube, including many which we found essential for its use in an active location-aware environment. The integration of the Tube with other environment features to support mobile multimedia applications is described later in section 6.

For a host to be able to send and receive mobile agents it must run an instance of a Tube site. Agents are written in the Scheme language (a dialect of Lisp); they are transmitted over a network as marshalled expressions. If tagged as executable, they are passed to an interpreter for execution. The marshalling respects duplicate objects and cyclic references. We are able to transmit the full range of expressions, even closures[1] and continuations[2]. This means that these agents can create arbitrary functions and send them elsewhere, with their closing environments.

Through being able to transmit continuations, an agent can be stopped, moved and restarted elsewhere in a single call. The Tube does much more than just execute byte-code compiled scripts remotely – agents can modify themselves, create and dispatch other agents and treat their functions and state as first class, transmissible data. Agents execute in a safe interpreter and are only allowed to access those functions that the platform's owner makes available.

Tube sites are multi-threaded. Full access to a POSIX threads interface provided by the underlying operating system is given to agents. A noticeboard is provided, which an agent can post messages to or read messages from. Data values can thus be left by an agent for others that may arrive later to use. The noticeboard can be divided into different areas; an access control list is associated with each.

Tube sites can dynamically load at run-time compiled libraries of code and call functions contained in them. This allows facilities to be added without having to stop and recompile.

[1]A closure is a function together with its defining scope.

[2]A continuation is a closure that represents the current state of execution. Calling a continuation results in the computation resuming from where the continuation was captured.

A user interface toolkit has been integrated with the Tube. Functions to create and manipulate widgets are available to agents. We have also enhanced the toolkit with the ability to return the state of any user interface, along with any callbacks registered on widgets, as a series of bytes. A corollary function takes a saved user interface and recreates it in a visible form. This allows an agent to create a user interface on one machine and retain it as embedded state. We have also written a Netscape plug-in that allows the Tube to be used for writing applets embedded in World-Wide Web pages.

The core state-saving functionality of the Tube is written entirely in Scheme. It is portable across Scheme interpreters and compilers, and thus also across operating systems. Code to interface with other systems, such as POSIX functions or a user interface toolkit, is not portable between operating systems. Currently, the implementation is Unix-based. We use the Bigloo [Ser94] Scheme compiler/interpreter and the XForms [ZO96] user interface toolkit. Versions are running on DEC Alphas under Digital Unix, SUN SPARCs under Solaris, Intel Pentiums under Linux and HP 9000s under HP-UX.

4 Mobile Multimedia Endpoints

4.1 Requirements

The use of distributed multimedia introduces new problems for mobile applications. In this discussion we are assuming a system is available which offers an object-based abstraction of media source and sink endpoints, such as cameras, microphones, video displays and speakers. These objects can be created on distributed hosts and connected together flexibly using stream abstractions, e.g. a video stream connecting a live video source to a video window. This is reasonable since such systems have been developed successfully [W$^+$94, BB95].

To move one or more objects involves transfer of object communication state. For example, assume David is looking at a video window of Kerry which is connected to a live video capture object at Kerry's location. If Kerry moves then we must remap the video source onto the nearest camera and reconnect to David's window. Such re-routeing of streams is known as *handoff*. This is commonly provided by the network layer (see for example [Raj95] or [PH94]), though application support for handoff has been proposed [Pop96]. Our implementation of handoff, described below, is done entirely in user-space and is specialised for multimedia objects which can tolerate data loss. We plan to capitalise on research which is being carried out into allowing application-specific policy for mobility to be closely bound at runtime with network control functionality [Roo97].

Media endpoints can be implemented as mobile agents. It is more likely that they would be built using more traditional distributed programming techniques (e.g. an enhanced CORBA platform) and created dynamically at run-time using distributed factories. The assumptions behind this reasoning are as follows:

- that an application involving media connections will have some form of intelligent controlling program, responsible for setting up the connections. In the case of a multi-user program, each user may have such a program. Therefore this program can be responsible for location-awareness and subsequent moving of connections.
- media objects generally perform simple memory-less functions and therefore do not need the advanced state-saving of a mobile agent system such as the Tube.
- media functions generally involve the manipulation of large amounts of data and thus should be as fast as possible. Mobile agent systems involve more heavyweight execution overheads due to

code interpretation and migration mechanisms. It is therefore often desirable to use compiled code for media objects.

4.2 A Media Object Implementation

In our implementation we support mobility of media endpoints in user-space, by saving the communication state of media objects and using this information to re-instantiate a copy in the new location (see figure 4-6). Media objects are themselves responsible for re-establishing connections to the objects they were previously connected to.

We have developed media endpoints as active objects, autonomous entities which can send and receive messages and process data concurrently. Each object class is developed to support distribution, with an interface that can include invokable methods, events that the object can notify (see section 2.2) and connectable stream endpoints. Stream endpoints are named and typed, for example `VidSource` and `VidSink` of type `VideoStream`.

In order to configure a multimedia session, first the appropriate objects must be created on the appropriate hosts and then they must be connected together. Media objects are created by making a creation request of an object factory on the appropriate machine, by spawning an object as a new process or by linking dynamically to the object code and instantiating a new instance. Factories are services at which objects of certain classes can be created. For instance, on every workstation with live video capture facilities there will be a factory capable of creating live video source objects.

If an object must be created remotely or a connection to an existing object is required then an interface reference for the appropriate object/factory must be located. A trader is used to this end. Object references are stored in the trader using attributes to aid retrieval (see section 5.3). For example, a live video source factory can be annotated with the `location` attribute set to `Meeting room`. If multiple cameras are available in one location then they can be distinguished with an attribute describing their proximity to computing equipment or active badge sensors. When objects are created they are allocated unique attributes to distinguish them from other objects of their class.

In order to support stream operations we have extended a distributed programming platform. Our implementation is available on DEC Mips running Ultrix and Alpha platforms running Digital Unix. Enhanced stub generation tools build functions to read and write stream data and generate interfaces to allow third parties to set up and connect stream endpoints. Third parties use a library, written to manage this process. Once interface references have been obtained, the third-party client invokes a `Connect` operation with the two objects in question as parameters. The underlying implementation is that the source object is sent a message detailing the sink object. The source and sink objects make a stream connection, handshake and exchange their unique naming information. Data can then be sent from source to sink using stubs generated by processing the object interface. Our implementation of the stream library uses sockets. Appropriate protocols can be selected to satisfy particular application requirements.

The stream library supports a mechanism to allow a third party to access the communication state of an object. The object stream library supports using this information to re-instantiate the streams when required. The trader from the domain in which the other object(s) existed is contacted (see section 5.3) to get the reference to its stream interface. A stream connection is then re-established.

4.3 Implemented Mobile Media Objects

We have developed various media object classes using our enhanced distributed programming platform. These include sources for live and stored video and audio, audio players, video windows and shared drawing and text components. There is also a component which enhances a Web browser for mobility. It queries the URL of the page currently being accessed by the Netscape browser on a particular host, using Netscape's socket API. Using the API on another host it can instruct the browser running there to display the saved URL.

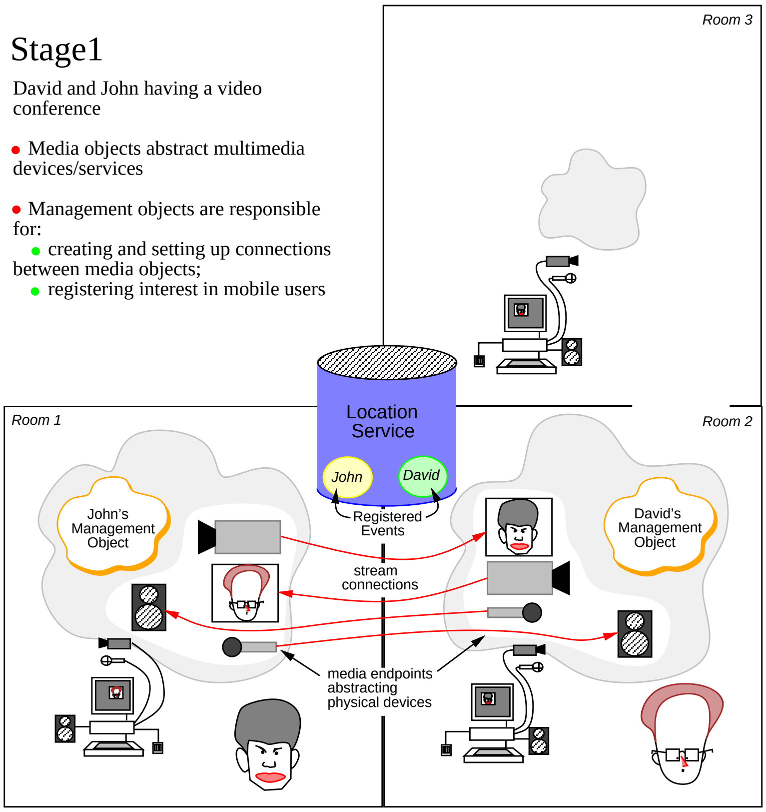

Figure 4: Stage 1 in moving an application instance

5 Naming and Location of Mobile Objects

5.1 Requirements

We have introduced two types of mobile object: mobile agents and mobile multimedia endpoints. For objects to communicate, the remote object must first be located (in physical terms a mapping to host and port) and the two objects bound. There are several circumstances in which location of remote objects for communication must be determined, for example:

- If a specific class of multimedia endpoint is required on a specific host, then the factory on this host must be located.

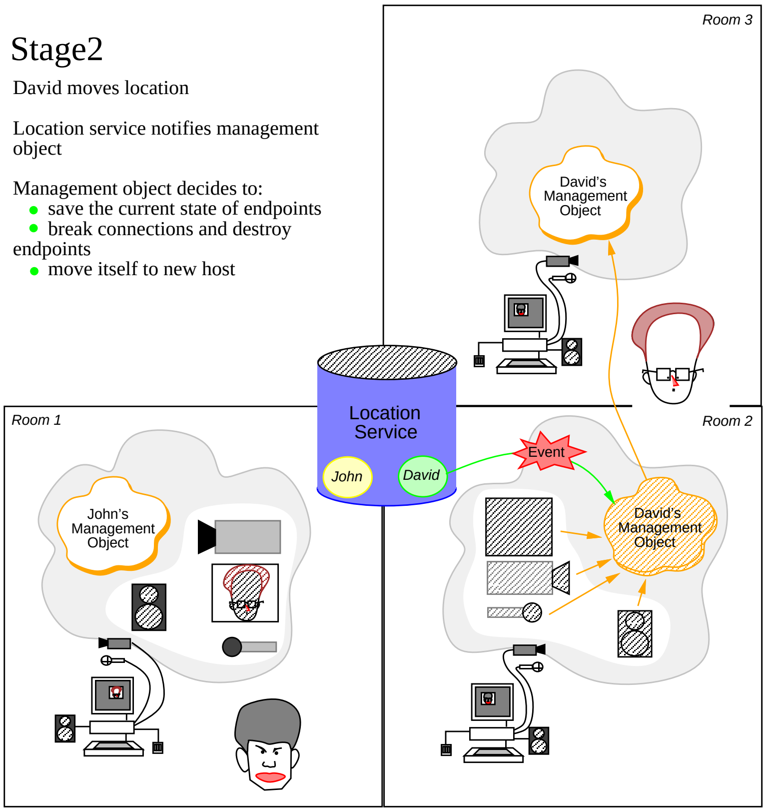

Figure 5: Stage 2 in moving an application instance

- For a new user to bootstrap themselves into a distributed cooperative application, it may be required to bind to another user's instance and run a join protocol.
- If two multimedia endpoints are joined using a stream and one endpoint moves, then the stream must be re-established. If this is done in user space then after movement a third party must locate and rejoin both ends or one of the end-point objects must find the other and reconnect.

5.2 Trading Services

The traditional way of locating an object is to use a trading service. An object's interface reference is required to bind with the object for communication purposes. Interface references are registered in a trader's namespace, indexed by the object class and other naming information. If objects move, interface references are of no use, since they encode physical location information. Regular access to trading information is thus a requirement in a mobile object system. An object naming scheme should allow the advertisement of specific characteristics to assist in locating a specific instance, for example name, class, owner and the host on which it runs.

If an object moves then a mechanism to alert applications which may try to contact it is required. During movement, the object can be marked in a trader as *mobile* or the object can remain alive to inform clients of its status. If the object moves to another domain it must be possible to locate it. One method is to federate the namespaces of trading services. Another is to leave *tombstones* (forwarding information) in a trader to enable objects to be located in their new domain. Updating references is at the discretion of moving objects. Additionally, this method introduces a problem of garbage collection. A further mechanism is to associate a home trader with each object and keep it

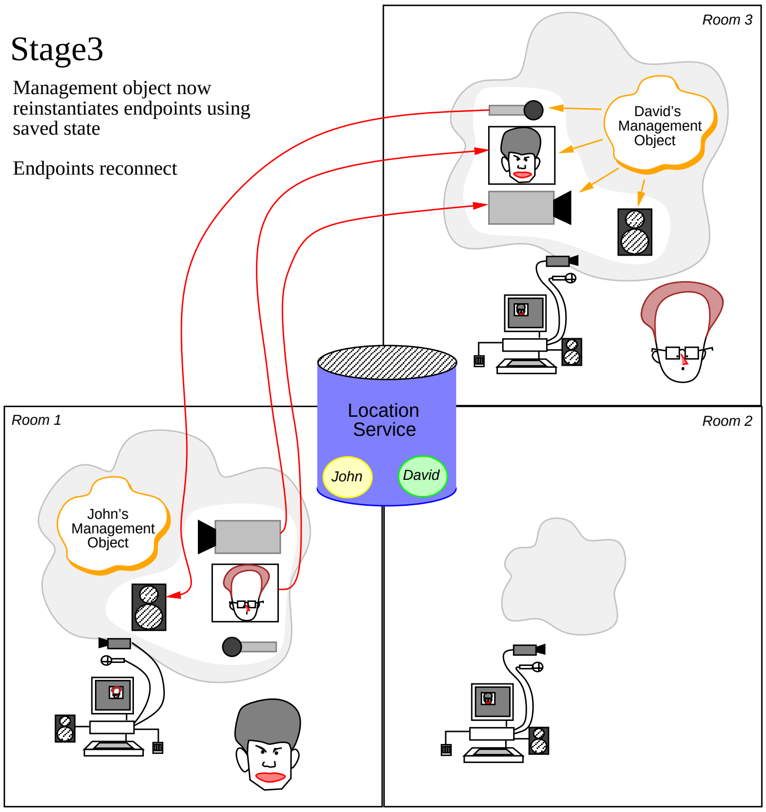

Figure 6: Stage 3 in moving an application instance

up-to-date with the object's current location.

5.3 An Example Implementation

We have built a trading service which supports a rich object naming scheme, based on SGML tags. An object advertises its functions with a trading service, e.g. the video source object in figure 7. Clients of the trader can query objects associatively, for example a search `objectName contains "John", objectType == "Tub VideoCamera"` would return the object from figure 7 and any others matching the search criteria. Tags are used at the discretion of applications. For example, the `html` tag allows arbitrary HTML markup to be inserted for use by Web browsers. We have implemented a graphical interface to the trading service, using the Web (see figure 8). By using information from traders to update large information repositories (analogous to Web search sites like `http://altavista.digital.com`) it is possible to search and locate active services worldwide, based on their characteristics.

In our implementation, an object in the process of moving can mark its reference in the trader as mobile. When an object has moved and the handoff process is complete, the reference can be updated with the new object location. If an object has moved to another domain, it can also register with the trader there; local applications that wish to bind to the object can thus avoid communication with external services.

```
<Agency>Tube</Agency>
<Naming>address</Naming>
<addressType>content</addressType>
<addressContent>
<objectType>Tub VideoCamera</objectType>
<html><br>
<img src=http://www.cl.cam.ac.uk/users/dah28/images/camera.gif>
</html>
<objectName>John's camera</objectName>
<objectOwner>John.Bates@cl.cam.ac.uk</objectOwner>
</addressContent>
<addressQuery>
#BAND(#FIELD(objectType=Tub) #FIELD(objectType=VideoCamera)
      #BAND(#FIELD(objectName=John's) #FIELD(objectName=camera)))
</addressQuery>
<resolvesType>Tube</resolvesType>
<resolvesHostname>britten.cl.cam.ac.uk</resolvesHostname>
<resolvesPortnum>1270</resolvesPortnum>
```

Figure 7: Advertisement for a video source object

6 The Integrated Mobility Framework

This section describes how the location awareness and mobile object technologies were integrated. Since mobile agents are the controlling entities in a mobile application, integration work was centred on the enhancement of our mobile agent system.

6.1 Enhancing Mobile Agents for Management

The Tube has been enhanced to make it location-aware and able to configure distributed multimedia objects. In our integrated architecture, the Tube is a client of both the location service and the multimedia framework. To enable this we linked the relevant client code into the Tube.

Mobile agents are able to register interest with the location service and, due to the multi-threaded implementation of the Tube, handlers can be set up in scripts to respond to asynchronous notification of user movements.

Scripts can also create objects on remote hosts, connect stream endpoints together, invoke operations on media objects and save communication state. Other versions of media objects have been developed which enable dynamic linking. These are more tightly coupled to a mobile agent as they run in the same address space. It is also possible to write user interface components using the Tube directly since it contains the state-savable XForms library.

6.2 Communication Systems

The ability to move applications partially or wholly between computing domains introduces the issue of communication system interoperability. It is acceptable to assume that communication standards, such as TCP/IP sockets and CORBA will be available. Incorporating events and streams into the CORBA standard will support inter-domain operation of all features described in this paper. There may be circumstances when the use of other communication technology will be required. A mobile agent system should provide the ability for agents to bind dynamically to code for different communication mechanisms.

In our specific implementation three communication systems were available: TubeRPC, MSRPC and CORBA RPC. It is possible for a mobile agent system to use any of these.

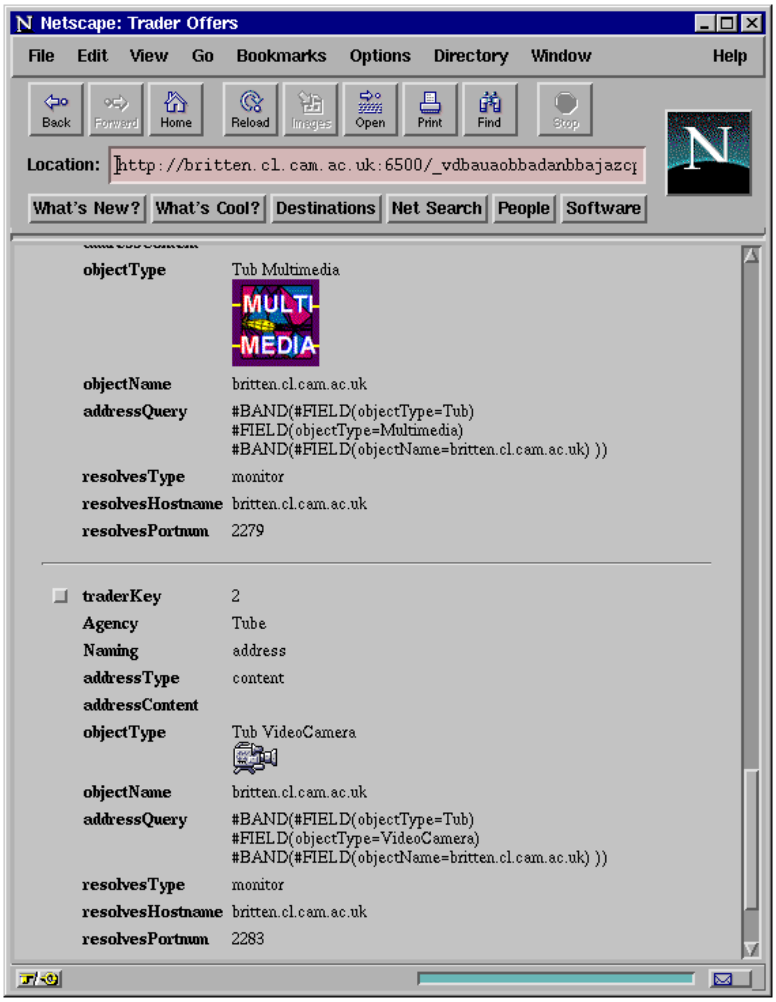

Figure 8: Trader advertisements in a Web browser: a multimedia object factory and a video camera

TubeRPC is a novel mechanism which involves installing an optimised Tube interpreter within a compiled object. Scripts can be despatched to execute within an object, using its local interface as an API. Some experimental multimedia objects feature a TubeRPC interface. TubeRPC also allows the publishing of object proxies in traders to abstract above communication semantics.

7 Experiments

Multimedia workstations on our local area network run instances of multimedia object factories and Tube sites. We deployed individual instances of our trading and location services. We wrote a number of experimental applications using this apparatus.

7.1 Experimental Schedule

The first phase of experiment involved building a number of small applications to illustrate rapid prototyping using Scheme scripts. We illustrated the principle of specifying host machines by their

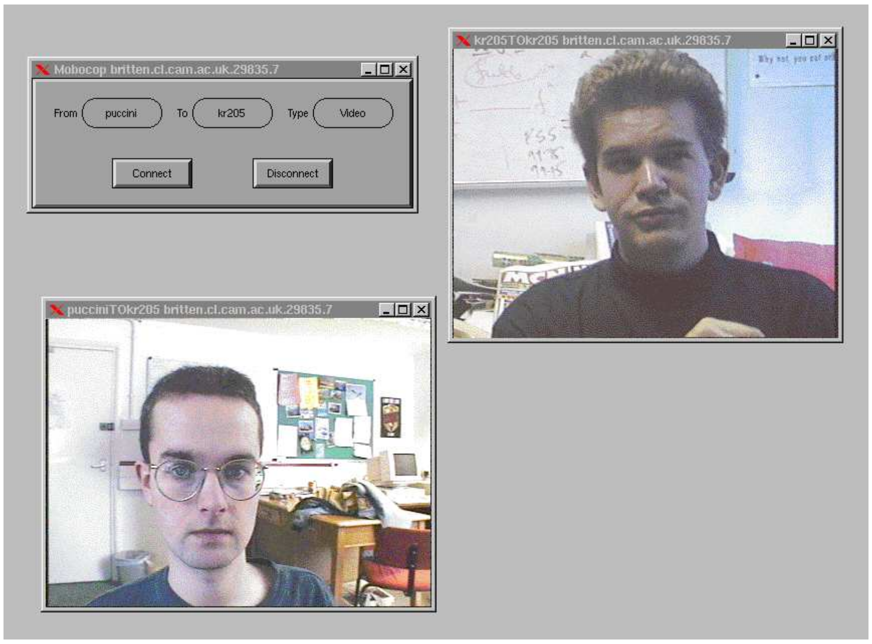

Figure 9: The Mobocop video conference manager

proximity to users rather than their name. Tube mobile agents have the ability to go directly to a particular Tube site or perform a traversal of a number of sites. Using the location-awareness information, it is possible instead to traverse the hosts nearest to one or a number of users. For example, a questionnaire about the time of a meeting can be sent to all proposed participants, one after another. The next iteration is signalled by clicking on a button.

For the second phase of experiments we carried out various multimedia experiments. One example is *follow-me* video and audio. Follow-me video is useful if one wants to monitor a particular person continuously, e.g. a child-minding application. The video display window follows the application user, and the live video source follows the person being monitored. The follow-me audio application allows music to follow a user around their environment. Additonally, we have developed a follow-me Netscape Web browser, which pops up showing the last used Web page whenever a user moves.

The third phase of experiments involved developing a complex illustration of our architecture in action: the mobile multimedia conferencing application, shown in figures 4-6. We built two versions, centralised and decentralised, in order to study their comparative behaviour and performance. In the decentralised version, each user has a mobile agent controlling their source and sink endpoints. The agent registers interest in the movement of its own user. When the user moves, the agent receives an event detailing his/her new location, saves the state of its objects, moves itself to the new location and recreates the objects using the saved state information. The objects are responsible for reconnecting themselves to the media endpoints of other users.

Instead of using mobile agents, the centralised conference, known as *Mobocop* (see figure 9), holds all information about connections in an agent managed by the chairman. The Mobocop agent only moves to follow the chairman. This version has the advantage that it is much simpler to author and user movement is managed slightly more rapidly (see below). The decentralised version is more elegant and scalable.

7.2 Measurements

Timings were taken using the conference applications described in the previous section. An audio and video conference was set up between three people with access to three DEC Alpha computers, each in a different room, equipped with a video camera, microphone and speakers, connected to a local-area ATM network and running a Tube site under Digital Unix. One of the users was designated chairman and the Mobocop application was able to follow him as he moved.

Each user was able to see and hear the others as they moved between rooms. All the timings given below include a constant overhead of 2 seconds for the active badge system to report badge sightings to Tube sites.

The average time taken for a user's multimedia objects to appear at his destination when he moved between rooms was 8.9 seconds (815 milliseconds to close the old connections, 1.614 seconds to destroy the old objects, 2.934 seconds to create the new objects and 1.509 seconds to recreate the connections between the new objects). The time taken between him arriving at a new room and the other users seeing him there was 9.2 seconds on average. Movement of the controlling agent adds an additional 500 milliseconds to this total. This is experienced in the centralised version only when the chairman moves and in the decentralised version when anyone moves.

These times proved adequate for conferences with only sporadic movement. They are also respectable given the heavy load placed on the computers. Each user had outgoing audio and video connections to the other users and to himself. That is, each user had six outgoing connections and there were eighteen connections in total. The video objects had to send and receive 1.33 megabytes of data over each connection per second (there were 12 frames per second, each of which was 384 by 288 pixels in 8-bit colour). The audio objects had to send and receive 22 kilobytes of data over each connection per second (it was sampled at 11kHz, using 16 bits per sample).

The DEC Alpha computers were overloaded by this amount of data, so that the playing of audio became intermittent and the display of video jerky. Handling of user mobility had to compete with the processing of this multimedia data, without help from the Digital Unix scheduler which cannot impose quality of service guarantees. Use of an operating system designed to provide such guarantees would have helped here [L$^+$96]. Performing stream handoff at a lower level, instead of tearing down and re-building connections, would have reduced the amount of software involved in re-routeing data (see section 4.1). Using a separate device for sending data from video cameras onto the network [FOR97] would have reduced the load placed on the computers.

Note that the related projects cited in section 1.2 give no performance measurements. Our system is experimental and we plan to work on improving these times significantly.

8 Summary and Conclusions

We have investigated general-purpose support which allows applications to follow mobile users, for example by hopping to the nearest computer. We have built and deployed a framework to test our ideas.

A location monitoring system is required to track users and to provide information about resources available in various locations. The provision of filtering support in a service, so a client is only informed about location events of interest, is realistic and cuts down on network traffic and client processing. Clients should be informed asynchronously when events of interest occur.

A mobile agent system allows script-level programs to move from site to site whilst retaining user interface and execution state. Multimedia endpoints can be made mobile by providing support for the saving of stream connections and their re-establishment after movement. A mechanism for locating mobile objects such as agents and multimedia endpoints is required. Standard trading technology can be used here, assuming an appropriate naming scheme and a policy of indicating objects which have moved to another domain.

Our integrated architecture places mobile agents at the centre of application coordination. We have found this paradigm works well since scripts are easily prototyped and agent support works flexibly between heterogeneous domains. We have also found that compiled media endpoints, built using distributed programming mechanisms, such as CORBA, are appropriate as mobile media endpoints. Our initial experiments are encouraging; we have found we can support complex applications, such as a multimedia conference in which users can move between hosts and connections to other users are re-established automatically.

References

[AG96] K. Arnold and J. Gosling. *The Java Programming Language.* Addison-Wesley, 1996.

[Bat96] J. Bates. A framework to support large-scale active applications. In *SIGOPS European Workshop.* ACM, 1996.

[BB95] J. Bates and J. Bacon. Supporting interactive presentation for distributed multimedia applications. *Multimedia Tools and Applications*, 1(1):47–78, March 1995.

[BBHM95] J. Bacon, J. Bates, R. Hayton, and K. Moody. Using events to build distributed applications. In *2nd International Workshop on Services for Distributed and Networked Environments.* IEEE, 1995.

[BC95] K. Bharat and L. Cardelli. Migratory Applications. In *Proc. ACM Symposium on User Interfaces Software and Technology*, Pittsburgh, USA, November 1995.

[DHDD96] D. DeRoure, W. Hall, H. Davis, and J. Dale. Agents for Distributed Information Management. In *Presented at Practical Applications of Intelligent Agents and Multi-Agents (PAAM '96)*, London, UK, 1996.

[FOR97] FORE Systems, Inc. *FORE ATM Video Solutions*, 1997.
URL *http://www.nemesys.co.uk/products/video/index.html*.

[Gro91] Object Management Group. The Common Object Request Broker: Architecture and specification. Technical Report 91.9.1, Object Management Group, December 1991.

[Hal97] David Halls. *Applying Mobile Code to Distributed Systems.* PhD thesis, University of Cambridge Computer Laboratory, June 1997.

[HH94] A. Harter and A. Hopper. A distributed location system for the active office. *IEEE Network*, 8(1), 1994.

[HR97] D. Halls and S. Rooney. Controlling The Tempest: Adaptive management in advanced ATM control architectures. Position paper, January 1997.

[L+96] I. Leslie et al. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communication*, September 1996.

[Nel97] G. Nelson. *System Support for Location Aware Computing.* PhD thesis, University of Cambridge Computer Laboratory, 1997. In preparation.

[PH94] J. Porter and A. Hopper. An ATM Based Protocol for Wireless LANs. Technical Report 94-2, Oracle Olivetti Research Limited, Cambridge, 1994.

[Pop96] S. Pope. *Application Support for Mobile Computing.* PhD thesis, Jesus College, University of Cambridge, October 1996.

[Raj95] B. Rajagopalan. Mobility Management in Integrated Wireless ATM Networks. In *Proceedings of Mobicom*, Berkeley, 1995. IEEE.

[RBMH94] T Richardson, F Bennett, G Mapp, and A Hopper. Teleporting in an X Window System Environment. *IEEE Personal Communications*, August 1994.

[RGK96] D. Rus, R. Gray, and D. Kotz. Autonomous and adaptive agents that gather information. In *Proceedings of the AAAI 96 Workshop on Intelligent Adaptive Agents*, 1996.

[Roo97] S. Rooney. *The Structure of Advanced ATM Control Architectures.* PhD thesis, University of Cambridge Computer Laboratory, 1997. In preparation.

[Ser94] M. Serrano. *Bigloo User's Manual.* INRIA, B.P. 105, Rocquencourt, 78153 Le Chesnay Cedex, France, 1.7 edition, 1994.

[W^{+}94] S. Wray et al. The Medusa applications environment. In *1st IEEE International Conference on Multimedia Computing and Systems*, 1994.

[WH96] M. Wachowicz and S. G. Hild. Combining location and data management in an environment for total mobility. In *Proceedings of the International Workshop on Information Visualization and Mobile Computing*, Rostock, Germany, February 1996.

[ZO96] T. Zhao and M. Overmars. XForms.
URL `http://bragg.phys.uwm.edu/xforms`,
1996.

Acknowledgements

We acknowledge EPSRC's support for this work, through Dr Halls' graduate studentship. Dr Bates would like to thank Michael and Morven Heller and St Catharine's College, Cambridge for supporting his research. All three authors extend their thanks to ICL for general support of their research group.