Quality of service support in a mobile environment: an approach based on tuple spaces

G.S. Blair, N. Davies, A. Friday and S.P. Wade
Distributed Multimedia Research Group,
Department of Computing,
Lancaster University, Lancaster, LA1 4YR, U.K.
Tel: +44 (0) 1524 65201
E-mail: most@comp.lancs.ac.uk

Abstract

There has recently been considerable interest in quality of service management architectures for high speed networks. In contrast, however, there has been less research on appropriate architectures for mobile computing. This paper addresses the issue of quality of service in a mobile environment. In particular, we describe a distributed systems platform, Limbo, which is intended to support the development of demanding, mobile-aware distributed applications in a heterogeneous networking environment. This platform is based on the concept of tuple spaces and is extended with support for quality of service management. The emphasis of quality of service management is on monitoring and adaptation, although support is also provided for other functions such as admission control and resource reservation. The paper argues that there are significant benefits from using a tuple space paradigm in such an environment, particularly in terms of the ability to adapt to changes in network connectivity or the more general environment.

Keywords

Mobile computing, quality of service management, adaptation, tuple spaces.

1 INTRODUCTION

There has recently been considerable interest in the development of architectures for quality of service (QoS) management (Hutchison, 1994). Such architectures are generally motivated by the requirement to support multimedia applications in a networked environment and enable the user to specify their nonfunctional requirements in the form of a QoS contract. The QoS management architecture then provides a range of functions to achieve and sustain the QoS requirements as specified in this contract. Examples of such functions include negotiation, admission control, resource reservation, monitoring and renegotiation. By selectively applying such functions, it is possible to provide varying levels of QoS guarantee (such as best effort, predictive or deterministic).

Existing QoS architectures, however, generally make implicit assumptions about the underlying network. In particular, they assume that the network offers a relatively stable environment in terms of connectivity, available throughput, delay and delay jitter. They ensure sufficient resources are available through admission control and resource reservation and compensate for *small* changes in the underlying network service through software feedback mechanisms. However, such approaches are not appropriate in a mobile environment, where the end system may roam from network to network experiencing *dramatic* changes in throughput and connectivity. In such systems, the emphasis for QoS management must change from providing levels of guarantees to supporting monitoring and adaptation.

This paper considers the role of quality of service management in a mobile environment. We are particularly interested in QoS management to support demanding distributed applications (requiring multiparty multimedia communications). We propose a novel approach based on tuple spaces, exploiting the properties of time and space decoupling exhibited by such systems. We demonstrate how tuple spaces can be enhanced to support QoS management functions within an overall QoS architecture.

2 QOS AND MOBILITY

2.1 Heterogeneous Networks

Early research in mobile computing focused on mechanisms to enable access to services across wireless networks such as GSM. Current research, however, is investigating techniques to access services over a *heterogeneous* networking infrastructure where the precise network will vary over time (Katz, 1994). Given such a heterogeneous infrastructure, the most dominant characteristic of mobile computing is *change*. For example, end systems can experience dramatic changes in the available bandwidth and bit error rates, or discover new functionalities are available in the underlying network. At the most extreme, end systems can move from periods of high to partial connectivity, through to complete disconnection.

The approach in existing QoS architectures is, where possible, to mask out change and hence present a *stable* service to the application (effectively providing a level of network transparency). This is achieved through combining QoS functions such as admission control, resource reservation and maintenance. In our view, such an approach is not sustainable in a mobile environment. In such an environment, the emphasis should not be on transparency but on making information available to applications and empowering applications to make the necessary changes. Such an approach relies on the provision of QoS management functions supporting monitoring and adaptation. To complement this, the underlying system must be amenable to adaptation, including the ability to adapt to periods of disconnection. This is not to say that existing approaches to QoS management are invalidated. Rather, strategies such as admission control and resource reservation can be used for component networks and can provide guarantees while the end system remains connected to that particular network. The architecture must, however, accommodate changes in underlying network service through the additional mechanisms we seek.

2.2 Other Factors

Mobile hosts must also deal with other changes in their environment. One important environmental factor is that of power availability and consumption. Despite significant developments in areas such as low-power processors, the availability of power remains an important concern for mobile users. By providing access to power information, applications may utilise hardware power saving functionality as appropriate.

Another important environmental factor is the current physical location of the end system. Such information can be exploited in numerous ways. For example, such information can be used for the *proximate* selection of services (Neuman, 1993) whereby services are selected on criteria such as physical distance from the mobile host and current cost of access. Schilit has also proposed the more general concept of context-aware applications (Schilit, 1994). One example put forward is that of a memory jogger which allows users to specify that a certain message should be displayed when a series of location and temporal based conditions are met, e.g. 'remind me to do task t next time I meet users x and y'.

In essence, we are proposing to broaden the scope of QoS monitoring and adaptation to include general environmental characteristics such as power availability, rate of consumption of power, physical location and time. In this way, applications can adapt to changes in the underlying environment. However, this introduces a problem. Existing systems provide access to QoS functions through abstractions over communications. For example, more recent distributed systems platforms support the concept of bindings as abstractions over communications services. Crucially, bindings generally offer interfaces for QoS management functions such as monitoring and adaptation. This implies, however, that there

would be one means of accessing and disseminating information about the network and another means of dealing with other environmental characteristics.

The close liaison between communications and QoS implies that communications must be established before QoS information can be obtained. This approach prohibits the system from making speculative decisions. For instance, the system cannot check the QoS between itself and some remote system to determine whether it wishes to contact that system without first establishing the binding to obtain a handle for gathering the QoS information. We therefore seek a uniform approach to QoS management which accommodates both communications and more general environmental considerations. Furthermore, such an approach need not be dependent on the existence of actual bindings between hosts.

2.3 Overall Analysis

In our opinion, the most crucial role of QoS management in mobile environments is to facilitate adaptation. To support this, the underlying distributed systems platforms must collate and manage QoS information for presentation to higher layers, enabling feedback and control throughout the architecture. In addition, we believe the close association of QoS with communications present in existing approaches to be less than ideal. The remainder of this paper describes a new distributed systems platform, Limbo, based on the tuple space paradigm which can provide a uniform mechanism for integrating QoS into applications. Firstly, however, it is necessary to provide some background information on tuple spaces and the rationale for tuple spaces in distributed mobile environments.

3 BACKGROUND ON TUPLE SPACES

3.1 What are Tuple Spaces?

Tuples comprise of collections of typed data fields. Each field is either an *actual*, if it contains a value, or a *formal*, if it does not. Collections of (possibly identical) tuples exist in shared data objects called tuple spaces. Tuples can be dynamically deposited in and removed from a tuple space, but not be altered while resident in it. However, changes can be made to a tuple by withdrawing it from the tuple space, amending and then reinserting it (Gelernter, 1985b). Tuple spaces are shared between collections of processes, all of which have equal access to its contents.

The tuple space paradigm was conceived by researchers at Yale (Gelernter, 1985a) and was embodied in a coordination language called Linda. Linda is not a standalone computational language, instead Linda operators are embedded in host computational languages (e.g. C or Java). The model defines four basic operators:

- `out` inserts a tuple, composed of an arbitrary mix of actual and formal fields, into a tuple space.

- `in` extracts a tuple from a tuple space, using its argument as a template, or *anti-tuple*, against which to match. Actuals match tuple fields if they are of equal type and value; formals match if their field types are equal. If all corresponding fields match the tuple is withdrawn and any actuals assigned to formals in the template. Tuples are matched nondeterministically and `in` blocks until a suitable tuple can be found.
- `rd` is a version of `in` which does not withdraw matches from the tuple space.
- `eval` is similar to `out` but creates *active* rather than *passive* tuples, spawning separate processes to evaluate each field. Such active tuples subsequently evolves into ordinary, passive tuple resident in the tuple space.

3.2 Why Tuple Spaces?

The tuple space paradigm has two very important properties:

- *Time decoupling*: In tuple spaces, tuples are persistent and the sender and receiver do not have to exist at the same time (Bjornson, 1991). A sender may place a tuple into tuple space and then terminate execution. At some point in the future, a receiver can invoke the `in` or `rd` primitives and obtain this tuple. It is quite possible that the receiver did not exist when the tuple was created.
- *Space decoupling*: The sender does not need to be aware of the identity of the receiver (or multiple receivers if `rd` operations are used). The sender simply places the tuple in tuple space and expects one or more processes to access the tuple at some point in the future. Their identity is unknown (i.e. communication is *anonymous*). It is, however, also possible to produce tuples for an identified consumer process, termed *directed communication*, by encapsulating destination information in tuples. Several such schemes have been proposed, including an approach based on Amoeba ports (Pinakis, 1992).

It is recognised that these properties support the construction of flexible and fault-tolerant parallel applications. We contend that the same properties can support the construction of distributed applications in a mobile environment. The majority of existing distributed system platforms are based on the remote procedure call (RPC) paradigm. In this style of interaction, clients locate an appropriate service through, for example, a trading service. They then bind to this service and invoke operations on it over a period of time. This makes the assumption that the sender and receiver exist at the same point in time and synchronise in order to exchange data (with the precise degree of synchronisation varying from service to service). Consequently, such systems have problems during both long and short periods of disconnection. A number of platforms overcome this deficiency by introducing an element of buffering. In this way, the platforms attempt to maintain RPC semantics in the face of variations in network connectivity. Tuple spaces, however, overcome this problem by decoupling senders and receivers in time.

Space decoupling also offers benefits for mobile computing. The advantage of space decoupling is that there is no binding between a client and a server. Rather, a

number of processes can service a particular request (as specified in a tuple). This increases fault-tolerance as the architecture naturally deals with failure of particular nodes. In a mobile environment, this property can be exploited to deal with the unavailability of particular services. For example, an end user can request a print service by inserting an appropriate tuple in tuple space. Any print service can respond to this request. As print services become unavailable, other services can take over. In this example, it would also be possible to exploit location information (if available) to dynamically select the nearest service.

The combination of space and time decoupling implies that there is no direct concept of a connection in tuple spaces. Consequently, QoS requirements must be associated with the overall tuple space or with individual tuples. The result of this is that the system has more flexibility to reconfigure itself to attain the desired level of QoS. We return to this aspect in section 4.2.3 below.

Finally, tuple spaces provide a very natural means of disseminating information to a variety of potential recipients. The paradigm directly supports multiparty communications and encourages the ready availability of global information. We exploit this later property to make QoS management information available to any interested parties. This aspect is discussed in more detail in section 4.2.2.

4 A DISTRIBUTED SYSTEMS PLATFORM BASED ON TUPLE SPACES

4.1 General Approach

We have designed a new distributed system platform, called Limbo, to support demanding distributed applications in a mobile environment. This platform is based on the concept of tuple spaces, augmented by *mobile agents*. Mobile agents reside above tuple spaces and interact with the tuple space, carrying out a particular computation. They can be written in any language supporting the creation of mobile code (e.g. Java or TCL) as long as the language provides a conformant interface to tuples spaces (thus supporting the requirement for interoperability and portability required by modern middleware platforms). Note that agent mobility is naturally supported by the tuple space model; agents simply stop interacting with the tuple space, relocate and then restart their interaction. As an enhancement, agents are also generally *stateless*; all state is assumed to be in the tuple space. As tuples are persistent and globally available, replicating agents is trivial. In other words, there is no need for a consistency algorithm; this is directly provided by tuple spaces. Agents are classified as system agents, already provided in the environment, and application agents, introduced into the environment by the programmer. This distinction is, however, not particularly rigid. The application programmer is free to introduce additional system agents into the environment.

We support the following key extensions to this basic architecture: i) multiple tuple spaces which may be specialised to meet application level requirements, e.g.

consistency, security or performance, ii) an explicit tuple type hierarchy supporting dynamic subtyping, and iii) a QoS management architecture supporting monitoring and adaptation. We look at the first two extensions in this section and the QoS management architecture in section 4.2.

The original tuple space model was designed to support parallel programming on shared-memory multi-processor systems and features a single, global tuple space. More recent models have proposed the introduction of multiple tuple spaces to address issues of performance, partitioning and scalability (Hupfer, 1990). This is important for performance in a distributed environment and critical in an environment where communications links are costly and unreliable.

We provide a class of system agent which can create new tuple spaces for applications. These tuple spaces are configurable to meet application specific requirements (Hupfer, 1990). For example, in addition to general purpose tuple spaces we allow the creation of tuple spaces with support for security (user authentication), persistence and tuple logging (for accountability in safety critical systems). Crucially, it is also possible to create a range of QoS-aware tuple spaces.

In order to create a new tuple space clients communicate with the appropriate system agents via a *universal tuple space*. Clients specify the characteristics of the desired tuple space and place a `create_tuple_space` request into the common tuple space. The appropriate system agent accesses this tuple, creates a tuple space with the required characteristics and places a tuple of type `tuple_space` in the common tuple space. The fields in this tuple denote the actual characteristics of the new tuple space (which may be different to those requested in best-effort systems) and a handle through which clients can access the new space.

Agents can select between tuple spaces by means of a `use` primitive. This primitive communicates with a *membership agent* through the universal tuple space, returning a handle if the tuple space exists and certain other criteria are met. The precise criteria vary from tuple space to tuple space and can include checks on authentication and access control functions or relevant QoS management functions. We return to this latter aspect in section 4.2 below. Handles can later be discarded by an agent using a `discard` primitive. This places an appropriate tuple in the universal tuple space so that the membership agent can take appropriate steps. A tuple space can be destroyed by placing a tuple of type `terminate` in it. Such tuples are picked up by system agents within the tuple space which invoke a system function to gracefully shut down the tuple space.

Note that this model can be applied recursively. It is possible to access a tuple space through the universal tuple space and find this tuple space has system agents supporting the creation of and subsequent access to further tuple spaces. This recursive structure provides a means of creating private worlds with fine grain access control.

In our model, we allow an optional type to be associated with each tuple. Typed tuples can be organised to form a hierarchy by establishing subtyping relationships between them. The benefits of subtyping in a distributed environment have been comprehensively investigated within the ODP community as part of their work on

interface trading (ISO/ITU-T, 1997). In this model subtyping enables added flexibility when matching service offers to client requests.

4.2 QoS Support

4.2.1. Overview

As mentioned above, the Limbo architecture supports the creation of multiple tuple spaces through a number of system agents. In order to support quality of service, we provide a range of system agents to create *QoS-aware tuple spaces*. A QoS-aware tuple space is one that supports a number of QoS management functions (the precise details vary from system agent to system agent). This architecture is completely extensible; the programmer is free to define new system agents enabling the creation of different styles of QoS-aware tuple spaces.

QoS-aware tuple spaces will typically have a more sophisticated membership agent which will perform certain QoS-management functions in order to provide guaranteed or predictive classes of service. In particular, the membership agent will cooperate with admission control and resource reservation agents in order to provide the required class of service. The membership agent will only issue a handle if the admission control test is passed and resources requested available. The two system agents can implement any of the available algorithms for admission control and resource reservation. In addition, the architecture is completely open. Different tuple spaces can have different admission control and resource reservation agents. The programmer is also free to extend the available services, hence offering a new class of tuple space. Note that these agents are not compulsory for QoS-aware tuple spaces; if not present, the tuple space will offer best-effort service. In general, this architecture allows a wide range of tuple spaces to be created offering best-effort, predictive or guaranteed classes of service for a variety of different traffic types (e.g. constant bit rate, variable bit rate, periodic or bursty traffic).

In QoS-aware tuple spaces, individual tuples can also have designated QoS attributes specifying the *expiration time* and the *priority* of the tuple. In the case of a tuple which is the subject of an `out` operation, the expiration time refers to the time the tuple is allowed to reside in the tuple space before being deleted. In the case of a tuple template used as an argument to `in` or `rd` operations, the expiration time refers to the time for which the request can block before timing out. The priority attribute is used by the tuple space implementation in both the end system and network to reorder traffic upon congestion. (Note that we are also currently debating whether to replace the priority field with a deadline or indeed with a combination of both values as proposed, for example, in (Nieh, 1995)).

The approach described above is designed to resolve the conflict between the need for QoS-guarantees on services and the problems that can occur in a mobile environment. With appropriate admission control and resource reservation functions, a tuple space can guarantee to make tuples available to agents using that tuple space with certain QoS guarantees if and only if significant changes do not occur in the underlying communications infrastructure. Thus, although the

approach is asynchronous and connectionless in nature, tuple spaces can be used to provide real-time guarantees on, say, a video stream (again, provided the available network QoS remains relatively stable). In a mobile environment this cannot, however, be guaranteed as nodes will roam between networks experiencing dramatic changes in the available bandwidth. In such cases, it is both undesirable and impossible to continue with the desired quality of service level. The advantage of tuple spaces (coupled with mobile agents) is that, in such circumstances, they naturally support reconfiguration. The persistent nature of tuple spaces also helps to overcome periods of disconnection. However, to fully exploit this potential, support must be provided for QoS monitoring and adaptation, as described below.

4.2.2. Support for Monitoring

Every Limbo site has an associated management tuple space together with a number of QoS monitoring agents. These agents monitor key aspects of the system and inject tuples representing the current state of that part of the system into the management tuple space. The precise configuration of QoS monitoring agents can vary from site to site. As above, the architecture is open and new QoS monitoring agents can readily be added to the configuration. Monitors can observe events at various points in the system including the: rate of injection of tuples into a given tuple space, rate of access to tuples in tuple space (through `in` or `rd` operations), total throughput currently achieved from that node, the cost of the current channel, level of connectivity, power availability, rate of consumption, processor load and current physical location of that node. In this way, the architecture deals uniformly with a range of QoS parameters relating to both communications and the general environment. In addition, the architecture can provide information relating to a particular tuple space or to the node in general.

This architecture has the advantage that information pertaining to a node may be globally accessible. By placing this information in tuple space, other sites can access this information through that tuple space. This means agents on different nodes can find out about the location of a particular site, its current processor load, the throughput it is experiencing, etc. A site can, instead, keep this information private by selecting a membership agent which prevents access from other sites.

4.2.3. Support for Adaptation

The Limbo architecture supports a variety of mechanisms for adaptation. Such mechanisms are typically employed on detecting a significant change as a result of QoS monitoring. One of the main techniques for achieving adaptation is that of *filtering agents*. Filtering agents are a special form of *bridging agent*. A bridging agent links arbitrary tuple spaces together and controls the propagation of tuples between them. In its simplest form, a bridging agent is a process which performs repeated `rd` operations on one tuple space and `out`'s corresponding tuples into a second tuple space. Filtering agents are then bridging agents which perform transformations on tuples to map between different levels of QoS. They rely on typing information to identify the subset of tuples to be filtered.

Filtering agents could be used to translate between different media formats. More commonly they are used to reduce the overall bandwidth requirements from the source to target tuple spaces. For example, a filtering agent could act between two tuple spaces dealing with MPEG video and only propagate I-frames to the target tuple space. The filtering agent could also perform more aggressive bandwidth reduction, for example by performing colour reduction on I-frames. The importance of filtering agents is that they allow parallel tuple spaces to offer the same service, e.g. the propagation of video frames, at radically different levels of QoS. An agent can therefore select between the different levels depending on their level of connectivity. On detecting a drop (or an increase) in available bandwidth, they can switch to another tuple space.

The architecture also supports several other forms of adaptation. For example, on detecting QoS violations, a sending agent can choose to adapt the rate at which tuples are injected into tuple space. This is, however, a rather crude mechanism in an environment supporting multiple receivers with potentially different levels of connectivity. More interestingly, a receiver can selectively `in` or `rd` certain types of tuple and ignore others on detecting a drop in *their* connectivity. For example, they can select I-frames only and ignore P- and B-frames (achieving a similar effect as above). Similarly, they can select base encodings in hierarchical encoding schemes. With the appropriate allocation of priorities on these tuple types, it is also likely that the underlying implementation will be able to discard the lower priority tuples on detecting congestion, implying that they need not be transmitted over a lower bandwidth link.

Finally, mobile agents provide considerable scope for adaptation. For example, it is possible to migrate an agent to a different site. Similarly, it is possible to create a new agent to act as your proxy on a well-connected site. Finally, with replicated agents, it is possible to access a replica that is either cheaper to access or is at a location with better levels of connectivity.

5 IMPLEMENTATION DETAILS

The platform has a decentralised architecture and is designed to be scalable: the state of each tuple space is decentralised across participant sites. Essentially, each site accessing a given tuple space maintains a local cache containing a (partial) view of that tuple space. The architecture uses a multicast protocol to manage consistency of the different views. Crucially, this uses the Scalable Reliable Multicast (SRM) protocol which operates as a lightweight service on top of IP multicast and offers a highly scalable means of achieving reliable group communications within the Internet (Floyd, 1995). The protocol has previously been used to implement a variety of services including a collaborative whiteboard tool, wb (Floyd, 1995) and a distributed file system, Jetfile (Grönvall, 1996).

The protocol operates as follows. Every tuple space is allocated an IP multicast group address. SRM then uses IP multicast to deliver messages to all interested parties (i.e. the sites currently using that tuple space). IP multicast is, however,

unreliable and hence it is necessary to deal with lost messages. The most obvious approach would be to have acknowledgements from every receiver in the group, but this would suffer from acknowledgement explosion and hence compromise scalability. To overcome this problem, SRM does not acknowledge receipt of messages. Instead, on detecting missing messages, the protocol issues a repair request to the group. Any member of the group can respond to this request and sites wait for a random period based on their perceived distance from the requester before responding. They also snoop to see if anybody else has responded in the meantime. With this approach, it is likely that the closest site will respond to the repair request by retransmitting the lost data.

The SRM protocol adopts an application level framing philosophy, leaving key policy decisions to the application. In particular, applications must detect missing messages and request repairs as necessary (the protocol simply provides the *mechanism* to enable this to happen). We have therefore layered a protocol on top of SRM to maintain consistency of multiple views of a tuple space. This protocol relies on the use of a monotonically increasing logical tuple timestamps in order to allow detection of missing messages. The protocol also exploits the ability to snoop in SRM to improve the performance of the protocol.

The benefits of this approach are that SRM is inherently scalable and highly efficient, being directly supported by IP multicast. The resultant platform is also portable to any network environment supporting IP multicast. The QoS architecture can also exploit existing Internet protocols for resource reservation. In particular, we assume the availability of RSVP (or equivalent) in our prototype implementation to provide QoS guarantees. This can then be supplemented by other QoS management functions such as admission control, monitoring and adaptation as described above.

6 CONCLUDING REMARKS

This paper has considered the role of QoS management in a mobile environment, where end systems are interconnected by heterogeneous networks. The paper argues the emphasis of QoS management in such environments should be on support for monitoring and adaptation. Furthermore, the scope of QoS management should be extended to include a variety of environmental factors including the level of connectivity, the current cost of connectivity, power availability and rate of consumption, and the current physical location and time.

We have described the Limbo distributed system platform which is intended to operate in a mobile environment. This platform is based on the tuple space concept and enables the creation of QoS-aware tuples spaces supporting admission control, resource reservation, monitoring and adaptation. This architecture can provide guarantees while underlying levels of connectivity remain fairly constant but also supports adaptation when significant changes occur for one or more end systems.

The advantages of the proposed approach are i) guarantees can be offered as with conventional architectures (assuming a relatively stable network), ii) there is

a single means of monitoring QoS, iii) all monitoring data is potentially globally available, iv) tuple spaces naturally support adaptation through mechanisms such as agent migration or replication, or selective tuple filtering, and v) tuple spaces accommodate periods of disconnection through the persistent nature of tuples.

7 ACKNOWLEDGEMENTS

The work described in this paper was carried out under the auspices of the 'Supporting Reactive Services in a Mobile Computing Environment' project (EPSRC Grant No. GR/K11864).

8 REFERENCES

Bjornson, R., Carriero, N., Gelernter, D., Mattson, T., Kaminsky, D. and Sherman, A. (1991) Experience with Linda. *Technical Report YALEU/DCS/TR-866, Department of Computer Science, Yale University, New Haven, Connecticut, U.S.*

Floyd, S., Jacobson, V., McCanne, S., Liu, C. and Zhang, L. (1995) A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. *Proc. ACM SIGCOMM '95, Cambridge, Massachusetts, U.S., ACM Press,* 342-356.

Gelernter, D. (1985a) Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems,* **7(1)**, 80-112.

Gelernter, D., Carriero, N., Chandran, S. and Chang, S. (1985b) Parallel Programming in Linda. *Proceedings of the International Conference on Parallel Processing,* 255-263.

Grönvall, B., Marsh, I. and Pink,S. (1996) A Multicast-Based Distributed File System for the Internet. *Proc. 7th ACM SIGOPS European Workshop, Connemara, Ireland.*

Hupfer, S. (1990) Melinda: Linda with Multiple Tuple Spaces. *Technical Report YALEU/DCS/RR-766, Department of Computer Science, Yale University, New Haven, Connecticut, U.S.*

Hutchison, D., Coulson, G., Campbell, A. and Blair, G.S. (1994) Quality of Service Management in Distributed Systems. *Network and Distributed Systems Management (ed. M. Sloman), Addison-Wesley,* 273-302.

ISO/IEC 13235-1 | ITU Recommendation X.950 (1997) Open Distributed Processing - Trading Function: Specification.

Katz, R.H. (1994) Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Communications*, **1(1)**, 6-17.

Neuman, B.C., Augart, S.S. and Upasani, S. (1993) Using Prospero to Support Integrated Location-Independent Computing. *Proceedings of the USENIX Symposium on Mobile and Location Independent Computing, Cambridge, Massachusetts, U.S.,* 29-34.

Nieh, J. and Lam, M.S. (1995) Integrated Processor Scheduling for Multimedia. *Proc. 5th International Workshop on Network and Operating System Support for Digital Audio and Video, Durham, N.H.*

Pinakis, J. (1992) Providing Directed Communication in Linda. Proceedings of the 15th Australian Computer Science Conference, Hobart, Tasmania.

Schilit, B., Adams N. and Want R. (1994) Context-Aware Computing Applications. *Proc. 1st Workshop on Mobile Computing Systems and Applications (MCSA '94), Santa Cruz, California, U.S., IEEE Computer Society Press,* 85-90.

9 BIOGRAPHY

Gordon Blair is currently a Professor in the Computing Department at Lancaster University. He has been responsible for a number of research projects at Lancaster in the areas of distributed systems and multimedia support and has published over a hundred papers in his field. His current research interests include distributed multimedia computing, the impact of mobility on distributed systems and the use of formal methods in distributed systems development.

Nigel Davies graduated from Lancaster University in 1989 and later that year joined the Computing Department as a research associate. After a spell as a visiting researcher at the Swedish Institute of Computer Science (SICS) he returned to Lancaster, first as site manager for the MOST mobile computing project and subsequently as a lecturer in the Computing Department. His current research interests include mobile computing, distributed systems platforms and systems support for multimedia communications.

Adrian Friday graduated from the University of London in 1991. The following year he moved to Lancaster and participated in the MOST project involving Lancaster University and E.A. Technology. In 1996 he was awarded a Ph.D. for his work on 'Infrastructure Support for Adaptive Mobile Applications' and is currently a research assistant in the Computing Department.

Stephen Wade has been a research student in the Computing Department since graduating from Lancaster University in 1995. He is an active participant in the 'Supporting Reactive Services in a Mobile Computing Environment' project and is working towards his Ph.D. on 'Using an Asynchronous Paradigm for Mobile Distributed Computing'.