Limbo: A tuple space based platform for adaptive mobile applications

N. Davies, S.P. Wade, A. Friday and G.S. Blair

Distributed Multimedia Research Group,
Computing Department,
Lancaster University,
Lancaster, LA1 4YR,
U.K.
Telephone: +44 (0)1524 594337
E-mail: most@comp.lancs.ac.uk

Abstract

Mobile computing environments are characterised by significant and rapid changes in their supporting infrastructure and, in particular, in the quality-of-service (QoS) available from their underlying communications channels. Applications which can operate in these environments and take advantage of changing QoS require distributed systems support platforms. The current state-of-the-art in such platforms attempt to provide synchronous connection-oriented programming paradigms reflecting their fixed network origin. In this paper we argue that these paradigms are not well suited to operation in a mobile environment and instead propose a new platform called Limbo based on the tuple space communications paradigm. The design of Limbo is presented together with details of two prototype implementations. The use of the platform to re-engineer a number of existing adaptive mobile applications is also discussed.

Keywords

Mobile Computing, Distributed Systems, Adaptive Applications, Tuple Spaces, Linda.

1 INTRODUCTION

Mobile computing environments are characterised by significant and rapid changes in their supporting infrastructure and, in particular, in the quality-of-service (QoS) available from their underlying communications channels. Previous research has demonstrated that in order to operate effectively in mobile environments applications are required to adapt in response to these changes [Davies,94a], [Katz,94]. Such applications are termed *adaptive applications*.

Adaptive applications require distributed systems support, and a number of platforms have recently been developed which address this requirement. Example platforms include Mobile DCE [Schill,95], the MOST platform [Davies,94b] and the Rover Toolkit [Joseph,95]. These mobile platforms attempt to provide application programmers with traditional computational models and communications' semantics consistent with those normally found in platforms designed for fixed networks. In particular, the three major mobile distributed systems platforms all implement RPC semantics with (to a greater or lesser extent) additional interfaces allowing

applications to monitor and adapt to changes in QoS. Clearly, procedure call semantics can not be provided during periods when mobile hosts are experiencing very low levels of communications QoS or during periods of disconnected operation. To address this problem the platforms all include support for buffering remote procedure calls during periods of disconnection ready for transmission when the network QoS improves.

In this paper we argue that the procedure call paradigm is not well suited for use in mobile environments and suggest an alternative paradigm based on *tuples* and *tuple spaces* [Gelernter,85a]. This paradigm has been widely used in the parallel computing community but there has, to our knowledge, been no work on applying the paradigm in mobile environments. We describe the design, API and use of a tuple space based platform for mobile computing called *Limbo*. The platform includes support for QoS monitoring and control by adaptive applications.

Section 2 presents a critique of the three foremost mobile distributed systems platforms. Section 3 then provides background information on the tuple space paradigm and section 4 presents the design for our new platform. The use of this platform to support a number of existing mobile applications is then described in section 5. Finally, section 6 contains our concluding remarks.

2 CRITIQUE OF MOBILE DISTRIBUTED SYSTEMS PLATFORMS

To date there have been three significant research efforts aimed at producing general purpose distributed systems platforms for mobile environments. In the following sections we briefly review each of the resulting systems and then discuss their commonalities and shortcomings.

2.1 Mobile DCE

The Mobile DCE initiative at the University of Technology, Dresden aims to augment a standard DCE platform with new features for operation in a mobile environment.

The overall system architecture is based on the concept of domains. These are logical groupings of machines with shared resources managed by a domain manager. Mobile clients move between domains and hence have access to different resources. Manager processes on each client interact with the domain managers and a matrix specifying resource characteristics in order to ensure service provision as the client changes domains. In more detail, as clients cross domain boundaries their managers decide (for each service) whether to continue remote access to a service in the original domain, to re-bind to a new service in the destination domain or, during periods of disconnection, to emulate the service and replay messages when connectivity is restored.

Mobile DCE has been implemented under the Windows/NT operating system and a number of applications have been developed including mobile e-mail. The use of the industry standard DCE/Microsoft RPC protocols allow the platform to be integrated with existing applications.

2.2 The MOST Platform

Lancaster University's MOST platform provides support for adaptive mobile applications within an Open Distributed Processing (ODP) [ISO,92] based framework. The platform augments an existing ODP compatible platform called ANSAware [APM,89] with new services, protocols and API calls. In particular, the platform incorporates a new protocol called QEX [Friday,96] which is able to adapt to changes in the QoS of its underlying communications infrastructure and pass this information on to interested client applications.

The QEX protocol is layered above a low-level service called S-UDP which provides dial-up UDP connections over GSM. S-UDP and QEX both allow messages to be tagged with deadlines and messages from the mobile host to the fixed network are sent in earliest-deadline-

first order. The system uses the message deadlines to determine when to establish and break connections. Furthermore, messages can be buffered during periods of disconnection until either they are sent or their deadlines expire (in which case an exception is raised at the client).

The MOST platform has been implemented on Sun workstations and notebook PCs running a variety of flavours of UNIX and using a range of communications technologies including GSM. The platform has been used to support a range of mobile applications including e-mail, a collaborative geographic information system and a job dispatch application for field engineers.

2.3 Rover

The Rover toolkit from M.I.T. is designed to support the development of mobile applications. This support is based on the twin notions of relocatable data objects and queued remote procedure calls (QRPCs). In essence, the platform allows the creation of data objects with well defined interfaces which can be migrated at run-time between the mobile client and servers on the fixed network. This allows decisions regarding application configuration and the client-server computation trade-off to be made (and re-evaluated) at run-time as the network QoS and resource availability change. Communication between objects is carried out using the toolkit's QRPC protocol. In addition to being able to re-bind to objects which have migrated, QRPC also provides support for periods of disconnection by buffering messages destined for remote sites until network connectivity is restored.

A number of applications have been ported to the Rover toolkit including a web browser and an e-mail application. However, unlike both MOST and Mobile DCE, Rover is not based on an existing standard and applications must be re-engineered to operate in a mobile environment.

2.4 Discussion

All of the above platforms offer mobile clients connection-oriented RPC-based communications. The implementations of these communications services all relax the synchronous nature of RPC interactions by allowing messages to be buffered (Mobile DCE), delayed (MOST) or queued (Rover). However, the programming model presented to application writers is still essentially synchronous in nature. Indeed, all of the platforms attempt to maintain RPC semantics in the face of variations in network connectivity. Furthermore, all of the models are connection-oriented: clients select services to be used, bind to their interfaces and then invoke operations on these interfaces. As the network QoS and service availability change the platforms use a range of techniques in an attempt to maintain the illusion of connection-oriented communications. For example, in Mobile DCE the RPC protocol transparently re-binds clients to local proxy services during periods of disconnection. In all of the platforms application programmers can determine the QoS of the underlying network and hence construct applications which adapt to changes in this QoS.

Our experiences with developing and working with platforms of this type have led us to question the suitability of the paradigms on which they are based for use in a mobile environment. In particular, as network QoS degrades, providing a model of synchronous, connection-oriented communications becomes increasingly difficult. In addition, the emphasis placed on communications in these platforms has thus far prevented a general model of QoS monitoring and adaptation emerging. More specifically, while explicitly modelling bindings (as in MOST) provides a convenient interface for monitoring communications QoS, it does not provide a general mechanism for informing applications of changes in other QoS parameters (e.g. power availability). These changes must be propagated to clients using an alternative mechanism, e.g. operating system signals or environment variables as in [Schilit,94].

We believe that adaptive applications are best written using an asynchronous, primarily connectionless, programming paradigm with generalised support for QoS control and monitoring. In this paper we propose a system architecture based on the tuple space paradigm which fulfils these requirements. In particular, the tuple space paradigm supports inter-process

communication across time as well as space [Bjornson,91] thus offering implicit support for periods of disconnection. We demonstrate the advantages of our approach by considering the re-engineering of a number of existing adaptive mobile applications.

3 THE TUPLE SPACE PARADIGM

The tuple space paradigm has been extensively researched by the parallel programming community for over a decade. Tuples are typed data structures and each tuple consists of a collection of typed data fields. Each field is either termed 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, though they can not be altered while resident in it. Changes can, however, be made to a tuple by withdrawing it from the tuple space, amending and then reinserting it [Gelernter,85b]. Tuple spaces are shared between collections of processes, all of which have access to the tuples contained within.

In classic distributed environments processes communicate across virtual channels described by bindings and formed from pairs of endpoints, c.f. Chorus ports and UNIX BSD 4.3 sockets [Coulouris,94]. The tuple space paradigm is fundamentally different because processes communicate exclusively through tuple space; this has been termed *generative communication* [Gelernter,85a]. As processes no longer interact directly with one another, the implicit need for bindings is removed and inter-process communication can actually progress *anonymously*. It is, however, also possible to achieve *directed communications* whereby tuples are produced for an identified consumer process by encapsulating destination information in the tuples themselves. Several schemes have been proposed to achieve this, including an approach based on Amoeba-like ports [Pinakis,92]. Because tuple spaces contain persistent tuple objects, as opposed to messages, inter-process communication is supported across time and space [Bjornson,91].

The tuple space paradigm was conceived by researchers at Yale University [Gelernter,85a] 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 Pascal). The original Linda model defines four basic operators:

- `out` inserts a tuple, composed of an arbitrary mix of actual and formal fields, into a tuple space. This tuple becomes visible to all processes with access to that tuple space.
- `in` extracts a tuple from a tuple space, with its argument acting as the 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 of a tuple match the template the tuple is withdrawn and any actuals it contains are assigned to formals in the template. Tuples are matched non-deterministically and `in` operations block until a suitable tuple can be found.
- `rd` is syntactically and semantically equivalent to `in` except that a matched tuple is copied, not withdrawn, from the tuple space and hence remains visible to other processes.
- `eval` is similar to `out`, except it creates *active* rather than *passive* tuples. The tuple is active because separate processes are spawned to evaluate each of its fields. The tuple subsequently evolves into a passive tuple resident in the tuple space.

In addition to the basic model and API described above more than a decade of research by the parallel programming community has led to a number of refinements and extensions to the paradigm. For example, many implementations support two new operators, `inp` and `rdp` [Leichter,89] which are non-blocking versions of `in` and `rd` and evaluate to boolean values indicating their success. More significant extensions include distributed tuple spaces [Pinakis,91], [Pinakis,93a], [Pinakis,93b], multiple tuple spaces [Carriero,94], [Hupfer,90], rules governing the use of tuple space for communications in open systems [Minsky,94] and removal of the distinction between tuples and tuple spaces [Carriero,94].

4 A TUPLE SPACE PLATFORM FOR MOBILE APPLICATIONS

4.1 Platform Overview

We have designed a new platform, Limbo, aimed at providing better support for adaptive mobile applications. This platform is based on the Linda model which has been described in section 3. However, the new platform includes a number of significant extensions which address the specific requirements necessary for operation in mobile environments. In particular, our system incorporates the following key extensions:

- multiple tuple spaces which may be specialised to meet application level requirements, for example consistency, security or performance;
- an explicit tuple type hierarchy with support for dynamic sub-typing;
- tuples with explicit QoS attributes;
- a number of system agents that provide services for QoS monitoring, the creation of new tuple spaces and the propagation of tuples between tuple spaces.

In the following sections we explain each of these extensions in detail.

Multiple Tuple Spaces

The original Linda model was designed to support parallel programming on shared-memory multi-processor systems and features a single, global tuple space. Many recent models have proposed the introduction of multiple tuple spaces to address issues of performance, partitioning and scalability. In particular, supporting multiple tuples spaces removes the need to maintain a consistent view of a single global tuple space on all machines: important for performance in a distributed environment and critical in an environment where communications links are costly and unreliable.

We propose to provide a class of system agent which can create new tuple spaces for applications. These tuple spaces will be configurable to meet application specific requirements [Hupfer,90]. For example, in addition to general purpose tuple spaces we propose to allow the creation of tuple spaces with support for security (user authentication), persistence and tuple logging (for accountability in safety critical systems). A number of further specialisations are possible which aim to increase application performance. Examples of these specialisations include support for dedicated homogeneous tuple spaces (c.f. the default heterogeneous tuple space) and support for reduced consistency models which permit convenient low level mapping onto optimised multicast and group data protocols.

In order to create a new tuple space clients communicate with the appropriate system agents via a common tuple space. Clients specify the characteristics of the desired tuple space and place a `create_tuple_space` request into the common tuple space. The service agent accesses this tuple, creates a tuple space with the required characteristics and then 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.

Applications can make use of the new tuple space by means of a `use(handle)` primitive which sets the destination tuple space for subsequent operations. Handles to tuple spaces can be passed between clients to enable the establishment of new shared tuple spaces.

Tuple spaces are destroyed by placing a tuple of type `terminate` into the tuple space. These tuples are picked up by system agents within the tuple spaces themselves and invoke a system function to gracefully shut-down the tuple space.

Tuple Type Hierarchy

We propose to type all tuples and to organise types in a hierarchy. This scheme has a number of advantages over the notional typing found in many tuple space implementations. In addition to the usual benefits associated with type signatures, it allows for the use of sub-typing when

attempting to match tuples to `in` requests. In more detail, `in` requests for a tuple of a given type can be matched with existing tuples of an equal or sub-type. The conversion between types and sub-types (simply a matter of omitting fields when returning the matching tuple) can be handled by the tuple space. The benefits of sub-typing in a distributed environment have been comprehensively investigated within the ODP community as part of their work on interface trading [ISO,92]. In this model sub-typing enables added flexibility when matching service offers to client requests. We hope to accrue similar benefits by supporting sub-typing in Limbo.

QoS Attributes

Existing mobile distributed systems platforms such as MOST allow QoS attributes to be associated with both bindings and messages. Since Limbo is connectionless all of our QoS attributes are associated with messages. In particular, tuples and tuple requests can be annotated with deadlines. This enables messages to be re-ordered by the system to make optimum use of the available network connectivity or buffered during periods of disconnection. In the case of a tuple which is the subject of an `out` operation the deadline refers to the time the tuple is allowed to reside in the tuple space before being deleted. In the case of tuples which are used as arguments to `in` or `rd` operations the deadline refers to the time for which the requests can block before timing out. Once again, this timing information can be used by the system to re-order messages.

Note that by supporting time-outs on tuple space operations we are able to avoid having to provide special support for `inp` and `rdp`, the non-blocking forms of `in` and `rd` found in many tuple space implementations. Furthermore, tuple time-outs assist garbage collection.

In addition to deadlines we can also associate costs with tuples. In this way applications can provide guidelines to the tuple space regarding the propagation of tuples over expensive wireless communications links. The issue of supporting additional QoS parameters for the transmission of continuous media is a topic for further study.

System Agents

All interaction between the system and applications is via tuple spaces. In addition to the tuple space creation agents discussed above, we define two additional types of system agent: bridging agents and QoS monitoring agents.

Bridging agents provide the means of linking arbitrary tuple spaces and controlling the propagation of tuples between these spaces. In their simplest form bridging agents are processes which carry out repeated `rd` operations on one tuple space and then `out` the corresponding tuples into a second tuple space (with appropriate mechanisms to avoid the problems caused by the non-deterministic nature of the `rd` operation). However, bridging agents can also provide more intelligent tuple propagation based on a number of factors including tuple types and QoS parameters. For example, bridging agents can be configured to only propagate tuples subject to a set of constraints. Bridging agents can also be used to provide gateways between specialised tuple spaces. For example, a bridging agent could be configured to carry out format conversions between homogeneous and heterogeneous tuple spaces or to act as a firewall to prevent the propagation of unauthenticated tuples to secure tuple spaces.

It is important to stress at this point that tuple spaces may span multiple hosts; bridging agents provide a mechanism for propagation of tuples *between tuple spaces* and are *not* usually required for the propagation of tuples between separate hosts.

QoS monitoring agents are responsible for making QoS information available to applications. They monitor a range of environmental factors and communicate this information to interested applications via the tuple space. Examples of QoS monitoring agents include:

- Connectivity monitors: Watch over the characteristics of the underlying communications infrastructure and make available information such as the current throughput between a host and the tuple space.

- Power monitors: Review the availability and consumption of power on a host. In particular, applications can obtain power information on host peripherals and may utilise hardware power saving functionality as appropriate.
- Cost monitors: Determine the cost associated with the current communications link between a given host and the tuple space.

Copies of these agents can run on multiple hosts within the system and produce as output standard tuples which may be accessed by applications in the usual manner.

Tuple spaces and QoS monitoring agents provide us with a uniform way of informing applications about changes in their environment. Furthermore, since QoS monitoring agents produce tuples as output QoS information can be made available within the system to all interested parties (c.f. signals which are only accessible by local applications).

4.2 Prototype Implementation

We are currently developing two prototype implementations of the Limbo platform with different architectures. The first, based on a centralised architecture, requires each tuple space to be managed by a single entity. This makes implementation extremely simple and enables features such as persistence, security and logging to be easily incorporated. While a centralised architecture does not scale well it is ideally suited to local-area and wide-area wireless networks which support a central message exchange (e.g. TETRA [Yeadon,96]). The second prototype we are developing is based on a distributed architecture whereby each tuple space is distributed between all interested hosts and managed collectively.

Each of our prototypes are described in more detail in the following sections.

Centralised Architecture

This prototype provides a limited experimental platform to enable us to evaluate the system. In detail, the prototype supports the `in`, `out` and `rd` operations for tuples with the standard Linda semantics. Tuples are optionally typed and type information is used for matching purposes. Currently, types are not organised hierarchically and sub-type relations are not allowed. However, multiple tuple spaces are supported and tuples can have QoS fields which specify time constraints on all the supported operations. The prototype implementation does not currently support bridging agents or QoS monitoring agents.

The prototype is implemented on top of the MOST distributed systems platform described in section 2.2 with an additional gateway to enable communication with Windows/NT machines. The use of the MOST platform provides us with basic object creation and communication capabilities in a heterogeneous environment. The key feature of the platform which we utilise is the QEX protocol. This enables us to support operation in mobile environments and, significantly, to have access to information regarding the QoS of the underlying communications channels.

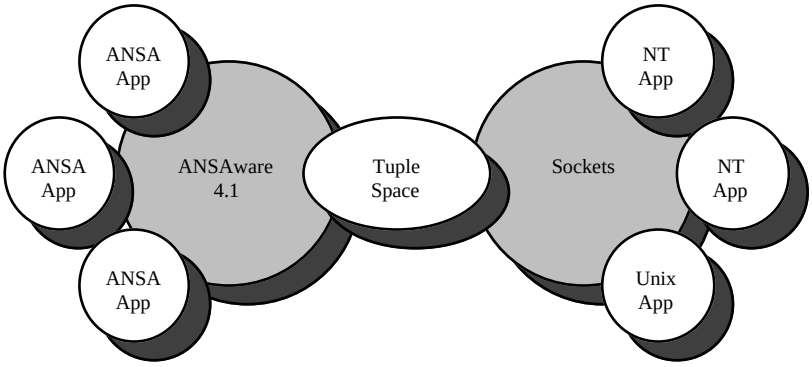

Figure 1 Centralised architecture prototype

The Limbo prototype comprises one or more tuple space servers which can be accessed by local or remote clients. The interface to the tuple space server (in ANSAware IDL) is shown in Figure 2.

```
TupleSpace: INTERFACE =

in :    OPERATION      [ RequestId : TS_Ticket;
                         DeliveryRef : ansa_InterfaceRef;
                         Format : TS_Format;
                         Tuple : TS_Tuple ]
        RETURNS        [ TS_Status; TS_Tuple ];

out :   OPERATION      [ Tuple : TS_Tuple ]
        RETURNS        [ TS_Status ];

rd :    OPERATION      [ RequestId : TS_Ticket;
                         DeliveryRef : ansa_InterfaceRef;
                         Format : TS_Format;
                         Tuple : TS_Tuple ]
        RETURNS        [ TS_Status; TS_Tuple ];
```

Figure 2 The tuple space server interface

Clients link with a special Limbo library to provide them with support for the `in`, `out` and `rd` tuple space operations. The arguments to each operation consist of a string specifying the format of the tuple followed by the tuple itself. Valid format characters are *i*, *c* and *s* to denote integers, characters and strings respectively. Each format character is preceded by a modifier, either *%* to signify an actual parameter or *&* to signify a formal parameter.

The Limbo library maps tuple space operations onto invocations to the tuple space server via the interface shown in Figure 2. The `out` operation registers a tuple with the tuple space server which then resides in the tuple space until its deadline has expired or it is removed by an `in` request; `in` and `rd` operations supply the server with a template tuple and request a match. If a matching tuple is available the operation returns immediately with this tuple, removing it from the tuple space in the case of an `in` operation. If no match is available the tuple space server returns a `delivery_postponed` status code to the client libraries which then block the calling thread awaiting a subsequent call back from the server. This call back will contain either a matching tuple if one subsequently becomes available, or notification of the expiry of the request's deadline.

The communications between clients and the tuple space can be monitored using the underlying QEX protocol and we intend to use this information to develop QoS monitoring agents for Limbo.

Since Limbo is layered on top of MOST it operates on all platforms currently supported by MOST, i.e. Sun SPARCs running SunOS and PCs running either USL System V Release 4 or Linux 1.2.13, and, via a gateway, Windows/NT machines.

De-Centralised Architecture

We are also currently developing a decentralised Limbo prototype which is been designed to be a highly scalable alternative to the centralised approach. Primary motivations for this prototype are improved performance through increased availability and stronger resilience to failures and network partitioning. It is essential that such a distributed implementation does not apply locking strategies for operations which remove tuples and avoids algorithms which lead to acknowledgement implosion, both of which critically affect performance. Fortunately there are a number of features of the tuple space paradigm which greatly simplify the implementation task. Firstly, the model does not specify how long it takes for tuples to propagate to and from the tuple space: as long as tuples are matched non-deterministically and tuples can never be `in`'d

more than once the model's semantics are preserved. Similarly, tuple operations are not causally ordered and total ordering does not have to be maintained.

In contrast to Distributed Linda [Pinakis,91] which centralises each tuple type at a specified server, our decentralised prototype is fully distributed. In order to ensure consistency we allocate owners to tuples and tuples may only be withdrawn from the tuple space by the owner. Changes of ownership are supported as is the concept of 'nominated owners' enabling optimisation of RPC-like communications [Friday,97].

Our implementation is based on IP multicast and borrows application level framing concepts from SRM the scalable multicast transport which underpins wb [Floyd,95] and Jetfile [Grönvall,96] with each distributed tuple space modelled as a multicast group. The design and implementation of our de-centralised prototype is described in more detail in a companion paper [Friday,97].

5 CASE STUDY

Early mobile computing research at Lancaster was aimed to develop open systems technologies to support field workers within the utilities industries [Davies,95]. In addition to the platform described in section 2.2, a collaborative toolkit application was developed that provides support for a number of day-to-day tasks carried out by field engineers. In particular, the application includes a number of groupware modules which enable field engineers to exchange electronic job instructions, collaboratively edit geographic data and access remote databases. In the following sections we describe the current implementation of each of these modules based on the connection-oriented MOST platform and then discuss how their implementation would be affected if they were re-engineered using Limbo.

Job Dispatch

In the current implementation job dispatch instructions map directly onto e-mail messages. The engineers' control centre issues job instructions and forwards them directly to the required engineer. In the case of job instructions requiring multiple engineers or which can be serviced by one of a number of engineers e-mail aliases can be used.

The transmission of e-mail messages in the MOST system is engineered using the QEX protocol and hence connections must be established to each destination site (as in regular e-mail systems). In contrast, tuple spaces inherently provide temporal de-coupling between message sender and message recipient. Furthermore, because of the dynamic nature of tuple matching, job requests can be issued and matched subject to a wide range of parameters such as job types, physical location, required skills and/or equipment.

Collaborative Editing

The MOST application provides support for collaborative editing of geographical information by groups of field engineers. Shared pointers are not supported because of the limited bandwidth, high-latency communications channels used and the desire to support interoperability between a range of windowing systems and display types. As an alternative, the application allows the annotation (red-lining) of diagrams using a selection of drawing tools.

In the current implementation group state is distributed between the collaborating engineers. Annotations and group membership changes are propagated using a sequence of unicast QEX messages. This requires applications to maintain a list of conference participants and deal with failures of messages between one or more group members. In addition, it is difficult in this type of application to bring new or partitioned members up to state with the remainder of the group. Furthermore, in safety critical applications it is essential to be able to log all interactions between group participants in such a way that the behaviour of the system can be reproduced. In the current implementation this is difficult because messages can originate from any member of the group and are not necessarily propagated to all other members.

Using Limbo substantially reduces the complexity of this type of application. In particular, support for groups is provided automatically by tuple spaces since individual tuples may be read by multiple processes. Tuples which remain in the tuple space can be read by latecomers to a group in order to bring themselves up to state. Deadlines can be used to ensure that tuples relating to transient group state is automatically removed from the tuple space. However, tuple spaces can be specialised to provide facilities such as tuple logging, allowing the state of the system to be captured for audit purposes. The use of separate tuple spaces for different collaborative sessions would help to partition system state and limit message propagation to interested parties only.

Database Access

The MOST application enables remote access to databases from mobile hosts. A key issue in engineering this application is matching the quantity of information passed between the client and server to the available communications resources. In the current implementation the client issues a standard database request to a remote server using QEX. The server then consults the platform to determine the network QoS and returns the results of the query as a set of partially complete records. The degree of completeness of these records is determined by the QoS and the number of records to be returned. So, for example, if a query which matches a large number of records is issued by a weakly connected mobile host the server will only return selected fields for each matching record to improve response time. The client is then able to refine its query or request complete records as appropriate.

In Limbo clients and servers are not (conceptually) linked and hence it is not possible to determine the QoS between a client and a server. Instead, clients and servers are able to determine their connectivity with respect to the tuple space using information supplied by QoS monitoring agents. Given this model, we would engineer the above application in the following manner. The database server would assess its current connectivity and issue an `in` request for database query tuples of an appropriate size. Clients wishing to query the database would assess their current connectivity to the tuple space and issue a database request, again of the appropriate size. Hence, servers would only offer to provide services which they could support with reasonable performance over their communications link, and clients would only request services whose results they could receive within a reasonable time frame. The subtyping mechanism described in section 4.1 can be used to ensure compatibility between requests and responses.

6 CONCLUDING REMARKS

Current distributed systems platforms designed for mobile environments are based on synchronous, connection-oriented communications. In this paper we have described Limbo, a new platform based on the tuple space communications paradigm. Clearly the work is at early stage and we have yet to fully address issues such as consistency during periods of disconnection. However, we are encouraged by the successful extended versions of the Linda model, such as distributed and multiple tuple space versions, which have been documented. We believe the tuple space paradigm is an interesting approach that addresses many of the shortcomings of existing mobile support platforms. We have presented evidence to justify this belief by illustrating how Limbo can be used to significantly simplify the implementation of a number of existing mobile applications.

Two Limbo prototypes are under development. To date, the centralised implementation is layered on top of an existing distributed systems platform and supports the basic Limbo model, the required communication primitives and some QoS features. Our short-term plan is to finish adding the proposed services to this prototype and to re-engineer them in the prototype based on our de-centralised architecture. We then intend to make the source code of both prototypes available to the research community for experimentation and evaluation.

7 ACKNOWLEDGEMENTS

This work was carried out under the auspices of the "Supporting Reactive Services in a Mobile Computing Environment" project (EPSRC Grant No. GR/K11864)

8 REFERENCES

[APM,89] APM Limited (1989) ANSA: An Engineers Introduction to the Architecture. *Technical Document release TR.03.02*, APM Cambridge Limited, Poseidon House, Castle Park, Cambridge, CB3 0RD, U.K.

[Bjornson,91] 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.

[Carriero,94] Carriero, N., Gelernter, D. and Zuck, L. (1994) Bauhaus Linda. *Selected Papers from the Workshop on Models and Languages for Coordination of Parallelism and Distribution (ECOOP '94)*, Bologna, Italy, 66-76.

[Coulouris,94] Coulouris, G.F., Dollimore J. and Kindberg, T. (1994) *Distributed Systems: Concepts and Design (Second Edition)*. Addison-Wesley.

[Davies,94a] Davies, N., Pink, S. and Blair, G.S. (1994) Services to Support Distributed Applications in a Mobile Environment. *Proceedings of the 1st International Workshop on Services in Distributed and Networked Environments (SDNE'94)*, Prague, Czech Republic, 84-89.

[Davies,94b] Davies, N., Blair, G.S., Cheverst, K. and Friday, A. (1994) Supporting Adaptive Services in a Heterogeneous Mobile Environment. *Proceedings of the 1st Workshop on Mobile Computing Systems and Applications (MCSA'94)*, Santa Cruz, California, U.S., 153-157.

[Davies,95] Davies, N., Blair, G.S., Cheverst, K. and Friday, A. (1995) Experiences of Using RM-ODP to Build Advanced Mobile Applications. *Distributed Systems Engineering Journal*, 2(3), 142-151.

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

[Friday,96] Friday, A.J., Blair, G.S., Cheverst, K.W.J. and Davies, N. (1996) Extensions to ANSAware for Advanced Mobile Applications. *Proceedings of the 1st International Conference on Distributed Platforms (ICDP'96)*, Dresden, Germany.

[Friday,97] Friday, A., Wade, S.P., Davies, N. and Blair, G.S. (1997) Using Tuple Spaces for Adaptive Mobile Computing. *Technical Report MPG-97-03*, Computing Department, Lancaster University, Bailrigg, Lancaster, LA1 4YR, U.K.

[Gelernter,85a] Gelernter, D. (1985) Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1), 80-112.

[Gelernter,85b] Gelernter, D., Carriero, N., Chandran, S. and Chang, S. (1985) Parallel Programming in Linda. *Proceedings of the International Conference on Parallel Processing*, 255-263.

[Grönvall,96] Grönvall, B., Marsh I. and Pink, S. (1996) A Multicast-Based Distributed File System for the Internet, *Proceedings of the 7th ACM SIGOPS European Workshop*, Connemara, Ireland.

[Hupfer,90] 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.

[ISO,92] ISO (1992) Draft Recommendation X.901: Basic Reference Model of Open Distributed Processing - Part 1: Overview and Guide to Use. *Draft Report*.

[Joseph,95] Joseph, A.D., deLespinasse, A.F., Tauber, J.A., Gifford, D.K. and Kaashoek, M.F. (1995) Rover: A Toolkit for Mobile Information Access. *Proceedings of the 15th Symposium on Operating Systems Principles (SOSP'95)*, Copper Mountain Resort, Colorado, U.S., 156-171.

[Katz,94] Katz, R.H. (1994) Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Communications*, 1(1), 6-17.

[Leichter,89] Leichter, J.S. (1989) Shared Tuple Memories, Shared Memories, Buses and LAN's - Linda Implementations across the Spectrum of Connectivity. *Ph.D. Thesis*, Department of Computer Science, Yale University, New Haven, Connecticut, U.S.

[Minsky,94] Minsky, N.H. and Leichter, J. (1994) Law-Governed Linda as a Coordination Model. *Selected Papers from the Workshop on Models and Languages for Coordination of Parallelism and Distribution*, Bologna, Italy, 125-146.

[Pinakis,91] Pinakis, J. (1991) The Design and Implementation of a Distributed Linda Tuple Space. *Proceedings of the 2nd Department of Computer Science Research Conference*, Department of Computer Science, University of Western Australia, Nedlands, WA 6009.

[Pinakis,92] Pinakis, J. (1992) Providing Directed Communication in Linda. *Proceedings of the 15th Australian Computer Science Conference*, Hobart, Tasmania.

[Pinakis,93a] Pinakis, J. (1993) Remote Thread Execution, *Proceedings of the 16th Australian Computer Science Conference*, Brisbane, Queensland, Australia.

[Pinakis,93b] Pinakis, J. (1993) Using Linda as the Basis of an Operating System Microkernel. *Ph.D. Thesis*, Department of Computer Science, University of Western Australia, Nedlands, WA 6009, Australia.

[Schilit,94] Schilit, B., Adams, N. and Want, R. (1994) Context-Aware Computing Applications. *Proceedings of the 1st Workshop on Mobile Computing Systems and Applications (MCSA'94)*, Santa Cruz, California, U.S., 85-90.

[Schill,95] Schill, A. and Kümmel, S. (1995) Design and Implementation of a Support Platform for Distributed Mobile Computing. *Distributed Systems Engineering Journal*, 2(3), 128-141.

[Yeadon,96] Yeadon, N. (1996) Using TETRA to Support Distributed Multimedia Applications. *Technical Report*, Computing Department, Lancaster University, Bailrigg, Lancaster, LA1 4YR, U.K.

9 BIOGRAPHY

Nigel Davies graduated from Lancaster University in 1989 and later that year joined the Computing Department as a research associate investigating storage and management aspects of multimedia systems. As a result of his work in this area he was awarded a Ph.D. in 1994. After a spell as a visiting researcher at the Swedish Institute of Computer Science (SICS) where he worked on mobile files systems 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.

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".

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.

Gordon Blair is currently a Professor in the Computing Department at Lancaster University. He completed his Ph.D. in Computing at Strathclyde University in 1983. Since then, he was a SERC Research Fellow at Lancaster University before taking up a lectureship in 1986. 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, operating system support for continuous media, the impact of mobility on distributed systems and the use of formal methods in distributed systems development.