4

Extensions to ANSAware for advanced mobile applications

A.J. Friday, G.S. Blair, K.W.J. Cheverst and N. Davies

Distributed Multimedia Research Group,
Department of Computing,
Lancaster University,
Lancaster,
U.K.
Fax: +44 (0)1524 593608
Telephone: +44 (0)1524 65201
E-mail: adrian, gordon, kc, nigel@comp.lancs.ac.uk

Abstract
Significant advances have been made in recent years in tackling the problem of heterogeneity in distributed systems with ISO/ ITU-T standards for a Reference Model for Open Distributed Processing (RM-ODP) now approaching international agreement. It is important, however, that such standards and related technologies are responsive to new developments in computer and communications technologies. This paper reports on experiences of using the RM-ODP based platform ANSAware in the development of mobile applications to support field workers in the electricity industry. A number of extensions to ANSAware are described including QoS-managed bindings, an enhanced trading service and a new communications protocol. These extensions are evaluated against the requirements of mobile applications.

Keywords
Mobile computing, ANSAware, RM-ODP, Quality-of-Service.

1 INTRODUCTION

Significant advances have been made in recent years in tackling the problem of heterogeneity in distributed systems with ISO/ ITU-T standards for a Reference Model for Open Distributed Processing (RM-ODP) now approaching international agreement [ISO95a, ISO95b, ISO95c]. Similarly, specific technologies such as APM's ANSAware, OMG's CORBA and OSF's DCE are commercially available. However, it is important that such standards and technologies remain responsive to new developments in computer and communications technologies. This paper reports on experiences of applying ODP standards, and more specifically the ANSAware distributed systems platform, in the new area of mobile computing [Duchamp92, Katz94].

The paper is structured as follows. Section 2 summarises the main body of research carried out in the MOST Project (Mobile Open Systems Technology for the Utilities Industries) at Lancaster, which has looked at the potential of RM-ODP technology and wireless communications to support mobile utilities workers [Davies95a]. In particular, this section describes a prototype application developed over GSM technology which is designed to support field engineers in the power distribution network. Section 3 then describes the underlying distributed systems platform developed in MOST. This platform is based on ANSAware but with extensions to support operation over mobile environments. These extensions include the new functionality of QoS-managed bindings and peer-to-peer linking of traders and a new underlying communications protocol, QEX. Section 4 evaluates the extensions to ANSAware and includes some performance figures for QEX. Finally, section 5 presents our concluding remarks.

2 AN EXPERIMENT IN DEVELOPING ADVANCED MOBILE APPLICATIONS

2.1 The applications scenario

Supporting field engineers
All work within a regional electricity company is traditionally co-ordinated by a single control centre. The engineer supervising a particular repair job files a schedule with the centre some days in advance. This schedule describes in detail the stages involved in carrying out the work and, in particular, the sequence of switching which must be carried out to ensure that the work can be conducted safely (i.e. the section of network being operated on is isolated and earthed) and with the minimum of disruption to users. The control centre checks the switching schedule against its central diagram of the network state (this may be held on its computer system) and approves or rejects the schedule accordingly. Expert systems may be used by both the control centre and the field worker in the development of the switching schedule [Cross93] although at present these systems are not integrated with the centre's representation of the global network state.

Once the schedule has been approved, the work may be carried out. On the day of the work, field engineers are dispatched to the appropriate switching points. The control centre then uses a voice-oriented private mobile radio system to instruct the staff as to which switches to operate. The use of a central co-ordinator helps to ensure the work is carried out in the correct order and allows the centre to maintain an up-to-date picture of the network's state. Once the work has been carried out the engineer must wait until returning to the office before completing the associated paper work.

Introducing mobile communications
There are clearly a number of disadvantages with the approach described above, in particular the lack of availability of global network state for the engineers in the field and the reduction in efficiency caused by the bottle-neck of a central point of control. The latter of these points becomes particularly important when faults occur requiring multiple unscheduled work items to be carried out. These disadvantages could be overcome by providing field engineers with a shared up-to-date picture of the network state. Providing such a picture would, of course, require a high degree of real-time synchronisation between field engineers operating on the network to ensure consistency between views of the network state.

Our aim is to exploit the capabilities of emerging mobile networks such as GSM to provide the means for field engineers to have at hand the information they require in the field and to enable them to collaborate with one another to discuss relevant aspects of this information. For example, we aim to allow field engineers who are not physically co-located to view and manipulate shared diagrams and information and to communicate using voice. This would ease congestion at the control centre which, in addition to supplying individual engineers with information, is also responsible for coordinating collaboration between engineers. Hence, the application is novel since much of the communications involved is of a time critical, peer-to-peer nature (i.e. mobile field engineer to mobile field engineer) rather than conforming to the client-server architecture more commonly found in mobile applications.

2.2 The prototype application

Overall functionality

The prototype application we have developed allows field engineers to view geographical information (e.g. network diagrams) using a public-domain Geographical Information System (GIS) called GRASS [Westervelt91]. Support for collaboration with other field engineers is provided in two ways. Firstly, field engineers are able to synchronise the operations of their GIS with those of one or more remote engineers. In this way a common shared view of a particular piece of information can be maintained. Once a shared view has been established the application allows engineers to annotate the display with highlight marks which are subsequently propagated to all relevant remote engineers. Secondly, the application supports communication between field engineers via audio communications.

An example screen shot of the application's user interface is shown in figure 1. The conference manager or group co-ordinator component of the application is shown in the top left of the diagram. This component makes extensive use of icons to represent users and their applications. On the left hand side of the group co-ordinator component are icons representing the sub-applications or modules which are available to the user (for example, the globe represents the GIS services) and on the right hand side are icons representing users that can participate in conferences. In the centre is a list of the current conference participants. Under the icon of each participant, a column of additional icons represent which modules that user is currently running. By selecting icons from the rows of module icons the user is able to easily direct messages to a subset of the conference members. The user is able to select (by clicking on a single button) whether or not an operation (for example drawing a line in the GIS module) is propagated to those group members whose module icons have been selected.

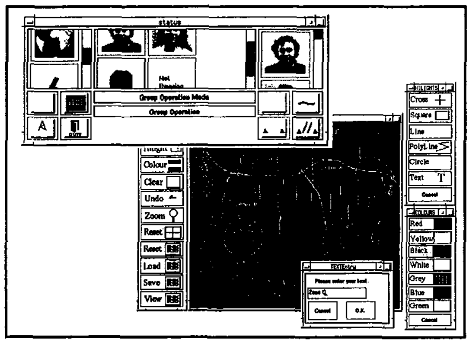

Figure 1 The user interface with the GIS module active during a three-party conference.

Considerable emphasis has been placed on reporting to the user the state of the communications links between local modules and remote users' modules. In our prototype this is achieved by colouring the module icons under each user in the central portion of the display to reflect connectivity (using a spectrum of green representing full connectivity through to red representing no connectivity). Figure 1 has been simplified in the interest of clarity to show only the group coordinator and a single GIS module.

Design philosophy

Our key design philosophy is one of extensibility. From the users point of view this manifests itself in the tool-box feel to the interface. In particular, the list of modules supported can be expanded at any time and we anticipate field engineers configuring their applications to use sub-sets of the available modules. For example, the current implementation supports modules providing remote database access (to look up customer records), a job dispatch application and an e-mail facility as well as the GIS and audio conferencing modules discussed above. We rely on the host's windowing system to provide features to allow users to make best use of their display (e.g. facilities to iconise and re-size windows or virtual desktops to optimise the use of a small screen).

In terms of implementation, the application is structured as a number of RM-ODP compatible service providers (objects) with all communication carried out via object invocation. Hence, we prohibit, for example, applications using the X protocol or X multiplexors (as used for example in XTV [Abdel-Wahab91]) to implement collaborative interfaces. This approach has a number of benefits. Firstly, we are able to support operation in a truly heterogeneous environment: since objects only communicate using invocations, new implementations of our application need only support the basic invocation mechanism and the prescribed service interfaces in order to inter-work with all existing implementations. In addition, since all communication is explicit we avoid building in dependencies on windowing systems (e.g. X or Microsoft Windows) and the risk of incurring hidden overheads which are often associated with windowing system protocols. For example, the difficulties associated with using the X protocol for communication to mobile hosts is reported by Kantarjiev [Kantarjiev93].

The drawback with our approach is that we require customised applications. However, since there has been little positive experience with using conventional applications in a collaborative context [Lauwers90], we believe that the benefits of our approach outweigh this drawback, particularly given the specific nature of our application domain. In order to simplify the development of collaborative applications for use in our environment we have implemented support for conference management within a single service, the group co-ordinator. This ensures that all applications share a common view of the participants in a conference and reduces the complexity of the applications. Further details of the design of the prototype application and the associated RM-ODP services can be found in [Davies95c].

3 THE UNDERLYING PLATFORM

3.1 Background on RM-ODP

Motivation and goals of RM-ODP

We have developed a distributed systems platform to support the trial application described above. This platform is based on the emerging RM-ODP to enable operation in a heterogeneous environment.

The RM-ODP is designed to be a generic, technology and application independent framework for developing open distributed systems. It is important to stress that the Reference Model provides a general framework for developing open distributed systems; this Reference

Model can, in turn, lead to specific instantiations in a given application domain (e.g. TINA in the telecommunications sector).

The Reference Model provides a set of general concepts (a vocabulary) and an object-oriented modelling approach to support distributed systems development. The Reference Model also provides a methodology based on the division of a specification into a set of 5 viewpoints: enterprise, information, computational, engineering and technology (representing the full range of perspectives on a distributed system development from business objectives to detailed implementation choices). The focus of our work in the MOST project is on the computational viewpoint, and this aspect of the RM-ODP is described below. The important concepts of distribution transparency and trading are also described. Note that a more complete discussion of the features offered by the RM-ODP can be found in the literature [Raymond93].

The ODP computational viewpoint

The ODP computational viewpoint is based on a location independent object-based model of distributed systems. In this model, interacting entities are treated uniformly as objects, i.e. encapsulations of state and behaviour. Objects are accessed through interfaces which can be of two types: signal interfaces and stream interfaces. Signal interfaces define named signals together with constraints on their occurrence. Stream interfaces define named data flows that constitute abstractions of sets of interactions which are not visible from the outside. Communication between objects using signals or flows is only possible through explicit or implicit bindings between interfaces.

Operational interfaces are special cases of signal interfaces. Operational interfaces define named operations together with constraints on their invocations. Operations can be either an interrogation (a two-way operation, comprising an invocation, later followed by a termination carrying results or exceptions), or an announcement (a one-way operation, comprising an invocation only). Implicit binding is only available for operational interfaces. Activity takes place in the model when objects communicate via signals or flows supported by explicit or implicit binding objects.

Objects offering services are made available for access by exporting interfaces to a database of service interfaces known as a trader. An object wishing to interact with a service interface must import the interface by specifying a set of requirements in terms of an interface type and attribute values. This will be matched against the available services and a suitable candidate selected. Any number of traders can exist and these may be linked or federated to allow access to services in different administrative domains.

Also central to the ODP computational model is the notion of transparency whereby selected aspects of systems can be made invisible to applications. This is achieved by means of notional transparency functions interposed between the application and the support layers (it should be noted that the application of transparency functions is under user control and hence transparencies are selective). An important example of a transparency is group transparency which allows multiple services to be invoked via a single interface. Other transparencies identified in ODP include location, access, concurrency, replication, migration and failure transparencies.

3.2 The ANSAware distributed system

The ANSAware computational model

The ANSAware computational model is a subset of the corresponding computational model defined in RM-ODP. In ANSAware, interacting entities are treated uniformly as objects. Objects are accessed through operational interfaces which define named operations together with constraints on their invocations. Operations are accessed through a binding; at present in ANSAware, such bindings are established implicitly before first access. ANSAware also supports the following transparency functions: location, access and group. RM-ODP features not supported by ANSAware include: signals, flows, explicit bindings, environmental contracts including QoS annotations, and more general transparency functions.

ANSAware provides an implementation of the trading function. Again, the ANSAware trader is a subset of trading as defined in RM-ODP. The ANSAware trader organises offers according to service type and service properties. Offers are also organised into (hierarchical) contexts. Traders can also be links; the current implementation, however, limits the resultant graph to a one-level hierarchy. One trader is nominated as the master trader with all other traders binding a local/ master context to this trader. The ANSAware implementation of linking also prescribes the policy for searching the resultant namespace; the local trader is always searched first and, if a match is not found, the master trader is then contacted. The ANSAware trader also supports the concept of proxy offers. If a client attempts to import a proxy offer, then the import request is forwarded to another trader which should return the appropriate service.

Implementation of ANSAware

To provide a platform conformant with the object model, the ANSAware suite augments a general purpose programming language with two additional languages. The first of these is IDL (Interface Definition Language), which allows interfaces to be precisely defined in terms of operations as required by the computational model. The second language, DPL (Distributed Processing Language) is embedded in a host language, such as C, and allows interactions to be specified between programs which implement the behaviour defined by these interfaces. Specifically, DPL statements allow the programmer to import and export interfaces, and to invoke operations in those interfaces. A number of system services are supplied which include a trader service and a factory service for creating new objects.

In the engineering infrastructure, the binding necessary for invocations is provided by a remote procedure call protocol known as REX (Remote EXecution protocol) or a group execution protocol known as GEX (Group EXecution Protocol). REX is a remote procedure call package which has been designed with two basic styles of interaction: the first, rapid interaction with small amounts of data, and the second, bulk data transfer. REX uses the bulk transfer mechanism when packets exceed the fragmentation threshold. In addition, either style of interaction may be synchronous or asynchronous. REX is then layered on top of a generic transport layer interface known as a message passing service (MPS). A number of additional protocols may be included at both the MPS and the execution protocol levels and these may be combined in a number of different configurations. The infrastructure also supports lightweight threads within objects allowing multiple concurrent invocations.

All the above engineering functionality is collected into a single library, and an instance of this library is linked with application code to form a capsule. Each capsule may implement one or more computational objects. In the UNIX operating system, a capsule corresponds to a single UNIX process. Computational objects always communicate via invocation at the conceptual level but, as may be expected, invocation between objects in the same capsule is actually implemented by straightforward procedure calls rather than by execution protocols. ANSAware currently runs on a variety of operating systems platforms including various flavours of UNIX, VMS and MS-DOS/ Windows.

3.3 Required extensions to support the prototype application

We have extended the basic ANSAware platform to support the transmission of continuous media and also to operate more efficiently in a mobile environment. The key changes are discussed below.

Enhanced functionality

Explicit QoS-Managed Bindings The first major change we have made to ANSAware has been to extend the support for bindings. A useful side-effect of this work is that our version of the ANSAware platform is now aligned closer to the current RM-ODP standard. To meet the requirement for an abstraction of real-time data flow over time, we have added the concept of explicit stream bindings (as described in the RM-ODP) to our ANSAware based platform. Stream bindings provide an end-to-end abstraction over continuous media communication and support arbitrary m:n connections, i.e. they allow m sources to be connected to n sinks.

Within our platform stream bindings are established using an explicit bind operation which takes as parameters the source and sink interfaces to be bound and a further set of desired QoS parameters. These parameters can include a specification of the desired throughput, latency and jitter associated with the binding (more details of the specification of QoS parameters for continuous media bindings can be found in [Coulson95]). Clients are returned a binding control interface as a result of an explicit bind operation. To control the QoS of the flow once the binding has been established the control interface includes a pair of operations setQoS() and getQoS(). These operations take as arguments a set of QoS parameters which can then be passed by the stream binding to the underlying transport protocol. A call-back mechanism is also provided to inform client objects of QoS degradations reported by the underlying transport service.

We have also added a new class of explicit binding for use with operational interfaces. These bindings are established using the binder$Bind operation as above but take as arguments operational interfaces. The resulting binder control interface is identical to that used for stream bindings except that clients are allowed to specify and monitor a different set of QoS parameters associated with the binding. This enables, for example, a client to ask to be informed when the QoS service supplied by the binding falls below a specified threshold. Of particular relevance to mobile applications is the ability to monitor the possibility of sending or receiving messages via a specified binding without having to explicitly send application level test messages, i.e. applications can delegate responsibility for guaranteeing QoS assertions to the system. This is of significance since it allows mobile applications to be structured in an event based fashion (c.f. polling). For example, through the use of our bindings it is possible to assert that the absence of messages on a given interface is a result of their being no traffic intended for the specified interface rather than a result of communications failure. In addition, QoS driven bindings allow the system to optimise the use of test messages which might otherwise be duplicated if left to individual applications, e.g. if multiple applications wished to test QoS assertions between the same pair of objects. Further details of the structural changes in applications and system services possible as the result of the introduction of QoS driven bindings can be found in [Davies94]).

Enhanced Trader Finally, as described above, the implementation of the trading function in ANSAware assumes a hierarchical pattern of linking. This hierarchical arrangement while possibly adequate for fixed networks is inappropriate for mobile systems where each portable computer is likely to have its own trader in order that it can continue operation during periods of disconnection. Moreover, given the nature of the communications network a link scheme which requires traversal up and down a hierarchy (possibly requiring multiple dial-up connections to be established) is clearly unsuitable.

We have addressed this issue by introducing a mechanism to enable peer-to-peer linking of traders. Links can also have associated constraints ensuring that they are traversed only when there is a strong likelihood of the remote trader being able to satisfy the request. For example, we can specify that services owned by a particular user are always found on a specified machine and traverse the link to that machine only when looking for that user's services. Again, this change means that our implementation of trading is more closely aligned to the RM-ODP standard.

Underlying protocols

Introducing QEX The current execution protocols in ANSAware are not well suited to operation over mobile networks. In our work, we have introduced a new protocol called QEX (Quality-of-service driven remote EXecution) to replace the REX protocol. We are also planning to replace GEX with a protocol G-QEX although this is still at the design stage. We describe the implementation of QEX below.

Currently, the REX protocol takes no account of the characteristics of the underlying network. More specifically, parameters such as the number of retry attempts and the interval between these attempts are fixed at installation time. Thus, when a system configured to operate over an Ethernet is run over a low-speed network the absence of congestion control within REX means that almost no data is actually communicated between user processes. Instead, the network becomes overloaded with REX control messages. In addition, if the parameters of the REX protocol are modified to operate over a low speed link, it no longer performs optimally over Ethernet. To overcome these difficulties, QEX analyses the characteristics of the communications medium for each interaction and adjusts itself to make the best use of the link. QEX maintains backward compatibility with REX to allow us to interact with existing ANSAware services.

Estimating Characteristics of the Channel The general approach in QEX is to estimate the underlying characteristics of the channel based on information obtained from round-trip times and sizes of messages. As a single object may be communicating with remote objects via links with potentially different network QoS, channel information is recorded for each active session at both ends of the interaction. The memory overhead of the channel analysis grows approximately linearly with the number of concurrent interactions at any given time.

Round-trip time calculations can not take into account asymmetric connections which may occur in mobile environments. However, estimating end-to-end channel delay would require a time stamping scheme using synchronised clocks. This option was avoided due to the overhead of running clock synchronisation algorithms over low bandwidth connections.

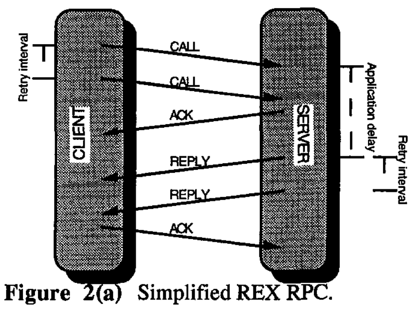

Figure 2(a) Simplified REX RPC.

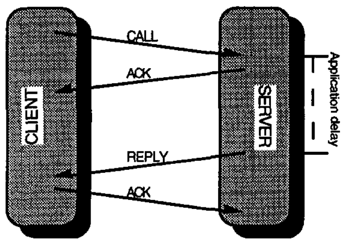

Figure 2(b) Simplified QEX RPC.

Figure 2(a) shows a REX RPC interaction (slightly simplified for clarity: for a full explanation see the ANSAware reference manual [APM93]). A call is sent to the server process. Providing it arrives, the server processes the request and responds with its reply. In case the call was lost, the client will send a retry after a fixed interval until either an acknowledgement or the reply is received. When the server detects a duplicate call it sends an acknowledgement to the client to inform it that it is still processing the request.

From this figure, we can see that the only pair of messages we can use to calculate round-trip times is the call/ ack or reply/ ack interactions since we have no method to determine the application delay introduced into other message pairings (which could change arbitrarily each time, particularly where user interaction is possible). These messages, although we believe them to occur frequently in most real application scenarios, cannot be relied upon to keep us up to date on the current characteristics of the communications medium. To solve this problem, we chose to make our protocol respond immediately with acknowledgements on receipt of a call or reply message. Application delay is therefore kept to a minimum and a reasonable approximation of round-trip time for a particular message size is obtained (note that this extra acknowledgement is not a significant overhead as in practice REX often retries at least once before returning results). The round-trip statistics are smoothed using a moving average calculation and fed back into the protocol to load the retry interval timers. The approach adopted in QEX is illustrated in figure 2(b).

Backoff Strategy While establishing the rate over a particular channel, backoffs play an important role. Over Ethernet it is customary to use exponential backoffs as the assumption is that dropped packets are caused by congestion. Over low bandwidth networks it is far more likely that dropped packets are caused by the relatively high error rates experienced by wireless networks. In QEX we choose different backoff strategies depending on the stability of our measurements. While working out the rate it is important not to flood a low bandwidth network. At this stage, we therefore employ exponential backoffs. When the round-trip calculations stabilise, if a low bandwidth network is indicated, we switch to a linear backoff strategy, the assumption being that packet loss is now more likely due to errors. Setting the threshold at which linear backoffs are used is critical, high speed mobile networks (such as WaveLAN) although offering almost wired LAN throughputs exhibit far higher error rates [Cáceres94] and thus require linear backoffs.

Dealing with Fragmentation When the message size is greater than the fragmentation threshold (which is dependent on the transport service), REX fragments the message and sends groups of fragments periodically to the server. The server then informs the client after a fixed interval which fragments it hasn't seen (a negative acknowledgement). This style of interaction is depicted in figure 3(a). In this approach there is no scope for calculating round-trip times as negative acknowledgements are not sent in direct response to client messages.

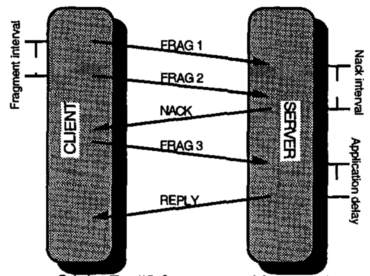

Figure 3(a) REX fragmented interaction.

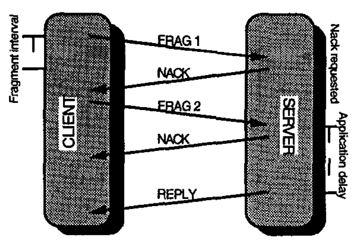

Figure 3(b) QEX fragmented interaction.

To overcome this problem we abandoned the negative acknowledgement time-out interval and explicitly send negative acknowledgements on the receipt of fragments (the negative acknowledgements indicate which fragments are still to be received). This approach is depicted in figure 3(b). To avoid the strict synchronisation this imposes on what is intended to be a bulk transfer mechanism, we only acknowledge packets that are tagged for acknowledgement. The proportion of tagged to untagged fragments is then calculated based on our confidence the system has in the throughput approximation. The number of fragments sent is worked out based on the current estimate of the underlying network characteristics.

Identifying Retries In REX any retry of a packet is identical to the original transmission; there is therefore no way to tell a transmission and a retry apart. This is catastrophic for round-trip calculations as it is not clear which call an acknowledgement matches (particularly under failure where original messages may be lost). In QEX the sending process tags packets with an identifier which the receiver then includes in its response, allowing the sender to match messages to responses. The identifier is hidden in the REX message header in a field which REX ignores. It can therefore serve a dual purpose: in addition to identifying the packet, it also identifies whether the partner is running the original REX or QEX protocol.

3.4 Implementation status

The application and associated platform has been fully implemented and demonstrated running in a heterogeneous environment consisting of SUN workstations, desktop PCs and portable PCs. Mobile communications is provided by either analogue cellular phones, operating at 2.4Kbits/sec, or GSM at 9.6Kbits/sec. Alternatively, a network emulator can be used allowing us to simulate the varying degrees of connectivity likely to be experienced by field engineers during a typical operational cycle [Davies95b]. The QEX protocol has been implemented to run over a UDP like protocol (which we call Serial-UDP) which handles serial and Hayes compatible dialup connections.

4 EVALUATION OF THE EXTENDED PLATFORM

4.1 Functional aspects

QoS-Managed bindings and adaptive services

The concept of QoS-managed bindings has proved to be particularly important in supporting the MOST application. Firstly, this feature enables the application to specify the required quality of service (QoS) from a particular binding. This information can then be used by the infrastructure to guide resource management and scheduling.

Example: Introducing deadlines

The MOST application features a number of different styles of invocations ranging from those supporting real-time interaction to those carrying asynchronous mail messages. In a low bandwidth environment, however, trade-offs must be made between the different styles of traffic. In the current implementation, deadlines on messages enables invocations for real-time interaction to take priority over e-mail messages.

Secondly, applications can obtain feedback on the QoS provided by the underlying network. This is particularly important as we anticipate end-systems which will either be disconnected, weakly inter-connected by low speed wireless networks such as GSM, or fully inter-connected by high speed networks ranging from Ethernet to ATM with this level of connectivity varying over time as a consequence of the mobility of the modern computer user. It is therefore crucial that applications can be informed of the current QoS to enable them to adapt their behaviour.

Example: Remote database access

The service providing remote database access can adapt its behaviour by varying the information returned depending on the available QoS. For example, while connected to a high speed network, this service can return full details of the results of a query. If, however, bandwidth is limited, the database service can return the number of hits together with name fields of the records. Individual records can then be pulled over on request.

Quality of service information can also have a fundamental impact on the structuring of distributed applications as illustrated by the following example.

Example: Weather report bulletins

Consider a simple ANSAware application designed to inform field engineers of approaching bad weather (for example, to provide warnings of impending lightning storms for engineers working on pylons). The application consists of a single, central service which has access to national weather information and a number of client objects (one for each field engineer) which can query the service. Engineers need to be warned when bad weather is recorded in areas adjoining the one in which they are working. Hence, there are two ways of structuring this application: either the client applications register an interest in the weather in particular areas and the server notifies them of any subsequent changes in condition, or, the client applications poll the server at regular intervals for the weather reports of the relevant areas. In a wired environment the former solution would almost certainly be adopted to avoid the communications and processing overheads incurred by polling. However, in a wireless environment there are additional factors to consider. Specifically, a call-back based approach assumes that communications are reliable. If communications are unreliable, a lack of information can either mean that messages are getting lost or that there are no reports of bad weather. This ambiguity is clearly undesirable.

The provision of both stream and operational bindings also proved to be helpful in constructing the MOST application. In particular, the complexities of dealing with audio communications are largely encapsulated behind stream bindings. This also enables different implementations of stream bindings to be provided. For example, the application could readily be ported from GSM to a PMR system.

The authors have considerable experience in the use of stream bindings for the support of multimedia applications; these experiences are discussed in detail in [Davies92].

Flexible model of trading

From our experiences, care must be taken in designing a trader service to ensure the implementation is flexible enough for a mobile environment. This is particularly important when traders are linked. As reported in section 3.2, the particular policies for linking in the ANSAware trader proved to be too prescriptive for such an environment. As a minimum requirement, we believe that traders should be flexible enough to allow peer-to-peer linking. Our experiences with the implementation of such a strategy in ANSAware have been positive.

4.2 Performance aspects

This section analyses the performance of the QEX protocol. The results are based on the use of the network emulator mentioned in section 3.3.

Experiment 1: Comparing REX and QEX (No Fragmentation)

The aim of the first experiment is to carry out an investigation of the performance of QEX against REX for a variety of network bandwidths. The packet size is kept constant and below the level of fragmentation. More specifically, requests are 215 bytes and replies 140 bytes (including header size in both cases).

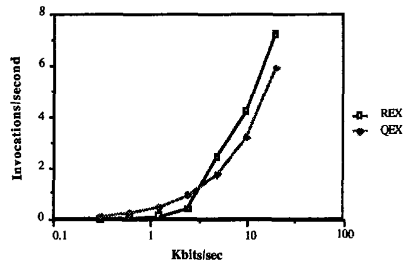

Figure 4(a) Comparison of throughput against data rates.

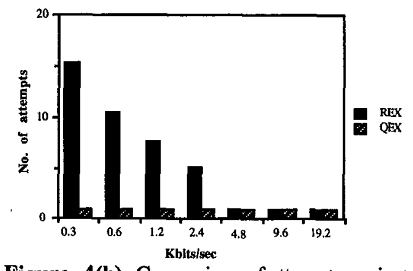

Figure 4(b) Comparison of attempts against data rates.

Figure 4(a) plots the number of invocations achieved per second against varying bandwidths. This figure shows that QEX works slightly better at low bandwidth (i.e. up to approximately 4.8 Kbits/s). At higher data rates however REX performs better. The reason for QEX performing better at low bandwidths is illustrated in figure 4(b). This shows the number of attempts to send a packet against data rates. The large number of attempts for REX at low bandwidth can be explained by the inappropriate retry interval. The better performance of REX at higher bandwidths is explained by the optimised behaviour of the protocol when message transmission time is small (small packet size or fast network) and application delay is very low. In this case, the reply is received by the client before the retry time-out and so the retry/ack interaction is unnecessary. If the next call to the server is ready to go immediately on receipt of the reply then the next call informs the server that its reply was received and the reply retry/ack interaction doesn't occur. Therefore, in experiment 1, with repeated invocations of small message sizes, REX uses less messages than QEX.

Note that this experiment highlights a worst case scenario for QEX, a single client repeatedly sends small packets to a single server that replies as quickly as possible. In real scenarios object interactions are more complex; typically messages are more sporadic and experience more pronounced application delays. In these scenarios REX will behave as shown in figure 2(a), sending more messages than QEX.

Experiment 2: Comparing REX and QEX (With Fragmentation)

The aim of the second experiment is to examine the performance of QEX against REX as before, though with an invocation payload that causes the fragmentation mechanism to be used. The amount of data transferred each time was 2K/bytes.

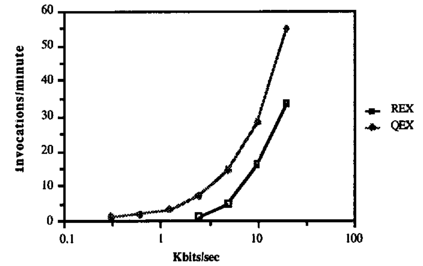

Figure 5(a) Comparison of throughput against data rates.

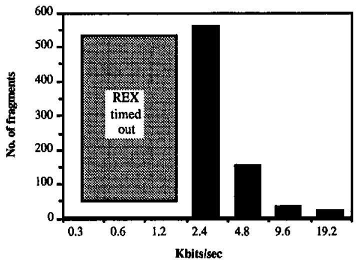

Figure 5(b) Fragments sent by REX for different data rates.

Figure 5(a) plots the number of invocations achieved per minute against varying bandwidths. This figure shows that QEX works consistently better than REX. The reason for this is illustrated in figure 5(b). This shows the number of attempts made by REX to send a two fragment invocation against a range of data rates. After one invocation at 2.4Kbps, REX has sent and queued over 590 fragments, a second invocation would not be possible. Below this rate, the invocation times-out before a single request/reply can get through (over 6,000 fragments waiting to be sent). This is because the REX fragmentation mechanism does not operate any form of congestion control once invocations become fragmented. In contrast, QEX maintains a consistent rate of two fragments (i.e. the minimum possible) over all the data rates tested once adaptation is completed.

Experiment 3: Adaptation of QEX

The third experiment shows how quickly QEX adapts to sudden changes in channel bandwidth. To do this we measure the number of unnecessary retries before the protocol adapts to the new bandwidth. This is measured against different fractional rate changes. A fractional change of 1/2 implies the bandwidth is dropped to a half of its original value. The figure is calculated by averaging over different ranges (i.e. 9.6-4.8 Kbits/s, 4.8-2.4 Kbits/s, etc). The highest figure is simply adopting the default tunings of REX; the lowest figure is 300 bits/s.

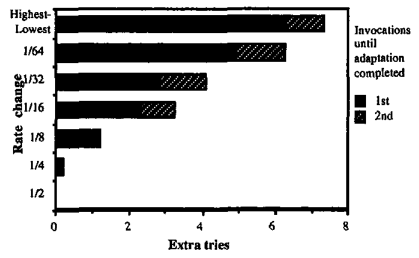

Figure 6(a) Extra tries per invocation until adaptation completed.

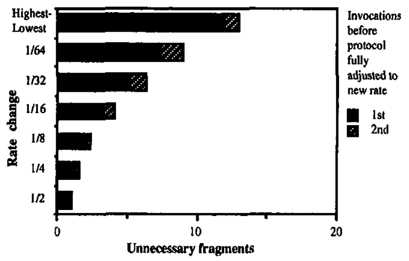

Figure 6(b) Extra fragments per invocation until adaptation completed.

Figure 6(a) illustrates adaptation for non-fragmented invocations (identical size to those used in experiment 1). As the change in rate is instantaneous, the protocol continues transmitting at the current rate until a change in round-trip time is detected. Once the protocol realises the rate has changed it begins adapting to the new rate, so less extra packets are transmitted for the next invocation, and so on. The figure shows the number of unnecessary reties for the first and then the second message after the change. In practice no more than two invocations are needed before the protocol has fully adapted to the new rate. From this figure we can also see that where bandwidth halves, no unnecessary retries occur in adapting to the new rate. When bandwidth drops to an eighth of the former rate, on average one unnecessary retry is sent. Over the maximum drop in bandwidth less than 8 unnecessary retries are sent.

Figure 6(b) illustrates adaptation for fragmented invocations (with a payload size of 2K/bytes as used in experiment 2). As for non-fragmented adaptation, the protocol continues transmitting at the current rate until a change in round-trip time is detected. Again, once the protocol realises the rate has changed, it begins adapting to the new rate, requesting explicit negative acknowledgements after every fragment until it has sufficient round-trip times to transmit no redundant fragments. Once the rate becomes stable the tight fragment/nack coupling is relaxed more and more over time. Although extra fragments are transmitted while the protocol is adapting, these are only redundant if they are unnecessary retransmissions of fragments that the partner has already received. If the change in bandwidth occurs during a long invocation (with many fragments) then these fragments would have to be transmitted anyway and can be treated as congestion with the usual backoff strategy. In this case the protocol needs no more than two invocations before full rate adaptation is complete. Less than two additional fragments are transmitted when the bandwidth halves.

5 CONCLUDING REMARKS

This paper has reported on experiences with developing mobile applications using the ANSAware distributed systems platform. From our experiences, ANSAware provides the necessary infrastructure to enable services to be accessed in a heterogeneous environment spanning multiple organisational domains. However, changes were required to both the binding and trading functions to enable operation in a mobile environment. In particular, explicit QoS-managed (stream and operational) bindings were added and changes were made to the linking policy in trading. These changes mean that our modified platform is more closely aligned to the emerging RM-ODP standard. In addition, the REX protocol has been replaced by QEX, a protocol which adapts to the underlying network bandwidth and provides feedback to the application on the quality-of-service obtained. The paper concluded with an evaluation of the extended platform and the results of a number of performance tests on QEX. Although theoretically the QEX protocol can lead to more messages being transmitted than with REX, we believe the additional overhead is justified because QEX's congestion control and rate adaptation enable operation over a variety of bearer services. In addition, in practice we have found it rare that REX can optimise to the simple call/ reply case and once the rate settles QEX's fragmentation mechanism uses less messages than REX's since the number of negative acknowledgements required are tuned by QEX.

REFERENCES

[Abdel-Wahab91] Abdel-Wahab, H.M., and M.A. Feit. "XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration." *Proceedings IEEE Tricomm '91: Communications for Distributed Applications and Systems*, 1991, Chapel Hill.

[APM93] A.P.M. Ltd. "ANSAware 4.1 Application Programming in ANSAware", Document RM.102.02, A.P.M. Cambridge Limited, Poseidon House, Castle Park, Cambridge CB3 0RD, UK, February 1993.

[Cáceres94] Cáceres, R., and L. Iftode. "The Effects Of Mobility on Reliable Transport Protocols." *Proceedings 14th International Conference on Distributed Computer Systems*, Poznan, Poland, 22-24 June, 1994. pp. 12-20.

[Coulson95] Coulson, G., G.S. Blair, F. Horn, L. Hazard, and J.B. Stefani, "Supporting the Real-Time Requirements of Continuous Media in Open Distributed Processing", *To appear in Computer Networks and ISDN Systems, Special Issue in Open Distributed Processing*, 1995.

[Cross93] Cross, A.D., J.R. Brailsford, and A.T. Brint, "Expert Systems to Support Network Switching", *Proceedings 12th International Conference on Electricity Distribution*, CIRED 1993, Birmingham, U.K., pp 17-21 May, 1993.

[Davies92] Davies, N., G. Coulson, N. Williams, and G. S. Blair, "Experiences of Handling Multimedia in Distributed Open Systems", *Proceedings of the 3rd Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS III)*, Newport Beach, CA, pp. 173-190, March 1992.

[Davies94] Davies, N., S. Pink, and G.S. Blair, "Services to Support Distributed Applications in a Mobile Environment", *Proceedings 1st International Workshop on Services in Distributed and Networked Environments*, Prague, Czech Republic, June 1994.

[Davies95a] Davies, N., G.S. Blair, A. Friday, A.D. Cross, and P.F. Raven, "Mobile Open Systems Technologies For The Utilities Industries", CSCW Issues for Mobile and Tele-Workers, Dix, A. (ed), Springer-Verlag, 1995.

[Davies95b] Davies, N., G.S. Blair, K. Cheverst, and A. Friday, "A Network Emulator to Support the Development of Adaptive Applications", *Proceedings 2nd USENIX Symposium on Mobile and Location Independent Computing*, Ann Arbor, Michigan, April 1995.

[Davies95c] Davies, N., G.S. Blair, K. Cheverst, and A. Friday, "Supporting Collaborative Applications in a Heterogeneous Mobile Environment", *To appear in Computer Communications, Special Issue of Mobile Computing*, 1995.

[Duchamp92] Duchamp, D., "Issues in Wireless Mobile Computing", *Proceedings 3rd Workshop on Workstation Operating Systems*, Key Biscayne, Florida, U.S., pp. 2-10, 1992.

[ISO95a] ISO/IEC Draft Recommendation X.902, International Standard 10746-1, "ODP Reference Model: Overview", January 1995.

[ISO95b] ISO/IEC Recommendation X.902, International Standard 10746-2, "ODP Reference Model: Descriptive Model", January 1995.

[ISO95c] ISO/IEC Recommendation X.903, International Standard 10746-3, "ODP Reference Model: Prescriptive Model", January 1995.

[Kantarjiev93] Kantarjiev, C.K., A. Demers, R. Frederick, R.T. Krivacic, and M. Weiser. "Experiences with X in a Wireless Environment." *Proceedings USENIX Symposium on Mobile and Location Independent Computing*, Cambridge, Massachusetts, U.S., pp. 117-128, August 1993.

[Katz94] Katz, R.H. "Adaptation and Mobility in Wireless Information Systems.", IEEE Personal Communications Vol. 1 No. 1. pp 6-17, 1994.

[Lauwers90] Lauwers, J.C., and K.A. Lantz. "Collaboration Awareness in Support of Collaboration Transparency: Requirements for the Next Generation of Shared Window Systems", *Proceedings CHI'90*, pp. 303-310, 1990.

[Raymond93] Raymond, K., "Reference Model of Open Distributed Processing: A Tutorial", *Proceedings 2nd International IFIP Conference on Open Distributed Processing*, Berlin, Germany, pp. 3-14, September 1993.

[Westervelt91] Westervelt, J., "Introduction to GRASS 4", GRASS Information Centre, U.S. Army CERL, Champaign, Illinois, U.S. July 1991.

BIOGRAPHY

Adrian Friday graduated from the University of London in 1991. Since 1992 he has been a research student at Lancaster University, working towards his PhD on "Infrastructure Support for Adaptive Mobile Applications". He has been an active participant in the MOST project involving Lancaster University and E.A. Technology.

Gordon Blair is currently a senior lecturer in the Computing Department at Lancaster University. He completed his PhD in Computing at Strathclyde University in 1983. Since then, he was an 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 system development.

Keith Cheverst is a research assistant with the Computing Department at Lancaster University working on a project concerned with research into reactive services for mobile environments. This project builds on the research issues identified in the MOST project with which he was formally associated. He is also currently involved in research for his PhD which focuses on the special requirements of groupware applications designed to operate in weakly connected environments. In particular, he is concentrating on establishing ways of increasing the dependability of groupware applications.

Nigel Davies graduated from Lancaster University in 1989 and later that year joined the 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 PhD in 1994. After a spell as a visiting researcher at the Swedish Institute of Computer Science (SICS) where he worked on mobile file 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.