A Resource and QoS Management Framework for a Real-Time Event System in Mobile Ad Hoc Environments

Hector A. Duran-Limon[1], Gordon S. Blair[1], Adrian Friday[1], Thirunavukkarasu Sivaharan[2], George Samartzidis[1]

Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, UK.

{duranlim,gordon,gs,adrian}@comp.lancs.ac.uk[1], t.sivaharan@lancaster.ac.uk[2]

Abstract

A new class of applications can now be envisaged with the emergence of both mobile ad hoc computing and ubiquitous computing. Applications of such kind are characterised by being largely distributed and proactive, i.e. able to operate without human intervention. The anonymous and asynchronous paradigm, which is advocated by event models, has shown to be well-suited for this kind of applications. However, current real-time event-oriented middleware technologies do not provide a complete solution for the requirements of mobile ad hoc environments. In this paper, we present the research carried out on both a resource and a QoS management framework to achieve real-time support for an event system operating in mobile ad hoc environments. The framework makes use of both reflection and component technology. The implementation of our resource system is developed in OpenCOM, which is a lightweight, efficient and reflective model based on Microsoft's COM.

1. Introduction

Over the last few years we have seen the proliferation of embedded mobile systems such as mobile phones and PDAs. Ubiquitous computing is also taking off in which multiple cooperating possibly embedded controllers are used. A new kind of applications can now be envisaged with the emergence of both mobile ad hoc computing and ubiquitous computing. Applications of such kind are characterised by being largely distributed and proactive, i.e. able to operate without human intervention. A set of further characteristics are also involved such as context awareness as a means to sense the surrounding environment. Examples of these applications include automatic car control systems in which cars are able to operate independently and cooperate with each other to avoid collisions. Another example is an air traffic control system whereby thousands of aircraft are proactively coordinated to keep them at safe distances from each other, direct them during takeoff and landing from airports and ensure that traffic congestion is avoided. Smart room systems can also be foreseen in which the intensity of light, room temperature and some other features are automatically tuned according to the user preferences of the persons present in the room.

The CORTEX Project[1] is examining fundamental issues relating to the support of such applications, including the development of middleware for this domain. The CORTEX approach is based on anonymous and asynchronous event models. Such models are well-suited to ad hoc environments including a large number of autonomous processing units. These many-to-many communication scenarios are well supported by the anonymous dissemination of information. In addition, asynchronous communication is well-suited to systems where frequent disconnection is anticipated as blocking conditions are avoided. Further requirements include support for mobility and non-functional properties such as timeliness and reliability since some of these applications are time- and safety-critical. However, current event-oriented middleware technologies do not provide a solution for all the challenges imposed by these applications. A higher-level programming model is also needed to deal with the complexity of multiple event interactions with different QoS demands. In addition, current programming models do not generally differentiate between functional and non-functional properties. The result of this is a code tangled with different aspects which is more difficult to write, read and reuse.

As regards timeliness requirements, we believe that resource management plays an important role in providing support for real-time applications. In particular, the mechanism that allocates resources in the system should ensure that critical activities will be provided with enough resources to carry out their tasks in a predictable way. Changes in the availability of network resources and periods of disconnection are frequently experienced as mobile computing environments are highly dynamic. Furthermore, mobile applications typically operate on devices with scarce resources, e.g. CPU capacity, system memory and battery life. Therefore, support for the predictable and efficient management of the system resources as well as resource reconfiguration capabilities for achieving adaptation are required. An example of the latter is a redistribution of both CPU-time and memory to

the set of activities that the system performs, thus, ensuring that time-critical activities are not disturbed.

In this paper, we present both a resource management framework and a quality of service (QoS) management framework for the real-time support of an event system in mobile ad hoc environments. A higher-level programming model for the resource management of event systems is introduced. In addition, aspect-oriented techniques [1] are used for separating out non-functional aspects from functional aspects. This practice reduces the complexity of programming real-time event systems.

The paper is structured as follows. Section 2 introduces some background on the CORTEX architecture and associated middleware. Section 3 presents the approach for resource management and QoS management support. The implementation of the system's prototype is then described in section 4. Section 5 draws some concluding remarks.

2. Background on the CORTEX architecture

Central to the CORTEX architecture is the notion of a *sentient object* [2] which is defined as an entity that is able to both consume and produce events. That is, sentient objects are entities that receive events, process them and generate further events. Input events are received either from sensors or from other sentient objects. Similarly, output events are sent either to actuators or other sentient objects. Sentient objects are autonomous entities that are able to sense their environment. Interestingly, sentient objects have a proactive role in that they are capable of taking decisions and performing some actions (i.e. generate further events) based on the information sensed. Hence, sentient objects include a control logic which realises the decision-making mechanism.

We are building a middleware platform in order to provide support for CORTEX applications and hence to support the sentient object abstraction described above. In order to tackle the requirements imposed by ad hoc environments, configuration and reconfiguration capabilities are introduced in the middleware architecture. Based on previous experience in the construction of reflective middleware [3], we make use of *reflection*, *component technology* and *component frameworks (CFs)*. Reflection is a means by which a system is able to inspect and change its internals in a principled way [4]. A reflective system is able to perform both self-inspection and self-adaptation. To accomplish this, a reflective system has a representation of itself. This representation is causally connected to its domain, i.e. any change in the domain must have an effect in the system, and vice versa. On the other hand, component technology introduces more configuration and reconfiguration capabilities into distributed applications and increases the level of reuse. Within the context of CORTEX a component is basically "a unit of composition with contractually specified interfaces and explicit dependencies only" [5]. The granularity of a component may be diverse ranging from components that realise only a part of the machinery of a single sentient object to components that encompass two or more sentient objects.

The environment support for the interaction of sentient objects is also conceptualised as a componentised middleware platform. In fact, the middleware is structured in terms of component frameworks [5] as shown in figure 1. Essentially, component frameworks are "collections of rules and interfaces that govern the interaction of components 'plugged into' them" [5]. In other words, a component framework is a reusable architectural design targeting a specific domain whereby desired architectural properties and invariants are enforced. The publish/subscribe CF realises the CORTEX event model [6]. The functionality of the control engine of a sentient object is provided by the context CF. Services can be dynamically discovered by the use of the service discovery CF [7]. Facilities for multicast in ad hoc environments are then provided by the multicast CF. The QoS management CF arbitrates the allocation of resources and provides facilities for monitoring and adaptation of QoS. Lastly, the resource management CF controls the resources used by all the CFs. Central to this paper are the resource management CF and the QoS management CF which are presented in the following sections. Some essential details of the publish/subscribe model are also presented below.
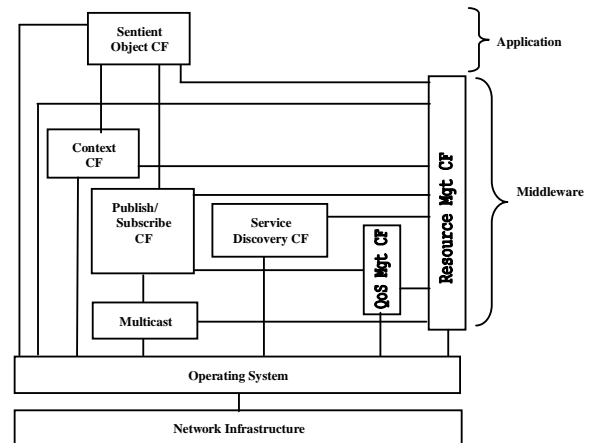

Figure 1. The middleware architecture of CORTEX

The publish/subscribe CF is based on an event model called STEAM [6] that allows for the dissemination of events in an ad hoc network environment. Producers publish events and consumers subscribe to event types to be able to receive events. An implicit event model is followed in which there is not an event broker or mediator; instead brokering functions are implemented at both the consumer and the producer side. This is useful in an ad hoc environment where permanent communication with a broker is unlikely to be maintained. An *event type*,

from which *events* are instantiated, is basically characterised by a subject, data information, context information (e.g. location, time, etc) and is associated with QoS properties (e.g. timeliness). Interestingly, event filtering is achieved by performing both subject-based and proximity filtering at the producer side. The former describes the particular event types that the consumer is interested in. The latter specifies the geographical area within which event types are valid. The consumer uses content-based filtering in which an expression is evaluated against a set of values included in an event.

3. Real-time support

3.1. Motivation

Several middleware services using the publish/subscribe communication model have been developed such as the OMG's CORBA Event service specification [8] , Sun Microsystems's Java distributed event specification [9], TIBCO's TIB/Rendezvous^TM, SIENA [10] (a wide area event notification service), Gryphon [11] (a distributed content based notification service), and JEDI [12] (a Java event based distributed infrastructure). However, most of them assume a fixed network infrastructure and require the use of stationary event brokers. Furthermore, most existing platforms have limited support for the negotiation or enforcement of QoS requirements. Some research has been carried out to overcome this situation, for example TAO's Event service [13] and the CORBA Notification Service [14]. However, these systems still require the existence of a fixed infrastructure. In addition, current approaches to QoS management in mobile ad hoc networks [15, 16, 17] only provide support for a limited number of service classes (usually two) and the effect of hidden terminals is generally not addressed. Other approaches such as the one described in [18] provide further support for service differentiation but operate at the MAC layer which makes the system deployment more difficult.

Real-time event systems in mobile ad hoc environments impose a number of new unsolved challenges. Such environments are characterised by being highly unpredictable. A peer-to-peer communication model is generally used in ad hoc networks. Importantly, nodes act as routers to reach nodes that are out of the transmission range. Communication delays between nodes may vary unexpectedly as the number of hops to reach the destination changes. In addition, a geographical area may unexpectedly become congested, resulting in the lack of communication resources. Moreover, periods of disconnection are likely to happen at any time due to the conditions of the geographical area. The transmission signal can be severely affected by bad weather conditions and obstacles such as trees, hills and buildings. In the worse case, there may be a network partition whereby one or more nodes are unreachable. In addition, unpredictability in ad hoc environments is also dictated by the fact that new event types with different QoS demands may be dynamically discovered, e.g. new event types are advertised at various locations. In this situation, no a priori resource consumption assumptions can be made.

Further challenges include the development of suitable mechanisms and protocols for the resource management of resource constrained devices such as PDAs and sensors. This kind of device has a limited battery life, hence, power management is needed to make an efficient use of the energy. Memory and CPU resources are also limited, therefore, suitable resource management mechanisms are required. As any node may be a router, resource scarce devices need to evaluate the amount of traffic they are able to route without having a negative impact in pursuing their main goal.

A considerable amount of event types with different QoS requirements may be involved in a real-time event system. This fact makes the programming of real-time event systems tedious and error-prone. Importantly, there is a lack of programming abstractions able to present a global view of how resource management is performed along multiple nodes to pursue common goals. Furthermore, resource management of complex relationships among event types are usually not represented by current event programming models. That is, there are not programming abstractions capturing the resource management of event flows across the network.

Under the current state of the art in mobile ad hoc network technologies, it is not feasible to offer hard real-time guarantees for communication resources except in special cases. Such guarantees can only be offered if certain specific conditions are met such as a scenario whereby a limited number of nodes are moving in an obstacle-free area at the same speed and direction. Although, hard real-time guarantees can still be provided for local resources such as CPU, only soft real-time guarantees can be offered for communication resources in most cases whereby these resources are dynamically allocated according to deadlines.

Another tough challenge arises here: how can we deal with applications demanding hard real-time guarantees when the underlying infrastructure is only capable of offering soft real-time guarantees. A number of issues have to be considered to address this problem. A high probability of meeting deadlines has to be offered. We also believe that such infrastructure has to be adaptable and flexible to deal with the highly dynamic and unpredictable nature of the environment. Resource management plays an important role in this adaptation process in terms of both resource awareness and dynamic reallocation of resources. Crucially, fail-safe mechanisms are needed to bring the application to a safe state when timing failures are detected. Therefore, QoS management support is required for monitoring QoS violations and triggering both adaptation and fail-safe procedures when required. In addition, QoS management is required to

arbitrate the allocation of network resources whereby admission control tests are performed and more resources are conceded to events with shorter deadlines and higher criticality.

This paper does not intend to provide solutions for all the problems outlined above. Rather, the paper focuses in providing a) a higher-level programming model for resource management, b) support for resource awareness and resource adaptation, and c) QoS management support. The first two are addressed by the resource management CF, i.e. the task model and resource model, respectively. The last one is tackled by the QoS management component framework.

3.2. Resource management component framework

3.2.1. Overview. The resource management CF encompasses both a task model and a resource model. The task model permits the user to model resource management of both coarse- and fine-grained interactions. Hence, the task model allows for the high-level analysis and design of resource management for event systems. There is a close relationship between the task model and the resource model. Tasks have an associated pool of resources which is defined by the resource model. More specifically, the resource model allows us to model different types of resources at multiple levels of abstraction. Both coarse- and fine-grained resource configuration and reconfiguration are feasible. Further details of the task and resource models are introduced in turn below.

3.2.2. The task model. From the programmatic point of view, a task may involve either a single invocation sequence or multiple invocation sequences. The simplest case for a sequence is where only one operation is invoked. A *task* is defined as a logical unit of computation which has an amount of resources allocated. Examples of tasks are activities performed by the system such as transmitting audio over the network or compressing a video image, which has an amount of resources assigned for its execution. The task model is concerned with both application services and middleware services. Thus, we take a task-oriented approach for managing resources in which services are broken into tasks and are accommodated in a task hierarchy. Top-level tasks are directly associated with the services provided by a distributed system. Lower-levels of this hierarchy include the partition of such services into smaller activities, i.e. *sub-tasks*. Sub-tasks are denoted as follows:

task.sub-task.sub-sub-task…

This approach offers resource management modelling of both coarse- and fine-grained interactions. The former is achieved by defining coarse-grained tasks (i.e. tasks spanning components and address spaces boundaries) and the latter is done by using task partitioning. A task may span the boundaries of a component, an address space and even those of a node. *Composite tasks* include two or more sub-tasks and

may involve either a single or multiple operation invocation sequences. In the former case, sub-tasks are interleaved whereas in the latter case sub-tasks are disjoint. Sub-tasks that are not further partitioned are called *primitive tasks* and are only related to a single operation invocation sequence. However, *distributed tasks* involve two or more nodes. It should be noted that sub-tasks may also be composite and even distributed.

Finally, different tasks may be interconnected. For instance, a component running one task may invoke another component concerned with a different task. Such a method invocation represents a *task switching point*. Thus, a task switching point corresponds to a change in the associated resource pool to support the execution of the task that has come into play.

3.2.3. Resource model. The most important elements of the resource model are *abstract resources*, *resource factories* and *resource managers* [19]. Abstract resources explicitly represent system resources. In addition, there may be various levels of abstraction in which higher-level resources are constructed on top of lower-level resources. Resource managers are responsible for managing passive resources such as memory or disk. Furthermore, *resource schedulers* are a specialisation of managers and are in charge of managing processing resources such as threads or virtual processors (or kernel threads). Lastly, the main duty of resource factories is to create abstract resources. For this purpose, higher-level factories make use of lower-level factories to construct higher-level resources. The resource model then consists of three complementary hierarchies corresponding to the main elements of the resource model. Importantly, *virtual task machines* (VTMs) are top-level resource abstractions and they may encompass several kinds of resources (e.g. CPU, memory and network resources). There is a one-to-one mapping between tasks and VTMs. Hence, a VTM represents a virtual machine in charge of supporting the execution of its associated task.
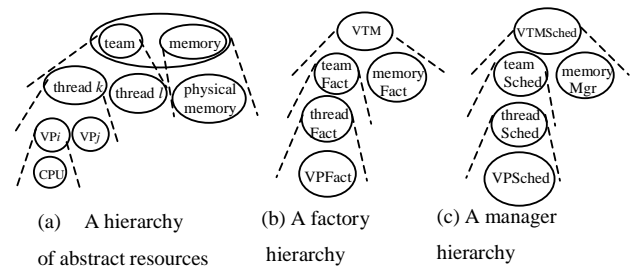

(a) A hierarchy of abstract resources

(b) A factory hierarchy

(c) A manager hierarchy

Figure 2. A Particular Instantiation of the resource model

As an example, a particular instantiation of the framework is shown in figure 2 (note, however, that the framework does not prescribe any restriction in the number of abstraction levels nor the type of resources

modelled). At the top-level of the resource hierarchy is placed a VTM, as shown in figure 2 (a), which encompasses both memory buffer and a team abstraction. The team abstraction in turn includes two or more user level threads. Moreover, a user level thread is supported by one or more virtual processors (VPs), i.e. kernel level threads. At the bottom of the hierarchy are located physical resources. In addition, a VTM factory is at the top of the factory hierarchy and uses both a memory and a team factory. The team factory then is supported by both the thread and the virtual processor factory as depicted in figure 2 (b). Finally, the manager hierarchy is shown in figure 2 (c). The team scheduler and the memory manager support the VTM scheduler to suspend a VTM by temporally freeing CPU and memory resources respectively. The thread scheduler in turn allows the team scheduler to suspend its threads. Finally, the VP scheduler supports the pre-emption of virtual processors. Conversely, this hierarchy also provides support for resuming suspended VTMs. Further details of the resource model and details of how dynamic resource reconfiguration is achieved can be found in [19].

3.3. Resource management support

Our general approach for resource management involves providing hard, soft and non real-time guarantees for local resources such as CPU and memory. In contrast, soft and non real-time guarantees are provided for communication resources. A static analysis is carried out for the hard guarantees which takes into account even the most rare event with a hard deadline. In the case of soft guarantees, most of the analysis is carried out at run-time. More specifically, events disseminated in the system can be periodic, aperiodic and sporadic. In fact, it is anticipated that there is a substantial share of aperiodic and sporadic traffic in the targeted applications. Using a pessimistic approach for resource reservation of sporadic events negatively impacts the utilisation of the resources system. Notably, the approach presented in [20] achieves a more efficient utilisation of the resources whereby network bandwidth is reserved for hard real-time events whereas no resource reservation is made for soft and non real-time events. As there is a risk of soft real-time tasks to become overloaded, we additionally define reservations for soft real-time tasks obtained from two sources. Firstly, resources are obtained from those that were not reserved by hard real-time tasks. The second source is a bounded overlapping region with the resources assigned to hard real-time tasks. As a result, some soft real-time events will miss their deadlines when hard real-time events use this region but the probability of meeting their deadlines will increase. We then have the best effort soft real-time and non real-time class whereby no resource reservations are made. Events belonging to the two soft real-time classes are scheduled according to their deadlines and laxity values. Criticality values are used to distinguish between soft real-time tasks and best effort soft real-time tasks. Finally, non

real-time events are scheduled when no hard- and soft-real time events are awaiting execution.

Table 1. Example of the mapping of tasks to event types

Task	Event Type
CarControl.emergencyStop	EmergencyStop
CarControl.touristInfo	restaurant, museum, theatre, cinema, map

The steps involved in the use of both the task and resource models for the real-time support of the publish/subscribe systems are listed below.

1. Task model design.
2. Resource requirements and schedulability analysis.
3. Specification of resource and task model in a description language.
4. The description language definitions are processed. As result, both the application and the middleware are configured.

Although we envisage the dynamic discovery of tasks in the targeted applications, the first step involves identifying the initial tasks that the system will encompass along with their criticality, associated event types and task graph configurations. One or more events can be associated with a task. Table 1 shows an example of mapping tasks to events. Task graph configurations involve the definition of task switching points, sub-tasks and of how tasks are interconnected. In addition, tasks are defined at both the application and the middleware level. As an example of application tasks consider a car control application associated with the top-level task "carControl" with a number of subtasks including "carControl.carLocation" and "carControl.sportNews". The former task is more critical as it informs of the other cars' current position whereas the latter, in charge of providing sport news information, is a non-critical task. The task "carControl.emergencyStop", however, represents a higher critical task as it informs of cars suddenly braking. Examples of middleware tasks include "carControl.eventFiltering" and "carControl.eventDispatching" which are in charge of the filtering and dispatching of events, respectively. Finer grained control can also be achieved by defining middleware tasks on a per event type basis. For instance, the task "carControl.emergencyStop.eventDispatching" allocates specific resources for the dispatching of emergency stop events.

The second step regards the mapping of application-level QoS values to resource parameter values. This may be achieved by mathematical translation or trial-and-error estimations as described in [21]. The specific resource

requirements are then stored in a local repository. For this purpose, distributed tasks are partitioned into primitive tasks whereby each primitive task is related to a particular node with specific needs. For instance, the distributed task "carControl.carLocation" is divided in the primitive tasks "carControl.carLocation.node$_m$" and "carControl.carLocation.node$_n$". Hence, diverse resource requirements may be needed for a distributed task in different nodes, as shown in figure 3. Events carry their dynamic scheduling parameters (such as deadlines) across the visited nodes whereas other QoS parameters (such as priorities, criticality, period, etc.) are locally obtained at the endpoints. In a more dynamic scenario, however, new event types associated with new tasks are discovered. The resource requirements of the new task are obtained from a remote task information repository. This repository is most likely to reside in the site of the event advertiser which may be either a mobile or fixed node. In case the information about such requirements is not available, a profiling technique can be used to obtain the requirements, i.e. resource usage is dynamically measured by profilers [22]. The consumer proceeds then to create at run-time a local subtask and an associated VTM able to cover the QoS demands of the new event type. Lastly, the schedulability analysis of the resources allocated to the hard real-time and the soft real-time classes is carried out off-line.
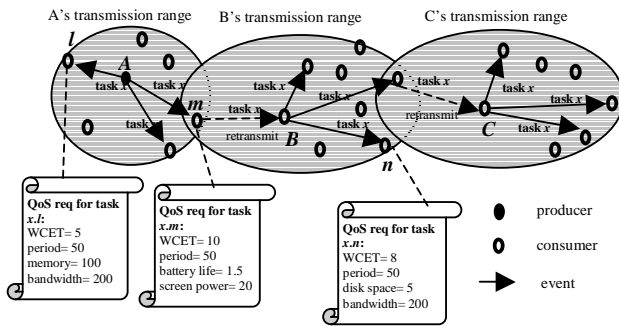

Figure 3. Example of task dissemination

Step 3 regards the use of an extension of the *resource configuration description language (RCDL)* [23]. The RCDL is a suite of aspect-oriented languages[2] that allows us to define a) the mapping between tasks and events, b) the component interfaces in which a particular task is triggered or switched, c) the system resources that are associated with each task and d) the resource requirements of a task instance for a particular deployment platform. The two former are achieved by the *task switch description language (TSDL)* whereas the last two are addressed by the *task description language (TDL)* and the *resource description language (RDL)*, respectively.

---

2 Aspect-oriented programming [1] allows for the decomposition of a program into aspects that cross-cut each other.

In the fourth step, RCDL definitions are processed by an interpreter that is in charge of configuring both the application and the middleware according to such definitions. Code is generated for achieving task switching. Generated code also includes a resource initialisation component in charge of creating the defined VTMs (i.e. resource reservation). A configuration file is also produced which is used at the system's load-time to plug in the generated components.

3.4. QoS management support

The highly dynamic nature of ad hoc networks makes the coordination of nodes competing for resources an essential issue. Therefore, *task managers* are introduced to control the allocation of resources according to both the QoS demands and the criticality of the tasks. There is a task manager per node. The resource management CF provides important support for the task managers in terms of resource awareness, resource reservation and resource reallocation. Resources for hard and soft real-time classes are statically reserved according to the defined tasks. Task instances obtain dynamic reservations when task managers receive requests from publishers (i.e. announce(), publish()) and consumers (i.e. subscribe()). For instance, when subscribing to a video on demand service supported by task *y*, resources are allocated if no other video sessions have been set up or enough resources are left for the new session. The resource requirements of announced and subscribed event types are obtained from a task information repository. As a result, resource allocation is granted for publishing and consuming events in the case of hard and soft real-time tasks. For that purpose, the task manager is in charge of requesting the appropriate resource managers to perform both the admission control tests and the reservation of resources when the test is successful.

Monitoring of QoS violations and adaptation procedures are carried out by the *timely computing base (TCB)* [24]. TCB is a framework that provides crucial time related services. More specifically, TCB supports the detection of timing failures and enforces the timely execution of safe-procedures and adaptation strategies. Fail-safe procedures ensure that the system is taken to a safe state when a critical failure is detected. In addition, two possible forms of adaptation are considered: a) tuning of application parameters and component reconfigurations (of the application and the middleware) and b) a redistribution of the resources used by tasks. The latter is achieved by the use of the resource management CF. In terms of monitoring, the TCB supports local measurements (e.g. the time that an operation takes to execute) and distributed measurements (e.g. the time taken to transmit a message). A coverage stability facility is also provided whereby the probability of meeting a deadline can be defined. In case going below the specified value,

adaptation procedures are triggered. The TCB employs a dedicated radio channel for the transmission of control messages. Thus, the payload is not burdened with extra traffic and TCB's control channel will have plenty of bandwidth for the timely transmission of its messages. Lastly, the TCB is conceptualised as an extension of the operating system services.

Network QoS management is achieved as follows. Firstly, every node is able to listen to traffic as the dissemination of packets is carried out by using an application level multicast protocol. Secondly, available bandwidth is fairly distributed among the nodes within a transmission area. For this purpose, a fully distributed protocol is used. Thirdly, a weighted fair scheduling policy [25] is used to allocate bandwidth within a single node to multiple service classes, each one of them associated with a particular task. A service class (i.e., a task) can be partitioned into sub classes (i.e., subtasks). In addition, classes and subclasses have a priority value. More critical classes have a higher priority. Different policies can be defined in case of resource contention. For example, critical messages can be scheduled according to their priority and use resources of lower level classes. Another example is a policy whereby a service class is provided with an exclusive bandwidth portion that is not shared even in resource contention conditions to avoid starvation.

The general principles behind the QoS management protocol are as follow. Every node is associated with a supported transmission rate (Stx) and a downgraded transmission rate (Dtx). The Stx defines the maximum rate at which a node is able to receive messages. A node obtains this value by fairly allocating a portion of the bandwidth according to both the amount of traffic and the number of nodes that are listened.
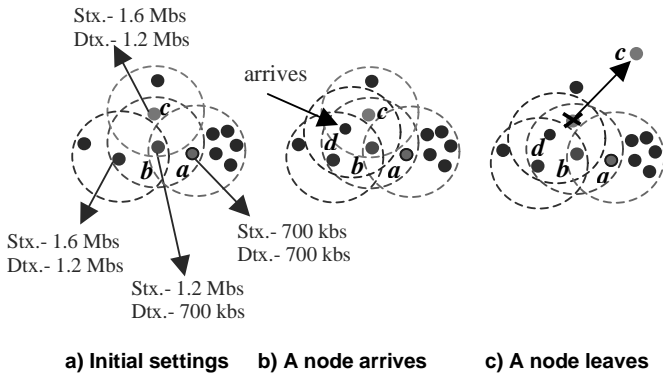

Stx.- 1.6 Mbs
Dtx.- 1.2 Mbs

arrives

Stx.- 1.6 Mbs
Dtx.- 1.2 Mbs

Stx.- 700 kbs
Dtx.- 700 kbs

Stx.- 1.2 Mbs
Dtx.- 700 kbs

a) Initial settings   b) A node arrives   c) A node leaves

Figure 4. Example of a mobile ad hoc scenario

The Dtx determines the maximum rate at which a node can transmit messages without negatively affecting the neighbouring nodes. In addition, a node periodically broadcasts its Stx and Dtx values to the neighbours (i.e., the nodes located within the transmission range). As a consequence, the Dtx is set to the lowest Stx received. As an example consider the scenario depicted in figure 4 (a)

whereby node a is located in a highly populated area and can receive packets at 700 kbs. Node b which is in a less populated area, is unaware of the traffic behind node a. Such a situation could negatively affect the availability of network resources of node a. That is, node b could transmit at a rate higher than 700 kbs considering it is suitable to do so as the sensed traffic is low. However, after exchanging a number of messages these two nodes, node b becomes aware of the maximum supported rate of node a. Similarly, node c sets its transmission values according to the maximum supported values of the vicinity. Furthermore, the bandwidth of a node is further distributed among the node's service classes. As a result, each service class is also provided with their own Stx and Dtx values.

Consider now the case of node d arriving to the vicinity as shown figure 4 (b). After a period of time, this node detects new traffic and request the QoS settings (i.e., the Stx and Dtx values) to the nearest nodes. As a consequence, these values are provided and the QoS settings of node d are updated. Node d then informs of its new settings and the neighbouring nodes update their settings by taking into account the bandwidth that the new node will use. Figure 4 (c) shows the case of node c leaving the area. After a timeout has expired, the neighbouring nodes assume this node has left when messages from this node are no longer received. As a result, the bandwidth released by node c is fairly distributed among the nodes within the vicinity. That is, each node allocates itself a portion of the bandwidth according to the QoS settings of the neighbouring nodes.

4. Implementation

An initial implementation effort has been carried out using OpenCOM [26], a reflective component model developed at Lancaster University. In addition, a prototype of a STEAM-like publish/subscribe system has been implemented on both Windows NT and Windows CE 3.0. A particular instantiation of the resource framework was also developed to extend the event system with resource management capabilities. Support for the management of CPU resources is provided at this stage. The resource management system offers dispatching predictability of real-time events. CPU reservations are organised in rounds. A round (or dispatch table) contains a number of time slots which are assigned to hard and soft real-time tasks. The framework has been realised by a two-level scheduling model implemented in Windows CE.

5. Conclusions

We have presented a novel approach for the real-time support of event models in mobile ad hoc environments. More concretely, we introduced both a resource

management framework and a QoS management framework for the real-time support of a publish/subscribe system. The presented task model allows for the high-level analysis and design of the resources system whereby both coarse- and fined-grained resource reconfiguration are feasible. The task model considerably diminishes the complexity of programming real-time event models. In addition, rather than burdening the application programmer with the job of defining the QoS requirements, these requirements are defined in a series of aspect-oriented languages by a possible different programmer, e.g. a QoS programmer.

QoS management support was also presented whereby task managers arbitrate resource reservation. In addition, the QoS management system makes use of the TCB services to detect timing failures in which case adaptation and fail-safe procedures are triggered. Also, a QoS management protocol supporting multiple service classes was presented.

Ongoing work regards simulation of the QoS management protocol and the experimental evaluation of the thread scheduling system. Also an implementation of the QoS management framework is underway. Furthermore, we are working on the implementation of an automatic car control application. The application includes a number of car robots which are controlled by Pocket PC handheld devices running Windows CE. Future work includes the development of an RCDL processor tool.

References

[1] Kiczales, G., J. Lamping, et al. "Aspect-Oriented Programming." *Proc. 11th European Conference on Object-Oriented Programming (ECOOP'97)*, Jyvaskila, Finland. June 1997. 220-241

[2] Verissimo, P. and A. Casimiro. "Event-Driven Support of Real-Time Sentient Objects." *Eighth IEEE International Workshop on Object-oriented Realtime Dependable Systems (WORDS 2003)*, Guadalajara, Mexico. January 2003.

[3] Blair, G. S., G. Coulson, et al. "The Design and Implementation of Open ORB version 2." *IEEE Distributed Systems Online Journal* **2**(6): 2001.

[4] Maes, P. "Concepts and Experiments in Computational Reflection." *Proc. ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'87)*, Orlando, FL USA. October 1987. 147-155

[5] Szyperski, C. "Component Software: Beyond Object-Oriented Programming." Harlow, England, Addison-Wesley. 1998.

[6] Rene Meier, V. C. "Steam: Event-based Middleware for Wireless Ad Hoc Networks." *In Proceeding of the International Workshop on Distributed Event-Based Systems (ICDSC/DEBS'02)*, Vienna, Austria. 2002. pp. 639-644

[7] Grace, P., G. Blair, et al. "ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability." *In proceedings of International Symposium on Distributed Objects and Applications (DOA 2003)*, Catania, Sicily, Italy. November 2003.

[8] Object Management Group, "CORBAServices: Common Object Services Specification." *95-3-31*, Dec 1998.

[9] Sun Microsystems, Inc, "Java Distributed Event Specification." July 1998. http://www.javasoft.com/products/javaspaces/specs.

[10] Carzaniga, A., Rosenblum, D. and Wolf, A. "Design and Evaluation of a Wide-Area Event Notification Service." *ACM Transactions on Computer Systems* **19**(3): pp 332-383, 2001.

[11] IBM Research. "Gryphon: An Information Flow Based Approach to Message Brokering." IBM Research. 1998. http://researchweb.watson.ibm.com/gryphon/home.html

[12] Cugola, G., Di Nitto, E., and Fuggetta, A. "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS." *IEEE Transactions on Software Engineering* **9**(27): pp 827-850, September 2001.

[13] Tim Harrison, D. L., Douglas C. Schmidt. "The Design and Performance of a Real-time CORBA Event Service." *Proceedings of OOPSLA '97*, Atlanta, GA, ACM. October 1997.

[14] Object Management Group, "Notification Service Specification." *telecom/99-07-01*, July 1999.

[15] Ahn, G.-S., A. T. Campbell, et al. "Supporting Service Differentiation for Real-Time and Best-Effort Traffic in Sateless Wireless Ad Hoc Networks (SWAN)." *IEEE Transactions on Mobile Computing* **1**(3): July-September 2002.

[16] Lee, S.-B. and A. T. Campbell. "Insignia: In-Band Signaling Support for QoS in Mobile Ad Hoc Networks." *Intl. Workshop on Mobile Multimedia Communication (MoMuc'98)*, Berlin. October 1998.

[17] Xue, J., P. Stuedi, et al. "ASAP: An Adaptive QoS Protocol for Mobile Ad Hoc Networks." *Procdings of the 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Beijing, China. 2003.

[18] Veres, A., A. T. Campbell, et al. "Supporting Service Differentiation in Wireless Packet Networks Using Distributed Control." *IEEE Journal of Selected Areas in Communication* **19**(10): October 2001.

[19] Duran-Limon, H. A. and G. S. Blair. "Reconfiguration of Resources in Middleware." *Seventh IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2002)*, San Diego, CA. January 2002.

[20] Kaiser, J., C. Brudna, et al. "A Real-Time Event Channel Model for the CAN-Bus." *11th Annual Workshop on Parallel and Distributed Real-Time Systems, held in conjunction with the International Parallel and Distributed Processing Symposium IPDPS*, Nice, France. April 2003.

[21] Nahrstedt, K., H.-h. Chu, et al. "QoS-Aware Resource Management for Distributed Multimedia Applications." *Journal of High-Speed Networking, Special Issue on Multimedia Networking* **7**: 227-255, 1998.

[22] Kalogeraki, V., P. M. Melliar-Smitm, et al. "Dynamic Scheduling for Soft Real-Time Distributed Object Systems." *In Proc. of Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, Newport Beach, CA, USA. March 2000.

[23] Duran-Limon, H. A. and G. S. Blair. "QoS Management Specification Support for Multimedia Middleware." *Accepted for publication in The Journal of Systems and Software. Elsevier Science Publisher.*: 2003.

[24] Verissimo, P. and A. Casimiro. "The Timely Computing Base Model and Architecture." *Transaction on Computers - Special Issue on Asynchronous Real-Time Systems* **51**(8): August 2002.

[25] Demers, A., S. Keshav, et al. "Analysis and simulation of a fair queueing algorithm." *Journal of Internetworking Research and Experience*: pp. 3-26, October 1990.

[26] Clarke, M., G. Coulson, et al. "An Efficient Component Model for the Construction of Adaptive Middleware." *IFIP/ACM Middleware'2001*, Heidelberg, Germany. November 2001.