An Architecture for Automated QoS Resolution in Wireless Systems

Behzad Bordbar[1] and Rachid Anane[2]
[1]*School of Computer Science, University of Birmingham, UK.*
B.Bordbar@cs.bham.ac.uk
[2]*School of Mathematical and Information Sciences, Coventry University, UK.*
R.Anane@coventry.ac.uk

Abstract: The pervasive nature of mobile and wireless systems has led to increased concerns over Quality of Service (QoS). In the prevailing models for QoS management, QoS resolution is achieved by table look-up, a feature that makes table access the focal point of activity. This approach suffers from two limitations, namely, an inability to deal with unexpected QoS requests, and a reliance on human intervention for update of information. This paper is concerned with the presentation of an architecture for supporting automated QoS resolution through verification. The architecture is modular and the QoS resolution function is performed by a subsidiary component, which combines knowledge base with resolution mechanism. This separation of concerns and the support for flexible QoS management has the advantage of accommodating new forms of QoS requests, and of minimising human intervention. An implementation of the QoS resolution architecture is presented in terms of Timed Automata. In addition, a framework is also introduced for extending QoS architectures such as ITSUMO.

1.Introduction

The pervasive nature of mobile systems has given rise to an increasing interest in Quality of Service. The provision and guarantee of an acceptable level of Quality Of Service (QoS) is seen as critical for a successful integration of wireless systems into IP network protocol [6][7][12].

Recent efforts to address these issues have focused on *QoS management* in distributed systems. In addition to the allocation of QoS resources, the main function of QoS management consists in accepting a QoS request and determining whether the request can be satisfied. Architectural support for such a fundamental function requires the setting up of a repository, where information about system state, QoS commitments, resources and their parameters can be stored and accessed, according to a pre-defined classification. QoS resolution is effectively achieved by table look-up, thus making table access the focal point of activity of QoS management. This tight coupling between QoS management and repository is a feature of many architectural models for QoS management. The Wireless Quality Enhancer (WQE) [12], for example, makes use of a pre-defined database for holding QoS information. The database is queried when a QoS request is made. This results in the identification of a suitable policy that will meet the QoS requirements. On receiving an unclassified QoS query, however, WQE throws an exception and adopts a "best effort" policy (see page 502 of [12]). This approach to QoS specification and resolution, although relatively easy to implement, presents some limitations, not least because of the static nature of the database. It is widely accepted that the inherent characteristics of mobiles systems and their usage are bound to give rise to unexpected requests, and therefore to unclassified types of QoS statement. In addition to its inability to accommodate new QoS parameters, the table/database needs to be maintained and updated manually.

This paper is concerned with the presentation of an architecture for supporting automated QoS resolution. The task for QoS resolution is delegated to component module, called the QoS Evaluation Module (QEM). This architecture combines knowledge base with resolution mechanism as a means of automating the verification of QoS requests. This approach presents many advantages. It is flexible and is likely to identify various combinations of resources as potential candidates for QoS request resolution. It is also able to accommodate new forms of requests, through its resolution mechanism. One major consequence of this approach is the subsequent reduction in human intervention in updating QoS information. As an illustration of the proposed approach, the QoS resolution architecture is combined with the QoS manager of the ITSUMO QoS Architecture [6][7] as a way of implementing a QoS management system.

The remainder of the paper is organized as follows. Section 2 introduces QoS management in Wireless systems and identifies the main requirements for QoS resolution. Section 3 describes the QoS resolution architectural components, and presents an implementation in terms of Timed Automata. Section 4 presents an XML based implementation of a QoS management system through an extension of the ITSUMO architecture [6][7] with the QoS resolution component. Section 5 gives a brief evaluation of the work with pointers for further work, and Section 6 concludes the paper

2. QoS in wireless systems

Quality of Service (QoS) is a general term used to differentiate between the performance aspects of a distributed system and its functional aspects. In general, since the function of the components of a system may be subject to delay or error, the service provided by the overall system is determined by the quality of the provision of such functions. There are various classes of QoS covered in the literature, with attempts at reconciling them. For example, [15] presents an attempt to standardise such views and create a *UML profile*, a dialect of UML, for the specification of QoS. Since the focus of this paper is on Timeliness properties in multimedia systems [4][5] such as *Throughput*, *Latency* and *Jitter*, it is necessary to define these terms. Throughput is the total number of signals per second. For example, if the signal represents dispatch of a media frames, a throughput of *k* frames means there are *k* frames dispatched per second. The latency between two signals is the time between the two signals such as the latency between the generation and final display of a frame. Jitter, also known as *non-anchored jitter*, is the variation of nominal latency suffered by the successive occurrence of the same signal. A formal definition of the Timeliness QoS properties is given in [4][5].

2.1 QoS management

This section introduces a motivational scenario involving a simple wireless system consisting of two laptops. Consider the system in **Figure 1**, in which an application running on Laptop1 dispatches video frames to Laptop2 via Access Point AP1, the Internet and Access Point AP2. An application running on Laptop2 subsequently displays the frames.

The specification of the behaviour of the system can be modelled in different levels of abstraction. For example, the specification of the creation of the frames in the application running on Laptop1, may involve either stating the encoding mechanism and multiplexing operations, or simply modelling the application as a source that produces frames periodically.

To demonstrate the difference between the QoS aspects and behavioural aspects, assume that the system outlined in **Figure 1** is a part of a video conferencing system where the provision of a suitable video display requires the system to present frames arriving in Laptop2 with a jitter of an upper limit of 10 ms.

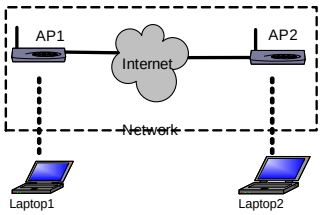

Figure 1: A basic networked system

A jitter above that value may result in low quality video display. Such a constraint is purely performance related and is independent of the behaviour of the system. In general, the effect of QoS statement is to impose restrictions on the behaviour of the system, by requesting a restricted course of actions. Sometimes, such restrictions are beyond the physical capabilities of the system, not least because various components of the system may be in competition for resources and for a better QoS.

Most modern system architectures treat QoS as a first class entity. For example, Quality Global Server (QGS) in the ITSUMO QoS architecture [6][7] and Wireless QoS Enhancer (WQE) in [12] are examples of the such entity. This entity will be referred to as the *QoSManager*[1] of the system. Apart from deciding what level of QoS can be allocated and assigned to each component, a *QoSManager* keeps a record of the availability of resources and their characteristics. A *QoSManager* often interacts with another component, the "Repository", to obtain information to act upon. In some cases, *QoSManager* is designed to take into consideration various policies, which are appropriate for the system (see page 502 of [12]).

2.2 QoS management requirements

The implementation of the repository either by a table or a relational database has the advantage of speed and ease of implementation. Although querying a database can be flexible the main constraint is that results are returned in terms of pre-defined classifications. This approach has also the disadvantage of requiring constant human intervention in the update of QoS information in the repository.

The rapid development of new applications and the creation of new software and hardware platforms with higher performance, means that there is a clear need for methods of updating automatically the repository used by the QoS manager. The limitations of the

[1] The term "*QoSManager*" has been widely used in the ODP, see chapter 17 of [13].

traditional approach to QoS resolution identify two fundamental requirements:

- The introduction of a scheme that allows a more sophisticated approach to QoS resolution.
- A reduction in human intervention in the update operation, by introducing schemes that can accommodate unexpected QoS requests.

A knowledge base approach combined with a resolution mechanism offers more scope for interpretation in the verification of QoS requests and can support an adaptive approach to the update of QoS information.

3 A QoS resolution architecture

This section is concerned with the presentation of a conceptual model of the architecture for automated QoS resolution. It also introduces the necessary formalisms for describing the different components and the resolution process.

3.1 Architectural components

The core components that make up the architecture are presented in Figure 2. Under this scheme the *QoSManager* calls upon the QoS Evaluation Module (QEM) to determine whether a QoS request can be satisfied or not. The resolution process itself requires three components. These are described below.

QoS Evaluation Module (QEM): On the arrival of a new user, which requires a new QoS *q*, the *QoSManager* interacts with a QEM to check if *q* is achievable. In order to achieve this aim three aspects of the system need to be considered:

- a model of the behaviour of the system
- a model that captures the QoS statement, and
- a mechanism that validates the QoS statement against the behaviour of the system
-

Behavioural Model Repository (BMR): The study of the feasibility of the system requires a behavioural model of the system. BMR is a repository containing various templates, representing the behaviour of parts of a model such as communication channels, sources and sinks. Such templates define building blocks, from which the overall behaviour of the system can be composed. QEM uses the templates in BMR to instantiate different parts of a model, and creates a behavioural model for the overall system.

QoS Model Repository (QMR): Similarly, QMR consist of a set of templates that can be used to provide formal models representing a QoS statement. QEM uses the templates in the QMR to instantiate formal representation of QoS aspects of the system.

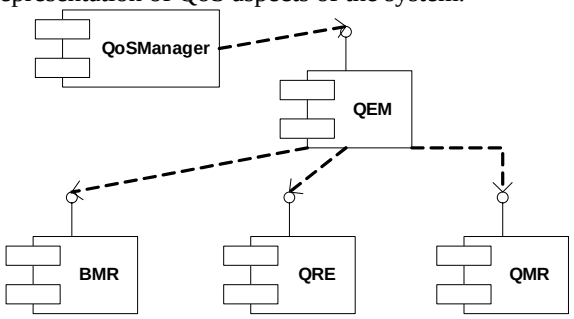

Figure 2: An architecture for QoS resolution

QoS Resolution Engine (QRE): QRE operates on the models generated by BMR and QMR, which represent behaviour and QoS statement, respectively. A QRE is a component that receives a model of the behaviour and a QoS statement and *checks* the validity of the QoS statement against the behaviour of the model. The interaction between the different components of the architecture is shown in Figure 3 in the following sequence diagram [14]. Following the request to check the validity of a QoS statement, QEM instructs BMR and QMR to instantiate the behavioural model *(m:Model)* and the QoS statement *(q:QoS)*, which are transferred to the QRE, represented by *Input2AE*(m) and *Input2AE*(q). The QRE carries the *check* and returns the *result* to QEM.

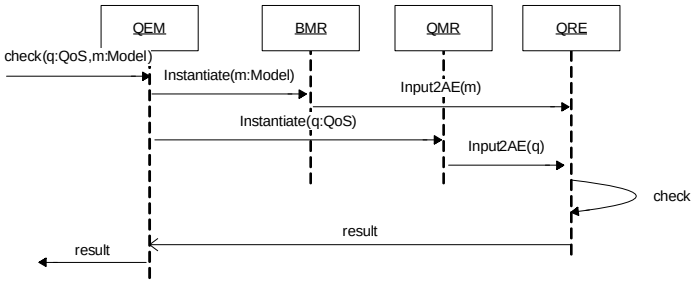

Figure 3 : QoS resolution

3.2 Implementation using Timed Automata

As the main aim of this approach is to automate the process of QoS resolution, a formalism for describing the behaviour of the system is required. There are various formal languages for representing the behaviour of the models such as various timed extensions of Petri nets [11] and CSP [8]. The architecture presented in the paper is independent of such modelling languages.

Behaviour specification: The modelling of the behaviour of the various parts of a system will be in terms of Timed Automata [2]. Figure 4 depicts a

network of Timed Automata modelling the dispatch of video frames from Latop1 dispatching video frames via the *Network* to Laptop2 as follows:

- The application running on Laptop1 dispatches the frames at the rate of *k* frames per sec.
- The latency between the dispatch from Laptop1 and arrival at Laptop2 is at most α.

Simplified behavioural models of Laptop1, Network and Laptop2 are represented by three Timed Automata *TA_Laptop1*, *TA_Network* and *TA_Laptop2*, respectively. The start is from *TA_Laptop1*, which has only one location (*l1*) and one transition marked with the action called "*L1Disptach!*" representing the event of the *Dispatch* of one signal from Laptop1. If the rest of the information depicted in the Timed Automaton is ignored, i.e. "*X<=1/k*", "*X ==1/k*" and "*X := 0*", then this results in a conventional automaton (state machine) which creates the following word,

(L1Disptach!) = L1Disptach! L1Disptach! L1Disptach!….*

which describes a periodic occurrence of the event *L1Disptach!*. It is necessary to model the occurrence of *k* signal in 1 sec, i.e. one occurrence of *L1Disptach!* every *1/k* sec. In order to achieve this, the concept of conventional Automaton is extended by including *Clocks*, which are non-negative real-valued variables. The resulting Timed Automaton *TA_Laptop1* has only one clock *X*. Within each period of *1/k* sec (*X<=1/k*), the control location is stays in *l1*. At exactly "*X ==1/k*" an event *L1Disptach!* occurs and the time is set to zero (*X := 0*) and a new cycle of waiting for a period of *1/k* sec starts. The Timed Automata corresponding to Network and Laptop2 can be explained similarly.

Concurrency and synchronization between Timed Automata are modelled by a CCS [10] style of parallel composition operators. For example, the event *L1Dispatch!* and *L1Dispatch?* are two half actions of the action marking the dispatch of a frame and *must occur at the same time*. Similarly, *L2Arrival!* and *L2Arrival?* are two halves of one action. Using half action it is possible to compose a number of Timed Automata to create a *network of Timed Automata* [9].

For example, the network of Timed Automata of Figure 4 is said to be the *parallel Composition* of *TA_Laptop1*, *TA_Network* and *TA_Laptop2*. This is denoted by

TA_Laptop1 || *TA_Network* || *TA_Laptop2*.

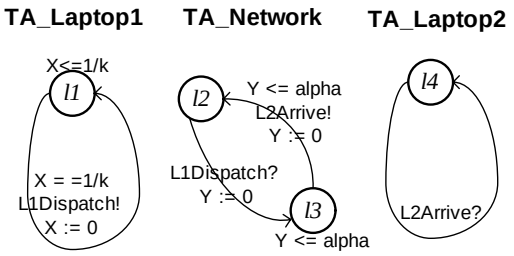

Figure 4: A network of Timed Automata

The network of Timed Automata in Figure 4 gives a high level view of the behaviour of the system presented in .

QoS specification: The specification of QoS aspects of system has received considerable attention For example, [4] presents a method of verification of Timeliness QoS properties such as jitter, throughput and latency, which draws on the idea of ***Test Automata*** [1][9]. Assume that *A* is a network of Timed Automata representing the behaviour of the system. Suppose that σ is a Timeliness properties such as jitter, throughput and latency. Corresponding to σ, there is a network of Timed Automata (or a single Timed Automaton) T_σ, which has a location called **failure** such that, the system modelled via *A* always satisfies σ if and only if the parallel composition $A \parallel T_\sigma$ never reaches a state with the location labelled as **failure**.

Consider the example in the previous section. Suppose the application that is sending the data dispatches video frames in the rate of 25 per sec i.e. *k=25*. Moreover, assume that the network is such that it takes α = 7 msec for the network to send each frames from Laptop1 to Laptop2. Jitter is one of the main factors for a good quality view of a video. Suppose that we demand a jitter of 10 msec for the signal arriving at Laptop2, i.e. we expect the time gap between the two consecutive arrival of the frames to be less than or equal to 10 msec (= 1/100 sec). The question is whether the system is able to provide such a QoS.

Figure 5 shows an instantiation of the QTA for jitter. There is a clock *Z* that measures the time gap between two occurrence of *L2Arrive?*. If the time gap between two occurrence is more than β = 10 msec the control location changes to **failure**. This signals a violation of the jitter condition.

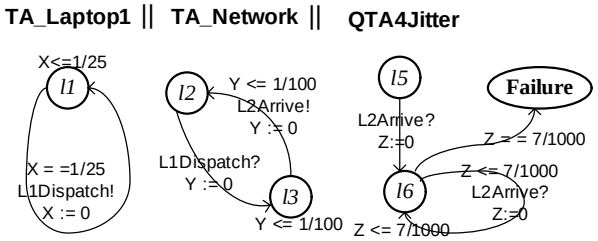

Figure 5: A QTA for jitter verification

UPPAAL and QoS: The study the behaviour of the system of Figure 5, can be performed with the help of CASE tool such as the model checker UPPAAL [9] Using UPPAAL it can be verified that the system reaches a state which includes **failure**. This means that with the current specification, it is possible for the system to produce a set of frames with the jitter above 10. However, if another protocol can be used to speed up the delivery of the signal such that the value of α reduces to below 5 msec, it can be seen that the system will not violate the jitter condition of 10. To do this we must replace the value of 1/100 with 1/200 in the Timed Automata TA-Network of Figure 5. Then, using UPPAAL, it can be verified that **failure** is not reachable.

4 A QoS management framework

The introduction of a new approach to QoS management was motivated by the desire to automate QoS resolution and widen its scope. Combined with a QoS manager, the QoS resolution architecture can be used to implement an enhanced QoS management system. This approach can be applied to the ITSUMO architecture by extending it.

4.1 Extension of the ITSUMO architecture

The extension of the ITSUMO architecture involves the incorporation a version of QEM. As mentioned earlier, in the ITSUMO architecture the role of *QoSManager* is performed by QGS. Within the proposed architecture UPPAAL performs the role of QoS Resolution Engine. In UPPAAL, models of the system are stored as XML documents. Each UPPAAL XML Model (UXM) consists of the following four main components; Declaration, Templates, Instantiation and System. Declaration is a part of the (UXM), which includes various variables, constant etc. Each UXM consists of one or more Templates. An instantiated Timed Automaton is created from a Template Timed Automaton by assigning values to the parameters. The Instantiation part of a UMX includes instantiation of one or more Timed Automata from the templates in the Templates section of the UMX. Finally, the System part of the UMX creates a parallel composition of a number of instantiated Timed Automata to create a network of Timed Automata.

4.2 An XML based implementation

The initial task for this implementation creates a UMX with a `Templates` and `Declaration` part, which includes *templates* for all possible types of components in the system such as

- source and sink, similar to Laptop1 and Laptop2 of
- models of various types of buffer, decoder etc. [4]
- models of communication protocol [3]
- QoS Timed Automata for various types of QoS such as Jitter, Throughput etc [4].

The above items are parametric representations and by instantiation (assigning value to parameters) it is possible to generate potentially an infinite number of combinations of behavioural models. Let us refer to this document as *Main.xml.*

Figure 6 details the process of checking if a QoS statement is achievable. The *QoSManager* (QGS in ITSUMO) wants to check a QoS condition specified as an xml file *q.xml.* Then QGS forwards *q.xml* and a model of the system *m.xml*. to QEM. The model specifies the current state of the system in terms of its components. The rest of the process is supervised by the QEM.

First, the QEM has to use *m.xml* to instantiate a model of the system. To do so, the BMR modifies the Instantiation and System part of the *Main.xml*. This is denoted by *Write_instatiation&system*(*m*). Similarly, the QEM uses the QMR to modify the Instantiation and System part of the *Main.xml*, by first instantiating suitable QoS Timed Automata (QTA) corresponding to the QoS statement specified in *q.xml* and then including them into the System part of the *Main.xml.* After modifying the *Main.xml*, it can be uploaded to UPPAAL and analyzed to check if the state **failure** of the QTA is reachable. The *result* is returned to the QEM. In case of **Failure**, the *QoSManager* is notified that the QoS request is not achievable.

5. Evaluation and future work

This component-based design to QoS management offers more flexibility than a monolithic scheme [6][7]. Components can be implemented using different formalisms and interchanged while still providing the core functionality of QoS resolution and QoS allocation. This clear separation of concerns provides a greater scope for a rapid configuration of the system, a feature that contrasts with the use of a centralised, static and passive repository. As the range of

applications of mobile systems and their capabilities increases, an adaptive approach to QoS resolution becomes imperative**.** The choice of an implementation that combines knowledge base and resolution mechanism represents a significant step towards addressing these issues.

The implementation of the architecture is independent of the formalism used for expressing behaviour and QoS. QoS statements in this implementation are expressed in terms of Timed Automata. More specifically, the use of UPPAAL is computationally demanding. For example, verification of a system consisting of 7 components takes around 600 sec CPU time [4]. This is due to the state-explosion problem. As a result, there is a clear scope for investigating other formalisms and techniques, such as temporal logic, in order to ensure better performance.

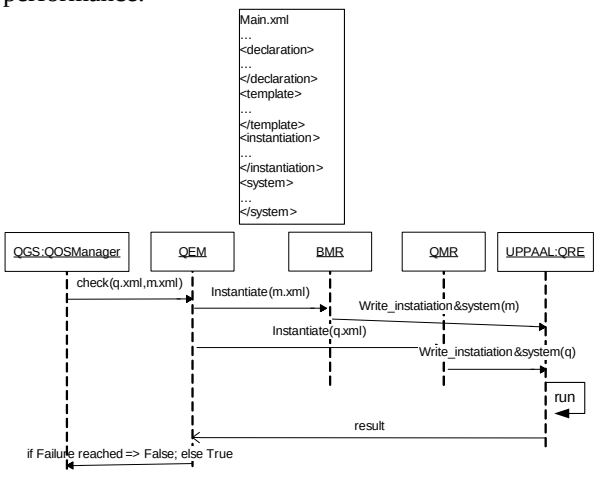

Figure 6: Extension of the ITSUMO architecture

The focus of the research at this stage is on a full implementation in order to evaluate the architecture.

6. Conclusion

In this paper a generic architecture for enhanced QoS provision was introduced as a means of addressing the limitations of the prevailing model for QoS management. The proposed architecture incorporates knowledge base and resolution mechanism, for flexible and adaptive QoS resolution, in contrast to the limited scope of table look-up. Closely linked to such an approach, is the introduction of a framework that promotes the creation of enhanced QoS management systems. This is achieved through the extension of an existing QoS manager by a QoS resolution architecture. Although a specification based on Timed Automata was put forward as proof of concept, there is, however, a need for the investigation and evaluation of other formalisms and techniques in the search for efficient implementations.

7. References

[1] L. Ageto, P. Bouyer, A. Burgueo and K. G. Larsen The Power of Reachability Testing for Timed Automata, in Theoretical Computer Science 300(1-3)411-475, 2003.

[2] R. Alur, D Dill A Theory of Timed Automata Journal of Theoretical Computer Science. 126(2): 183-235 (1994)

[3] J Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi Automated Analysis of an Audio Control Protocol Using Uppaal, J In Journal of Logic and Algebraic Programming, volumes 52-53, pages 163-181, Holger Hermanns and Joost-Pieter Katoen (eds.). July-August, 2002

[4] B. Bordbar and K. Okano, Verification of Timeliness QoS Properties in Multimedia Systems, Proceeding of 5th International Conference on Formal Engineering Methods, Lecture notes in Computer Science, pp 523-540, 2003

[5] H. Bowman, G. Faconti, and M. Massink Specification and verification of media constraints using UPPAAL, In Proceedings of Design, Specification and Verification of Interactive Systems '98, Markopoulos and P. Johnos, editors, pp. 261--277 Springer, 1998.

[6] J.C. Chen, A. McAuley, A. Caro, S. Baba, Y. Ohba, P. Ramanathan. A QoS Architecture for Future Wireless IP Networks. In Twelfth IASTED International Conference onParallel and Distributed Computing and Systems (PDCS 2000), Las Vegas, NV, November 2000

[7] J. C. Chen, A. McAuley, V. Sarangan, S. Baba, and Y. Ohba, Dynamic Service Negotiation Protocol (DSNP) and Wireless DiffServ, ICC'02, New York city, April 2002

[8] C. A. R. Hoare Communicating Sequential Processes Prentice Hall, 1985

[9] K. G. Larsen, Paul Pettersson and W. Yi UPPAAL in a Nutshell Springer International Journal of Software Tools for Technology, 1(1+2), 1997

[10] R. Milner Communication and Concurrency, Prentice Hall, 1989

[11] J. L. Peterson, Petri net Theory and Modelling of systems, Prentice Hall, 1981

[12] G. Pau, D. Maniezzo, S. Das, Y. Lim, J. Pyon, H. Yu, M. Gerla, A Cross-Layer Framework for Wireless LAN QoS Support", IEEE International Conference on Information Technology Research and Education, ITRE 2003, Newark, New Jersey, USA, August 10-13, 2003.

[13] J. R. Putman Architecting with R-ODP Prentice Hall, 2001

[14] Unified Modelling Language: specification, available at www.omg.com

[15] UML profile for QoS and Fault Tolerance, available at [14] UPPAAL www.uppaal.com