Support Components for Quality of Service in Distributed Environments: Monitoring Service

D. A. Reed and K. J. Turner
Dept. Comp. Sci. And Math., Stirling University, Stirling, UK.
E-mail:dar@cs.stir.ac.uk, kjt@cs.stir.ac.uk

Abstract
One of the services required in a Quality of Service (QoS) oriented open distributed environment is monitoring. Instead of a bespoke monitoring system for each distributed application it is proposed that the optimum solution is a generic distributed service which is adapted to suit the application. The following approach can encompasses many of the QoS characteristics proposed in the draft ISO QoS Framework Standard.

Keywords
Quality of Service, monitoring, distributed environments

1 INTRODUCTION

Until recently Quality of Service (QoS) was not a vital issue in the design of open 'middleware' systems but, since the introduction of multimedia services with time-critical characteristics and the need for synchronisation between data streams, this is changing. The Open Software Foundation (OSF) is investigating a number of QoS initiatives. Architecture Projects Management Ltd. (APM) is defining the Distributed Interactive Multimedia Architecture (DIMMA) (Otway, 1995), an extension of the real-time Advanced Network Systems Architecture (ANSA).

This paper outlines a generic distributed QoS monitoring service which is tailored according to the distributed application. The methodology progresses naturally from formal temporal specification techniques, thereby aiding development and design of the distributed application. Also included is a brief description of QoS support services that a designer/programmer might expect of a QoS-oriented distributed environment.

2 SERVICE COMPONENTS

Five services are proposed for supporting QoS within QoS-oriented distributed environments: co-ordinating, monitoring, negotiating, networking and scheduling. The co-ordinating service provides the pivotal role, interfacing with the other services. The monitoring service deals with the operational concerns of a distributed application, warning of potential failures. The negotiating service deals with establishment concerns, particularly the QoS requirements between components of the distributed application, and between the same components and the distributed environment. The networking service deals specifically with the establishment and management of communications resources. The scheduling service deals specifically with the establishment and management of local resources. Both the networking and scheduling service act as access control mechanisms.

3 MONITORING SERVICE

Monitoring is an essential service in a QoS-oriented distributed environment. There are three basic requirements of a monitoring service. The monitor must report QoS violations promptly, it should use few resources, and it should minimise interference when monitoring the application. As with many systems, the act of measuring within a system can perturb it. It is necessary to reduce this perturbation to a minimum.

The proposed service is based on a 'witness' monitor. Each node in a distributed environment has a local monitor which collects information from components of distributed applications located at that node. Each distributed application has an associated global monitor which collates data collect from the local monitors associated with that application. A local monitor may collect information from many applications but a global monitor collates data for only one distributed application.

For the monitoring service to be generic there cannot be application-specific information conveyed to it from the distributed application. Events must therefore be generic in format. The conditions encapsulating the statement of allowed/disallowed behaviour are presented to the monitoring service in a general notation which may be applied to these generic events. Establishment of these conditions consists of the application first presenting a number of formal identifiers representing the events and a set of conditions expressed using these formal parameters to the monitoring service. This then issues currently available identifiers to the application. Each event is assigned to one identifier, although this event may occur more than once. The identifiers are referred to as *colours*.

The concept of pair-wise interaction is the underlying model of this system. In a pair-wise interaction there is an action event and later in time a reaction event which is causally linked. This paper considers the basic case of 1:1 action/reaction pairs but the n:1 case can be treated as n 1:1 cases.

When an event occurs the application component must inform the local monitor immediately: *signal ≡ (colour, sequence, signature)*. The *colour* identifies the event, *sequence* identifies the occurrence of the event and the *signature* represents data at the event. The signature must be generated by the application to accurately reflect the data; the signature is used to check data integrity. The local monitor time-stamps the *signal* to create an *event tag ≡(signal, time-stamp)*. If a particular condition cannot be evaluated at the local monitor then *event tags* associated with that condition are collected into *tag lists* and periodically sent to the global monitor: *tag list* $\equiv$*(colour, (*$signature_0$*,* $time\text{-}stamp_0$*), (*$signature_1$*,* $time\text{-}stamp_1$*), ..., (*$signature_n$*,* $time\text{-}stamp_n$*))*.

The event and *signal* are considered atomic. Whether this requires critical sections of code will depend on the accuracy required of the time-stamp, frequency of the event and system overhead of critical sections. The local monitor does not acknowledge *signals*; the call is one way to reduce the impact on the components. *Tag lists* are only sent to the global monitor if necessary and the periodicity is determined by the required response time. Whether a condition can be evaluated locally or globally depends on the distribution of the application's components. Co-location of some components could be beneficial.

Conditions are composed of a number of functions and operators which may be applied to the event tags and to condition expressions. There are two basic functions which can be applied directly to an event tag: *t(colour, sequence)* returns the time-stamp and *s(colour, sequence)* returns the signature of the event tag with the matching colour and sequence number. The function *c(colour)* returns the number of times the event with the matching colour has occurred. Functions for comparing signatures and functions which count the number of times a condition has been true or false are also included. The basic logical operators $\neg$ (`not`), $\wedge$ (`and`), $\vee$ (`or`) are allowed to create complex conditions although it is not yet clear whether the language would benefit from $\forall$ and $\exists$ operators.

This is the basis for a language which can be used to express QoS characteristics. It is not complete. In particular operators to flush the event counter and disable or enable conditions dependent on events or other conditions are of substantial interest. There is also the possibility of defining stopwatches within the local monitors to improve the range and response of condition statements.

It is not anticipated that a distributed application would state its QoS conditions directly in these basic functions. Common QoS characteristics would be expressed using this language consisting of a set of macros which the application could instantiate. This set of macros would support an international standard such as the ISO QoS framework document (ISO, 1995). The next section illustrates how such macros can be constructed.

4 ISO QOS

The ISO QoS framework document outlines 54 QoS characteristics grouped into eight categories. Not all of these characteristics are suitable for this 'tagging' methodology. This section gives as examples the time delay characteristic and the non-specialised characteristics of capacity, accuracy and availability.

The simplest characteristic is the 'time delay' characteristic. This is a building block of many other QoS characteristics. The example is interpreted as the time of the reaction event (`colour2`, nth occurrence) minus the time of the action event (`colour1`, nth occurrence) - possibly a latency.

$$\text{time delay} \equiv t(\texttt{colour2},n) - t(\texttt{colour1},n)$$

Many of the capacity characteristics can be measured by counting the number of action or reaction events. The example below is interpreted as the action event (`colour`) where *m* is a constant representing the value of one unit of capacity, for example a packet size. This condition therefore calculates the mean throughput.

$$\text{capacity} \equiv \frac{c(\texttt{colour}) \times m}{t(\texttt{colour},n) - t(\texttt{colour},0)}$$

The accuracy characteristics require the application to distinguish between correct and incorrect reaction events. The following example is interpreted as the number of correct reaction events (`colour2`) over the number of action events (`colour1`).

$$\text{accuracy} \equiv \frac{c(\texttt{colour2})}{c(\texttt{colour1})}$$

The availability characteristics require the application to indicate when a resource is available or unavailable. The example below reflects the definition of availability as Mean Time Between Failures (MTBF) divided by MTBF plus Mean Time To Repair (MTTR). The action event (`colour1`) is interpreted as 'becomes available' while the reaction event (`colour2`) is interpreted as 'becomes unavailable'. There are naturally other expressions which reflect availability.

$$\text{availability} \equiv \frac{\sum_{n=0} t(\texttt{colour2},n) - t(\texttt{colour1},n)}{\sum_{n=0} t(\texttt{colour1},n+1) - t(\texttt{colour2},n) + \sum_{n=0} t(\texttt{colour2},n) - t(\texttt{colour1},n)}$$

5 FUTURE WORK

At present this monitoring methodology is tailored towards systems in which the monitored data is identical at the action/reaction events. The data is then used, in part, to establish the causality between the action/reaction events. This is particularly suited to monitoring network communications such as found in protocol stacks. Current work is implementing a version of this monitoring service between Stirling, Scotland and BT Labs, Suffolk UK. Principal problems include accessing a high resolution clock on the end-systems and establishing a global clock reference. These problems are non-trivial for a middleware solution. The lack of a high-resolution clock can be obviated by recasting the QoS conditions into statistical forms thereby reducing the need for such a clock, but only at the expense of the response time to QoS violation. Establishment of a global clock reference to a sufficient accuracy over an unreliable network is also cause for concern.

Future work will extend this methodology to systems in which the data is transformed between events and to causally relate events between which there is no data transfer. This will naturally require the designer to embed a method of establishing the action/reaction pairs. It is anticipated that tools for automating this procedure will be developed, thus minimising the impact of this methodology on the designer and programmer. This work is also being evaluated with respect to the work at the University of Michigan, USA particularly ARMADA and the real-time fault tolerant monitoring systems (Jahanian, 1994).

6 REFERENCES

ISO (1995), *Information Technology - Quality of Service - Framework - Final CD.* Open Systems Interconnection, data management and Open Distributed Processing; ISO/IEC JTC1/SC 21.

Jahanian, F., Rajkumar, R. and Raju S. (1994), *Runtime Monitoring of Timing Constraints in Distributed Real-Time Systems*; CSE-TR-212-94, University of Michigan, USA.

Otway, D. (1995), *DIMMA overview*; APM.1439.02, APM Ltd., Cambridge, UK.

7 BIOGRAPHY

Daren Reed is a PhD student in his final year investigating specification and analysis of QoS in distributed systems. This wide ranging work includes formal specification methods, operating systems, networking and distributed system environments. His research is supported by the Distributed Systems Group, BT Labs, UK. He holds an MA in Chemistry, Oxford University, and an MSc in Software Engineering (distinction), Stirling University.

Prof. Ken Turner's research interests include system and communication architectures, formal methods, hardware/software codesign, and QoS with respect to distributed multimedia. During his career he has worked for ICL and has participated in international standardisation committees. He holds a BSc in Electrical Engineering, Glasgow University, and a PhD in Machine Intelligence, Edinburgh University.