IPP Hurray!

www.hurray.isep.ipp.pt

Technical Report

An Ada Framework for QoS-Aware Applications

Luis Miguel Pinho

Luis Nogueira

Ricardo Barbosa

TR-050601

Version: 1.0

Date: June 2005

An Ada Framework for QoS-Aware Applications

Luis Miguel PINHO, Luis NOGUEIRA, Ricardo BARBOSA

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: {lpinho, luís, rbarbosa}@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

In this paper we present a framework for managing QoS-aware applications in a dynamic, ad-hoc, distributed environment. This framework considers an available set of wireless/mobile and fixed nodes, which may temporally form groups in order to process a set of related services, and wherethere is the need to support different levels of service and different combinations of quality requirements. This framework is being developed both for testing and validating an approach, based on multidimensional QoS properties, which provides service negotiation and proposal evaluation algorithms, and for assessing the suitability of the Ada language to be used in the context of dynamic, QoS-aware systems.

An Ada Framework for QoS-Aware Applications

Luís Miguel Pinho, Luis Nogueira, Ricardo Barbosa

Department of Computer Engineering, ISEP, Polytechnic Institute of Porto,
Rua Dr. António Bernardino Almeida, 431, 4200-072 Porto, Portugal
{lpinho,luis,rbarbosa}@dei.isep.ipp.pt

Abstract. In this paper we present a framework for managing QoS-aware applications in a dynamic, ad-hoc, distributed environment. This framework considers an available set of wireless/mobile and fixed nodes, which may temporally form groups in order to process a set of related services, and where there is the need to support different levels of service and different combinations of quality requirements. This framework is being developed both for testing and validating an approach, based on multidimensional QoS properties, which provides service negotiation and proposal evaluation algorithms, and for assessing the suitability of the Ada language to be used in the context of dynamic, QoS-aware systems.

1 Introduction

Quality of Service (QoS) is considered an important user demand, receiving wide attention in real-time multimedia research [1][2]. However, in most systems, users do not have any real influence over the QoS they can obtain, since service characteristics are fixed when the systems are initiated. Furthermore, multimedia applications (and their users) can differ enormously in their service requirements as well as in the resources which need to be available to them [3]. These applications present increasingly complex demands on quality of service, reflected in multiple attributes over multiple quality dimensions.

At the same time, the use of laptop computers coupled with wireless network interfaces is growing rapidly. Recent technological development lead to the fusion of wireless ad-hoc networks, peer-to-peer computing and multimedia content. As devices move within the range of each others a local ad-hoc network forms spontaneously, creating a new, highly dynamic and decentralized environment for multimedia applications.

Such an environment is expected to be heterogeneous, consisting of nodes with several resource capabilities. For some of those there may be a constraint on the type and size of applications they can execute with user's acceptable quality of service. For example, video conferencing systems often use compression schemes that are effective, but computationally intensive, trading CPU time for limited network bandwidth. A mobile client with limited CPU and memory capacity, but sufficient link speed, with nearby more powerful (or less congested) devices, can divide the computational intensive processing into tasks and spread it among different neighbours.

It is obvious that these requirements for more flexible QoS-aware applications impact on the available support from the underlying environment (language, middleware, operating system). In what languages are concerned, Ada has been for a long time considered suitable for the development of traditional, static, real-time applications, but is often considered to be limited concerning the support to more dynamic real-time applications. It is our belief that this latter idea is not true, and, moreover, that Ada is an enabling technology for supporting QoS-aware type of real-time applications

Therefore, in this paper we present a framework which is currently being built for testing and validating a QoS applications support approach which is currently being specified [4]. This framework is being implemented in Ada, using the currently available mechanisms, which will allow providing sufficient insight on the suitability of the language. The rest of this paper is structured as follows. The next section provides a brief description of the considered model for the system, and the used approach for QoS requirements representation and service requests. Section 3 presents an overview of the Ada framework, considering its structure and main functionalities, while sections 4 and 5 present, respectively, how negotiation and acceptance of services is performed, and how resource managers are implemented. Finally, section 6 presents some conclusions.

2 System Model

In this work, we consider a system where wireless/mobile nodes may dynamically enter the range of each other, and of wired infrastructures (even clusters of nodes [5]), opportunistically taking advantage of the local ad-hoc network that is spontaneously created, forming a temporary coalition for service execution (Figure 1). Coalition formation is necessary when a single node cannot execute a specific service, but it may also be beneficial when groups perform more efficiently when compared to a single node performance.

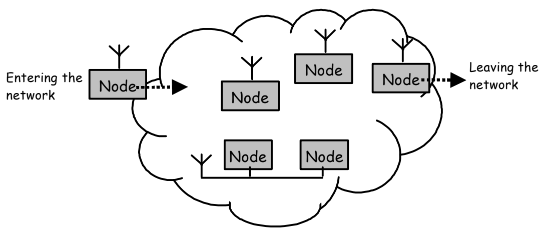

Fig. 1. System Overview

Ad-hoc networks, i.e., networks without any fixed network infrastructure (such as base stations, etc.) are gaining much interest in research as well as in industry. With ad-hoc network mechanisms, clients that are in sufficiently close proximity are able to communicate directly without the need of further, externally provided, infrastructure. At first glance, an individual mobile device may not have sufficient capacity and computation power for an effective integration in a distributed multimedia processing environment. However, if we exploit the aggregated mobile power instead of single, individual power and consider the exponential rise of mobile devices and the continuous developments in wireless technology, then one may conclude that this collaborative processing can be a valid solution.

This provides a generic model that enables a distributed service allocation, i.e., without a central authority distributing the services among nodes. Given a set of services, a distributed environment must seek the maximization of the associated QoS constraints. The nodes shall reach efficient service allocation by themselves, seeking a maximal outcome. This will be achieved via the formation of a temporary group of individual nodes (coalitions), which, due to its higher flexibility and agility, is capable of effectively respond to new, challenging, requirements.

It is clear that such a group presents very significant challenges, especially at the architectural level. Major developments are required in the fields of communications protocols, data processing and application support. Our goal is to develop the architecture which enables the creation of a new generation of mobile nodes that can effectively network together, providing a flexible platform for the support of distinct network applications. In this model, QoS-aware applications must explicitly request the service execution form the underlying QoS framework, thus providing explicit admission for controlling the system, abstracting from existent underlying distributed middleware and from the operating system. The model itself abstracts from the communication and execution environments.

2.1 QoS Requirements representation

In [4], QoS requirements are described through a scheme that defines dimensions, attributes and values, as well as relations that maps dimensions to attributes and attributes to values:

$$QoS \circledR \{Dim, Attr, Val, DA_r, AV_r, Deps\}$$

where *Dim* is the set of QoS dimensions (e.g. Video, Audio), *Attr* is the set of attributes identifiers and *Val* is the set of attribute's values identifiers. DA_r is the relationship that assigns to each dimension in *Dim* a set of attributes in *Attr*, AV_r is the relationship that assigns to each attribute in *Attr* a specific value in *Val* and *Deps* is a set of relationships defining the dependencies between attributes' values. Values are represented by a type (integer, float, enumeration) and domain (discrete, continuous).

As an example of this requirement description, a video streaming application may define a set of dimensions (and their attributes) that might be associated with a particular application (the following list is not intended to be exhaustive):

```
Dim  = {Video Quality, Audio Quality}
Attr = {color depth, frame rate, sampling rate, sample bits}
Val  = {{1,integer,discrete},{3,integer,discrete},...,
       {[1,...,30],integer,continuous},...}
```

It is clearly infeasible to make the user specify the utility of every quality choice, for all the QoS dimensions of a particular application. There are simply too many choices. Instead, a preference order is imposed over the dimensions, its attributes and their values on user's service request [4]. While a semantically rich request is provided, so that the system tries to achieve a service the more closely related to user's preferences, a user is actually able to express his preferences in his request.

Suppose that, in a remote surveillance system, video is much more important to the user than audio. Assuming that for a particular user a grey scale, low frame rate is fine for video, his request could be as follows:

```
1. Video Quality
    (a)frame rate    : [10,...,5], [4,...1]
    (b)color depth   : 3, 1
2. Audio Quality
    (a)sampling rate : 8
    (b)sample bits   : 8
```

The relative decreasing order of importance imposed in dimensions, attributes and values expresses user's preferences, that is, elements identified by lower indexes are more important than elements identified by higher indexes. In the example above, video is more important than audio, and frame rate is more important than color depth in the Video Quality dimension. In a similar way, the audio sampling rate is more important than the sampling size. For each of these attributes, a preference order for the QoS values is as well expressed.

3 The Ada QoS-Aware Framework

Figure 2 presents the structure of the proposed framework. Central to the behaviour of the framework, is the *QoS Provider*, which is the responsible for all the process of both distributed and local resource requests.

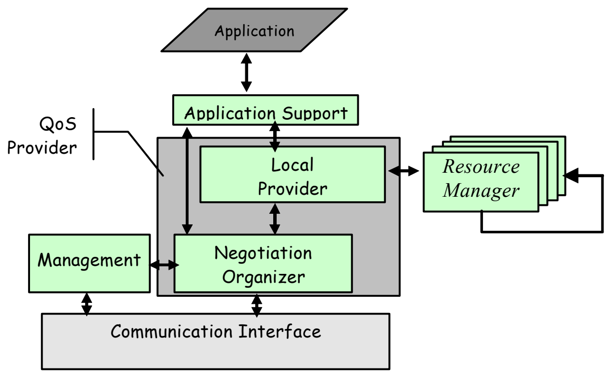

Fig. 2. Framework Structure

The *QoS Provider*, rather than reserving resources directly, contacts the *Resource Managers* to grant specific resource amounts to the requesting task. It is the *QoS Provider* which receives the user's preferences for all the QoS dimensions of a particular application, which are then distributed among the *Resource Managers*.

Within the *QoS Provider*, the *Negotiation Organizer* is the responsible for the collaboration of all of the current nodes in the system, by implementing the negotiation and proposal evaluation algorithms of [4], atomically distributing service requests, receiving the individual nodes' proposals for each service and deciding which node(s) will provide the service. Note that for now we consider the existence of an atomic broadcast mechanism [6] in the system, thus by guaranteeing that all nodes will receive the same service requests and proposals in the same order, we guarantee that the decision will be the same in all nodes of the system.

The *Local Provider* is responsible for replying to negotiation requests, by making a proposal using a heuristic algorithm [4] inspired in the local QoS optimization heuristic of [7]. This module is also responsible for maintaining the state of the resource allocations and services provided in each node.

The *System Manager* module will be responsible for maintaining the overall system configuration, due to the dynamics of nodes entering and leaving the system, and for detecting coalition operation and dissolution. The *Resource Managers* are the modules that manage a particular resource. These modules interface with the actual implementation in a particular system of the resource controllers, such as the device driver for the network, the scheduler for the CPU, or by software that manages other resources (such as memory). It is obvious that, although we consider a collaborative environment, proper resource usage must be monitored in run time [8], in order for system resource managers be able to decide based on the actual resource usage of the system, not only on the resource assumptions of executing services.

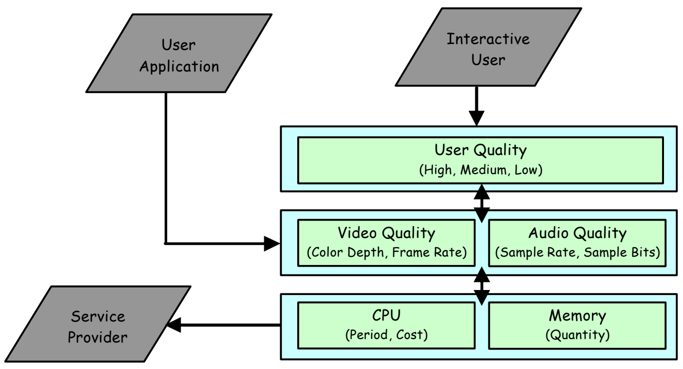

Fig. 3. Resource Managers Layering

One important issue is the ability of resource managers to use each other, in order to allow systems to be built supporting QoS requirements either from the point of view of the user (*e.g.* high quality), of applications (*e.g.* frame rate) or of the system (e.g.

period and cost). Nevertheless, special care must be taken that a service request is not performed with accumulative resource requests on these different levels.

As an example, a particular system may provide the resource manager layering of Figure 3. With this layering, an interactive user application could be more friendly and easier to use by providing only high-level user perceptive quality, whilst other user applications could be programmed to use application-related QoS constraints. Finally, service providers would collect the service requirements at the system level.

4 The QoS Provider

Figure 4 presents the structure of the *QoS Provider* module. New service requests are made by the communication interface (applications call this interface in order to guarantee the order of service requests). Applications may request information concerning the actual QoS values of currently executing (in this node) services. The module also receives/sends proposals from/to other nodes concerning a service being negotiated.

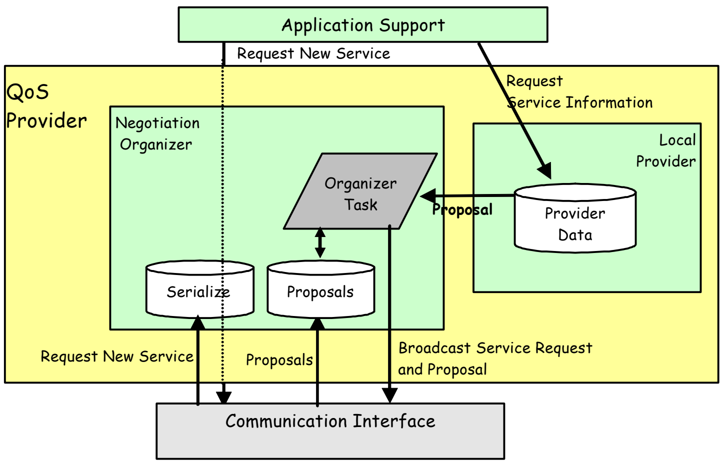

Fig. 4. *QoS Provider* structure

In order to guarantee that new service requests are serialized in the provider, a protected object is used (Figure 5). New service requests queue on the *New_Service* entry, in order to request for the service. Then, requests are re-queued on *Wait_New_Service*, waiting for the end of the negotiation process. Note that while this process takes place, the barrier in *New_Service* is closed.

Concerning the proposals (Figure 6), there is no need to serialize access to deliver a proposal, therefore only mutual exclusion is provided, but an entry is provided for the *Organizer Task* to wait for the arrival of the proposals of the other nodes.

```
protected Serialization is
   entry New_Service(Preferences     : in QoS_Values_Type;
                     Accepted_Values : out QoS_Values_Type;
                     Accepted        : out Boolean;
                     Id              : out Service_Id);

   entry Get_Service(Preferences : out QoS_Values_Type);

   procedure Service_Answer(Accepted : in Boolean;
                            Proposal : in QoS_Values_Type;
                            Id       : in Service_Id);
private
   entry Wait_New_Service(Preferences : in QoS_Values_Type;
                          Accptd_Val  : out QoS_Values_Type;
                          Accepted    : out Boolean;
                          Id          : out Service_Id);

   Pref, Acc        : QoS_Values_Type;
   Accepted_Service : Boolean;
   Serv_Id          : Service_Id;

   New_Service_Request  : Boolean := false;
   New_Service_Response : Boolean := false;
   Organizer_Available  : Boolean := true;
end Serialization;
```

Fig. 5. *Serialization* protected object

```
protected Proposals is
   procedure Set_Proposal(Node    : in SM.Nodes_Type;
                          Proposal: in QoS_Values_Type);

   procedure Clean;
   entry Wait_Decide(Accepted_Values : out QoS_Values_Type;
                     Accepted        : out Boolean;
                     Node            : out Nodes_Type);
private
   Proposals : Nodes_Proposals;
   Proposed  : Nodes_Proposed := (others => false);
   Complete  : Boolean        := false;
end Proposals;
```

Fig. 6. *Proposals* protected object

The *Organizer Task* (Figure 7) is normally blocked in the *Get_Service* entry of the *Serialization* object. Upon a service request is made, the task atomically broadcasts it to the system, and requests a proposal to the *Local Provider*. Note that the broadcast of the proposal thus not need to be atomic, since there is no necessity of order between the proposals. The task then blocks waiting for a decision of the *Proposal* object. It then informs the *Local Provider* of the state of the request. Note that since the decision is the same in all nodes then it is not necessary to broadcast it. The *Local Provider* (Figure 8) is a simple mutual exclusion object, protecting the manipulation of resource managers and services information. Note that Resource Managers must register with the provider upon initiation, in order to be asked for proposals concerning their particular dimension.

```
task body QoS_Organizer_Task is
  -- declarations
begin
   loop
      Serialization.Get_Service(Preferences);
      Comm.Atomic_Broadcast(Preferences);
      Local.Service_Request( Preferences,
                             My_Node_Proposal,
                             Id);
      Comm.Broadcast(My_Node_Proposal);
      Proposals.Set_Proposal(This_Node,
                             My_Node_Proposal);
      Proposals.Wait_Decide( Decided_Values,
                             Accepted,
                             Node);
      if Node = This_Node then
         Local.Accepted_Service(Id);
      else
         Local.Rejected_Service(Id);
      end if;
      Serialization.Service_Answer( Accepted,
                                    Decided_Values,
                                    Id);
   end loop;
end Qos_Organizer_Task;
```

Fig. 7. *Organizer* task

```
protected Provider_Data is
    procedure Register(Resource  : in Manager_Access;
                       Dimension : in QoS_Dimensions_Type;
                       Old       : out Manager_Access);
    procedure Service_Request(Pref : in QoS_Values_Type;
                              Prop : out QoS_Values_Type;
                              Id   : out Service_Id);
    procedure Rejected_Service(Id : in Service_Id);
    procedure Accepted_Service(Id : in Service_Id);
    procedure Terminated_Service(Id : in Service_Id);

    function Get_Service_Parameters(Id : SM.Service_Id)
         return QoS_Values_Type;
    function Get_Resource_Manager(D : in QoS_Dimensions_Type)
         return Manager_Access;

private
    Resources : Resources_Set := (others => null);
    Used_Service_Id : Service_State_Array := (others => Free);
    Services :  Service_Array;
end Provider_Data;
```

Fig. 8. *Local Provider* protected object

5 Resource Managers and Attributes

Resource attributes are supported by providing a tagged abstract type (Figure 9), which can be extended by resource managers, to define particular resource attributes. Attributes must implement the abstract function for difference, in order to support the evaluation of proposals of [4]. An attribute list is also provided for service requests to be able to specify a set (in decreasing importance order) of attribute values.

```
type Attribute_Value is abstract tagged null record;

type Attribute_Value_Access is access all
   Attribute_Value'Class;

function Difference(Preference, Proposal : Attribute_Value)
   return Float is abstract;

package Attr_List is new
   List(Attribute_Value, Attribute_Value_Access);
```

Fig. 9. *Attribute_Value* tagged type

```
type Manager_Type(Dim : QoS_Dimensions_Type) is abstract tagged
record
      Dimension: QoS_Dimensions_Type := Dim;
end record;

type Manager_Access is access all Manager_Type'Class;

-- Resource Managers must call register
procedure Register_Manager(Resource : Manager_Access;
                           Old      : out Manager_Access);

-- Services to implement
-- Depend of the particular Resource Manager
procedure Evaluate_Service(Resource    : in Manager_Type;
                           Preferences : in Attr_List.List;
                           Proposal    : out Attr_List.List;
                           Id          : in Service_Id)
   is abstract;
procedure Accepted_Service(Resource : in Manager_Type;
                           Id       : in Service_Id)
   is abstract;
procedure Rejected_Service(Resource : in Manager_Type;
                           Id       : in Service_Id)
   is abstract;
procedure Terminated_Service(Resource : in Manager_Type;
                             Id       : in Service_Id)
   is abstract;
```

Fig. 10. *Resource Manager* tagged type

Managers themselves are extensions of a tagged type (Figure 10), and must implement the abstract primitive subprograms that are used by the Local Provider to request the evaluation of a service, and to inform of service acceptance, rejection and termination. The implementation of a particular resource manager must ensure the consistency of the resource, guaranteeing that after a request is evaluated, it is considered as granted for other resource requests performed until rejected or terminated. Note that several resource requests may coexist for the same service request, due to the layering of resources presented in Figure 3.

As an example of attribute and manager instantiation, Figure 11 presents a *User_Quality* resource manager, which maps to the high-level user perception manager of Figure 3. A single attribute is defined, which is must be mapped by the *User_Quality_Manager* to actual values in other Dimensions (such as Video and Audio quality).

```
package User_Quality is

   -- Attributes

   type Possible_Values is (High, Medium, Low);

   type User_Value is new Attribute_Value with record
      Value : Possible_Values;
   end record;

   -- Implementation of Difference


   -- Manager

   type User_Quality_Manager is new Manager_Type with record
      -- mapping between User and Audio/Video
   end record;


   -- Implementation of Manager_Type abstract services
   -- This Resource manager will use the managers for
   -- Video Quality and Audio Quality

   Manager : aliased User_Quality_Manager(User);

end User_Quality;
```

Fig. 11. User perception quality manager

Figure 12 presents an example of how this manager could be used by an application to request for High quality in what concerns the overall service (in annex, another example is provided, for the dimensions of Video and Audio quality). Note that if a service is accepted, it is possible to get the actual obtained QoS values, not only for the requested dimension, but also in lower-layer dimensions. For instance in Figure 12, upon acceptance, the procedure call

```
Attr_List.Get_First(QoS_Accepted(Video).Attributes, User_Accepted);
```

will provide in `User_Accepted` an access value to the first video attribute.

```
procedure Create_QoS_Request is

   QoS_Values, QoS_Accepted : QoS_Values_Type;
   Accepted                 : Boolean;
   Id                       : Service_Id;
   User_Perception          : Attribute_Value_Access;
   User_Accepted            : Attribute_Value_Access;

   Quality_Value            : QoS_Dimension_Access;

begin

   User_Perception := new User_Value;
   User_Value(User_Perception.all).Value := High;

   Quality_Value := new QoS_Dimension_Values;
   Quality_Value.Importance := 1;

   Attr_List.Insert_First(Quality_Value.Attributes,
                          User_Perception);

   QoS_Values := (User => Quality_Value,
                  others => null);

   Application_Interface.Request_Service( QoS_Values,
                                          QoS_Accepted,
                                          Accepted,
                                          Id);

   if Accepted = true then
         Attr_List.Get_First(
             QoS_Accepted(Video).Attributes,
             User_Accepted);
   else
         -- if not accepted
   end if;
end Create_QoS_Request;
```

Fig. 12. Example of requesting service with User Perception value

6 Conclusions

In this paper we presented an overview of a framework which is being implemented for managing QoS-aware applications in a dynamic, ad-hoc, distributed environment. This framework is being used both for validating the group formation and processing approach of [4], but also to assess the suitability of the Ada language to be used in the context of dynamic, QoS-aware systems.

Currently, the resource managing support, the service negotiation and proposal processing is already implemented, allowing us to demonstrate that Ada provides the required mechanisms for the framework purposes. Nevertheless, work must still be done on the management of the collaborative system as a whole, and on real experience on actual systems.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions. This work was partially supported by FCT, through the CISTER Research Unit (FCT UI 608) and the Reflect project (POSI/EIA/60797/2004).

References

1. Clemens C. Wust, Liesbeth Steffens, Reinder J. Bril, and Wim F.J.Verhaegh. Qos control strategies for high-quality video processing. In Proceedings of the 16th Euromicro Conference on Real-Time Systems, Catany, Italy, June 2004.
2. Christina Aurrecoechea, Andrew T. Campbell, and Linda Hauw. A survey of qos architectures. Multimedia Systems, 6(3):138{151, 1998.
3. ARTIST (IST-2001-34820). Selected topics in Embedded Systems Design: Roadmaps for Research. Part III Adaptive Real-Time Systems for Quality of Service Management, May 2004. Available at http://www.artist-embedded.org/.
4. Luis Nogueira, Luis Miguel Pinho. Dynamic QoS-Aware Coalition Formation. In 13th International Workshop on Parallel and Distributed Real-Time Systems, Denver, Colorado, USA, April 2005.
5. Michael Ditze, Berta Batista, Eduardo Tovar, Peter Altenbernd, and Filipe Pacheco. Workload balancing in distributed virtual reality environments. In 1st Intl. Workshop on Real-Time LANs in the Internet Age, Satellite Event to the 14th Euromicro Conference on Real- Time Systems, 2002.
6. V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In Distributed Systems, Mullender, S. (Ed.), 2nd Ed., Addison-Wesley. 1993.
7. T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. Qos negotiation in real-time systems and its application to automated flight control. IEEE Transactions on Computers, Best of RTAS'97 Special Issue, 49(11):1170{1183, November 2000.
8. Ricardo Barbosa, Luis Miguel Pinho. Mechanisms for Reflection-based Monitoring of Real-Time Systems. Work-In-Progress Session of the 16th Euromicro Conference on Real-Time Systems. Catany, Italy, June 2004.

Annex. Video and Audio Quality Managers

```
package Audio_Quality is

   -- Attributes
   type Audio_Attributes is (Sampling_Rate, Sample_Bits);

   type Sampling_Rate_Values is (rate_8, rate_16, rate_24, rate_44);

   type Sample_Bits_Values is (bits_8, bits_16, bits_24);


   type Sampling_Rate_Value is new Attribute_Value with record
      Value : Sampling_Rate_Values;
   end record;
   -- Implementation of Difference

   type Sample_Bits_Value is new RM.Attribute_Value with record
      Value : Sample_Bits_Values;
   end record;
   -- Implementation of Difference


   -- Manager
   type Audio_Quality_Manager is new Manager_Type with record
      -- mapping between Audio and CPU/Memory
   end record;

   -- Implementation of Manager_Type abstract services
   -- This Resource manager will use the managers for CPU and Memory

   Manager: aliased Audio_Quality_Manager(Audio);

end Audio_Quality;


package Video_Quality is

   -- Attributes
   type Video_Attributes is (Color_Depth, Frame_Rate);

   type Color_Depth_Values is (bits_1, bits_8, bits_16, bits_24);

   type Frame_Rate_Values is range 1 .. 30;


   type Color_Depth_Value is new Attribute_Value with record
      Value : Color_Depth_Values;
   end record;
   -- Implementation of Difference

   type Frame_Rate_Value is new Attribute_Value with record
      Value : Frame_Rate_Values;
   end record;
   -- Implementation of Difference

   type Frame_Rate_Range is new Attribute_Value with record
      Low: Frame_Rate_Values;
      High: Frame_Rate_Values;
   end record;
   -- Implementation of Difference


   -- Manager
   type Video_Quality_Manager is new Manager_Type with record
      -- mapping between Video and CPU/Memory
   end record;

   -- Implementation of Manager_Type abstract services
   -- This Resource manager will use the managers for CPU and Memory

   Manager: aliased Video_Quality_Manager(Video);

end Video_Quality;
```

```
-- Example of use of the Video and Audio Quality Managers

procedure Audio_Video_Managers_Request is

   QoS_Values, QoS_Accepted:QoS_Values_Type;
   Accepted:                Boolean;
   Id:                      Service_Id;
   Attribute_Value:         Attribute_Value_Access;
   Video_Values,
   Audio_Values:            QoS_Dimension_Access;

begin

   -- Video First

   Video_Values := new QoS_Dimension_Values;
   Video_Values.Importance := 1;


   -- video frame rate
   Attribute_Value := new Frame_Rate_Value;
   Frame_Rate_Value(Attribute_Value.all).Value := 15;

   Attr_List.Insert_First(Video_Values.Attributes, Attribute_Value);

   Attribute_Value := new Frame_Rate_Range;
   Frame_Rate_Range(Attribute_Value.all).Low := 5;
   Frame_Rate_Range(Attribute_Value.all).High := 10;

   Attr_List.Insert_Last(Video_Values.Attributes, Attribute_Value);


   -- video color
   Attribute_Value := new Color_Depth_Value;
   Color_Depth_Value(Attribute_Value.all).Value := bits_8;

   Attr_List.Insert_Last(Video_Values.Attributes, Attribute_Value);

   Attribute_Value := new Color_Depth_Value;
   Color_Depth_Value(Attribute_Value.all).Value := bits_1;

   Attr_List.Insert_Last(Video_Values.Attributes, Attribute_Value);


   -- Audio Second

   Audio_Values := new QoS_Dimension_Values;
   Audio_Values.Importance := 2;


   -- Sampling rate
   Attribute_Value := new Sampling_Rate_Value;
   Sampling_Rate_Value(Attribute_Value.all).Value := rate_8;

   Attr_List.Insert_First(Audio_Values.Attributes, Attribute_Value);


   -- Sample bits
   Attribute_Value := new Sample_Bits_Value;
   Sample_Bits_Value(Attribute_Value.all).Value := bits_8;

   Attr_List.Insert_Last(Audio_Values.Attributes, Attribute_Value);


   -- QoS Values to request

   QoS_Values := (Video => Video_Values,
                  Audio => Audio_Values,
                  others => null);


   Application_Interface.Request_Service(QoS_Values,
                                         QoS_Accepted,
                                         Accepted,
                                         Id);

end Audio_Video_Managers_Request;
```