Distributed Federative QoS Resource Management

Günther Hölzl, László Böszörményi

Universität Klagenfurt, Institut für Informationstechnologie
Universitätsstraße 65–67, A–9020 Klagenfurt
{guenther, laszlo}@itec.uni-klu.ac.at

Abstract

In a distributed multimedia system QoS resources have to be managed carefully to utilize the resource pool in a way that bottlenecks can be avoided. Our key idea is to let the applications participate on the resource management. We propose a distributed architecture with a fine granulated, balanced resource management with explicit QoS characteristics. The architecture is based on a distributed cooperative resource manager which combines both the adaption and reservation principle for guaranteeing QoS. We have designed and implemented a prototype of our federative QoS resource manager (FQRM) in the Java environment.

Key words: QoS resource management; distributed resources; cooperative resource sharing

1 Introduction

Modern multimedia applications in general have an insatiable demand for resources. Not only the applications exist in a non-cooperative world but also the users want to run their applications with the best possible quality of service (QoS). Resources with a guaranteed QoS, which are used for running multimedia applications, are commonly overloaded [?]. One solution is, of course, to add as much resources as necessary (e.g. better communication links, more processing power, faster hard disks etc.). This approach has not only the disadvantage that it wastes a lot of resources, but it is actually only meaningful if it is just the *number* of running applications that has to be increased and the size of needed resources per application is fixed. Otherwise, there is no limitation on the growth of demand on resources.

As a result of the development of powerful graphics and processing devices multimedia computing systems have changed from the *control-only approach* to the *fully integrated approach*. Continuous media technology had an influence not only on the communication infrastructure but also on operating systems [**?**].

A multimedia application assigns different types of system resources for representing a given user QoS. The transformation from the user QoS to the system QoS allows combinations, e.g. a smaller amount of assigned communication resources could be compensated by a larger amount of processing power by using better compression techniques. Our key idea is to let the applications participate in the resource management. The advantage of such an approach is obvious: applications know exactly what they really need and can adapt to changing situations without loss of user-level QoS. For example, if an application knows that the available bandwidth is getting less then it may reduce the size of the video-window or reduce the resolution, or even close certain windows etc., without frustrating the users. If the operating system does this automatically then it may do it in a way which is unacceptable, but applications are supposed to know, which solution is appropriate for their users. However, there are two obvious disadvantages as well: applications must be cooperative and they have to take care of management problems that other systems may solve entirely hidden. For the first objection there is an easy answer: applications will be ready to cooperate, if this reduces their *costs*. For the second objection we answer: the participation of the applications must be made very easy – through a simple script language or through a simple API for resource allocation.

Our approach deals with “balancing” resources throughout the whole system. Beyond (the usual) negotiation in the admission phase, also *renegotiations* take place during sessions. The main goal of this approach is to schedule the limited set of different types of resources, characterized by QoS *parameters*, in a way that the number of the running and still “satisfied” multimedia applications will be maximized.

1.1 Adaption versus guaranteed resources in distributed systems

Currently most of the communication structures and also operating systems schedule their resources corresponding to the best effort principle, i.e. the scheduler tries its best but provides no guarantees for deadlines. This is principally unsuitable when dealing with QoS resources, where guarantees of availability are basic requirement. Mainly two strategies are applied to cope with this problem:

(1) *The adaption principle*[**?**]: the system components have to be aware of changing parameters and have to adapt to new situations. Adaption can be done at the operating- and communication system, the application or at the user domain. The goal of this approach is to widen the accepted resource space for a given user QoS. The application system is associated with a control system creating the feedback (see figure **??**). The use of the proper adaption algorithm will influence the overall system performance and a lot of new problems known from control theory arise, e.g. *instability.*

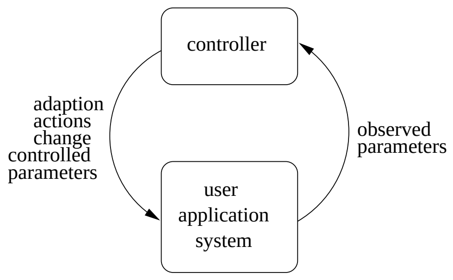

Fig. 1. Adaption mechanism

(2) *The reservation principle*: resources are assigned in advance to applications located at the end of a flow. The strategy is usual in operating- resp. communication systems supporting real time scheduling. Examples are the SMART real-time-scheduler[**?**], resp. isochronous channels in ATM[**?**] or IEEE-1394[**?**] and the use of RSVP[**?**]. In contrast to the former principle this technique needs an explicit resource management function which is handled by the operating system or dedicated daemons. Resources are assigned statically, e.g. the route in which bandwidth is guaranteed doesn't change over a session. Unallocated resources can be used for services with best effort characteristics.

Since applications are developed under an egocentric view of world they aren't aware of over-reserving bandwidth. For example an MPEG stream has due to the dynamic compression technique a very bursty characteristic[**?**]. At startup time the application can only estimate the needed resource bandwidth. Cell oriented techniques (like in ATM[**?**] networks) provide a possibility of dynamic resource consumption even for such cases. Unused resources can be utilized by traditional best effort communication services. However, unused reserved resources cannot be reassigned to other applications.

We examined these two basic principles and combined them in the *federative resource management principle.* The manager provides on the one side an interface for reservation of resources with guaranteed characteristics. On the other side, it also implements strategies for resolving bottlenecks by employing the adaption principle. In this paper we describe mainly the structure of the system and the areas of operation.

1.2 Design principles

The design principles of our QoS resource management system can be summarized as follows:

- *Low implementation costs*: the resource manager should incorporate easy to implement algorithms and resource models. If adaption to new management strategies is desired, there shouldn't be much effort to do this.
- *Easy to integrate – low usage costs*: the programmer shouldn't be burdened by a hard to understand API. Existing applications should be able to adapt to this new technology easily.
- *Efficient communication model*: the communication overhead created by the extra resource manager shouldn't stress the network and block valuable resources.
- *Fairness*: Requests for resources can be affiliated with priorities and sessions of applications therefore can be categorized. The manager shouldn't prefer any application of the same class.
- *Fault-tolerance*: Even if a node in the distributed system fails, the rest of the system shouldn't block.

2 Communication model and API

Resource management for distributed multimedia systems has become a challenge for developers because of the large variety of the different resource types. Not only communication bandwidth has to be reserved for the transmission of isochronous data. Also CPU time, graphics power, audio capabilities and DSP power are resource types needed by multimedia sessions. Resource management functions are in principle orthogonal to the applications tasks. Thus, there has to be a strict rule which says what has to be done in the manager and what in the application.

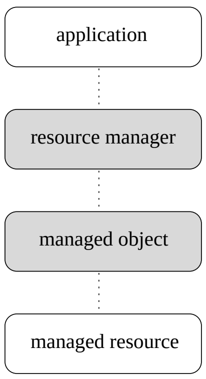

Fig. 2. Inserting layers of abstraction

Traditional operating systems manage native resources centrally within the kernel. Applications can deposit a demand for getting more resources, e.g. more processing power, and the operating systems *reacts* by inserting/updating parameters in the scheduler. Clients have only a coarse-grained control through *ad hoc* interfaces that makes modular design difficult [?].

Our approach inserts two layers of abstraction, one to the view of the application and one to the view of the resource manager (figure **??**). Applications use a dedicated resource manager which gives an easy and uniform access to systems resources hiding the details of the underlying structure. The resource manager in turn uses unified objects for accessing the actual resources. If we apply this structure on the top of a traditional operating system then some system resource management functions may be duplicated. However, we gain a *uniform* view of resource management. A new operating system can integrate this new principle from scratch.

2.1 Structure and objects

Federative resource management assumes cooperative applications. For this, application programmers must have an easy control over the fine-grained management, otherwise they will just reject it.

We designed and implemented an object oriented architecture both for the manager as for the resources. The system structure is shown in figure **??**.

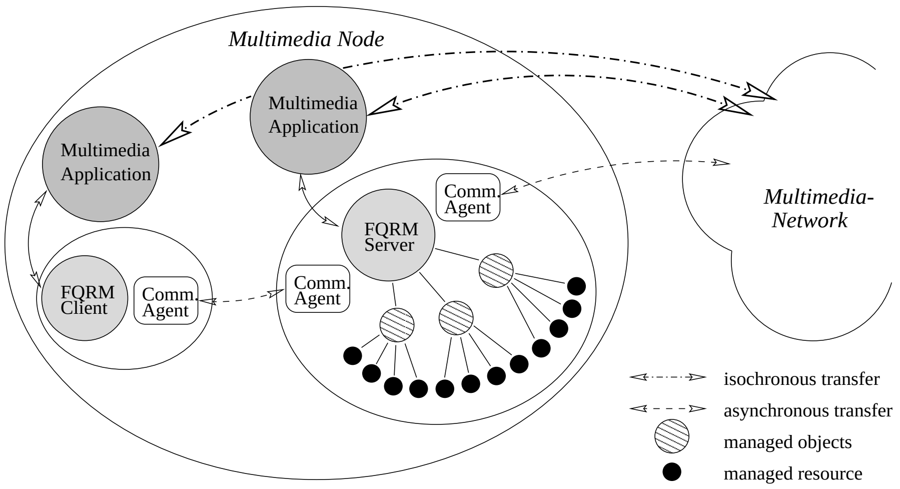

Fig. 3. Communication model of the resource manager

2.1.1 The manager objects

Applications create at startup a *federative QoS resource manager* (FQRM) *object*, which provides an interface for allocating and releasing of resources.

The first FQRM object created at a node starts as a server, all other ones are created as clients. Quasi stand alone servers with no dedicated binding to an application can be created by a dummy function. The client FQRM objects are active, i.e. they have their own thread of control, thus enabling them to continuously communicate to the server object. The FQRM objects communicate with each other through *communication agents*. The communication agents hide any communication details from the applications. The communication between agents is independent from the application and the specific management tasks.

The FQRM server object is the only local resource manager. Each server has bidirectional connections to its neighboring server objects. For achieving the *efficient communication model* goal resource requests should neither cause heavy network bursts, nor should they overload the processors. Demands from applications are passed to the local resource manager which tries to fulfill the requirements. In the case of a bottleneck or when no local resources have to be allocated, e.g. at parallel systems, the FQRM objects inform the network of the resource requirements. An implicit conclusion of the *efficient communication model* goal is that the system has to be scalable, since it has to be applied in a variety of distributed architectures. Solutions using trivial broadcasting methods, i.e. each server object sends the requirement information to all other server objects, need $O(n^2)$ messages for n servers and do not scale well. We had to adopt a principle for distributed systems that offers a short transmission delay for broadcasting and also has good scaling characteristics. These demands are quite similar to requirements of the World Wide Web (WWW) problems at actualizing links on pages without causing heavy network bursts. The *p-flood algorithm* [**?**] is a simple mechanism with all of the desired characteristics. It is also well established in the WWW. The principle is to forward requests not to all of the server's neighbors but only to a fixed or randomly selected subset of them, which in turn use the same strategy. All of the neighbors are reachable on multiple paths thus assuring that the information is passed to all of the FQRM server objects. By slightly increasing the transmission delay the p-flood algorithm avoids large communication peaks. As shown in figure **??** the server objects C and D are neighbors but D receives the requirement information not before time step 4. Servers which fulfill the resource requirements will return an offer for resource reservation. These offers are gathered, compared and finally the actual resource reservation can be done.

2.1.2 The managed objects

Resources are managed using *sets of objects*. For providing an abstraction of the managed resources we use an intermediate layer, the so called *managed objects* which provides a unified interface to the actual managed resources.

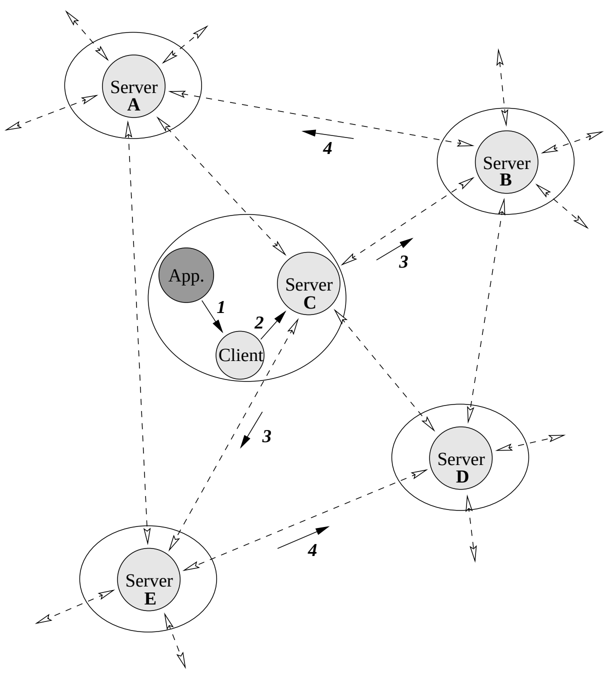

Fig. 4. Spreading of a resource request message using the *p-flood algorithm*

These resources are allocated at startup time or on demand from the operating system and are categorized to the different types of managed objects. Resource sets can be made persistent in a simple way, thus tolerating short node breakdowns.

In principle all resources are managed locally, there is no need of a central instance. This principle increases stability for the adaption principle by having a low latency for the control. The FQRM server manages reservable resources and tries to apply the adaption principle if a bottleneck arises. For fulfilling resource requirements and for applying the adaption principle we need a possibility of comparing different resource objects. We categorized the resource types in a hierarchy (see figure **??**) for providing uniform access methods. The structure was designed for applying in multimedia systems but can be extended easily for different purposes which are described in section 4.

To provide a unified access method to the manager objects the resources have to implement a unified interface. Each resource implements the methods `getAttribute()` and `setAttribute(Attribute)` for changing parameters. The methods `equals(QoSResource)` and `compareWith(QoSResource)` are the basis for the adaption principle thus making resources interchangeable.

For example resource interchanging will happen

- if the bandwidth of a channel resource is too low but there are plenty reserves of the processing resource. A comparison is done between the channel

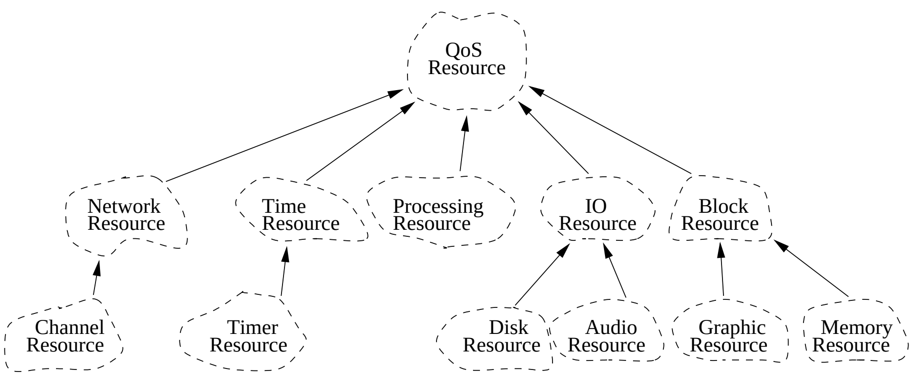

Fig. 5. Object hierarchy of the resources

resource and the processing resource with the appropriate attributes of the desired user QoS and as a result a solution with a better compression technique is proposed, thus decreasing the communication effort and increasing the needed processing resources.

- if memory resources can be interchanged by slower disk resources, if applicable, e.g. for storing of huge video data in video studios.
- if the user QoS is still satisfied, unavailable graphics resources can be replaced by audio resources.
- if timer resources are added to inform the application to release the resources.

We implemented the architecture in the Java environment using stream object sockets for intercommunication thus enabling the access to a wide range of applications in a homogeneous system.

2.2 API

As already mentioned, the client manager acts as an interface to the server manager. An application just declares a federative QoS resource manager (FQRM) which decides on its own how to start depending on whether a local server object is already installed or not.

To fulfill the easy to integrate design goal we provide a small but versatile API. The parameters of the API functions are *sets* or *multi-sets* of managed objects, described in section **??**, or *resource scripts* defining required resource characteristics of a session. In contrast to characteristic resource sets, resource scripts define a range of resource characteristics.

The following API functions are implemented:

- *get resources* is the main request function used by the application. Resources are allocated (and also released) in a bundle by using sets of resources. *Two-*

phase locking[?] is applied by the resource manager for preventing deadlocks in typical multimedia resource allocation situations. Resource sets are associated with unique identifiers, thus allowing applications to incrementally extend (or reduce) their allocated set of resources with just referring to the identifier.

  Required resources can be characterized by the application in the sense of *user QoS* or *system QoS*. Since managed resources only have system-QoS characteristics the former method implies a transformation done by the resource manager. This transformation also enables the employment of the adaption principle, described in section **??**.

  The application is able to determine the mode of accomplishment of management transactions, e.g. it can determine whether to block until the resources are reserved, or to block until a response from the server is given or to wait for the resource reservation with a polling strategy. This flexibility makes it easier to integrate the resource management into existing applications.
- *release resources* is the inverse function to *get resources*.
- *renegotiation* makes the resource allocation process more effective by enabling cooperative resource sharing. Allocated resources can also be pulled up when new resource requirements have to be processed and a bottleneck arises. The applications are informed via call-back functions or via messages when the allocated resources have set a *renegotiation flag*. For example, resources are pulled up by the manager, when the actual degree of an application's user QoS exceeds the desired minimum QoS and a bottleneck is caused by further resource requirements of different applications.

Figure **??** demonstrates the API within an interactive video on demand application.

3 Managing techniques

3.1 Managing policies

Distributed resource management (in contrast to central management) has the following advantages:

- *Scalability* by adding processing power proportional to resources.
- *Fault tolerance* by eliminating single points of failure.
- *Efficient communication* through the principle of locality.

The optimal resource management problem in principle is a multi-matching problem with constraints and is an NP hard problem. However, a lot of coop-

```
/**
  * declare a local client resource manager within a video on demand
  * application with interactive resource renegotiation. The resources
  * are managed by a local server manager.
  */
FQRM localRM = new FQRM("MPEG-Play");
int  setkey;
⋮
/**
  * get required resources from the local server resource manager
  */
QoSResource resources = new QoSResources[4];
/*----------------- specify channel parameters -------------------*/
resources[0]          = new ChannelResource();
videoServer.getChannelParameters(resources[0]);
/*---------------- specify graphics parameters -------------------*/
resources[1]          = new GraphicsResource();
resources[1].width    = userInputWidth;
resources[1].height   = userInputHeight;
resources[1].hwSupport= HWSUPPORTOVERLAY + HWSUPPORTMPEG;
/*--------------- specify processing parameters ------------------*/
resources[2]          = new ProcessingResource();
resources[2].procType = PROCFILTER;
resources[2].procQuant= 20;
/*------------------ specify timer parameters --------------------*/
resources[3]          = new TimerResource();
resources[3].timeSpan = PREVIEWTIME;

localRM.setMode(RMWAIT + RMRENEGOTIATE);

setkey = localRM.getResources(0, resources);
⋮
/**
  * resources aren't required any longer
  */
localRM.releaseResources(setkey);
```

Fig. 6. Sample code of a video client in Java using *FQRM*

erative sub-optimal scheduling techniques have been developed and classified [?]. Stochastic algorithms provide amazing solutions to this problem, e.g. *lottery scheduling* [?], which uses *tickets* and *currencies* to manage resources. By the way of *deflation/inflation* and *ticket transfer*, resources can be controlled quite well with a low processing effort.

These algorithms also allow the employment of a reasonable technique, called *resource balancing*, with the following properties:

- Resources should be located where the primary work has to be done (principle of locality).
- If possible, resources have to be managed in a way, that a fine granulated division can be realized.
- Resources can be shared among different applications.

We use this technique for integrating the *adaption principle* into the manager. Tickets are related to managed objects which in turn are related to applications. In case of a resource bottleneck the manager uses lottery scheduling for choosing and comparing of managed objects.

3.2 Mobile Resources

Unallocated resources belong to a locally installed server manager. By accessing resources from a foreign manager object, the control is passed to the foreign node by copying resource objects to foreign server objects. We call this technique "moving mobile resources". This technique avoids large communication overhead.

4 Operational Areas

4.1 Distributed multimedia architectures

This field is probably the main application area of the federative QoS resource manager. Film and audio studios have to handle isochronous data streams with different priorities, e.g. a simple query can be done with a lower priority than live recording. To provide a lot of communication power, host processors are interconnected very tightly and the task of the resource manager is to offer the resources to the applications in a way that the maximum system utilization can be reached.

Besides the management of dedicated hardware, e.g. DSP pools for real time video, filtering is one of the operational areas of the QoS resource manager.

4.2 Distributed hard real time applications

A problem in this area is the assignment of processing power to applications. What has to be done when a new task is starting? Which unallocated processing unit can fulfill the tasks requirements? Typical areas of operation are DSP pools for speech processing and robotics.

4.3 Embedded systems

The management of peripheral elements used by different distributed tasks is quite hard to handle. A synchronization between applications using shared resources is fairly difficult to establish and maintain, e.g. a task wants to get an allocated resource, how does it signal its requirement to the other task? The easy unified method for cooperating just with the resource manager simplifies the problem. In the worst case, a high prioritized application could preempt a task with a low resource priority using the manager. Not only the reservation but also the reassignment of resources is done more efficiently: when a task can't react on an inquiry in a certain time, the resource will be withdrawn by the manager.

4.4 Fault tolerant systems

Fault-tolerant systems are supposed to be able to switch to alternative resources in the case a shortage of resources arises. Traditional systems reserve resources statically at startup time even if the resources are used only in the case of emergency, e.g. backup structures. A federative resource management system will allow a more flexible allocation scheme.

5 Conclusion and further work

Distributed federative resource management is a technique to use the advantages of both adaption and reservation. The object oriented infrastructure gives an easy to implement interface for the applications and for the manager objects. The effort of integrating the system in an already existing application is moderate.

We have implemented a first version of FQRM in the Java environment for the purpose of managing small distributed multimedia systems. The next step will be the integration into a medical video on demand system. A teaching system using different classes of priorities is also planned.

Further research areas will be the management of different interconnect structures and also operating systems for applying the federative resource management principle to heterogenous architectures.

References

[1] T.L. Casavanat, J.G. Kuhl, *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems*, IEEE Transactions on Software Enginieering, vol. 14, no. 2, February 1988

[2] J. Gecsei, *Adaption in Distributed Multimedia Systems*, IEEE-Multimedia, April-June 1997

[3] IEEE Computer Society, *IEEE Std 1394-1995, IEEE Standard for a High Performance Serial Bus*, New York, August 1996

[4] F. Kappe, *A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems*, Journal of Universal Computer Science (J.UCS) Vol. 1, No. 2, pp. 84-104, Springer, February 1995

[5] J.Y. Le Boudec, *The Asynchronous Transfer Mode: A Tutorial*, Computer Networks and ISDN Systems, vol. 24, May 1992

[6] J. Nieh, M. Lam, *The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications*, Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, St. Malo, France, October 1997

[7] O. Rose, *Simple and efficient models for variable bit rate MPEG video traffic*, Performance Evaluation, No. 30, pp. 69-85, 1997

[8] H. Schulzrinne, *Operating system issues for continuous media*, Multimedia Systems, Springer Verlag, 1996

[9] A.S. Tanenbaum, A.S. Woodhull, *Operating Systems, Design and Implementation*, Second Edition, p. 179, Prentice Hall, 1997

[10] C.A. Waldspurger, W.E. Weihl, *Lottery Scheduling: Flexible Proportional-Share Resource Mangement*, Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI '94), pp. 1-11, Monterey, California, November 1994

[11] C.A. Waldspurger, W.E. Weihl, *An Object-Oriented Framework for Modular Resource Management*, Proceedings of the Fifth Workshop on Object-Orientation in Operating Systems (IWOOOS '96), Seattle, Washington, October 1996

[12] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, *RSVP: A New Resource ReSerVation Protocol*, IEEE Network Magazine, September 1993

Biographies

GÜNTHER HÖLZL received a Dipl.-Ing. degree in Telematik (telecommunications and computer science) from the Graz University of Technology, Austria in 1993. He joined Commend Communications Systems at Salzburg in 1994 where he developed a series of new generation intercom exchanges. In 1987 he got research assistant at the department of computer science at Klagenfurt University, Austria in the group of Prof. László Böszörményi in the fields of distributed and parallel systems and programming languages. He is member of IEEE.

LÁSZLÓ BÖSZÖRMÉNYI is a full professor of computer science at the University Klagenfurt, Austria. He received his M.S.Sc and Ph.D. from the Technical University Budapest, Hungary. His main research areas are distributed and parallel systems, multimedia systems and programming languages. He publishes regularly at international conferences and journals, he is organizer and program committee member of several international workshops and conferences. He is a member of ACM and IEEE.