Predictability and Resource Management in Distributed Multimedia Presentations

Costas Mourlas

Department of Computer Science, University of Cyprus,
75 Kallipoleos str., CY-1678 Nicosia, Cyprus
mourlas@ucy.ac.cy

Abstract. The continuous media applications have an implied temporal dimension, i.e. they are presented at a particular rate for a particular length of time and if the required rate of presentation is not met the integrity of these media is destroyed. We present a set of language constructs suitable for the definition of the required QoS and a new real-time environment that provides low-level support to these constructs. The emphasis of the proposed strategy is given on deterministic guarantees and can be considered as a next step for the design and the implementation of *predictable* continuous media applications over a network.

1 Introduction

The current interest in network and multimedia technology is focused on the development of distributed multi-media applications. This is motivated by the wide range of potential applications such as distributed multi-media information systems, desktop conferencing and video-on-demand services. Each such application needs Quality of Service (QoS) guarantees, otherwise users may not accept them as these applications are expected to be judged against the quality of traditional services (e.g. radio, television, telephone services). The traditional network environments although they perform well in static information spaces they are inadequated for continuous media presentations, such as video and audio.

In a distributed multimedia information system (see figure 1) there is a set of Web-based applications where each application is allocated on a different node of the network and can require the access of media servers for continuous media data retrieval. These continuous media servers can be used by any application running in parallel on a different node of the network. Each such presentation has specific timing and QoS requirements for its continuous media playback. This paper presents a new set of language constructs suitable for the definition of the required QoS and the real-time dimension of the media that participate in multimedia presentations as well as a runtime environment that provides low-level support to these constructs during execution.

2 The Proposed Language Extensions for QoS definition

Playing a set of multimedia presentations in a traditional network architecture two main problems are met. Firstly, the best-effort service model provided by

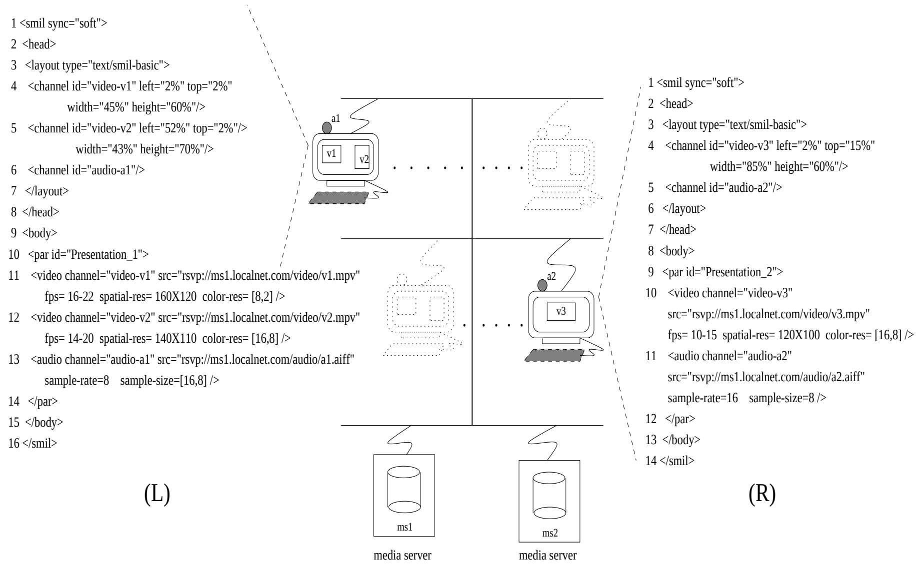

Fig. 1. *A Distributed Multimedia Information System*

the existing systems does not address the temporal dimension of the continuous media data during their retrieval and transmission phase. Resource reservation even if it is required, it is not the final answer to the end-users. The end-users actually care on how to exploit all the available (and reserved) resources in a best way such that the multimedia application will be presented according to the expected quality requirements. For example, a 10% reservation of the total bandwidth to a video presentation means that the video can be played either colored with a rate of 10 frames per second or grey-scaled with a rate of 18 frames per second. The decision has to be taken by the end-users and the multimedia authors, providing high-level language primitives and special annotation for the definition of any quality requirement. This new set of high-level language constructs will be presented in the following paragraphs and comes as a continuation of our previous work described in [6].

The language that will be extended is SMIL [9], a language for Web-based multimedia presentations which has been developed by the W3C working group on synchronized multimedia. These extensions are introduced along the lines of SMIL, and there is an attempt to reuse terminology wherever feasible. SMIL describes four fundamental aspects of a multimedia presentation: temporal specifications, spatial specifications, alternative behaviour specifications and hypermedia support. In this section we introduce and define a fifth aspect of a multimedia presentation, called *quality specifications*. In our extended SMIL language, the two continuous media objects can be described together with their quality requirements within a document via the following syntax:

- $\langle$`video cmo-attributes v-qos-attributes`$\rangle$, and
- $\langle$`audio cmo-attributes a-qos-attributes`$\rangle$.

The extensions are defined by the two new sets of attributes `v-qos-attributes` and `a-qos-attributes` for video and audio respectively. The set `cmo-attributes` is curenlty supported by SMIL to define the location and duration of the media object. The new `v-qos-attributes` and `a-qos-attributes` lists describe quality requirements using the attributes:

fps : The value of fps defines the temporal resolution of a video presentation by giving the number of frames per second. The value of this attribute can be any positive integer or a range of positive integers. For example giving `fps=14-18` as attribute to a video object, it means that the accepted values for this video presentation can be any rate between 14 and 18 frames per second (Figure 1 lines: L-11,L-12,R-10).

spatial-res : The *spatial-res* definition of a video presentation specifies the spatial resolution in pixels required for displaying the video object. In our model, the concepts of *layout* and *resolution* are separated. The *resolution* is a quality concept. If an ordered list of resolutions is given (e.g. *spatial-res=[180X130, 120X70]*) then the video object will be presented with the highest possible spatial resolution according to the availability of system resources and can be altered at run time (lines: L-11,L-12,R-10).

color-res : This attribute specifies the color resolution in bits required for displaying the video object. Typical values are 2, 8, 24 .... If an ordered list of integer values is given (e.g. *color-res=[8,2]*) then the video object will be presented with the highest possible color resolution. (lines: L-11,L-12,R-10).

sample-rate : The value of *sample-rate* for an audio object defines in KHz the rate that the analog signal is sampled. If we need, for example, telephone quality the analog signal should be sampled 8000 times per second (i.e. *sample-rate = 8*), (lines: L-13,R-11).

sample-size : This attributes of an audio object specifies the sample size in bits of each sample. If an ordered list of integer values is given (e.g. *sample-size=[16,8]*) then each sample will be represented with a number of bits equal with one of the values given. For telephone quality, each sample of the signal is coded with 8 bits whereas for CD quality it is coded with 16 bits. The highest value that can be used for every sample it is decided at run time according to the availability of the resources (lines: L-13,R-11).

The above language primitives form a complete set for QoS definition of every distinct continuous media that participate in a multimedia presentation. If several media streams have to be combined then inter-media synchronization is another important factor of quality specification but this subject has been extensively studied and completely supported by the standard SMIL language.

3 The Proposed Runtime Environment

We view every different multimedia presentation s_i as a *periodic* task τ_i with period T_i. Every periodic task τ_i is allocated on a different node of the distributed system and requires in each period the retrieval of a number of media blocks from the remote disk of the server. CS_j^i is the deterministic disk access time that task τ_i requires in every period to retrieve data for all of its streams from the server S_j (communication delays can be included in the evaluation of every CS_j^i). Every data retrieval section on a remote shared server S is guarded by a `lock(S)` statement. These locks are released after the data retrieval using the `unlock(S)` statement. The term "critical section" will be used to denote any data retrieval section of a task defined between a `lock(S)` and the corresponding `unlock(S)` statement.

We follow a rate monotonic strategy for priority assignments. Periodic tasks are assigned priorities inversely to tasks periods (ties are broken arbitrarily). Hence, task τ_i with period T_i receives higher priority than τ_j with period T_j if $T_i < T_j$.

The period T_i and the computational requirements CS_j^i of every task are determined by the desired QoS of the stream that the task represents as well as system resources (processor speed, disk access time). The formal procedure of transforming the set of distributed multimedia presentations with quality of service expectations to a set of periodic tasks is described in our previous work [7, 6]. We have to notice here that the scheduling analysis that follows does not consider ranges of QoS values and this task is left as future work.

A periodic task τ can have multiple non-overlapping critical sections, e.g.

$\tau = \{\ldots$ `lock(S`$_1$`)...unlock(S`$_1$`).....lock(S`$_2$`)...unlock(S`$_2$`)...`$\}$

but not any nested critical section. Each task is characterized by two components (CS^i, T_i), $1 \leq i \leq n$, where CS^i is the set $\{CS_j^i \mid j \geq 1\}$ that includes all the critical sections of the task τ_i. CS_j^i is the critical section of task τ_i guarded by statement `lock(S`$_j$`)`. We define as C_i the total deterministic computation requirement of all data retrieval sections of task τ_i, i.e $C_i = \sum_{x \in CS^i} x$.

Each server S_j can be either *locked* by a task τ_i if τ_i is within its critical section CS_j^i or *free* otherwise. Suppose that a task τ_i requires to lock server S_j and enter its critical section CS_j^i issuing the operation `lock(S`$_j$`)`. Then the following cases can occur:

1. The server S_j is *free*. Then, the server S_j is allocated to the task τ_i, the task τ_i proceeds to its critical section and the state of S_j becomes *locked*. A server S_j *locked* by task τ_i can not be accessed by any other task.
2. If case 1 does not hold, i.e. server S_j is currently *locked*, then after its release it is allocated to the highest priority task that is asking for its use. The task τ_i will proceed to its critical section if and only if server S_j has been allocated to τ_i.

By the definition of the protocol, a task τ_i can be blocked by a lower priority task τ_j, only if τ_j is executing within its critical section CS_l^j when τ_i asked for the use of the shared server S_l. Note also that the proposed synchronization protocol prevents deadlocks due to the fact that for any task τ_i there is no nested critical section. Thus, τ_i will never ask in its critical section for the use of any other server and so a blocking cycle (deadlock) cannot be formed.

We can easily conclude that a set of n periodic tasks, each one bound to a different node $\wp$ of a network can be scheduled using the proposed synchronization protocol if the following conditions are satisfied:

$$\forall i, 1 \leq i \leq n, \quad C_i + B_i \leq T_i \tag{1}$$

The term B_i represents the total worst case blocking time that task τ_i has to wait for the allocation of the required media servers in every period T_i. Once B_is have been computed for all i, the conditions (1) can then be used to determine the schedulability of the set of tasks.

3.1 Determination of Task Blocking Time

Here, we shall compute the worst-case blocking time B_l^i that a task τ_i has to wait the allocation of server S_l, following a response-time-analysis type formulation [3]. This longest blocking time occurs at the *critical instance* for τ_i.

Definition 3.1 A *critical instance* for task τ_i occurs whenever a request from τ_i to lock a server occurs simultaneously with the requests of all higher-priority tasks to lock this server. At that instance also, the lower priority task with the longest critical section executes its critical section holding the lock of that server.

Theorem 3.1 Consider a set of n tasks $\tau_1, \ldots, \tau_n$ arranged in descending order of priority. Each task is bound to a different node $\wp_i$ of the network and the proposed synchronization protocol is used for the allocation of the servers. Let

$H_l^i = \{CS_l^j \mid 1 \leq j < i\}$,	- set of critical sections used by tasks with higher priorities than τ_i accessing the same server S_l
$L_l^i = \{CS_l^j \mid i < j \leq n\}$,	- set of critical sections used by tasks with lower priorities than τ_i accessing the same server S_l
$\beta_l^i = \max(L_l^i)$.	- blocking time due to lower priority tasks

Then, the worst case blocking time B_l^i each time task τ_i attempts to allocate server S_l and execute its critical section is equal to:

$$B_l^i = \sum_{CS_l^j \in H_l^i} \lceil \frac{B_l^i + \Delta t}{T_j} \rceil * CS_l^j + \beta_l^i, \;\; 0 < \Delta t < 1 \qquad \text{if} \sum_{CS_l^j \in H_l^i} \frac{CS_l^j}{T_j} < 1 \tag{2}$$

Proof: The smallest integer value that satisfies equation 2 above represents the longest blocking time B_l^i for a task τ_i trying to enter its critical section CS_l^i at its worst-case task set phasing, i.e. at its critical instance.

If the worst-case task set phasing occurs at time $t_0 = 0$ then the right-hand side of the equation represents the sum of the computational requirements for server S_l for all inputs from higher levels at the time interval $[0, B_l^i + \Delta t)$ as well as the duration of one (actually the maximum) critical section of the lower priority tasks in L_l^i namely β_l^i. Task τ_i will enter its critical section at time B_l^i

when the server S_l becomes $free$, i.e. after its consecutive use from tasks during the worst-case phasing. At that time and during the interval $[B_l^i, B_l^i + 1)$, server S_l becomes $free$ for first time after t_0 and thus task τ_i will have the opportunity to lock S_l. The fact that server S_l is idle at time $t \in [B_l^i, B_l^i + 1)$ leads to the result that the sum of the computational requirements for server S_l over the interval $[0, t)$ equals B_l^i. Notice that an arbitrary value lying between zero and one is actually needed to check the load of the server at the interval $[B_l^i, B_l^i + 1)$, and this value is represented by the term Δt.

In all cases, the sum $\sum_{CS_l^j \in H_l^i} \frac{CS_l^j}{T_j}$ should be less than one. This sum represents the work load of server S_l or the utilization factor of the server due to higher priority tasks and should be less than one otherwise all these higher priority tasks could block repeatedly the task τ_i and in this case B_l^i will be unbounded (condition of formula 2). Hence the Theorem follows. □

Equations of the form 2 above do not lend themselves easily to analytical solution. However, a solution to this equation can be found by iteration. The total worst-case blocking duration B_i experienced by task τ_i is the sum of all these blocking durations, i.e. $B_i = \sum_{CS_j^i \in CS^i} B_j^i$. Once these blocking terms B_i, $1 \leq i \leq n$, have been determined, conditions (1) give a complete solution for the real-time task synchronization and scheduling in the distributed environment.

4 Related Work

A significant amount of work has been carried out for making resource allocations to satisfy specific application-level requirements. The Rialto operating system [2] was designed to support simultaneous execution of independent real-time and non-real-time applications. The RT-Mach microkernel [4] supports a processor reserve abstraction which permits threads to specify their CPU resource requirements. If admitted by the kernel, it guarantees that the requested CPU demand is available to the requestor.

The Lancaster QoS Architecture [1] provides extensions to existing microkernel environments for the support of continuous media. The QoS Broker [8] model addresses also the requirements for resource guarantees, QoS translation and admission control, so a new system architecture is proposed which provides all these issues. The Nemesis operating system is described in [5] as part of the Pegasus Project, whose goal is to support both traditional and multimedia applications.

We have to notice at this point that few of the above efforts address the problem of distributed multimedia applications and very few of all the current multimedia architectures provide any synchronization strategy and a theory for the analysis and the predictability of a set of multimedia applications executed in a distributed environment. Many CPU allocation schemes have been presented for multimedia applications based on the restrictive assumption that the applications are independent of one another and do not have access to multiple resources simultaneously.

5 Conclusions

In this paper, we studied a set of language extensions and a runtime environment suitable for creating and playing distributed multimedia information systems with QoS requirements. At the language level a set of language extensions for SMIL was presented suitable for the definition of the required QoS and the real-time dimension of the media that participate in a multimedia presentation. The runtime part is mainly focused on the maintenance of real-time constraints accross continuous media streams. It is based on a task oriented model that employs a *periodic-based service discipline* which provides the required service rate to a continuous media presentation independent of traffic characteristics of other presentations.

One direction of our future work will be on the ability of the runtime environment to support the required quality of service when the required quality lies within a range, by giving the minimal and the upper bound for the expected quality (e.g. *fps=18-22*). The runtime system will try to provide the best value in the range and it will be also authorised to modify this value at run-time towards the upper or the lower bound value according to the availability of the resources. This adaptation of quality of service will make the best use of the resources currently available to distributed applications and will give a fair solution to the presentation of continuous media applications over a network without sacrificing the ability to execute these applications *predictably* in time.

References

1. G.Coulson, G.S. Blair, P. Robin, and D. Shepherd. Supporting Continuous Media Applications in a Micro-Kernel Environment. In Otto Spaniol, editor, *Architectures and Protocols for High-Speed Networks*. Kluwer Academic Publishers, 1994.
2. M. B. Jones, D. Rosu, and M. Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, October 1997.
3. M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.
4. C. Lee, R. Rajkumar, and C. Mercer. Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach. In *Proceedings of the Multimedia Japan 96*.
5. I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1280–1297, September 1996.
6. C. Mourlas. A Framework for Creating and Playing Distributed Multimedia Information Systems with QoS Requirements. In *Proceedings of the 2000 ACM Symposium on Applied Computing, SAC 2000* (accepted for publication) .
7. C. Mourlas, David Duce, and Michael Wilson. On Satisfying Timing and Resource Constraints in Distributed Multimedia Systems. In *Proceedings of the IEEE ICMCS'99 Conference*, volume 2, pages 16–20. IEEE Computer Society, 1999.
8. Klara Nahrstedt and Jonathan M. Smith. The QoS Broker. *IEEE Multimedia*, 2(1):53–67, Spring 1995.
9. W3C. SMIL Draft Specification. *See:* `http://www.w3.org/TR/WD-smil`.