Automatic Scheduling of a Dynamic Multimedia Application with POLKA: a Case Study

I. Demeure and F. Singhoff

Ecole Nationale Supérieure des Télécommunications
46, rue Barrault
75634 Paris Cedex 13, France

F. Horn

Centre National d'Etude des Télécommunications
38,40 rue du Général Leclerc
92131 Issy Les Moulineaux, France

Abstract

POLKA is a system to dynamically and automatically schedule application that have temporal QoS constraints. This paper briefly describes the POLKA system and scheduling approach; it then introduces a multimedia application that is used as a case study to show the effectiveness of the approach. The performance measurements presented in the paper were performed on a CORBA based prototype of POLKA.

1 Introduction - Motivation

In applications that involve continuous media (e.g. audio and video), information is expressed not only in its individual value, but also by the time of its occurrence. To run smoothly, multimedia applications must therefore meet temporal **Quality of Service (QoS)** constraints.

In order to guaranty that the temporal constraints are met, several proposed multimedia platforms have tempted to use technics derived from the hard real-time field (see [6]). Such technics require a very good knowledge of the application: tasks requirements, in particular in terms of CPU time, as well as tasks dependencies must be known prior to execution. These technics do not therefore allow for much flexibility and adaptivity. In addition, they often lead to resource over-reservation.

In the work presented herein, we concentrate on multimedia applications that execute on a single machine and involve continuous flows as well as sporadic traffic. The number of flows may vary over time. We assume that we have no knowledge of the requirements of each flow in terms of required processing time, period, etc. However, we do have a knowledge of specific temporal constraints (intra and inter flow synchronization); these constraints may vary during the execution of the application. The problem we address is that of **automatically** and **dynamically** finding a schedule of the application such that temporal constraints are met if such a schedule exists, and scheduling the application accordingly.

In this paper, we present a case study application. We outline the POLKA approach in which a multimedia application is specified as a set of objects and QoS equations (a more complete overview of the approach is given in [4]). We explain how an application can be partitioned into a set of dependant preemptable tasks and how deadlines and release times are assigned to each task. The tasks are then scheduled following an earliest deadline first policy. We describe a prototype platform designed on top of a CORBA[1] ORB[2] [3]. We present performance measurements that show that the overhead of scheduling with POLKA is reasonable for the targeted applications, and that it leads to a schedule where QoS constraints are taken into account in a much better way than if we run the same application using a round robin scheduler (recall that the objective is to automatically schedule the application).

2 Presentation of the case study application

Our case study application emulates an interactive TV application running on a PC or a workstation. The user may view a recording of a movie award ceremony (e.g. The American Oscar award) and at the same time view parts of one or several competing movies. The video flows are displayed in separate application windows and a unique sound flow is displayed on a local audio device.

In the POLKA specification model an application is defined as a collection of objects and a set of QoS equations. Objects are encapsulation units (in particular, data declared inside an object may not be accessed by another object). They may export operations at their interface. Operation invocations can be synchronous or asynchronous.

Figure 1 shows the object model for our example application. This model involves three objects: a client

[1]**CORBA** for **C**ommon **O**bject **R**equest **B**roker **A**rchitecture.
[2]**ORB** for **O**bject **R**equest **B**roker.

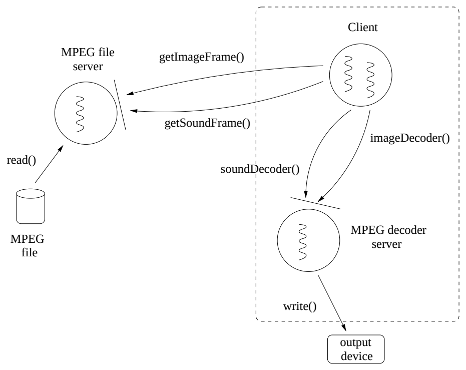

Figure 1: Movie Award Object Model

object, an MPEG2 file server object and an MPEG2 decoder object. Figure 2 describes the interfaces for the MPEG2 file server object and the MPEG2 decoder objects. These interfaces are described using the **I**nterface **D**efinition **L**anguage (IDL).

For each flow to be displayed, the client object loops on the invocation of two methods: the getImageFrame (resp. getSoundFrame) method at the MPEG file server interface to read MPEG image (resp. audio) frames from a disk; and the imageDecoder (resp. soundDecoder) method at the MPEG decoder interface to decode the frame and display it. Producer/consumer buffers are used to absorb the file system jitter. We used independent (demultiplexed) audio and video flows to experiment with fine-grain synchronization constraints (e.g. lip synchronisation).

```
module mpeg {
    interface mpegDecoder {
        long soundDecoder(in SoundFrame f);
        long imageDecoder(in ImageFrame f); };
    interface mpegServer {
        long getSoundFrame(out SoundFrame f);
        long getImageFrame(out ImageFrame f); }; };
```

Figure 2: IDL interfaces of interactive TV CORBA objects

Each time an invocation takes place, a series of what we term **significant events** can be observed at the object interfaces, namely: **IE** (Invocation Emission), **IR** (Invocation Receipt), **RE** (Response Emission) and **RR** (Response Receipt) events (see Figure 3).

Figure 4 shows a graph representation of our example application.

We use the followind notation: *Object.Op.Ev* where

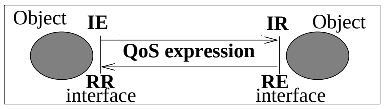

Figure 3: Observable events at an object interface.

Object denotes an object, *Op* is the name of an operation defined at the object interface and *Ev* is one of the four significant events. In the graph, the application code is partitioned into **portions of code** each ending with one of the above defined significant events. The graph captures significant data and control dependencies that will be used by the POLKA scheduler.

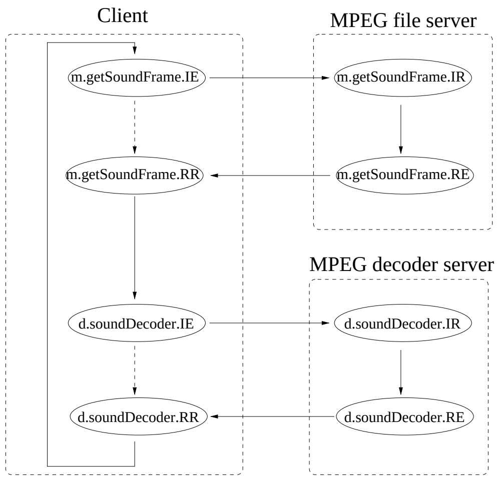

Figure 4: Movie Award Graph Model

Simple temporal QoS constraints are expressed in the form of simple QoS equations. We use the following notation [1]: $\tau()$ denotes the date function. $\tau(Object.Op.Ev,\ n)$ therefore corresponds to the date of the *nth* occurrence of the *Object.Op.Ev* event.

Figure 5 gives a set of QoS equations specified for our example application (d denotes an instance of the IDL interface *mpegDecoder*). Equations (1) to (4) define intra flow synchronization constraints on the sound and image flows. Equations (5) and (6) correspond to lip-synchronization constraints. Note that significant events appearing in a QoS equation are called **pertinent** events.

3 Scheduling

The first step in the automatic scheduling of the application is to translate QoS equation delays into task deadlines and release times. Nodes corresponding to pertinent events are assigned a deadline or a release time

Equations
(1)$\tau(d.soundDecoder.RR, n+1) - \tau(d.soundDecoder.RR, n) < x_1$
(2)$\tau(d.soundDecoder.RR, n+1) - \tau(d.soundDecoder.RR, n) > x_2$
(3)$\tau(d.imageDecoder.RR, n+1) - \tau(d.imageDecoder.RR, n) < x_3$
(4)$\tau(d.imageDecoder.RR, n+1) - \tau(d.imageDecoder.RR, n) > x_4$
(5)$\tau(d.soundDecoder.RR, n) - \tau(d.imageDecoder.RR, n) < x_5$
(6)$\tau(d.soundDecoder.RR, n) - \tau(d.imageDecoder.RR, n) > x_6$

Figure 5: MPEG Flows Synchronization

derived from the delay of the corresponding equation.

Let us illustrate how deadlines are assigned to nodes by going back to the graph describing our example application in Figure 4. Let us consider equation (1) (see Figure 5).

Let t denote the effective time at which the nth occurrence of the d.soundDecoder.RR event takes place; then the deadline of the node corresponding to the $n+1$th occurrence of the d.soundDecoder.RR event is $t + x_1 s$.

Initial **"parameterized"** deadlines of this kind can therefore be statically assigned to nodes corresponding to pertinent events. Effective deadlines are dynamically computed each time a pertinent node is ended. A difficulty is to assign a deadline to all nodes in a graph including those that do not correspond to a pertinent event.

In [2] Blazewicz proposed an algorithm that optimally schedules a set of dependent aperiodic preemptable tasks that may become ready at different time instants. For this purpose, a modified deadline is computed, for each task, using the following formula:

$$d_i = min(d_i, min(d_j \mid i \prec j))$$

where d_i denotes the deadline attached to task i and for any tasks i and j, $i \prec j$ defines a precedence relation that says that that task j must wait for the completion of task i to begin.

Therefore, task i inherits the smallest deadline of all its successor nodes. POLKA uses this formula to compute deadlines of all nodes including those that have not been assigned an initial deadline (it implicitly considers that they start with an "infinite" deadline).

In the example, the IE, IR and RE events corresponding to the d.soundDecoder invocation are not pertinent (i.e. they do not appear in a QoS equation); as soon as we know the effective time t at which the occurrence of the nth d.soundDecoder.RR event takes place, we can derive the deadline of the corresponding occurrence of the d.soundDecoder.IE, d.soundDecoder.IR and d.soundDecoder.RE event to be the same as that of the $n+1$th occurrence of the d.soundDecoder.RR event (namely $t + x_1 s$).

Release time equations are used to assign a release time to the corresponding tasks. If r_i denotes the release time computed for task i, then

$$r_i = max(r_i, max(r_j \mid j \prec i))$$

The POLKA scheduler maintains a list of all ready tasks, each of which has a deadline computed using the above formula. It also maintains a list of all tasks that are not to be released yet. The scheduler updates both lists each time it is invoked (each time a task terminates, a significant event occurs and when certain clock interrupts occur). It chooses the next task to be scheduled among the ready tasks using the Earliest Deadline First policy.

4 POLKA Platform

Our prototype platform framework uses the CORBA architecture. A POLKA application is therefore designed as a collection of CORBA objects (specified by IDL interfaces) and QoS equations.

In addition to the application objects, the platform involves a scheduler object and a loader object: they are passive objects that share the same address space.

When an applicative object starts executing, it calls the loader object. The loader reads the application dependency graph and the QoS equations (note that, as of today, graphs must be edited by the designer). It also activates all the objects that will be invoked by the new object if not already active. After registration, each time an object invokes a method, and for the four corresponding significant events, the POLKA scheduler is called. It blocks until the scheduler object gives it the processor.

This call to the scheduler is automatically inserted in the CORBA stubs and skeletons generated by the IDL compiler (we modified the IDL compiler to this purpose).

The CORBA object model well matches our specification model and using a CORBA ORB proved to be a good way to simplify the implementation of our prototype and to prepare the distribution of our approach. Note that plain CORBA is not well suited for the support of multimedia, and more generally for the support of real-time applications [5, 7]. It turned out to be a good support for us because we added a specific tool for the specification of QoS constraints, and because these QoS constraints are enforced through the POLKA scheduler.

5 Performance Evaluation and Conclusion

The experiments were conducted on a Sun Ultra Sparc 1 workstation using the omniORB2 CORBA 2 freeware from the Olivetti and Oracle research Laboratory on top of the Solaris 2.5 operating system. In our tests, we used equations (1), (3) and (5) with $x_1 = x_3 =$ 100 ms and $x_5 =$ 80 ms.

Evt	Sched. Overh.	Polka Efficiency	with Polka	without Polka
IE	690 us	% missed audio frame	0.405	2.355
IR	270 us	% missed video frame	0.393	6.629
RE	270 us	Audio average behind time(ms)	36.72	900.16
RR	220 us	Video average behind time(ms)	38.26	44.44

Figure 6: Measurements

We ran two series of experiments: one to evaluate the overhead of the POLKA scheduler; one to evaluate the POLKA scheduler efficiency.

The overhead measured on each of the four significant events corresponding to the soundDecoder and imageDecoder invocations is shown in Figure 6. The overhead is bigger for IE events because it is where initializations are done. The total overhead is therefore 1.4 ms for an invocation. It is significant compared to the audio frames processing time (2.11 ms) but turns out to be small compared to the processing time of a video frame (55.24 ms) and therefore reasonable once averaged over all flows.

We measured the number of times the delay between consecutive frames of a given flow is larger than the delay specified in the corresponding QoS equation and we evaluated the average additional delay. Again, as shown in Figure 6, POLKA performed well.

Finally, we evaluated the scheduler efficiency by measuring the number of times the delay between matching audio and video frames is larger than the delay specified in the corresponding QoS equation and by evaluating the average additional delay. Without POLKA all frames are late, because the round robin scheduler alternates the display of sequences of audio frames and sequences of video frames.

With POLKA the audio and video frames were properly interleaved, and no frame was late. This is an encouraging result that shows that the approach is effective even for fine grain synchronization.

A lot of work remains to be done: We are currently studying situations in which causal relationships between pertinent events cannot be easily established. We are also studying interaction with the operating system. We intend to include a monitoring object to the framework; the monitoring information will be used, in particular, to adapt the applications behavior. We are also interested in studying situations in which admission control could be used to provide guarantees. Last but not least, we are currently working on the distribution of the approach.

Acknowledgments

This research was supported by the ARCADE team of CNET (France Telecom research center).

We are grateful to Frank Girodengo, former ENST student, who developed the first version of the case study application.

References

[1] ARCADE,"Toward the integration of Real Time and QoS handling in ANSA architecture", *CNET- Paris A, technical note NT/PAA/TSA/TLR/3498*, July, 1993.

[2] J. Blazewicz, "Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines", *Gelende. H. Beilner (eds), Modeling and Performance Evaluation of Computer Systems*, Amsterdam, Noth-Holland, 1976.

[3] *OMG TC Document*, "The Common Object Request Broker : Architecture and Specification", June, 1995.

[4] I. Demeure J. Farhat-Gissler F. Gasperoni, "A Scheduling Framework for the Automatic Support of Temporal QoS Constraints", *proceedings of the Fourth International Workshop on Quality of Services (IWQoS), Paris*, March, 1996.

[5] D. Levine T. Harrison D. C. Schmidt A. Gokhale C. Cleeland, "TAO: a High-performance End System Architecture for Real-time CORBA", *Response to the OMG Real-time Special Interest Group Request for Information*, 1997.

[6] C. W. Mercer S. Savage H. Tokuda, "Applying Hard Real-Time Technology to Multimedia Systems", *Proceedings of the Workshop on the Role of Real-time in Multimedia/Interactive Computing Systems*, Raleigh-Durham, NC, December 1993.

[7] L.C. DiPippo R Ginis M. Squadrito S. Wohlever V. F. Wolfe I. Zykh, "Expressing and Enforcing Timing Constraints in a Real-Time CORBA System", *Technical Report number TR97-252*, University of Rhode Island, February 1997.