Rapport de Recherche

Numéro SINGHOFF-02-2002

Support of Temporal QoS Constraints for Distributed Object Oriented Multimedia Applications

I. Demeure †, L. Leboucher ‡,
N. Rivierre ‡ et F. Singhoff *

†Ecole Nationale Supérieure des
Télécommunications
CNRS URA 820
46, rue Barrault - 75634 Paris Cedex 13, France

‡France Télécom R&D
38,40 rue du Général Leclerc
92131 Issy Les Moulineaux, France

* LIMI/EA2215
Université de Bretagne Occidentale
20, av Le Gorgeu
29285 BREST

EA 2215
LIMI

Département, IUP d'Informatique
U.F.R. Des Sciences et Techniques
Université de Bretagne Occidentale
6, avenue Le Gorgeu
BP 817
29285 BREST cedex
FRANCE

February 2002

Abstract

In this paper we describe a specification technique and a middleware platform to support distributed multimedia applications. Temporal constraints are specified independently from the application itself and from the data flow graph that describes the multimedia system. The middleware is based on CORBA. It automatically schedules the applications in order to meet the QoS constraints, provided enough resources are available. The notions introduced are illustrated by an example. We also provide performance evaluation.

Contents

Chapter 1

Introduction

In the work presented here, we focus on the support of distributed multimedia applications involving continuous flows. By "continuous flow" (e.g. audio or video flows) we mean a sequence of data units that must be presented according to temporal quality of service (QoS) constraints; examples of such constraints are maximum delay between the display of two successive pictures of a given video flow, lip-synchronization and synchronization between animated objects.

In order to meet the temporal constraints of multimedia applications, classical solutions use the real-time properties of the underlying system scheduler and the real-time support provided by new generations of communication protocols (e.g. ATM [Vet95], RTP/RTCP [SCFJ96]). They also rely on specific components such as audio card and MPEG decoders.

These solutions have several limitations: First, schedulers such as Unix SVR4, provided by general purpose operating systems, were not designed to support continuous flows [NNH93]; a "multimedia" scheduler must provide abstractions similar to those used in multimedia applications.

Furthermore, it is not enough to consider each component of the multimedia system separately: it is often necessary to consider end-to-end quality of service [CAH96] and therefore to implement a global resource management policy.

When the scheduling directives are hard-coded, a small modification to the application code can induce complete re-scheduling of the application. A system for the support of multimedia applications should therefore make it possible to express temporal QoS constraints independently from the application itself. This is particularly important if portability is seeked. Also note that if the traditional solutions are well suited to static applications, they can hardly support dynamic ones (application where temporal constraints can change at runtime).

Finally, the support of multimedia applications with traditional solutions requires a very thorough knowledge of the underlying operating system, the underlying communication protocols and the specific components used (e.g. audio card, MPEG decoder).

The Polka specification model and middleware address these problems. With the specification model one can describe the application and the temporal QoS constraints that it must meet. The specification model must also define the components of the underlying platform and the data flow graph that represents the whole multimedia system (application + platform components) (see Figure 1.1).

The Polka middleware uses the specifications to automatically schedule applications in order to meet their temporal constraints, provided enough resources are available (e.g. processor, network bandwidth, memory). In Polka the application temporal constraints can be modified at runtime without modification to the application itself.

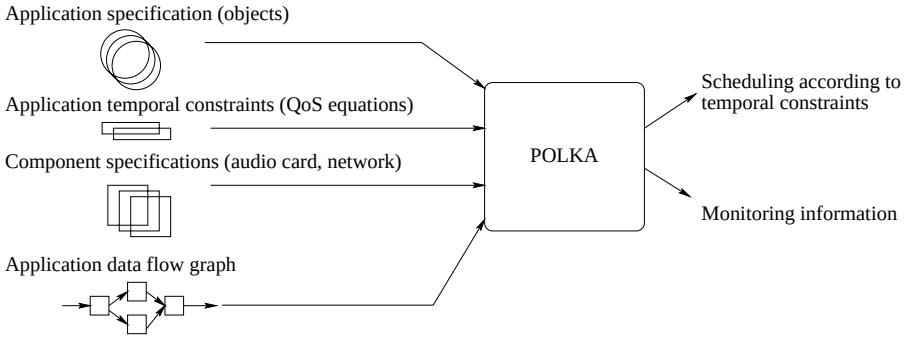

Figure 1.1: POLKA's model and tools.

The remainder of this paper is organized as follows: We first present the specification model. We introduce a distributed case study application involving an audio flow and a video flow that must be synchronized. The case study application is modeled using the POLKA specification model.

In Chapter 3, we present the goal and the architecture of our middleware. The middleware is comprised of an Object Request Broker (ORB) and a virtual machine layer. The virtual machine layer has been developed to increase portability of both the POLKA platform and the POLKA applications. As of today, POLKA supports LINUX and Solaris systems.

Chapter 4 is dedicated to performance evaluation. The measurements presented show that POLKA effectively supports the automatic scheduling of applications according to their temporal constraints. We compare two executions of our case study application: one using POLKA and one without POLKA. We show that synchronization is more effective in the first case than in the second one. We also show that, for the targeted applications, POLKA overhead is reasonable. Chapter 5, is dedicated to related work. We conclude in Chapter 6.

Chapter 2

Modeling a multimedia system

2.1 Application modeling

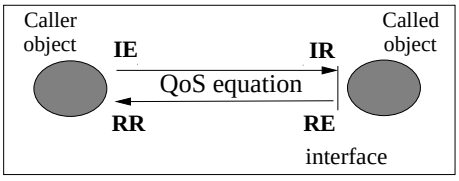

Figure 2.1: Events observed during an object invocation

In POLKA, an application is modeled as a set of objects that cooperate to process and display continuous flows [SBC+96]. Objects encapsulate code and data. Services exported by an object are described by an interface (currently a CORBA IDL interface). Objects cooperate through method invocations (shared memory is prohibited).

A multimedia flow is modeled by a sequence of object invocations. An object invocation leads to several observable events (see Figure 2.1): the invocation emission at the client side (IE event), the invocation receipt at the server side (IR event), the end of the invocation at the server side (RE event) and the response receipt at the client side (RR event). Asynchronous invocations only yield two events: IE and IR. **A multimedia flow therefore corresponds to a sequence of events.**

To illustrate the modeling of a POLKA application, let us consider an application that reads audio and video samples from movies stored in a file system (see Figure 2.2). Movies are encoded according to the MPEG standard [ISO95]. This application is composed of three objects: the *server* object reads MPEG frames from disk and sends them to the remote *mpegSource* object. The *mpegSource* object receives frames from the network and stores them in a buffer. The *mpegDecoder* object takes frames from the buffer, uncompresses them and displays the result on an output device (e.g. audio card, X11 server). In practice, POLKA objects are CORBA objects. Figure 2.3 gives the IDL specification of the *mpegDecoder* and the *mpegSource* objects.

Once the application objects have been described, the QoS constraints can be specified. They are described using equations that express temporal relationships between the events corresponding to object invocations (IE, IR, RE and RR events). Note that QoS equations can be hidden by a high level user interface; for example, the graphical interface of a video on demand application can include an "elevator" to indicate the desired video frame display rate.

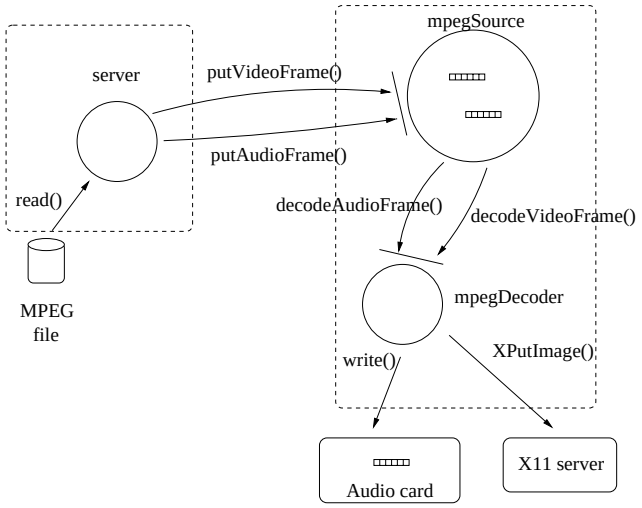

Figure 2.2: Case-study distributed application

```
interface mpegDecoder {
    long decodeAudioFrame(in frame f);
    long decodeVideoFrame(in frame f);
};
interface mpegSource {
    long putAudioFrame(in frame f);
    long putVideoFrame(in frame f);
};
```

Figure 2.3: IDL interfaces

2.2 Temporal QoS equations

QoS equations supported in POLKA are expressed in the QL (*QoS Language*) temporal logic [Ste93]. In this paper, we only study deterministic QoS constraints. Let us consider the following equation:

$$\forall\ n:\ \epsilon_1 \leq \tau(e, n+k) - \tau(f, n) \leq \epsilon_2 \tag{2.1}$$

where e and f are two events and $\tau(x, n)$ is an operator that gives the date of the n th occurrence of event x. This equation means that at least ϵ_1 units of time and at most ϵ_2 units of time must separate the n th occurrence of f and the $n+k$ th occurrence of e.

This equation can be used to define a periodic traffic of period p with a given jitter (if $\epsilon_1 = p - \epsilon$ and $\epsilon_2 = p + \epsilon$, the maximum jitter between the occurrences of events e and f is $2 * \epsilon$ units of time). Equation (2.1) can also be used to express a bound on the method invocation delay. If e and f denote the beginning and the end of an invocation then the response time on an invocation must be greater than ϵ_1 units of time and smaller than ϵ_2 units of time. Finally, equation (2.1) can express lip-synchronization constraints; in this case, e and f respectively map to a video flow event and an audio flow event.

Let us now consider the following equation:

$$\forall\ n:\ \epsilon_1 \leq \tau(e,n) - \tau(H_p,n) \leq \epsilon_2 \tag{2.2}$$

Here, H_p models a logical clock of period p, and e denotes an event. The date of the n th tick of the logical clock is given by $\tau(H_p,n) = n * p$. Equation (2.2) expresses intra-flow synchronization for a periodic traffic with jitter. The maximum delay between a flow event and the corresponding clock tick is $\epsilon_2 - \epsilon_1$. The delay between two successive occurrences of e is greater than $p-(\epsilon_2-\epsilon_1)$ and less than $p+(\epsilon_2-\epsilon_1)$ units of time. Note that modeling intra-flow synchronization with this kind of equation avoids flow skewing (contrary to equation (2.1)).

Finally, a tighter version of equation (2.2) is given by the following equation:

$$\forall\ n:\ \tau(e,n) = n * p \tag{2.3}$$

This equation models a strictly periodic flow (flow events occur every p units of time). Constant bit rate flows such as audio streams require this kind of equation.

Let us now design the set of equations ($S0$) corresponding to a set of temporal constraints on the case study application. In the case study application the audio flow is output to an audio card and the video flow is delivered by a X11 server. An audio card is an independent device involving one or several buffers. It reads data from its buffers and produces regular outputs (inputs and outputs are asynchronous). Over Solaris for instance, an *amd* audio device outputs one byte every 0.125 ms when the μ-LAW decoder is used. Therefore, the desired QoS at the output of the audio card can be expressed by:

$$\forall\ n:\ \tau(a,n) = n * 0.125\ ms \tag{$S0_1$}$$

where a denotes the event corresponding to the output of one byte by the audio card to an output device such as a speaker. In the remainder of this paper, Ha_{pa} denotes a logical clock whose period is $pa = 0.125\ ms$.

If the user wants to display 10 images per second (one image every 100 ms) with a maximum jitter of 40 ms, then the QoS observed at the output of the X11 server must be:

$$\forall\ n: 80\ ms \leq \tau(i,n+1) - \tau(i,n) \leq 120\ ms \tag{$S0_2$}$$

In this equation, i denotes the event produced when an image is displayed. Since one image is displayed every 100 ms and since $100/0.125 = 800$ audio samples are output to the speaker by the audio card in the same time interval, the equation corresponding to lip-synchronization is:

$$\forall\ n: -40ms \leq \tau(a, n*800) - \tau(i,n) \leq 40\ ms \tag{$S0_3$}$$

It can also be written:

$$\forall\ n: -40\ ms \leq \tau(Ha_{pa}, n*800) - \tau(i,n)\ \leq 40\ ms$$

For most users and movies, $80\ ms = [-40ms, 40ms]$ corresponds to a reasonable jitter on the lip-synchronization [SN95].

2.3 Modeling a multimedia system

In the previous chapter, we showed how temporal QoS constraints can be expressed in QL. However, defining QoS requirements is sometimes not sufficient since some important **components** have temporal behavior that cannot be ignored (e.g. audio card, network interface card).

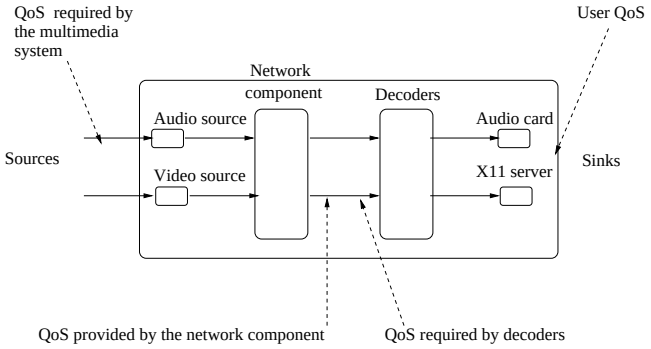

Figure 2.4: A data flow graph to model a multimedia system

A very common way to model multimedia systems is to use a data flow graph [And93, Jef92, SNH97, MNCK99]. This is the approach used in POLKA: a multimedia system is represented by a data flow graph where each node corresponds to a component and each arrow to a multimedia flow (see Figure 2.4). Flows are composed of a sequence of events (see Chapter 1).

The user specifies the QoS desired at the graph sink (e.g. lip-synchronization). We call it the **user's QoS**. At the source of the graph, QoS expressions correspond to the QoS required by the system to enforce the user's QoS. We call it the system **required QoS**.

Let us now see how the system components can be modeled. A component is a basic element of a multimedia system. Components can be composed in a sequential or in a parallel manner. A component is a "black box" that encapsulates the temporal behavior of a part of the system (hardware or software). It is defined by a clock and input and output flows. Finally, a component can use a buffer to store information acquired from its input flows.

A component temporal behavior is specified by QoS equations over input and output events. Equations are grouped in **QoS contracts** [BS98]. A QoS contract is defined by two sets of QoS equations: the first set (called **provided QoS**) expresses QoS the component must provide on its output flows. The second set (called **required QoS**) expresses QoS the component requires on its input flows to deliver the provided QoS. The contract says that if the required QoS is met at the input of the component, then the component will deliver the "provided QoS" at its output. QoS contract is a powerful abstraction to automatically deduce the QoS associated to a composition of components from the QoS of individual components [Leb98].

We distinguish between two kinds of components: those whose temporal behavior is known a priori, and those whose temporal behavior is discovered at runtime. In the second case, the component specification identifies the information that must be monitored by the platform (e.g. evaluate a round trip time for an asynchronous network component).

We use QoS contracts and composition operations to compute the QoS required by the system (at the sources) from the user's QoS and the component specifications. The resulting "required QoS" is a **sufficient condition** to enforce the user's QoS provided enough resources are available in the system. Since there is no resource reservation in the actual platform, we cannot guaranty that the user's requirements will be met. Note that if the platform does

not support resource reservation services, the model itself does not preclude such mechanisms. Also note that all the solutions in which the user's QoS can be guaranteed require resource information such as task execution times (e.g. [NL97, JRR97]). Such information is hard to evaluate (in particular for the targeted applications) and puts a heavy burden on the developer. Instead of using a resource reservation service, we advocate an approach in which the system is able to notify applications if a QoS equation cannot be satisfied. It is then the responsibility of the application to adapt its behavior accordingly. This is why POLKA was designed to allow dynamic modification of QoS equations. We believe that this "feedback-adaptation" procedure is adapted to most multimedia applications since they can tolerate transient QoS degradation.

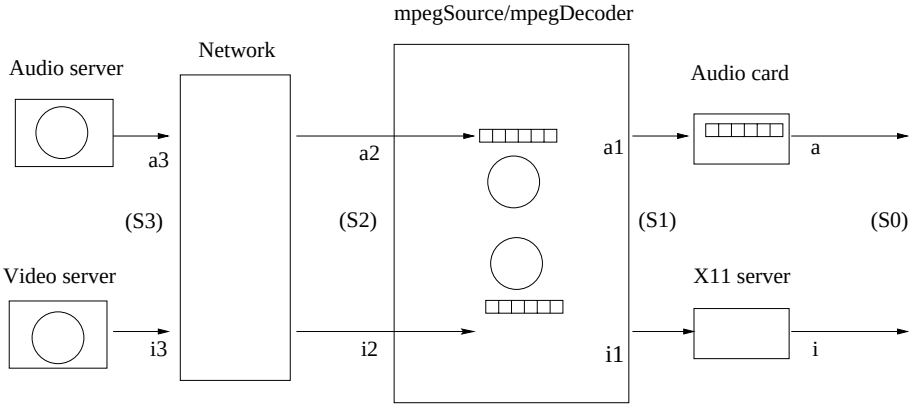

Figure 2.5: POLKA model of a distributed application

Let us now apply the component model to our case study application. Figure 2.5 gives the application data flow graph. On the right side of the graph, one component models the audio card; another models the X11 server. In the center, two components model the audio/video MPEG decoders and the network. On the left side two components model the video and the audio servers. As in Chapter 2.2, we call $(S0)$ the QoS equations corresponding to the user's QoS (provided by the audio card and the X11 server). Recall that the $(S0)$ set of equations is the following:

$$\forall\ n : \begin{cases} \tau(a,n) \ = \ n * 0.125\ ms \\ 80\ ms \leq \tau(i,n+1) - \tau(i,n) \leq 120\ ms \\ -40\ ms \leq \tau(Ha_{pa}, n*800) - \tau(i,n) \ \leq 40\ ms \end{cases}$$

$(S1)$ denotes the QoS equations required by the audio card and the X11 server that are sufficient to enforce $(S0)$ (see Figure 2.5). $(S1)$ is deduced from $(S0)$, the audio card QoS contract and the X11 server QoS contract. $(S1)$, corresponds to the QoS provided by the decoder component. We can similarly deduce $(S2)$, the QoS required by the decoder component, from $(S1)$ and the decoders component specification. Finally, $(S3)$ is deduced from $(S2)$ and from the network component specification. $(S3)$ is the QoS required by the system that is sufficient for the system to deliver the user's QoS $(S0)$.

This backward computation of QoS constraints from $(S0)$ to $(S3)$ is automatically done by the POLKA platform. In practice, the platform uses a library of the most frequently used components. In order to support QoS specifications reuse, parameterized components and component composition operations are provided.

Once the temporal model of the application is built, the flow events (events a, $a1$, $a2$, etc.) must be mapped to object events (IE, IR, RE and RR). This mapping binds QoS specification

to the functional specification of the application. For instance, in our case-study application, the $a3$ event (see Figure 2.5) is mapped to the $putAudioFrame.IE$ event and the $a2$ event to $putAudioFrame.IR$ event.

This last operation will allow the POLKA scheduler to decide which QoS equations apply to which method invocations.

In the next chapter we discuss the goals and the architecture of the POLKA middleware platform.

Chapter 3

Polka middleware platform

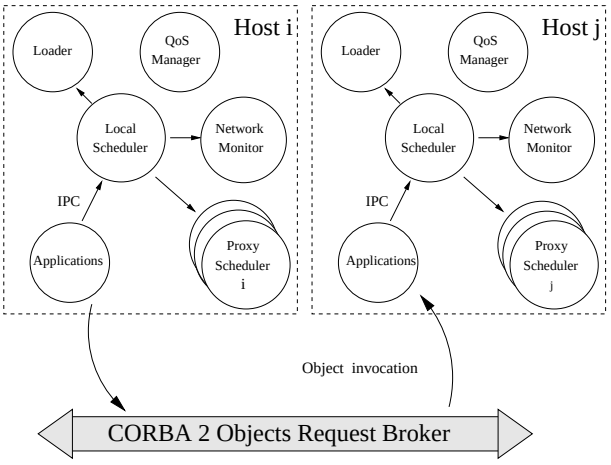

Figure 3.1: Platform architecture

As represented on Figure 1.1, POLKA takes as inputs the application specification (currently CORBA objects), a set of QoS equations, a data flow graph corresponding to the multimedia system specification and component specifications. POLKA then automatically schedules the application in order to meet the QoS constraints and provides monitoring information. Monitoring information is useful to detect when QoS constraints cannot be met (because of lack of resources) and to evaluate the temporal behavior of components when it is not known prior to execution.

The current platform is built on top of the CORBA omniORB2 Object Request Broker (ORB) [LP98]. The platform is comprised of a daemon, an IDL compiler and an QoS compiler. The IDL compiler was modified to generate stubs and skeletons instrumented with calls to the POLKA scheduler. These calls are meant to block and wake up the application threads in order to meet the QoS constraints. The QoS compiler helps designers at application modeling time. It provides tools to manage components library (e.g. making components compositions, etc). The daemon is a set of CORBA objects: the QoS manager, the loader, the local scheduler, the network monitor and the proxy schedulers (see Figure 3.1).

The **QoS manager** object reads QoS and component specifications and deduces the QoS equations each local scheduler is responsible for.

The **loader** object initializes data structures needed by the local scheduler to store scheduling information in memory.

The distributed application global scheduling is done through local schedulers cooperation.

Each **local scheduler** computes deadlines and release times according to its set of QoS equations. For example, assuming our case-study application is distributed on two hosts, a source host and a sink host, the "sink" host (resp. "source" host) will schedule its tasks according to the ($S1$) (resp. ($S3$)) set of equations.

The application is decomposed into tasks delimited by method invocations corresponding to the IE, IR, RE and RR events. These events are also the ones appearing in the QoS equations. The local schedulers use them to allocate deadlines and release times to the application tasks to be scheduled. Local scheduling follows an EDF *(Earliest Deadline First)* policy [LL73]. We do not explain the scheduling algorithm any further (see [DSH98, Leb98] for a detailed explanation of it).

Distributed scheduling is done using **proxy schedulers** to collect information from remote sites. These proxies provide a local view of remote scheduling information. On each host, there is a proxy scheduler corresponding to each remote scheduler.

The last object of the POLKA daemon is the **network monitor**. It is responsible for gathering network state information such as round trip time, packet lost ratio, jitter. The information collected by the network monitor depends on the properties of the underlying network. For instance, if the underlying network provides isochronous communication services, the network monitor does not need to collect round trip time during execution time (because it is known). Conversely, if the network does not have services with guaranteed temporal properties, the network state is computed through monitored information (e.g. using RTP/RTCP for IP networks).

Monitoring information is used by proxy schedulers to evaluate remote scheduling information.

Note that proxy schedulers do not use a global clock to provide remote scheduling information to the local scheduler: scheduling information received from a remote host is translated in the local clock time using the round trip time provided by the network monitor [SD98].

The use of the CORBA technology is a first step towards the portability of distributed multimedia applications. It is, however, not completely satisfactory. In particular, automatic scheduling of the targeted applications may require specific non standard services from the operating system. For example, a thread scheduling service is provided by many operating systems. Most of them implement POSIX threads, but unfortunately, some of them are not fully compliant with the POSIX standard.

To overcome this problem, a virtual machine layer has been inserted between POLKA and the underlying operating system. This layer is a set of *inlined* C++ classes. Abstractions provided by this layer are threads, synchronization tools, inter-process communication mechanisms and timers. We have developed a virtual machine layer for LINUX and Solaris systems.

Chapter 4

Performance evaluations

Let us now evaluate the POLKA overhead and POLKA effectiveness in meeting specified temporal constraints. The measurements are done using the case-study application. Notice that during the experiments MPEG frame decompression and presentation are simulated: instead of calling MPEG decoders, the application loops during 2.11 ms for each audio frame (resp. 55.24 ms for a video frame) - these durations are the actual MPEG decompression and presentation times measured when running the application. The application is ran first without POLKA and then with POLKA. Intra and inter-flow synchronization delays are measured. The application is ran on two LINUX computers (Pentium 200 MMX with 64 Mb of memory) connected by a 10Mbits/s Ethernet. The audio frame size is 627 bytes and the video frame size is about 4096 bytes.

Figure 4.1 summarizes the measurements done. The first curve shows the delay between the date at which the audio frames are effectively delivered and the theoretical date (time deduced from the QoS equations). With POLKA, as specified in the QoS equation, we observe a variation of 64 ms around the theoretical date (corresponding to the clock tick, see QoS equations ($S1$)). Without POLKA, frames are displayed too early (the curve drops down very fast).

The second curve measures lip-synchronization (delay between the display of a video frame and output of the corresponding audio frame). The time interval of 80 ms around the x axis corresponds to the delay authorized by QoS equation ($S0_3$). Again, without POLKA since audio frames are output too early, no lip-synchronization can be provided. With POLKA, frames are displayed according to QoS specifications.

Let us now evaluate the POLKA overhead. This overhead is mainly due to the time needed by the scheduler to compute task deadlines and release times from QoS equations and time taken by the proxy schedulers to share scheduling information with remote schedulers. Remember that all these treatments are made during method invocations since scheduler calls are inserted at compile time in stubs and skeletons[1]. On LINUX, we measured an overhead of 2,8 ms per method invocation for the distributed application. For a centralized version of the application, the overhead measured was 1.5 ms per method invocation (note that it was about 840 μs when running over a Solaris 167MHz UltraSparc 1 processor with 128 Mbytes of memory). Compared to the average time to uncompress a video frame (55.24 ms), the POLKA overhead stays reasonable (about 5 percent).

[1]However, there is no need to compile the application if the designer changes QoS description since QoS information is not hard-coded in the application source.

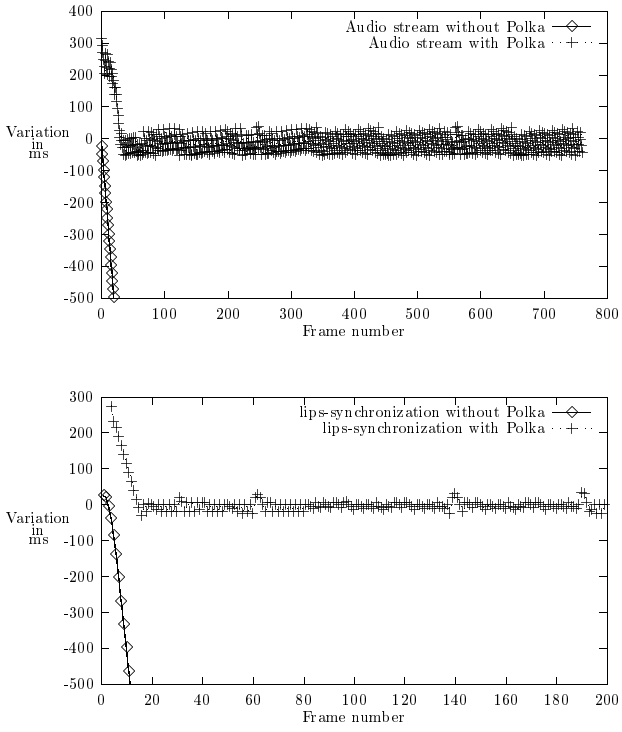

Figure 4.1: Measurements

Chapter 5

Related work

The POLKA project involves issues related to the scheduling of multimedia applications, the specification of temporal constraints, the modeling of multimedia systems, and middleware platforms for the support of multimedia applications.

A first approach to the specification and scheduling of multimedia applications is to borrow from the real-time system field. The periodic task model is often used [LL73]. Tasks are supposed to be independent and tasks execution times are known a priori. To schedule these tasks, two families of scheduling algorithms have been proposed: RM (*Rate Monitonic*) and EDF (*Earliest Deadline First*) [SSNB95, GRS96].

RM statically assigns a priority to each task (depending on its period: the shorter the period the higher the priority); EDF dynamically assigns a priority to each task (depending on its deadline).

Mercer et al have proposed a processor capacity reserve mechanism in which tasks reserve a percentage of the processor capacity [MST94]. The task model used is similar to the Liu and Layland task model. Tasks are scheduled with EDF or RM. Processor reservation is made based on the percentage of processor capacity requested and the task period. The system verifies that the tasks scheduled do not exceed the capacity reserved. A feedback mechanisms allows the application to adjust its resource reservation.

If solutions using classical scheduling algorithms are well suited to some static multimedia applications they often lead to resource under-utilization. In addition, they cannot support dynamic applications whose resources are not known a priori or can change at runtime.

Jeffay proposed an approach dedicated to multimedia applications [JB95]. Like us he uses a data flow graph to model the multimedia system. Each node in the graph models a task or a resource. A model called "rate-based execution" (RBE) is introduced. In this model, an "arrival rate" and a deadline are associated to each task. The arrival rate model is more general than the Liu and Layland one; here, each task is activated x times during a time interval y; there is no hypothesis on the time at which the x activations arrives in the y time interval. This model has a lot in common with the LBAP model used in the DASH system [And93]. Only the sink nodes in the data flow graph define their task arrival rate. For all preceding nodes, the arrival rate is computed from the arrival rate and the execution time of the next node in the graph. The RBE model was tested in the YARTOS operating system [JSS92]. YARTOS includes resource reservation services. Resource required by each new task is reserved to provide guarantees.

In the Rialto project temporal constraints are specified in the application code [JRR97]. At design time, the application is splitted into portions of code (this is also what we do in POLKA).

Temporal constraints (deadline, release time and criticity) can be attached to each portion of code. When the portion of code is submitted for execution, the scheduler takes the constraints into account to predict if temporal constraints can be met or not. It is the responsibility of the application to adapt its requirement to the available resources. The scheduling is deadline oriented (based on EDF).

The solution developed in SMART is similar to the Rialto one. Temporal constraints (deadlines) on portions of code are specified in the application [NL97]. The execution times of the portions of code are needed. The scheduler provides an upcall mechanism to call a function in the user address space when an application temporal constraints cannot be met. This function is meant to modify the application resource requirements. SMART can support applications with and without temporal constraints. Tasks are scheduled according to their **importance** and their deadline. To allocate the processor, the scheduler chooses the tasks with the greatest importance and gives the processor to the task with the earliest deadline (if a task with temporal constraint exists; otherwise all tasks with the same importance share the processor).

We have mentioned several projects that address QoS specification and task scheduling of multimedia applications. They often use very simple task models. In POLKA, the specification model is more general and can express most of the temporal constraints seen above (for instance, it should be easy to show how periodic, aperiodic and sporadic task can be modeled in POLKA). Our model does not require task execution times to be known a priori and task dependencies are automatically taken care of through the data flow graph. Finally, in POLKA it is easy to change the QoS equations to allow the designer to adapt QoS requirements to available resources.

Regarding the platform aspect, several efforts have been made to propose object request brokers adapted to applications with temporal constraints and in particular to multimedia applications.

The OMG *(Object Management Group)* real-time special interest group has made propositions for a real-time ORB. They are implemented in TAO, an object request broker that provides real-time services [GLS00]. TAO uses the classical real-time RM and EDF algorithms. TAO is better suited to the support of applications with static constraints than to applications with dynamic constraints or variable resource needs.

The Retina European project has specified an ORB that supports stream services as well as QoS extensions to CORBA. Jonathan is an ORB that implements the Retina specification [DHDtS98]. Other ORBs such as DIMMA and QuO propose flexible frameworks for QoS support [DFH+98, ZBS97]. The POLKA daemon and IDL compiler could be easily adapted to any of these ORBs. POLKA solution is similar to the QuO propositions. However, our solution is not CORBA specific and can be applied in each system where events can be monitored. Futhermore, QuO scheduling services are provided by TAO : contrary to POLKA, this platform does not automatically translate high level QoS information into scheduling directives.

GOPI *(Generic Object Platform Infrastructure)* is a multimedia platform designed to offer all the services needed by a multimedia middleware [Cou99]. It was design to be CORBA compatible. GOPI corresponds to an effort similar to the one we made in designing the virtual machine layer.

Chapter 6

Conclusion

In this paper, we have presented a model and a middleware to automatically schedule a distributed multimedia application based on an object oriented specification of the application, a data flow graph of the system, component specifications and a set of QoS equations corresponding to the constraints to be met.

Our actual implementation uses a CORBA ORB running over LINUX and Solaris. This choice has strongly simplified the implementation of the current POLKA release, but, algorithms and solutions presented in this paper can be applied to any other distributed system where events can be monitored. Measurements have shown POLKA effectiveness. However, the measurements performed on the case-study application have shown that POLKA overhead is significant. This overhead is mainly composed of local communications and synchronization operations (these operations cost about 85% of the overhead on Linux and 75% on Solaris). Indeed, general purpose operating system do not export sufficient scheduling information to design efficient user-level scheduler. To achieve optimal efficiency, POLKA should be ported to better suited execution environment (both ORB and operating system). In this context, experiments over Jonathan (a flexible object request broker promoted by France Telecom R&D) are planned.

The work presented herein can be extended in several ways. First, QoS constraints used in this paper are deterministic. Consequently, the applications are sometimes over-constrained [BCM+96]. In addition, most execution environments do not have a fully deterministic behavior and some multimedia applications need a way to express stochastic constraints (e.g variable bit rate traffic specifications). This is why we are now studying probabilistic constraints.

Second, in the work presented here, the applications are scheduled in order to meet the QoS constraints provided that enough resources are available. It is the task of the application to adapt its requirements to the available resources. We think this is an advantage over other approaches since it relieves the application developer from the difficult task of evaluating resource needs (and in particular task execution times). It is also particularly adapted to dynamic multimedia applications (where requirements are difficult to evaluate a priori and QoS constraints can be varied at runtime).

However, POLKA does not preclude the use of resource reservation services. We are currently studying the use of such services for network components such as ATM that can provide QoS guarantees [Dri00].

Chapter 7

Acknowledgments

This work was supported by the DTL/ASR team of France Telecom R&D.

Bibliography

[And93] D. P. Anderson. MetaScheduling for Distributed Continuous Media. *ACM Trans. on Computer Systems*, 11(3):226–252, 1993.

[BCM+96] V. Baiceanu, C. Cowan, D. McNamee, C. Pu, and J. Walpole. Multimedia Applications Require Adaptive CPU Scheduling. Workshop on Resource Allocation Problems in Multimedia Systems, Washington DC, December 1996.

[BS98] G. Blair and J. B. Stefani. *Open Distributed Processing and Multimedia.* Addison-Wesley, 1998.

[CAH96] A. Campbell, C. Aurrecoechea, and L. Hauw. A Review of Quality of Service Architectures. pages 173–194. University of Columbia, in Proceedings of the 4th international IFIP Workshop on Quality of Service, Paris, March 1996.

[Cou99] Geoff Coulson. A configurable multimedia middleware platform. *IEEE Multimedia*, pages 62–76, January-March 1999.

[DFH+98] D. I. Donaldson, M. C. Faupel, R. J. Hayton, A. J. Herbert, N. J. Howarth, A. Kramer, I. A. MacMillan, D. D. J. Orway, and S. W. Waterhouse. DIMMA - A Multi-Media ORB. pages 141–156. MIDDLEWARE'98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, 1998.

[DHDtS98] B. Dumant, F. Horn, F. Dang-tran, and J. B. Stefani. Jonathan : an Open Distributed Processing Environment in Java. pages 173–190. MIDDLEWARE'98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, 1998.

[Dri00] B. Driss. Support of multimedia applications by a corba and atm based distributed system. Master's thesis, ENST Paris, January 2000.

[DSH98] I. Demeure, F. Singhoff, and F. Horn. Automatic Scheduling of a Dynamic Multimedia Applications with Polka : a Case Study. pages 15–19. Fourth IEEE Real-Time Technology and Applications Symposium (RTAS'98), WIP, Denver, Colorado, USA, June 1998.

[GLS00] C. D. Gill, D. L. Levine, and D. C. Schmidt. The Design and Performance of a Real-Time CORBA Scheduling Service. *To appear in the International Journal of Time-critical Computing Systems*, April 2000.

[GRS96] L. George, N. Rivierre, and M. Spuri. Preemptive and Non-Preemptive Real-time Uni-processor Scheduling. INRIA Technical report number 2966, 1996.

[ISO95] ISO. Press Release, 29th Meeting of JTC 1/SC 29/WG 11. number 1110, March 1995.

[JB95] K. Jeffay and D. Bennett. A Rate-Based Execution Abstraction For Multimedia Computing. *In Lectures Notes in Computing Science, T. D. C. Little and R. Gusella, Springer-Verlag, Heidelberg*, 1018:64–75, April 1995.

[Jef92] K. Jeffay. On Kernel Support for Real-Time Multimedia Applications. In *Proceedings Third IEEE Workshop on Workstation Operating Systems*, pages 39–46, April 1992.

[JRR97] M. Jones, D. Rosu, and M. Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. 16th ACM Symposium on Operating Systems Principles in Saint-Malo (SOSP'97) - France, October 1997.

[JSS92] K. Jeffay, D. L. Stone, and F. D. Smith. Kernel Support for Live Digital Audio and Video. *Computer Communications*, 15(6):388–395, August 1992.

[Leb98] L. Leboucher. *Algorithmique et Modélisation pour la Qualité de Service des Systèmes Répartis Temps Réel.* Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications de Paris, septembre 1998.

[LL73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[LP98] S. L. Lo and S. Pope. The implementation of a High Performance ORB over Multiple Network Transports. pages 157–172. MIDDLEWARE'98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, 1998.

[MNCK99] S. Mitchell, H. Naguib, G. Coulouris, and T. Kindberg. A qos support framework for dynamically reconfigurable multimedia applications. pages 17–30. in Proc. of the Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems, Helsinki, Finland, June 1999.

[MST94] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Applications. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, May 1994.

[NL97] J. Nieh and M. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. 16th ACM Symposium on Operating Systems Principles in Saint-Malo (SOSP'97) - France, October 1997.

[NNH93] J. Nieh, J. N. Northcutt, and J. G. Hanko. SVR4 UNIX Scheduler are unacceptable for multimedia applications. In *Lecture Notes in Computer Science, Springler-Verlag, Proceedings of the 4th International Workshop on NOSSDAV*, volume 846, pages 49–60, Lancaster, U.K., November 1993.

[SBC+96] J.B. Stefani, G.S. Blair, G. Coulson, M. Papathomas, P. Robin, F. Horn, and L. Hazard. A programming model and system infrastructure for real-time synchronization in distributed multimedia systems. *IEEE Journal on selected areas in communications*, 14(1):249–263, January 1996.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC1889 : RTP : A Transport Protocol for Real-Time Applications. Network Working Group, pages 1-75, January 1996.

[SD98] F. Singhoff and I. Demeure. Environnement d'exécution pour les applications réparties sous contraintes temporelles : une solution CORBA-RTP. pages 53–57. 10'ième Rencontres Francophones du Parallélisme (RENPAR'10) - Strasbourg, juin 1998.

[SN95] R. Steinmetz and K. Nahrstedt. *Multimedia : Computing, communicating and applications*. Prentice Hall, innovative technology series, 1995.

[SNH97] S. Siewert, G. Nutt, and M. Humphrey. Real-Time Parametrically Controlled In-Kernel Pipelines. Third IEEE Real Time Technology and Application Symposium (RTAS'97), Work-In-Progress, Montreal - Canada, June 1997.

[SSNB95] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo. Implications of Classical Scheduling Results For Real-Time Systems. *IEEE Computer*, 28(6):16–25, June 1995.

[Ste93] J. B. Stefani. Computational Aspects of QoS in an object-based, distributed systems architecture. 3rd Workshop on Responsive Computer systems, Lincoln, NH, USA, September 1993.

[Vet95] R. J. Vetter. ATM concepts, architectures, and protocols. *Communications of the ACM*, 38(2):30–38, February 1995.

[ZBS97] John A. Zinky, David E. Bakken, and Richard Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1), 1997.