PERFORMING REAL-TIME SCHEDULING IN AN INTERACTIVE AUDIO-STREAMING APPLICATION

Julien Cordry, Nicolas Bouillot, Samia Bouzefrane

Laboratoire CEDRIC, Conservatoire National des Arts et Métiers, 292, rue Saint Martin, 75141 , Paris, France
Email:julien.cordry@auditeur.cnam.fr,{ bouillot, samia.bouzefrane}@cnam.fr

Key words : multimedia application, real-time scheduling, real-time system, distributed system.

Abstract: The CEDRIC and the IRCAM conduct since 2002 a project entitled "distributed orchestra" which proposes to coordinate on a network the actors of a musical orchestra (musicians, sound engineer, listeners) in order to produce a live concert. At each site (musician), mainly two components are active: the sound engine (*FTS*) and an auto-synchronisation module (*nJam*), two modules which must treat audio streams in real time and exchange them via the network. These components were first made to run under the Linux environment, where the available schedulers are imposed. For this purpose, we choose to use Bossa, a platform grafted on the Linux kernel in order to integrate new real-time schedulers.

1. INTRODUCTION

The distributed virtual orchestra is the result of a co-operation between the research laboratories of the IRCAM and the CEDRIC-CNAM [Bou., 2003; Loc. et al., 2003]. The aim of this project is to provide means to connect musicians that would play in real time via Internet. The current version runs over a multicast network. The musicians communicate via PCM audio streams, a constraint allowing a high quality hearing of the different audio streams. Each musician broadcasts towards the other musicians the music which he/she plays and hears the music that he/she has just played after a constant latency. Our purpose is to schedule the different processes of each site, particularly those generated by the sound device and the self-synchronization by using a suitable real-time scheduling technique in order to improve the global performances of the application. We have chosen to use Bossa, an event-based framework for process-scheduler development. This choice is motivated by the following points:

- to use a "Bossa" scheduling for "Linux native" applications, both Linux and application must be modified. However, a minimum of insertion of code (three lines of code by process to attach it to a specific scheduler) are required in an application code to benefit from BOSSA schedulers. Linux is automatically modified by Bossa: a set of rewriting rules are applied to the sources of the Linux core. This way allows us to test and configure easily new schedulers in an environment (Linux) where many applications are available. One could have used real-time Linux such as RTAI[1] but this requires the complete rewriting of the application considering the particular structure of the real-time tasks and the particular libraries to include.

- We will be able to use a real-time scheduler for the management of the processes of our multi-media application, rather than those of Linux (SCHED_FIFO or SCHED_RR).

The paper is organised as follows. In section 2, we describe the characteristics of our multi-media application, the distributed virtual orchestra. In section 3, we present the Bossa platform and we show how it can integrate new scheduling policies. In section 4, we determine the application processes which need to be scheduled in real time, by defining a scheduler hierarchy. Before concluding in section 6, section 5 presents our experiments, where we show that BOSSA helps our application to meet the timing constraints.

2. THE DISTRIBUTED VIRTUAL ORCHESTRA

The free-software team of IRCAM and the multimedia research team from CNAM-CEDRIC conduct a project since 2002 named the "distributed virtual orchestra". The aim of this project is to provide means for musicians to play across the Internet in real time (see Figure 1) [Bou., 2003; Loc. et al., 2003].

The application constraints are as follows:

- the musicians are physically separated but must play virtually "together" in real time.

- the sound engineer must be able to adjust in real time the audio parameters of the various sound sources (e.g., to add reverberation effects, etc).

[1] Real-Time Application Interfaces, developed in Dipartimento di Ingeniera Aerospaziale, Politecnico di Milano of Prof Paola Mantegazza (http://www.aero.polimi.it/~rtai/applications /)

- the public must be able to virtually attend the concert, either at home by a standard mechanism of audio/video streaming, or in a room with a dedicated installation.

In this paper, we are interested in the part concerning the musicians only, since it is a critical part in term of interactivity. Our application uses jMax, a visual programming environment dedicated to interactive real-time music and multimedia applications [Déch., 2000]. jMax has been developed by the IRCAM. It is composed of two parts: FTS for "faster than sound", a real-time sound processing engine and a graphical user interface which allows to add, remove or connect components that exchange audio samples or discrete values. Some examples of components available in jMax are the inputs/outputs of the sound device, the arithmetic operations and the digital audio filters. Since jMax is often used to make audio synthesis, it has an interface with the operating system, ALSA (for Advanced Linux Sound Architecture) for Linux.

As we are close to virtual-reality conditions, the sound quality, the feeling of presence, as well as synchronism among musicians are crucial conditions. For this reason, the technology developed for the distributed concert uses a non compressed sound (PCM samples at 44100Hz 16 bit, corresponding to the quality of an audio CD). Additionally we use the multicast with the RTP protocol [Sch. et al., 1998] for the communication among musicians. For the feeling of presence, during our experiments, a videoconference software allowed remote visualisation among the musicians.

Usually, the musical interaction (all musicians in the same room) is enabled thanks to a common perception among musicians: the sound and the visual events are perceived instantaneously, simultaneously and with a sound quality limited by the capacities of the human ears and eyes.

During networked performances, we can provide the better quality for the sound, but we cannot provide instantaneity. In fact, we estimate that 20ms is the threshold above which the human ear perceives the shifts. For this reason, we ensure a global simultaneity among musicians thanks to a synchronization mechanism described in [Bou., 2003; Bou. et al., 2004] and implemented inside nJam (for network Jam), a pluggin of jMax. This synchronization ensures that the return of the overall mix of the music is identical for all the musicians. nJam computes the diffusion of the sound through multicast with RTP, the synchronization of the audio streams, and the shift between the musicians. Thus, the musicians specify a tempo, as well as the shift in musical units (a beat, an eighth note, a sixteenth note, etc). This parameter enables them to have a shifted feedback, which is synchronized and matches the beats of the music they are playing on their instrument.

We can extract some constraints for the operating system and the network: each instance of nJam will send on the network only one audio stream and will receive N from them (if N sites are involved). Then, FTS manages at least one component corresponding to an input (microphone or instrument) and one component for each output of the sound device. Within the RTP protocol, the isochronism of the audio data is ensured by a time-stamping that corresponds to the number of audio samples. Additionally, each site is controlled by its own clock, that came from the local sound device. At each site-clock tick, a sample of 16 bits is produced and a sample coming from each source is consumed. Thus, the constraints of end-to-end temporal delivery are crucial, either for the network part or for the system part FTS/ALSA.

In this paper, we focus on the schedule of various application components by using a real-time scheduling technique.

3. BOSSA

The distributed virtual orchestra is an application written in C whose execution environment is the Linux system. From a system point of view, this application is confronted with the resource sharing, in particular in terms of access to the processor and to the peripherals (network device and sound device). However, a guaranteed periodic access to these resources is necessary. In this context, the use of a real-time Linux system (such as RT-Linux[2] or RTAI) would enable us to try out scheduling policies not available on the traditional Linux system. Nevertheless, to profit from these policies, the target application must respect the structure of the real-time tasks and include the particular function calls of the library of real-time Linux. To avoid modifying the source code that deals with the logic of the application, we choose to use rather Bossa. Indeed, to our knowledge, it is the only platform which can integrate real-time schedulers into the Linux core, thus allowing Linux processes to be scheduled according to a scheduling policy integrated in Bossa. The only modification to perform on the source code of the process is the insertion of a function call used to attach the process to the selected scheduler.

Before presenting the Bossa[3] platform, we describe the Bossa DSL (domain-specific language) used to implement new scheduling policies.

3.1 The Bossa DSL

The technique used by Bossa to integrate new scheduling policies in an existing operating system is the use of a dedicated language (DSL: Domain Specific

[2] http://www.fsmlabs.com

[3] http://www.emn.fr/x-info/bossa

Language). A DSL is a programming language providing high-level abstractions appropriate to a given domain and permitting scheduling-specific verifications and optimizations.

Each scheduling policy in Bossa is implemented as a collection of event handlers that are written in Bossa DSL and translated into a C file by a dedicated compiler. A Bossa scheduling policy declares: (i) a collection of scheduling-related structures to be used by the policy, (ii) a set of event handlers, and (iii) a set of interface functions, allowing users to interact with the scheduler.
Table 1 shows some of the declarations made by the Bossa implementation of the Linux 2.2 policy. The `process` declaration lists the policy-specific attributes associated with each process. As reflected by the `policy` field, the Linux 2.2 scheduling policy manages FIFO and round-robin real-time processes, as well as non real-time processes. The other fields of the `process` structure are used to determine the current priority of the associated process. Finally, the `ordering_criteria` declaration specifies how the relative priority of processes is computed. Table 1 shows also examples of event handlers of the Linux2.2 policy. For example, the event handler `block. *` moves the target process to the blocked process whereas the event handler `unblock. *` moves the target process from the blocked queue to the ready queue.

3.2 From the Linux kernel to Bossa

The developers of Bossa examined the problem of operating system (OS) evolution in the context of adding support for scheduler development into the Linux OS kernel. The goal of Bossa is to simplify the design of a kernel-level process scheduler so that an application programmer can develop specific policies without expert-level OS knowledge [Law. et al., 2002; Bar. et al., 2002]. A Bossa scheduling policy is implemented as a module that receives information about process state changes from the kernel via event notifications and uses this information to make scheduling decisions.

Table 1: Declarations of the Linux 2.2 policy

Declarations	Event Handlers
`type policy_t = enum {SCHED_FIFO, SCHED_RR, SCHED_OTHER}`	`On block.* { e.target => bocked; }`
`process = { policy_t policy; int rt_priority; time priority; time ticks; system struct ctx mm; }`	`On unblock.* { if (e.target in blocked) { e.target => ready; } }`
`ordering_criteria = { highest rt_priority, highest ticks, highest ((mm==old_running.mm)? 1:0)}`	

Preparing a kernel for use with Bossa requires inserting these event notifications at scheduling points throughout the kernel. The evolution of the Linux kernel to support Bossa is rather complex, for various reasons. First, Bossa would like to be used across the many sub-series of Linux releases, which do not contain new algorithms. A solution based on patches is not sufficient because the line numbers of the scheduling points as well as the code surrounding these points can differ across releases. Second, some of the changes required to support Bossa depend on control-flow properties. Detecting such properties by hand is error-prone even when considering a single version of Linux. Finally, making any changes by hand across multiple files of a large piece of software (Linux currently amounts to over 100MB of source code), is tedious and error-prone. Hence, the rewriting principle has been used to implement a crosscutting functionality that contains a collection of code fragments and a formal description of the points at which these fragments should be inserted into the target application. This functionality uses temporal logic to precisely describe code insertion points and thus resolve the context-sensitivity issue.

An example of a rewrite rule is the following as it is described in [Aber. et al., 2003]:

```
n:(call try_to_wake_up))
=>Rewrite(n, bossa_unblock_process(args))
```

This rule matches any call to the function `try_to_wake_up`. A node matching this pattern is given the name `n`. The use of `Rewrite` indicates that the call to `try_to_wake_up` is replaced by a call to `bossa_unblock_process`. The function `wake_up_process` shown below illustrates the effect of applying this rule.

```
wake_up_process(struct task_struct * p) {
#ifdef CONFIG_BOSSA
       return bossa_unblock_process
       (WAKE_UP_PROCESS, p, 0);
#else
   return try_to_wake_up(p, 0);
#endif
}
```

The Linux kernel is rewritten using over forty logical rules of a rather great complexity implemented in Ocaml and Perl via CIL (C Intermediate Language). Even with these methods which are supposed to guarantee a minimum of reliability, the error is always possible. Thus, when using Bossa with the distributed virtual orchestra, we could note the failure of a rewriting rule.

3.3. Bossa: a hierarchy of schedulers

A scheduler is a complex application since that it requires understanding the operation of multiple low-level kernel mechanisms. Ideally, to be able to implement new scheduling policies, the scheduler and the rest of the kernel must be completely distinct but perfectly interfaced.

Bossa proposes a specific abstraction level to scheduling domain. Instead of calling directly the functions of the scheduler (typically `schedule` ()), the drivers call a system of events. Indeed, the Bossa framework replaces scheduling actions in the kernel, such as the modifying of a process state or the electing of a new process, by Bossa event notifications. Event notifications are processed by Bossa run-time system (RTS) (see Figure 2) which invokes the appropriate handler defined by the scheduling policy.

To resolve the problem of coexistence of real- time and non real-time programs, Bossa introduced the concept of hierarchy of schedulers. A *process scheduler* is a traditional scheduler that manages the processes in order to allocate to them a processor time. A *virtual scheduler* is a scheduler that controls other schedulers. Thus, one can create a virtual scheduler with child schedulers to which it can give control according to well defined criteria (for example, priority) or according to a proportion (for example the virtual scheduler will give control once on three to the child scheduler number 1 and twice out of three to the child scheduler number 2). The system scheduler will thus have a tree form where nodes are virtual schedulers and leaves are process schedulers. The main difference between a process scheduler and a virtual scheduler is the handling of events. The Bossa run-time system sends the event to the first scheduler in the hierarchy. After receiving the event, a virtual scheduler forwards the event to the appropriate child scheduler and then updates the child scheduler state according to the result of the event treatment.

4. REAL-TIME SCHEDULING OF THE DISTRIBUTED ORCHESTRA

Many real-time scheduling algorithms are described in the literature [Cot. et al., 2002] nevertheless they are not implemented on usual (non real-time) operating systems. We have chosen Bossa because our multimedia application will continue running under Linux while using a real-time scheduling strategy for the processes.

In the remainder of this section, we will investigate the real-time processes of the distributed orchestra that must be scheduled by using a real-time scheduling policy. For this purpose, we will be interested in the FTS/jMax and nJam modules that constitute the heart of our application.

4.1 Analyzing FTS and nJam processes

As explained in section 2, at each site FTS, the audio engine, manages the jMax components like the audio inputs/outputs or nJam (the pluggin of jMax). During the initialization of FTS (when starting jMax), modules (like ALSA under Linux) will be loaded. Then, the user can define, connect and set parameters of the components via a graphical interface. The FTS engine has a loop structure(see the following code): the functions associated to the components are executed one by one (with a beat driven by the sound card). Indeed, the function `fts_sched_do_select` returns in `main_sched` the list of the functions to be executed.

```
void fts_sched_run(void)
{
while(main_sched.status! = sched_halted)
fts_sched_do_select(&main_sched);
}
```

FTS starts by analyzing the output of the graphical interface to deduce a set of dependences between FTS components. The execution of the functions associated to the components will allow audio-data exchange between components and a possible output over the sound device. This loop is critical since each function registered in FTS engine corresponds to a set of samples which must be available to the next cycle. During our experiments, the cycles were equivalent to 64 samples each, corresponding to a duration of 64/44100 seconds, i.e., 1.45 ms.

In addition to FTS, the nJam patch synchronize musicians to provide the perceptive consistency [Bou. et al, 2004]. Additionally, it keeps the isochronism from end to end by playing null sample when data come late (RTP is build on top of UDP). In this way, nJam needs periodical accesses to the sound card and the network interface nJam starts mainly a thread which loops on the reception and the sending of RTP packets until the end of connections. It is the greediest operation from the resources point of view.

4.2 The distributed orchestra under Bossa

To run the distributed orchestra under Bossa, we define a scheduler hierarchy. The development team of Bossa has already worked on multi-media applications [Conc. et al., 1998]. Indeed, they developed a version of mplayer that uses the EDF technique. This version requires to define the attachment of the application to the scheduler with a period and an execution time expressed in jiffies (CPU clock ticks). We define a tree structure with one level so that the root which corresponds to a virtual scheduler is composed of two child process schedulers: one process scheduler corresponds to the EDF version of mplayer and the other one is a traditional Linux process scheduler. The virtual scheduler is a fixed-priority based scheduler, in other words it handles two child schedulers having static priorities. In our case, the priorities are associated to the process schedulers so as to favour systematically EDF over a Linux scheduler. The following commands allow the creation of the schedulers hierarchy of Figure 3.

```
panoramix:/home/cordry # modprobe EDFu
panoramix:/home/cordry     #     modprobe
Fixed_priority
panoramix:/home/cordry # /bin/manager
Available schedulers:
0. Linux (PS, root, default)
1. EDFu (PS, not loaded, not default)
2.  Fixed_priority  (VS,  not  loaded,
default)

Default path:
Linux

Command: (the scheduler number uses)
C <P> <C> connect relative scheduler P to
child scheduler C
D <S> disconnect scheduler S
L list available schedulers
H print this help finely
Q quit

> C 2 0
int importance_10: 5
> C 2 1
int importance_10: 7
> L
Available schedulers:
0. Linux (PS, loaded, default)
1. EDFu (PS, loaded, not default)
2. Fixed_priority (VS, root, default)

Default path:
Fixed_priority - > Linux
> Q
```

All the processes will be executed by default under Linux, except for the principal loop of FTS (which makes audio computation) and the nJam thread (in charge of the emissions and receptions of RTP packets) which will be scheduled in real time.

Any process which must be scheduled under Bossa, must be attached to a scheduling policy. In the context of our application, FTS loop will be attached to EDF scheduler while specifying the worst case execution time of the loop and its deadline that is equal to its period.

To compute the execution time of FTS loop, it is necessary to evaluate the execution time of the various functions called. These functions associated with the components prepare 64 samples at each clock tick of the sound device, that is, a cycle which takes 64/44100=1.45 ms. Greater is the number of components, greater is the number of associated functions, which increases the execution time of FTS loop. In our experiments, according to the number of components defined, we limited the execution time of FTS loop to 4 jiffies (CPU clock ticks, on a modern hardware a jiffie approximates 10ms) and fixed its period to 5 jiffies.

The following code shows the modifications that were have carried out on FTS loop in order to attach it to EDF scheduler.

```
/* we include the definitions of EDFu */
 # include " user_stub_EDFu.h "


void FTS_sched_run(void)
{
int period = 5;
int wcet = 4;

/* we attach the current process to the
EDFu scheduler * /
if (EDFu_attach(0,period,wcet) < 0)
    FTS_post("Cannot   attach   (%s)\n",
strerror( errno));

while(main_sched.status! = sched_halted)
FTS_sched_do_select(&main_sched);
/*  we  loop  on  the  list  of  functions
called by FTS until the end */
}
```

Similarly, we attached the nJam thread to the EDF scheduler by assigning to the thread a period equal to 10 jiffies and a worst case execution time of 1 jiffie. The following code shows this attachment.

```
void start_routine(nJam_t * this)
{
struct timeval timeout;

pid_t my_pid;
 int wcet = 1;
int period = 10;
 my_pid = getpid();
if (EDFu_attach(0, period, wcet) < 0)
fts_post("Cannot    attach    %u    (%s)
\n",my_pid,strerror( errno));
pthread_exit(0);
}
```

5. PERFORMANCES EVALUATION

We have run the distributed orchestra application by using sound automates that generate a 16 bit PCM audio signal at a frequency of 44100 Hz (one automate on each site). The machine called "breton" has an Intel processor of 3 GHz with a memory of 1 GB and uses Linux as operating system (kernel 2.6.5). The machine "panoramix " has an Intel processor of 350 MHz with a memory of 256 MB and uses Linux with two kernels (Bossa and kernel 2.4.21). The curves we present correspond to the quantity of data stored in the buffers of nJam, each buffer corresponds to a musical source. These data are regularly consumed by FTS in order to feed the sound device. Since the production of audio samples as their consumption take place at the same rate theoretically (if we consider that the clocks of the sound cards do not derive), the quantity of data should be constant, modulo the jitter of the network. The measurements are made from the first communication between the machines, showing abrupt increasing due to the adjustment of latencies to synchronize audio streams.

Figure 4 shows the ideal situation for the machine panoramix (which runs under Bossa), i.e. when the system is not overloaded. In this case, the producer/consumer relation of the audio streams coming from panoramix and breton is correctly preserved (the curves are constant starting from the 33th second).

Figure 5 shows the case where the machine panoramix runs with a Linux system loaded thanks to a script and started at the 29th second on panoramix. From this moment, the audio samples are not heard any more at the output of the sound device, causing an imbalance in the producer/consumer relation of nJam. The curve of figure 5 shows the nJam buffer size of panoramix machine. We can see that the data coming from breton are not consumed since their quantity increases. However, the local stream remains constant, letting us assume that the data are not sent. This assumption is confirmed by the curve of figure 6, because the machine breton stops abruptly the reception of data from panoramix at the 50th second. We thus see clearly thanks to figure 5 that processor loads blocks completely the access to the sound device (FTS process) as well as the sending and the reception of the data on the network (thread nJam) .

We made the same load test with panoramix while running under Bossa. Process FTS as well as the thread nJam being scheduled with an EDF policy. In this case, in spite of the load, we can see on figure 7 that the machine panoramix is not disturbed by the load script, neither in relation with the network, nor regarding to the sound-device access. It shows that nJam meet its timing constraint, instead of an heavily loaded system.

6. CONCLUSION

The project on the distributed virtual orchestra aims to provide means to allow to remote musicians to play music via Internet. In this paper, we provide some execution guaranties, which help the application to satisfy the temporal requirement, both for the local device and for the network access.

In addition to FTS/jMax module, N. Bouillot proposed an audio-stream synchronization algorithm to provide synchronism among the musicians. This algorithm is implemented as a jMax pluggin. We wanted to show here how we proceed to schedule the various processes generated by these components by using a real-time scheduling technique in order to handle the temporal constraints of the application. We chose to use Bossa, an event-based platform which integrates easily new scheduling policies without changing the operating system. We used the concept of scheduler hierarchy defined in Bossa to schedule the real-time processes of our application according to an EDF-based technique more appropriate to the multi-media domain. Non real-time processes are handled by a traditional scheduler of Linux.

Finally, the experiments carried out show that we can test and configure specific schedulers in a widely deployed desktop environment (Linux). Thus, we argue that BOSSA allows us to test new schedulers easily with multimedia applications and with a small cost. However, the Linux kernel must be replaced by the kernel modified by the BOSSA rewriting rules. This modifications performed on Linux can be performed only by specialists.

Several search directions can be explored as a perspective to this work. Nevertheless the direction which seems to be essential to pursue this work concerns

- o first, a precise study of the parameters relating to the quality of service of the network which could influence the temporal characteristics of the real-time processes of the distributed orchestra and
- o second, the configuration of the scheduler. We have seen that the scheduler is configured with Jiffies. However, our constraints are expressed in time units which depend on the sound-device clock. Thus, we plan to modify Bossa to use system calls, as the access to the sound device, inside the scheduler configuration

7. REFERENCES

[Aber. et al., 2003] : Aberg R. A., Lawall J.L., Südholt M., Muller G. And Le Meur A.-F., "On the automatic evolution of an OS kernel using temporal logic and AOP", Automated Software Engineering, 2003.

[Bar. et al., 2002] : Baretto L. P. et Muller G., "Bossa: a language-based approach to the design of real-time schedulers", In 10th International Conference on Real-Time Systems (RTS'2002), pages 19-31, Paris, France,march 2002.

[Bou., 2003] : Bouillot N., "Un algorithme d'auto synchronisation distribuée de flux audio dans le concert virtuel réparti", RenPar'15, CFSE'2003, SympAAA'2003, pp.441-452, France, oct. 2003.

[Bou. et al., 2004] Nicolas Bouillot N. et Gressier-Soudan E.,"Consistency models for Distributed Interactive Multimedia Applications". A paraître dans Operating Systems Review. Volume 38, issue 3. Octobre 2004.

[Conc. et al., 1998] : Consel C. et Marlet R., "Architecturing software using a methodology for language development", Proc. of the 10th Intern. Symp. On Programming Languages, Implementations, Logics and Programs, Pise, Italy, pp. 170-194, 1998.

[Cot. et al., 2002] : Cottet F., Delacroix J., Kaiser C. et Mammeri Z., "Scheduling in Real-Time Systems", Wiley Ed., 261 pages, 2002.

[Déch., 2000] : Déchelle F., "jmax: un environnement pour la réalisation d'applications musicales temps réel sous Linux", Actes des journées d'Informatique Musicale, 2000.

[Law. et al., 2004a] : Julia L. Lawall, Gilles Muller, Hervé Duchesne, "language design for implementing process scheduling hierarchies", Invited application paper, in Proc. of the ACM/SIGPLAN Workshop Partial Evaluation and Semantics-Based Program Manipulation Proceedings of the 2004 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, pages 80-91, ISBN:1-58113-835-0, Italy, August 24-25, 2004.

[Law. et al, 2004b] : Julia Lawall, Gilles Muller, Anne-Francoise Le Meur, " On the design of a domain-specific language for OS process-scheduling extensions", in Proc. of the Third International Conference on Generative Programming and Component Engineering (GPCE'04), Vancouver, October 24-28, 2004.

[Loc. et al., 2003] Locher H.-N., Bouillot N. Becquet E., Déchelle F. & Gressier-Soudan E.,"Monitoring the Distributed Virtual Orchestra with a CORBA based Object Oriented Real-Time Data Distribution Service", International Symposium on Distributed Object Application, nov. 2003. Catagne, Italy.

[Sch. et al., 1998] Schulzrinne, Casner, Frederick and Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889. 1998.

Aknowledgments : we would thank Mr Gilles Muller, Professor at Ecole des Mines de Nantes (France) and a Bossa-team member, and Mr Jean-Ferdinand Susini, Associate-Professor at CNAM (Paris), for their relevant remarks that help us to improve the content of this paper.

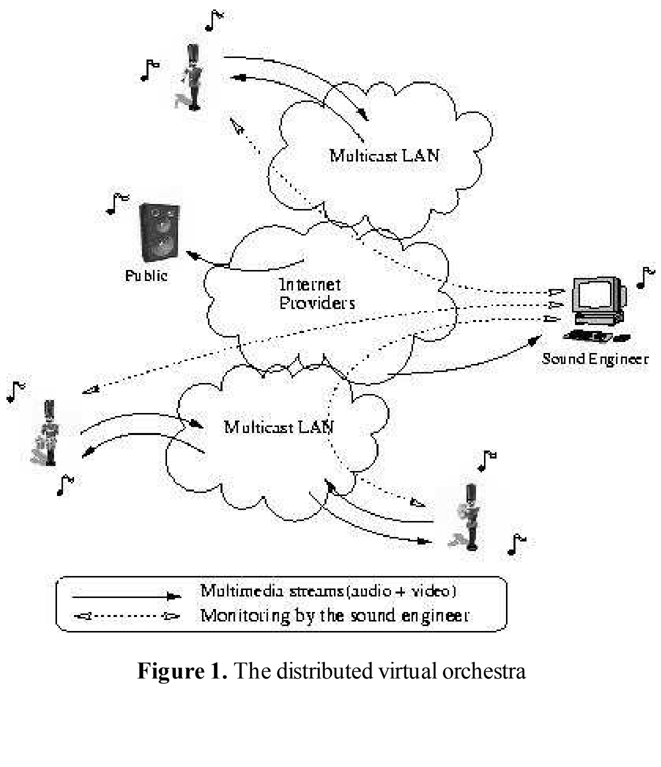
Multicast LAN
Public
Internet Providers
Sound Engineer
Multicast LAN
Multimedia streams(audio + video)
Monitoring by the sound engineer

Figure 1. The distributed virtual orchestra

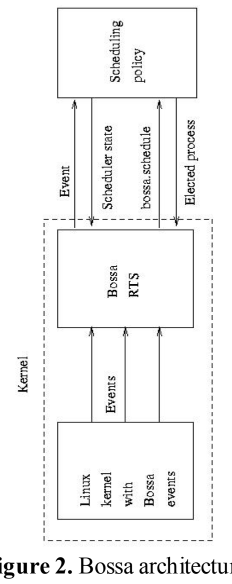

Figure 2. Bossa architecture

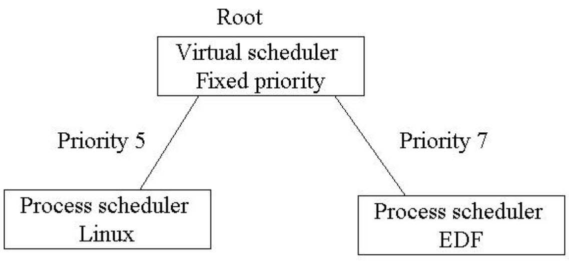

Figure 3. A scheduler hierarchy

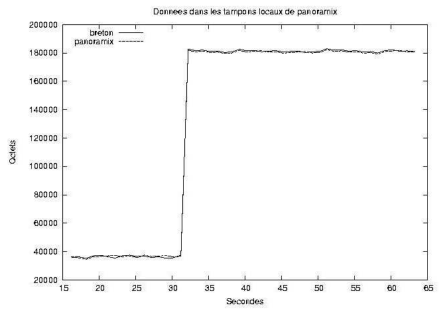

Figure 4. The behaviour of *panoramix* using Bossa and *breton* using Linux when the system is not loaded (from *panoramix* point of view)

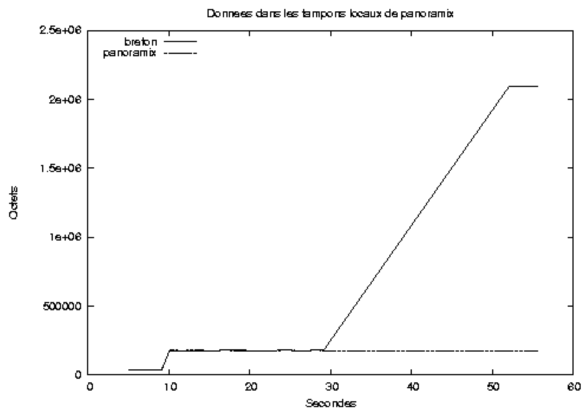

Figure 5. The buffers sizes of *panoramix* and *breton* when they run under Linux with a loaded system(from *panoramix* point of view)

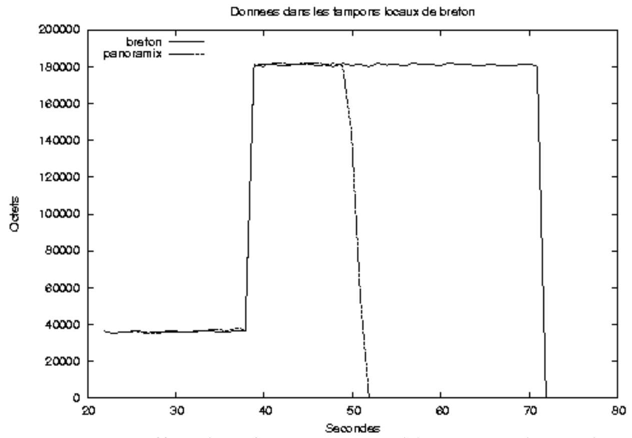

Figure 6. Buffers size of *panoramix* and *breton* running under Linux with a loaded system (from *breton* point of view)

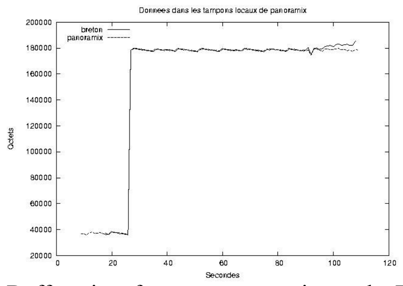

Figure 7. Buffers size of *panoramix* running under Bossa with a loaded system