# The Distributed Virtual Orchestra Project

Remy Bonafous[1], Nicolas Bouillot[1], Hans-Nikolas Locher[1], Joel Berthelin[1], Francois Dechelle[2], and Eric Gressier-Soudan[1]

[1] CEDRIC Laboratory, CNAM,
292 rue St Martin,
75 141 Paris Cedex 03, France
{bonafo_r, bouillot, locher_h, jb, gressier}@cnam.fr
[2] Opensource team, IRCAM,
1 place Igor Stravinsky,
75 004 Paris, France
francois.dechelle@ircam.fr

**Abstract.** This paper describes our distributed orchestra project and its current status. The aim of the project is to allow musicians to play across the Internet. The full Internet is too wide, and internetworking is limited to LANs, and MAN. The application based on jMax/FTS sound devices use multicast IP. The monitoring of the application is built over a real-time data service over CORBA. The music play is based on a constant latency obtained according to a self-synchronization algorithm. Each section describes the current results from experiments. The end of the paper concludes on the project and presents future works.

## 1 Introduction

Several experiments of real-time multimedia performances on Internet have already be tested. Most of them use the MIDI (Musical Instrument Digital Interface) standard. In [1], latency between the production and the consumption of sounds is not designed to be constant, so variations can be heard. In the context of piano lessons [2], a single musician is teaching while the pupils around Internet are listening. In this case, there is no need to play in a synchronized way. The team of Masataka Goto developed VirJa[3, 4], a tool of virtual session of Jazz in which someone can play music with two processors, but there is no mechanism of synchronization of audio streams specified. An experiment of remote music was born thanks to protocol RMCP used in VirJa. *Open RemoteGIG* [5] allows to play with remote musicians with MIDI streams and constant delays aligned on the tempo and the metric of the musical piece. In this system, the musicians from various places will not hear the same thing, indeed the return of what is played locally does not about latency. The example describes in [4] shows us this shift: the player improvises while listening to other player's performance that are delayed for the constant period of the repetitive chord progression (for example, a 12-bar blues chord progression). Because the progression is repetitive, the delayed performance can fit the chords. Despite the interest that such

performances represent, the MIDI protocol alleviates some constraints of PCM (Pulse Code Modulation) audio streams transmission. The descriptive format of the note in MIDI preserves bandwidth. But it also decreases the field of the transported sounds. Thus, it is difficult to make a direct analogy with the constraints raised by PCM audio streams.

The transport in real time of PCM audio streams has been tested in [6, 7]. The authors used a recording studio to sample a performance that takes place in different countries. Two experiments were tested with physically distant musicians with the Global Visual Music project [8]. The first one(*Lemma 1*), has been made on a local area network with two interacting musicians but at the same place. At the time of *Lemma 2*, the musicians were distant physically but did not hear and see each other in a synchronized way, "the piano and the percussions were analyzed on each location in order to make the information emerges in various ways at the other site". For example in the form of abstract image indicating aspects of the playing of the distant musician, such as the swiftness. This project is a cooperation between the opensource team of IRCAM, and the distributed system for multimedia research team from CNAM-CEDRIC. The aim of our distributed orchestra project is to provide means to musicians to play across the Internet in real-time. The full Internet is too wide, and internetworking is limited to LANs, and MAN, nation wide Internet will be tried in a second phase. Musicians exchange PCM audio streams. This is a constraint stated by IRCAM, and then eliminated the use of MIDI audio. Musicians multicast the music they play to each other. Latencies between musicians are kept constant. Each musician hears himself after a constant latency. They play together in a synchronized way: the constant latencies, brought by a self-synchronization algorithm, allow them to hear all sounds played by the distributed orchestra but with a constant late. This synchronization mechanism is developed as a component of the jMax[9, 10]. jMax is a visual programming environment for real time musical and multimedia applications. The overall application is monitored using an object oriented real-time data service. These features in a context of high and real-time interactivity promote this work as brand new project. As far as we know, there is no equivalent project at this time. Two enhancements will be added in a next phase. To provide a virtual concert, the public will be able to receive the concert through Internet over an ADSL network. To allow the public to hear the concert, a sound engineer will collect audio streams to perform a spatial and a traditional mixing. Spatial mixing consists in placing sources on a three dimensional space. In this way, the sound is multi-channel. Mixing will be done by remote control of audio devices or by sound processing after reception.

The paper is organized as follow. Section two describes the musicians' environment: jMax/FTS and self-synchronization. Section three presents the monitoring service. Section four shows the main features of our networking environment based on IP multicast. The last section concludes and presents future works.

## 2 Musicians in the Virtual Distributed Orchestra

### 2.1 jMax/FTS, the software sound processor

**jMax principles and interface** jMax is a visual programming environment dedicated to interactive real-time music and multimedia applications [11]. jMax is an implementation of a programming paradigm that exists in a large number of audio and music software. It can be seen as an extension of the patchable modular analog synthetizer [11]. The user builds "patches" by connecting modules together with "patch cords". Via the connections, control messages or signals are sent between the modules. These modules generally represent processing units, from simple arithmetic computations to complex transformations; they also can represent data containers like signal buffers or matrices as well as system inputs and outputs like for PCM and MIDI audio. Modules can be either primitives, called "objects", or patchers themselves, thus giving to patches a hierarchical structure. Some objects have graphical interactive behavior and are used as controllers, to change values in a patch, or as viewers, to display values computed by the patch. Figure 1 shows an example of a patch. This simple patch generates a wave using the well-known frequency modulation algorithm. In this patch, several types of object appear: control objects (for instance, the "mtof" box), signal objects (for instance, the "osc " boxes), a viewer object (the lower oscilloscope) and a graphic controller (the upper right slider).

**jMax architecture** jMax architecture can be depicted as a client/server architecture and separates the graphical user interface from the real-time sound processing engine [12, 11]. This engine, known as FTS (for 'Faster Than Sound'), is implemented in C and provides a number of services that will be detailed later. The graphical user interface that is shown in figure 1 is written in Java. This architectural choice has been driven by obvious portability and modularity considerations, but also allows to run the engine without a graphical user interface, for instance in plug-ins environments. The communication between the graphical user interface and FTS uses a very simple ad-hoc message passing protocol [13]. The transport layer can be any byte stream; current implementation uses either a Unix pipe or a TCP/IP socket (which allows to run the client and the server on different machines). An interesting consequence of this architecture is that FTS can be used with different graphical front-ends as long as they comply with the communication protocol. A client library provides a higher level access to the communication protocol. This library exists in several implementations, namely C++, Java and Python. The programmer has a maximum flexibility in the choice of the implementation language and the graphical toolkit when writing client applications. A consequence of this portability is that jMax runs on all standard platforms, namely Linux (using the ALSA audio and MIDI drivers), MacOS X and Windows. jMax is extensible: FTS public API allows the programmer to develop libraries of objects that will be dynamically loaded in FTS or in the Java graphical user interface [12].

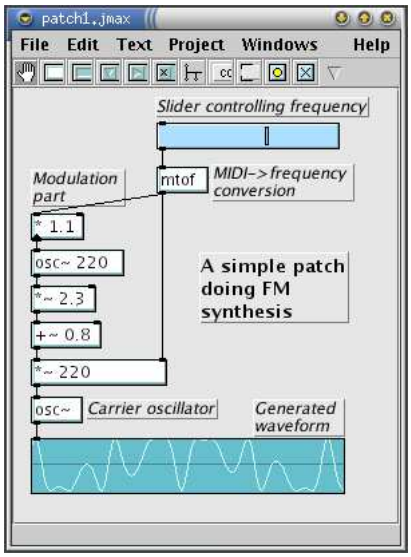



**Fig. 1.** *Screenshot of a jMax patch*

**FTS services** The FTS engine is organized around a simple object system [12]. The operators that appear in a patch are objects in the meaning of OOP: the connections end-points are message emission and receiving ports. Message receive triggers the call of a method of the receiving object class. One major simplification of this object system is that it does not provide inheritance.
A patch is then nothing more that a graph of objects, each object generating messages toward the object nodes to which it is connected in the graph. The emission of a message can be driven by a user action in the graphical user interface, the reception of a value (a MIDI note for instance) on an input object or by timed-tagged messages. The core of the FTS execution engine is then naturally a scheduler [12] that activates at each tick (a time "quantum") a number of sub-tasks that are: 1) the message interpreter 2) the timed-tagged messages execution 3) the computation of signal objects.
The computation of signal objects (objects that process or generates streams of audio samples) is optimized by a pre-compilation of the graph using a simple graph traversing and topological sort algorithm. This graph serializing produces a flat list of function calls, each function operating on vectors of samples (the typical length of this vector is 64 samples).

**jMax applications** jMax is used mainly by musicians for live performance. Using the built-in libraries that provide an extensive range of control and signal processing elements, jMax allows to build easily real-time sound generators and effects that can be controlled by standard MIDI equipments or by acoustic

instruments using score following techniques [14]. The separation of the FTS engine and the graphical front-end allows new range of applications using specific front-ends dedicated to an application together with its integration into virtual reality environments [15].

### 2.2 The distributed self-synchronization of audio streams

### 2.3 Motivation

In the context of live music, the musical interaction is done by the various visual signs and conventions predetermined on the piece of music played (for example a set of chords with a theme for a piece of Jazz). At the same time, the musical contents added by each musician inform the others on the possible evolution of the piece. For example, groups of percussions sometime use rhythmic sentences to call each other to change rythms. All these interactions are possible because the musicians are in the same environment (the scene, the part...). This environment enables them to ear each other at the same time in a synchronized way. In our context of distributed live music where musicians are remote, there is no help technology, networks and operating systems, are asynchronous Thus it is mandatory to add a mechanism that enables consistent listening.

To synchronize PCM audio streams, we choose to minimize the latency between the musicians and to compensate the jitter introduced by the network. Un-synchronized streams would have a terrible effect on the interactivity in an audio performance. In the case of Internet, we have a best effort asynchronous network without resource reservation. It is thus necessary to provision a significant amount of sounds for each musician (of a few milliseconds at several seconds) to compensate latency and jitter. Provisioning depends also on the load of the network and on the loss ratio. Some authors [16, 17] state that these variable latencies remain too significant to allow interactive distributed audio performances. Despite these difficulties, we believe to be able to allow an interaction between the various musicians. Let us recall that some kinds of musicians are able to read a partition with several measurements away of the time they are just playing. The example of reverberating acoustics of the cathedrals imposed on the Middle Ages constraints of tempo: the Gregorian songs had to be composed of long notes in order to make the texts comprehensible. Also let us recall that in a symphony orchestra, the orchestra conductor is always in advance with respect to the musicians.

In our synchronization framework for PCM audio streams, we targeted to minimize the latency between the musicians and to compensate the jitter introduced by the network. Un-synchronized streams would have a terrible effect on the interactivity in an audio performance. In the case of Internet, we have a best effort asynchronous network without resource reservation. It is thus necessary to provision a significant amount of sound samples for each musician (from few milliseconds to several seconds) to compensate latency and jitter. Provisioning

depends also on the load of the network and on the loss ratio. Some authors [16, 17] state that variable latencies remain too significant to allow interactive distributed audio performances. Despite these difficulties, we believe to be able to provide an interaction framework between the various musicians. Let us recall that some kinds of musicians are able to read a partition with several measurements in advance from the time they are just playing. The example of reverberating acoustics of the cathedrals imposed tempo constraints during the Middle Ages: the Gregorian songs had to be composed of long notes in order to make texts understandable. Also let us recall that in a symphonic orchestra, the conductor is always in advance with respect to the musicians. Then, playing/hearing late is not an impossible thing for musicians but latency should be kept constant.

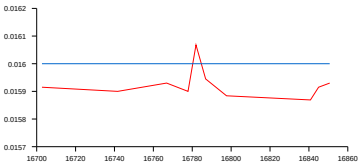
Considering these arguments, we think that we are able to provide a tool for the real time interaction of remote musicians. This tool will be associated to a dedicated mode of musical interaction, because of latencies perceived for each stream. By minimizing the latencies from beginning to end with a consistent listening, we aim to provide an interaction closest to the traditional one, e.g. musiciens in the same room. Notice that other interactions are possible, for example with latency getting allong allong with the tempo of the music. In order to make the leasting consistent beetwen each musician, we synchronise streams before playing them out, as described in [18]. In the streams, we stamp PCM units. All listener can calculate relative difference beetwen the different stamp played at the same time. After exchanging these values, each listener can adjust the playout of each streams, according to the worth receiver. At this point, all musiciens hear the same music and can play in a synchronised way.

### 2.4 Implementation

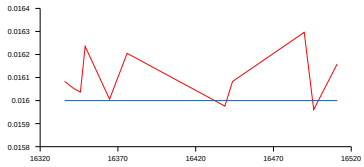
Distributed audio synchronization was deployed in the prototype of the distributed virtual orchestra. The reception of audio streams is done with the *rtpin* object developed in jMax. Audio stream transmission is implemented in the *rtpout* object. The sound samples are produced at a constant rate of 44100Hz and are consumed by the *rtpin* objects (at the same rate). The transport of audio streams is done using the RTP protocol [19]. We used the RTP library called UCL Common Code Library version 1.2.8 developed by the computer science department of University London college. Each *rtpout* object stamps the samples which it produces in the field *timestamp* of RTP. This field is incremented with each sample. The use of RTP allows the identification of the source of music (the field *ssrc*) of RTP. Each *rtpin* object consumes simultaneously a sample resulting from each *rtpout* object.
We tested the following configurations:

- two remote musicians out of two stations with a third machine sending computed sound. Only two *rtpin* objects allowed the hearing of the piece, one for each musician. This test has been made on local area network 100Mb/s

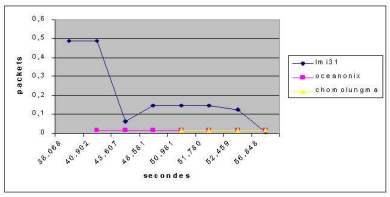
and the streams were PCM 44100Hz 16 bits streams. As the test took place on a local area network, we considered any packet losses. In this test, the two musicians were playing with the third machine in a synchronized way.
– Four sound automata, exchanging multicasts streams throw a multicast tunnel.



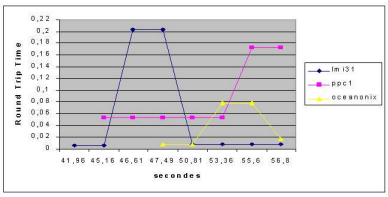

(a) *the LAN test: round trip time from first machine to first machine*




(b) *the LAN test: round trip time from thrid machine to first machine*




(c) *the multicast test: round trip time calculate for the chomolungma stream*




(d) *the multicast test: round trip time calculate for the oceanonix stream*

**Fig. 2.** *round trip time*

On the figure 2(b) and 2(a), we can see various measurements (in seconds) of tround trip times during the first experiment. The continuous line indicates the average value of the latency (16ms) measured on the machines (16ms). Note that we measured figure 2(a) the round trip time between the first machine and the first machine, as each musician have to hear his own feedback after the distributed audio streams synchronization. Some peaks and hollows appear in spite of the favorable conditions. Three streams 44100Hz in 16bits correspond to a load approximately 2,04Mbs with the RTP headers (those accounting for approximately 1,3% of stream load). This load is low considering the 100Mb/s throughput of the CNAM local area network. These great variations can be caused either by the real jitter of the network or by the non real-time scheduling of the operating systems (SuSE 7.1 and 8.0).

These measurements show that it is possible (considering delays) to play music on a local area network as if musicians were physically together. Indeed, the human ear does not perceive the shifts lower than 20ms, therefore higher than half of the round trip time measured (approximately 16ms on average). In the same configuration, we measured 6ms as the maximum shift introduced by the distributed audio synchronization. The tests between two musicians on the same network could confirm that human beings cannot ear the delay on a LAN. That let us consider the use of a such networkin recording studios, for example facilitating wiring.
On the figure 2(c) and 2(d), the round trip time exceed the 20ms that human being can perceive. The different lines represent the round trip time mesured with the RTP receiver report from lmi31, oceanonix and ppc1. lmi31 is the worst receiver, latencies might be higher when the streams is encapsulated in a multicast tunnel. The distributed self-synchronization of audio streams is here important to make musicians playing in a synchronised way.

## 3 Monitoring the Distributed Virtual Orchestra

### 3.1 Design Principles

The monitoring of the distributed virtual orchestra application is achieved using a real-time object oriented data service over CORBA. This service inherits from previous works on the IEC TASE.2 standard [20]. The main interesting properties for our concern are the virtual device abstraction, the data management model and the corresponding services. It defines a client role, the remote monitor, and a server role, the monitored device. As an application protocol in the OSI-ISO world, it can be extracted form the ISO stack as a functional specification of a real-time data exchange service. The specification has been used to implement TASE.2 over CORBA. The translation from an ISO stack toward CORBA is quite natural. This translation is limited to a subset of the TASE.2 services, basically: periodic data exchanges, and condition-based data exchanges. To provide an object oriented real time data service, we decided to adapt the TASE.2 services described previously over CORBA. ISO confirmed services become synchronous method invocations. ISO unconfirmed services become oneway method invocations or synchronous method invocations without result parameters (in our case, the two choices are allowed, it depends of specified QoS parameters in the Data Set Transfer Set).
In the distributed virtual orchestra application, the virtual device models a jMax/FTS sound automata and its associated resources. The real-time data service defines "Data Value" objects and "Data Set" objects managed by the server and subscribed by the clients. Data Value and Data Set management functions allow creation, destruction, configuration, read, write, etc of objects. Data values reference measurement points, indication points. They contain status information, analog values, and attributes. Indication points can have attributes

like timestamp, quality class, change of value counter... The real-time data service defines also Data Set Transfer Sets. They describe the way Transfer Reports must be pushed from the producer, the virtual device, toward the consumer, the remote monitor. They contain parameters defining under which conditions Data Values referencing indication points related to Data Sets are transmitted. The remote control and monitoring of sound automates use all these abstractions and their corresponding services over CORBA.

### 3.2 The Prototype

The CORBA server object on the server side supports the virtual device interface. The corresponding methods are classical invocation methods, thus they return results. The other abstractions (Data Set, Data Value, Data Set Transfer Set) are supported by the virtual device as objects implemented with the programming language, C++ in our case. The virtual device interface inherits from all basic objects interface. Local communications between the virtual device and the FTS sound processor use sockets and a library of callbacks. These mapping functions provide write operations from the virtual device to the jMax/FTS engine and vice versa. There is a CORBA server object on the remote monitor side, it supports the Transfer Report services. A generic real-time object oriented data service is specified using the CORBA IDL (which is the most popular one). This specification is freely available [21]. Figure 4. presents a short piece of the IDL specification. that deals with the Data Value object method interface. We called openTAZ[21, 22] prototype of our C++ CORBA based object oriented real-time data service over MICO. MICO [23]. Figure 7. hereafter gives an overview of the design of opentTAZ.
However, this real-time data service could be extended in a way that a producer provides the same data toward n subscribers if IP multicast and UDP (and also RTP/RTCP) were used. In a same way a client could trigger more than one server. Such extensions could be very useful to monitor the distributed virtual orchestra but need a special version of the underlying CORBA platform, that is not the case of the one we used.

The monitoring service has been used for different purposes : jMax/FTS engine activation for each entity (player based or computer based), the sound level, rtpin and rtpout objects initialization, and the "bang" of the orchestra (namely the start of the application) that means the transmission and reception of sound samples across the network. The monitoring service has been a useful tool. We plan to extend its use as the support of the metronome function, to learn the latency in the network, to initialize the parameters of the self-synchronization algorithm.

## 4 Multicast Networking

The whole set of musicians defines a group, naturally, it maps on an IP multicast group. All the musicians of the orchestra transmit and receive audio streams

```
typedef string DataValueNameType;
typedef sequence < DataValueNameType >
DataValueNameSeqType;
exception UnknownDataTypeXType { };
interface DataValueManagement {
  DataValueType
  getDataValue(in DataValueNameType name)
  raises (UnknownDataValueNameXType);
  void
  setDataValue(in DataValueNameType name,
  in DataValueType val)
  raises (UnknownDataValueNameXType);
  DataValueNameSeqType
  getDataValueNames();
  DataValueType
  getValueType(in DataValueNameType name)
  raises (UnknownDataNameXType);
};
```

**Fig. 3.** *IDL specification of Data Value Objects Interface*

on the same IP multicast address. IP multicast networking provides many advantages: 1 datagram for n receivers, factorizing traffic. We have seen that it allows also to implement self-synchronization easily. But IP multicast implies UDP, and there is no reliability nor ordering of datagrams that contain sound samples. RTP/RTCP bring the information needed to order samples, and audio allows also some losses. This section focuses on IP multicast used for the project.

### 4.1 IP Multicast routing for the distributed virtual orchestra

The first key issue was to choose an IP multicast routing protocol for the project. We use PCs that run Linux. This eased some decisions. For LAN and campus communications we were completely free of our choice. We took into account the following requirements: ? Knowing that campus routers were not able to support native IP multicast, the elected multicast routing protocol should run on Linux hosts and should use tunnels.
? According to the grouping of musicians, the group is naturally sparse. This eliminated routing protocols like DVMRP (mrouted), and also PIM-DM, or MOSPF.
? The selected multicast routing protocol must be compatible with the choice of our ISP, the french national research network RENATER.
These requirements led to the PIM-SM family. It is also the multicast routing protocol considered by RENATER in the context of native multicast deployment. Linux allows to run PIM-SDM. PIM-SDM can adapt the routing tree to the topology of multicast leaves. This feature is mandatory when we will consider the public where final users will be able to connect and disconnect more

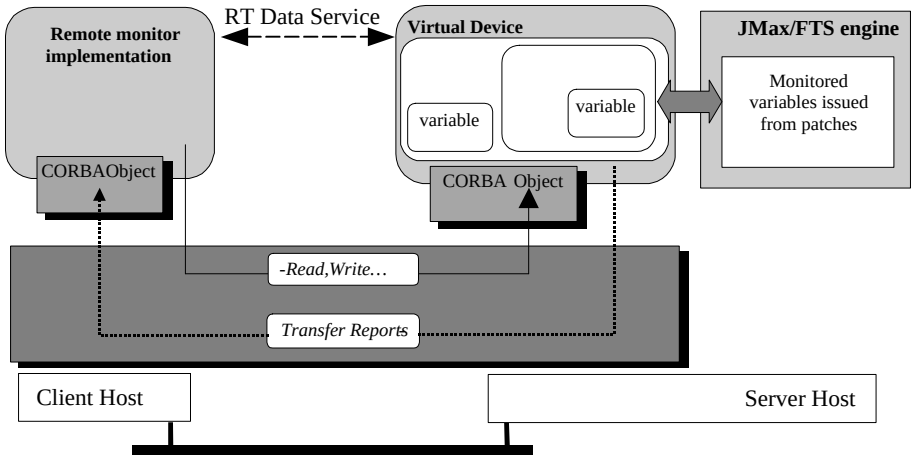



**Fig. 4.** *OpenTAZ : a CORBA based object oriented real-time data service implementation*

dynamically than musicians. The choice of PIM-SM implies the use of a shared tree to support multicast datagrams. It can be noticed that, in the case of a concert (orchestra and public), two shared multicast trees: one for the musician, the other for the public. For this last one, the root is clearly identified: the mixing host.

Indeed, contrary to DVMRP (where the packages are sent on many bonds not leading neither to a transmitter, nor with a member of the group), the scattered mode of PIM uses a mechanics of appointment between transmitter and member of the group. The messages are sent only in the event of attachment to the group. A dense mode, if the members of the group are close, was added: This mode is to be able to be easily carried in the scattered mode. Indeed, DVRMP has problems of conversion of metric with PIM SM

In our situation, our participants are grouped on geographical spot being able to be distant.

It is necessary thus that we can use the 2 modes of management of group: - on the level of a site, dense mode - between the sites, the scattered mode Contrary to DVMRP, PIM allow two operating modes (PIM DM and PIM SM) as well as an easy conversion between these 2 modes. Moreover, it is the protocol used on our 2 sites like on RENATER.

———————— To set up our service multicast, two possibilities are presented:

- IRCAM and CNAM are currently connected to the MBONE: a service of the French research network RENATER, until now experimental. Managed by an external company (CSSI), this service is specialized for the transport of flow multicast.
- To create a tunnel connecting our two sites.

The origin of the MBONE in France goes up at the end of the year 1993 thanks to the experimentation of Aristote. This association gave place to the creation of RENATER (FMBONE). From 1998 to 2000, traffic multicast of the FMBONE consists of a network of tunnel IP unicast in which traffic multicast is encapsulated. Today, a service of native transport multicast is deployed between the Regional Node of Distribution of RENATER. It is based on an infrastructure of private network (VPN ATM of 4Mb/s). The heart of the MBONE is made up routers multicast in the establishing NRD of the bonds by using protocols PIM SM, MBGP, MSDP. The FMBONE is implemented by a private operator (CSSI).

If we wish to connect an establishment other than those connected to MBONE, we must use an ISP having access to service multicast of RENATER by the terminal point of traffic Internet SFINX (www.SFINX.tm.fr). Today, little ISP is connected to this terminal point. Moreover, it is not obvious that a site connected to the MBONE deployed this service multicast in its internal infrastructure.

The establishment of a tunnel of site to site appears to be a better adapted solution. The installation of roads linux enables us to see a greater versatility. Thus we can carry out VPN to be able to accept musicians who would not have an easy access to the MBONE. The interest in this remote concert also comes from the accessibility of our service and the low costs of its installation.

The tunnel, enables us to create our own service: a bond multicast of quality with few intermediate interlocutors. Knowing our interlocutors, we thus have a minimum of not controled constraints. We can add a certain number of tools all to length of traverses our flow multicast to solve a blocking quickly. The measurements raised by these tools will make it possible to understand how the network reacts to the time constraints.

The solution brought by RENATER through MBONE is however not has to draw aside. In an immediate future, the MBONE will be in exploitation. We will be able to use the MBONE with respect to the sites connected to RENATER, of the tunnels to supplement existing it and the vpn to connect users to their residence.

**3/ Problems of routing multicast**

## 5 Conclusion and Future Works

The distributed virtual orchestra is a work in progress. Musicians can play on an IP multicast network that spans a LAN or a campus network. The aim of

our project is to scale to MAN networks with a tenth of musicians. We also plan trials across France between Lyon and Paris. These challenges will improve our prototype: self-synchronization, monitoring, and IP multicast. The guarantee of constant latency between the musicians is statistical by nature. In the future, it will be necessary to take into account the losses and to choose a strategy as interleaving or media specific Forward Error Correction [24]. During the test with the multicast tunnel, the application mesured 2,2% of loss in the tunnel. Also, a skew exists between the clocks of the different sound boards [25, 26]. Therefore, it will be necessary to solve this problem.
Our monitoring service runs correctly but it is not sufficiently flexible. We are able to monitor predetermined variables, but not user defined variables. We have to cover this lack, because musicians should be able to define what they want to be monitored. A long term goal is to connect our monitoring service to the numeric mixing device of a sound engineer.
Native multicast is a challenge by itself. We expect to deal with RENATER, the ISP of french academic research. But, we need also bandwidth provisioning, and it is a more complex task. If we use Linux based routers with GRE tunnels, we are able to handle QoS properly with iproute []. If we should use native QoS mechanisms in RENATER's routers, we will need to deal with a QoS broker not directly manageable by us. This is an open issue.
The current results enable to consider our platform in different use cases: a cheap recording studio, or a music school in order to automate the training of the musical interaction between instrumentalists. The automation of the training of elementary musical gestures is also possible.

## References

1. Elins, A., van Welie, M., van Ossenbruggen, J., Schnhage, B.: Jamming (on) the web. In: Proceedings of WWW6. (1997)
2. Young, J., Fujinaga, I.: Piano master classes via the internet. In: Proceedings of the International Computer Music Conference. (1999)
3. Goto, M., Hidaka, I., Matsumoto, H., Muraoka, Y.K.Y.: A jazz session system for interplay among all players. In: ICMC Proceedings. (1996)
4. Goto, M., Neyama, R., Muraoka, Y.: RMCP: Remote music control protocol, design and applications. In: ICMC Proceedings. (1997)
5. Goto, M., Neyama, R.: Open RemoteGIG:an open-to-the-public distributed session system overcoming network latency. IPSJ JOURNAL **43** (2002) 299–309 (en japonais).
6. Xu, A., Cooperstock, J.: Real-time streaming of multichannel audio data over internet. In: AES 108th convension, Paris (2000)
7. Cooperstock, J.R., Spackman, S.P.: The recording studio that spanned a continent. In: IEEE International Conference on Web Delivering of Music,WEDELMUSIC, Florence, Italie (2001)
8. Puckette, M., Danks, M., Steiger, R., Sorensen, V.: http://visualmusic.org/gvm.htm (1999)

9. Dchelle, F., Borghesi, R., Cecco, M.D., Maggi, E., Rovan, B., Schnell, N.: jmax: an environment for real-time musical applications. ComputerMusic Journal **23** (1999) 50–58
10. Dchelle, F.: jmax: un environnement pour la ralisation d'applications musicales temps rel sous linux. Actes des journes d'Informatique Musicale (2000)
11. Dchelle, F., Borghesi, R., de Cecco, M., Maggi, E., Rovan, J., Schnell, N.: jmax : An environment for real-time musical applications. In: Computer Music Journal. Volume 23-3. (1999) 50–58
12. Dchelle, F., de Cecco, M.: The ircam real-time platform and applications. In: ICMC : International Computer Music Conference, Banff (1995)
13. Dchelle, F., Borghesi, R., Orio, N., Schnell, N.: The jmax environment : an overview of new features. In: ICMC : International Computer Music Conference. (2000)
14. Orio, N., Dchelle, F.: Score following using spectral analysis and hidden markov models. In: ICMC : International Computer Music Conference, La Havane (2001)
15. Bargar, R., Dchelle, F., Choi, I., Betts, A., Goudeseune, C., Schnell, N., Warusfel, O.: Coney island : combining jmax, spat and vss for acoustic integration of spatial and temporal models in a virtual reality installation. In: International Computer Music Conference, San Francisco (2000)
16. Kon, F., Iazzetta, F.: Internet music: Dream or (virtual) reality. In: Proceedings of the 5th Brazilian Symposium on Computer Music, Belo Horizonte, Brazil (1998)
17. Bargar, R., Church, S., Fukuda, A., Grunke, J., Keislar, D., Moses, B., Novak, B., Pennycook, B., Settel, Z., Strawn, J., Wiser, P., Woszczyk, W.: Aes white paper: Networking audio and music using internet2 and next-generation internet capabilities. Technical report, AES: Audio Engineering Society (1998)
18. Bouillot, N.: Un algorithme d'auto synchronisation distribue de flux audio dans le concert virtuel rparti. Confrence Franaise sur les Systmes d'Exploitation (2003) CFSE 3. la Colle sur Loup, France. IN PROPOSAL.
19. Schulzrinne, Casner, Frederick, Jacobson: RTP: A transport protocol for real-time applications. RFC 1889 (1998)
20. Group, U.C.S.W.: TASE.2 services and protocol. version 1996-08. iccp inter-control centre communication protocol version 6.1. Technical Report IEC 870-6-503, IEC (1996)
21. Becquet, E., Abdallah, M., Gressier-Soudan, E., Horn, F., Bacon, L.: Object oriented timed messaging service for industrial ethernet: a fieldbus like architecture for power plant control and factory automation. In proceeding Fieldbus Technology (FeT'2001) (2001) IFAC. Nancy, France.
22. Becquet, E.: Opentaz home page (2003)
23. Rmer, K., Puder, A., Repository, I., Pilhofer, F., Schultz, A., Kersting, A., Gardas, K., Ltd, O.: Mico (2003)
24. Perkins, C., Hodson, O., Hardman, V.: A survey of packet-loss recovery techniques for streaming audio. IEEE Network Magazine (1998)
25. Orion, Hodson, Colin: Skew detection and compensation for internet audio applications (2000)
26. Fober, D., Orlarey, Y., Letz, S.: Clock skew compensation over a high latency network. In ICMA, ed.: Proceedings of the International Computer Music Conference. (2002) 548–552