A Component Model for Transmission and Processing of Synchronized Multimedia Data Flows

Emmanuel Bouix, Philippe Roose, Marc Dalmau, Franck Luthon

A Component Model for transmission and processing of Synchronized Multimedia Data Flows

Emmanuel Bouix, Philippe Roose, Marc Dalmau, Franck Luthon
LIUPPA – IUT, Computer Science Dept.
Château Neuf – Place Paul Bert 64100 Bayonne – France
{bouix, roose, luthon}@iutbayonne.univ-pau.fr, dalmau@ieee.org

Abstract

Our research deals with distributed multimedia applications built on software components. Currently, multimedia data are omnipresent on the Internet. However, this network is not designed to support and transmit multimedia data. In this perspective, it is necessary to introduce quality of service management in this kind of applications. In this paper, we are particularly interested in the inter-flow synchronization (e.g. audio and image flows of a video). We develop a component model in order to tackle the synchronization between multimedia flows from the source (e.g. media capture) to the destination (e.g. media player). This model is named OSAGAIA[*] and is made of two entities. The first one is called elementary processor. It is used as the runtime environment (component container) for multimedia components. The second one is called conduit. It is used to transport synchronous multimedia flows between elementary processors. So, distributed applications are composed of multimedia components (within containers) connected by multimedia flows (within conduits). We are working on a distributed prototype which validates the synchronization algorithms that we use. It is implemented with Java language using JMF (Java Media Framework) API and TCP/IP as network protocol.

1. Introduction

Our work deals with distributed multimedia applications through the Internet. These applications require a high flexibility to provide a correct Quality of Service (QoS). They need to adapt themselves in real-time to user's requirements and to the environment on which they are running [1]. Moreover, these requirements can evolve during runtime and these applications must be dynamically adapted to the new requirements. To achieve this flexibility, we choose a software component approach [2] in order to implement multimedia applications. By adding, removing or replacing components in real-time, the application can be adapted according to QoS requirements.

We propose a software architecture for distributed multimedia applications [3] which is divided in two parts: an application and a runtime platform. The application part is composed of distributed components connected to each other by multimedia data flows. The runtime platform deals with supervision: it detects critical situations and try to solve them by adapting the composition of the application to the new QoS criteria.

In this paper, we are interested in problems induced by the processing and manipulation of multimedia data flows. We present the OSAGAIA model which solves inter-flow synchronization by associating a time-stamp to each sample of each multimedia flow.

Section 2 presents an overview of the context and motivations of this research. Section 3 describes the OSAGAIA Multimedia Component Model by discussing all its elements. Section 4 presents how OSAGAIA solves problems described in section 2. Section 5 describes the dynamic configuration and reconfiguration process used by the runtime platform in order to keep acceptable QoS-levels. Section 6 presents a concrete implementation of OSAGAIA and Section 7 presents our conclusions and future work.

2. Context and Motivation

During these last years, software component paradigm has emerged in the software industry. The development of an application consists in the selection and assembly of pre-existing software components. Indeed, current software component models, e.g. EJB [4], COM [5], Fractal [6], etc., allow developers to design and implement applications. These models provide some non-functional properties such as persistence, transaction or security but, there is a lack when the interest is focused on a particular domain such as multimedia. For example, when handling multimedia data, one needs to take into account some important characteristics like synchronization between multimedia flows, QoS management, etc. In this way, non-functional properties provided by some models need to be extended by developers to integrate multimedia data characteristics. However, an extension

[*] OSAGAIA means software component in Basque Language

of CCM (Corba Component Model) is currently specified by the OMG [7]. This specification will have to address keypoints as topologies for flows, flow description and typing, QoS, etc.

To explain what is the inter-flow synchronization problem, we will take the example of TV news showing a speaker in front of a street background using two video cameras and two microphones (one for the speaker, one for the street). This application implements both bluescreen imaging and audio mixing. Bluescreen imaging is a technique which starts by filming a subject (the speaker) in front of a uniform monochromatic background (e.g. blue or green). Then, a compositing process replaces the selected background hue in the image with another background image (the street). Audio mixing technique allows to mix two audio flows in order to provide a single one. This configuration (shown in "**Fig. 1**") uses two processing components. The first one implements compositing process and the second one audio mixing function in order to present a composite audio signal to the user. On the same machine, we connect four capture devices: two video cameras for image capture, two microphones for audio capture. A complete video flow is composed of the audio flow and the image flow. At capture time, all flows are synchronous. When implementing this configuration with an existing model, e.g. EJB [4], COM [5], Fractal [6], the major problem is the loss of synchronization between flows since some of them are treated by different components. This problem is known as inter-flow desynchronization.

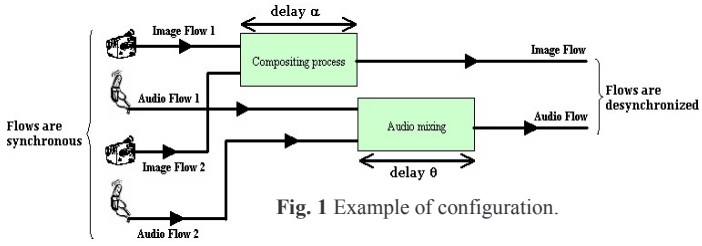

Fig. 1 Example of configuration.

Indeed, the two provided flows will no longer be synchronized like they were at the origin. This kind of problem is due to temporal relationships that exist between several flows (e.g. the sound and the image of a video). The flows of this configuration go across processing components in their path from source to destination accumulating different temporal delays (α and θ) as shown in "**Fig. 1**". In a complex application, the difference between temporal delays which affect related flows may be important and probably not acceptable by end-users because it affects the semantic of the data. In order to solve this problem, we propose the OSAGAIA model which keeps the synchronization between flows using temporal dependencies. The solution is implemented as a non-functional property [18]. A non-functional property is a concern independent from the business logic.

Researches in the multimedia domain try to take in consideration the characteristics of multimedia data. Exposito in his PhD [8], proposes to design a new QoS oriented transport protocol aimed at providing a large set of transport mechanisms to efficiently satisfy applications requirements using available resources and network services. Yavatkar presents a new transport protocol called Multi-Flow Conversation Protocol (MCP) [9] that provides two communication abstractions. The first one provides a token based mechanism for concurrency control among participants of a multipoint connection. The second one includes a novel communication abstraction called multi-flow conversation to allow temporal synchronization among traffic over multiple, independent flows. Demeure [10] describes a specification technique and a middleware to support distributed multimedia applications. Temporal constraints are specified independently from the application itself and from a dataflow graph that describes multimedia systems.

The originality of our work is to link the two domains: we try to consider the characteristics of multimedia data in order to develop a component model which includes these as non-functional properties. We will now describe the OSAGAIA model which allows the transport of multimedia flows taking into account the synchronization that may exist between several flows even if some are delayed by processing. Next section describes this model and the mechanism that we are using to support inter-flow synchronization.

3. OSAGAIA Multimedia Components Model

The OSAGAIA model is made of two entities which take care of inter-flow synchronization. The first one is the conduit that allows the transport of synchronous multimedia flows within the application. The conduit can be distributed through the Internet. The second one is the Elementary Processor (EP) that provides a runtime environment for a Business Component (BC). It keeps synchronization between processed and non-processed flows. The BC encapsulates the multimedia processing, i.e. the functional implementation [18] (functional properties). For example, a video capture BC implements only the necessary mechanism to provide the capture.

We present in the following section the multimedia flow characteristics and the synchronization mechanism used in this model.

3.1. The Multimedia Flows

3.1.1. Definitions. Multimedia applications handle several types of representation media such as images, sound, sub-title text, etc. The representation media is the type of data which defines the nature of the information as described in its coded format [11], e.g. MJPEG, MPEG, MIDI, etc. [12]

These data are called multimedia flows since they consist in a continuous sequence of finite size samples which have strict temporal dependencies. For example, an MJPEG flow is made of images with a frames rate of 25 images per second. Temporal dependencies may exist not only between samples of a same flow, but also between samples of different flows, e.g. between the images and the sound of a video.

Multimedia flows are voluminous (especially video) causing transmission bandwidth problems, so QoS management is necessary.

These applications handle multimedia flows and information flows too (e.g. events, text, messages, etc.), but this last kind of flow is not detailed in this paper.

3.1.2. Inter-Flow Synchronization Mechanism. In order to implement a synchronization mechanism, we use time-stamping of samples on each flow at acquisition or creation time. This mechanism is induced by the previous definitions [13].

"**Fig. 2**" shows the principle of this mechanism. At creation (here capture), a time-stamp is linked to each sample of each flow. Thus, we can transfer for each flow an information quantity which corresponds to a same time interval. For example, to transfer image and sound flow, for each image we transfer the audio samples associated to time-stamps equal or inferior to current image time-stamp.

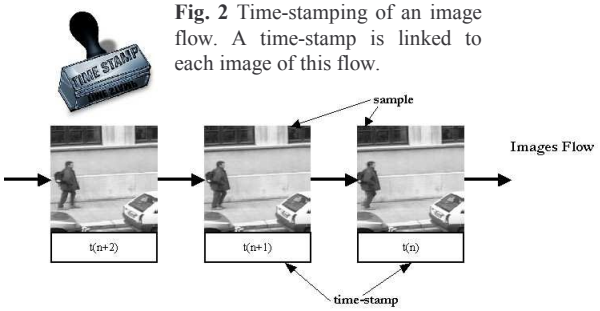

Fig. 2 Time-stamping of an image flow. A time-stamp is linked to each image of this flow.

We call that a "synchronous slice". The "synchronous slice" represents the temporal dependencies between samples of several synchronous multimedia flows.

The couple formed by a sample and a time-stamp is called a Temporal Unit (TU).

3.1.3. Input/Output connections: the Ports. The conduit and the EP use input/output ports to allow connections between them. Each has a couple of ports for each multimedia flow, e.g. if a conduit contains two flows then each flow will have an input and an output port. In the same way, if a component provides two flows then it will have two output ports. The ports export and import the required and provided data [14]. They can be compared to the pins of an electronic component.

The port is the structural unit of connection between the conduit and the EP. More precisely, it is used as a connectable element on the two elements of our model. "**Fig. 3**" describes the way to connect this two elements. The input conduit provides into its ports the data to be processed by the BC running inside the EP and conversely, the EP provides data to be transported by the output conduit. It should be noted that connection between EP and conduit can be performed with several conduits. Indeed, an EP can process several flows present in several conduits.

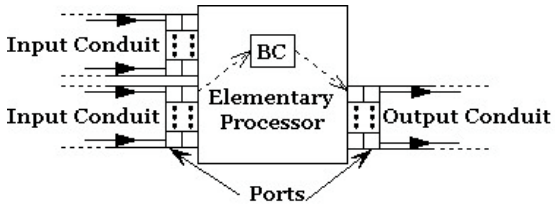

Fig. 3 Way to connect a Conduit and an EP in OSAGAIA.

A port accepts TUs as input or provides TUs as output. These are the required or provided data for the multimedia component, e.g. a component whose task is to process images requires images as input and provides processed images as output. A port can contain as many TU as necessary to constitute the "synchronous slice". Thus, a group of ports will contain a "synchronous slice". This technique is known as dynamic buffering because the size of ports can evolve during runtime. Their size is adapted to the size of the "synchronous slice" transferred at a given time.

3.2. The Business Component (BC)

In OSAGAIA, a BC implements a particular media processing. The BC represents the functional implementation of our model. For example, the "**Fig. 1**" configuration will have BCs for image capture,

audio capture, compositing process and audio mixing. The BC needs to be executed in a container named the EP which will be described in section 3.4.

The behavior of a BC is defined in terms of provided and/or required data. We distinguish three kinds of behavior:

- the first one only provides data. This kind of BC is known as capture component (e.g. audio capture component) ;
- the second one requires and provides data. This kind of BC is known as processing component (e.g. audio mixing component) ;
- the third one only requires data. This BC is known as rendering component (e.g. audio rendering component).

The BC is data driven, that means it needs data in order to apply its processing.

"**Fig. 4**" describes the generic structure of a BC.

```
Reading of TU     ut = pe.lireUniteTemporelle(this, numConduit, numFlux);

                  UniteTemporelle ut2 = (UniteTemporelle) ut.clone();

                  Image image = ( (ImageIcon) ut2.valDonnees()).getImage();
                  int x = image.getWidth(f);
                  int y = image.getHeight(f);
                  int masque[] = new int[x * y];

                  PixelGrabber pg = new PixelGrabber(image, 0, 0, x, y, masque, 0, x);
                  try
                  {
                    pg.grabPixels();
                  }
                  catch (InterruptedException ie)
Processing        {
                    System.err.println(
                        "ERROR: Interruption during processing.");
                  }

                  for (int j = 0; j < y * x; j++)
                  {
                    Color coul = new Color(masque[j]);
                    masque[j] = (255 << 24) | (255 - coul.getRed() << 16) |
                        (255 - coul.getGreen() << 8) | (255 - coul.getBlue());
                  }
                  ImageIcon ii = new ImageIcon(f.createImage(new MemoryImageSource(x, y,
                      masque, 0, x)));

                  ut2.affecterDonnees(ii);

Writing of TU     pe.ecrireUniteTemporelle(0, 1, ut2);
```

Fig. 4 General way to implement a BC (here, processing is a transformation on the grey-levels applied on each image of a flow).

First the BC must get data in order to run its processing. That is why, it initially executes a reading in an input port of the EP in order to have one or more TU to process. Then, it applies on data (i.e. part of TU) its own processing and writes the processed data in an output port of the EP.

The life cycle of a BC has four states which are Configured, Connectable, Connected and Disconnectable. On one hand, the first two states represent an activity where BC does not operate (initialized and ready to start). On the other hand, the two other states represent an activity where the BC applies its processing. This life cycle [23] allows the runtime platform to dynamically reconfigure multimedia application by adding, removing or replacing a BC by another one. So, it is necessary for the platform to know the activity of all the BCs which are used by the application at a given time. Thus, the platform can add, remove or replace a BC without disturbing the execution of the application.

3.3. The Conduit

We will now highlight the conduit which is the element used for multimedia flow transmission. Its major functionality is to transport synchronous multimedia flows. If several flows are put together in the same conduit, then these flows will be kept synchronous.

The conduit has input/output ports in order to allow its connection to EPs (see section 3.1.3). Each flow has a couple of ports (input and output). Each input port (respectively output) is connected to a buffer which stores the TUs before transport. The buffers are implemented with a queue data structure.

"**Fig. 5**" describes the path of the data within the conduit. A client/server approach is used to transfer TUs towards the output buffers for each flow. This mechanism can be implemented as well in the same machine or across the network. The synchronous transfer is performed with synchronous slices by using the time stamps linked to each sample at capture or creation time (see section 3.1.2). First, we look for the TU with the maximum time stamp in each multimedia flow of the conduit. This time stamp becomes the current time stamp. Then, for each flow, we transfer into corresponding output ports, all the TUs with time stamps lower or equal to the current one. After that, we get in output ports what we have called a "synchronous slice". It should be noted that each writing in output ports is signaled by an event used by the EP.

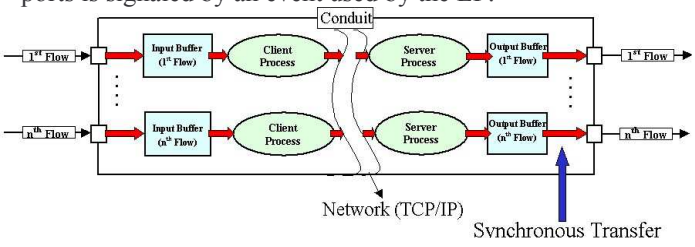

Fig. 5 Internal structure of the conduit with the path of data.

When applications are distributed through the Internet, the conduit allows network transfers. This is why the conduit encapsulates the TCP/IP [16] network protocol used for remote communication. In this case, flows are also transmitted in a synchronous way by using time-stamps of each sample and the "synchronous slice" is rebuilt by the receptor. "**Fig. 5**" shows an example of distributed conduit.

The conduit has a Control Unit (CU) which allows it to communicate with the runtime platform. Indeed, the conduit is also supervised by the platform. So, the CU reports various information on the behavior of the conduit, this information can be interpreted by the runtime platform for future reconfigurations. This evaluate a quantitative QoS-level, e.g. the filling rate of buffers for each flow. Overflow in some input buffer (respectively emptying of some output buffer) means that the rate of the network is low (respectively that some BCs are not suited).

3.4. The Elementary Processor (EP)

Now, we come to describe the EP and its internal functioning. The EP is a container for the BC. It gives some non-functional properties for a correct execution of the BC (functional properties) and of the whole application. We call it EP because its internal architecture has common points with a Von Neumann processor architecture [17]. The EP is composed of an Input Unit (IU), an Output Unit (OU) and a Control Unit (CU) as shown in "**Fig. 6**". All these units implement non-functional concerns [18] of the OSAGAIA model. The EP is an element supervisable by the runtime platform which can dynamically reconfigure the application (add, remove or replace EPs) in real-time according to QoS requirements.

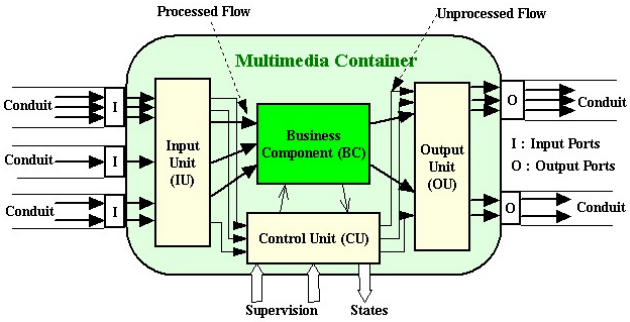

Fig. 6 Internal architecture of an EP.

As explained in section 3.1.3, the EP has a couple of input and output ports for each multimedia flow entering or outgoing. Like BCs, some can have only input or output ports, e.g. EPs that encapsulate capture or rendering BCs (see section 3.2). Each port is linked respectively to the IU or the OU. These units are interfaces between BC and multimedia flows. They contain methods used by the BC in order to read (respectively write) in the input (respectively output) ports. An introspection method allows accessing to the properties of the multimedia flows (e.g. the data type).

The CU manages the other entities of the EP. This unit communicates with events and specific methods as we shall see in details later. CU also manages the data circulation within the EP. The BC behavior is controlled by CU through its methods *init(), start() and stop()*.

The BC processes one ore more multimedia flows. In order to preserve synchronization between processed and non-processed multimedia flows, the PE is connected to the conduits transmitting the flows which can be processed. In this way, all data flows of the conduits circulate within the EP. The IU knows, by an event sent by the conduit, when new input information is present. In fact this new input always constitutes a "synchronous slice". Thus, when the BC requests a reading operation, the corresponding TUs are sent to it. During processing, IU stores the entiere "synchronous slice" corresponding to the TUs processed by the BC. When processing is achieved, the BC puts the results into the corresponding output ports via the OU. It requests another reading operation and the IU transfers to OU, the TUs of other synchronous flows stored previously. By this way, a new "synchronous slice" is now available in output ports of the EP.

"**Fig. 7**" is a sequence diagram that describes the event communication implemented between the different entities of the EP and the conduit using UML formalism [19]. This diagram details the case of an EP with only one conduit connected to its input and to its output in order to simplify explanation. For EP with several conduits connected to its input or to its output, the functioning is exactly the same. When data are written into the output ports of a conduit, an event is sent to the input ports of EP directly connected to this conduit (1). During this time, BC is in a work state and will need TUs in order to process them (2). So, a reading operation is performed. This operation initiates a transfer between output ports of the conduits and input ports of the EP (3) and (4). Then, TUs are available for the BC (5), so the BC receives the TUs requested in (2) and it can perform its processing on these TUs. During this processing time, input unit transfers to output unit the TUs corresponding to the same time interval than the processed TUs (6). For example, if processing is performed on one image, the IU transfers all sound samples which time stamp is lower or equal to the image time stamp. When BC has finished its processing, it can write produced TUs into the corresponding output ports (7). To do this, it uses a writing method of the OutputUnit. Then the BC needs TUs again in order to process them, so it performs another reading operation (8). After this second operation, TUs stored in Output Unit are transferred into output ports of the EP (10) and so a "synchronous slice" is constructed. These writing operations are

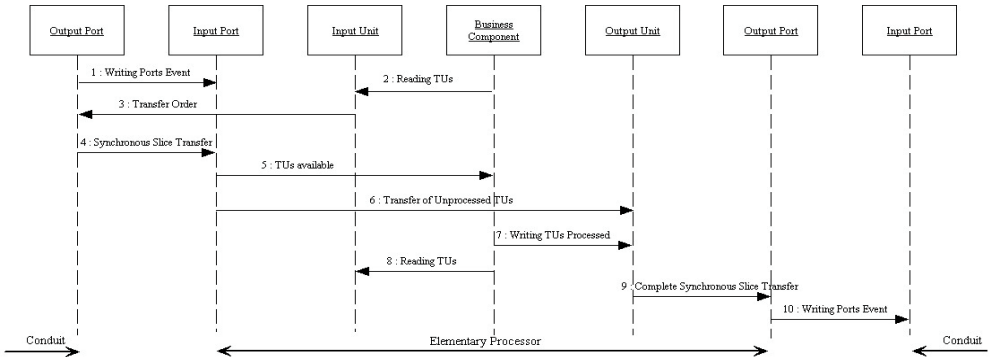

Fig. 7 Sequence diagram showing interactions between entities which compose the elements of the model.

signaled by a writing port event to the conduit connected at the right side of the EP (11).

The "**Fig. 8**" shows an example of the synchronous transfer of TUs inside the PE "synchronous slice" by "synchronous slice".

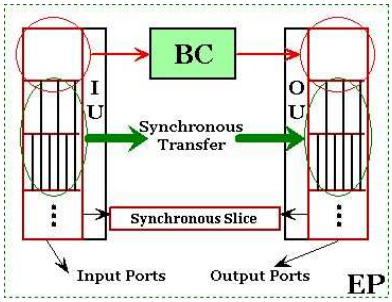

Fig. 8 Synchronous Slice Transfer.

4. Applying OSAGAIA to an application

In this section, we present how OSAGAIA solves the problems described in section 2. We apply OSAGAIA to a TV news application.

When we apply OSAGAIA to this application, each component is encapsulated into an EP. The multimedia flows are encapsulated within a conduit in order to keep their synchronization. This is shown on "**Fig. 9**.

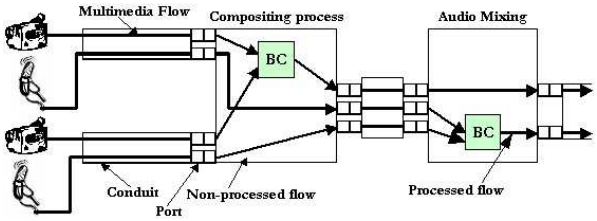

Fig.9 A TV news application with OSAGAIA.

When the two image flows are processed by the first component of the configuration (Compositing process), the two audio flows pass within the EP synchronously with the provided flow by the BC. In the output of this first EP, a conduit is connected with three data flows inside: the image flow provided by the compositing process and the two audio flows. In the same way, the image flow passes within the second EP synchronously with processed audio flows. Thus, as output of the application, we obtain a video composed of two synchronous flows: image flow plus audio flow.

5. Dynamic Configuration/Reconfiguration

The internal structure of a multimedia application may be adapted to the QoS requirements defined by both users and runtime environment [22]. We define in this section how dynamic configurations or reconfigurations are performed in our system.

Dynamic configuration is known as the ability for an application to be customized according to both the characteristics of the runtime environment and to the requirements of end-users. Dynamic reconfiguration is the ability for an application to be changed on-the-fly, i.e. to be able to change its internal components and configuration parameters to adapt to both changes of the runtime environment and to the requirements of end-users. For simplicity, the term dynamic configuration may be used to denote both dynamic configuration and dynamic reconfiguration [21] [15].

We remind some principles about the configuration of an application with this model in order to understand how dynamic configurations are performed. Between two EPs, there is as much conduits connected as data flows processed by BC. For example, "**Fig. 6**" shows a BC which processes three flows. This three flows are in different conduits that is why three conduits are connected at the input of this EP, i.e. a conduit is connected between two EPs if and only if one at least of its flows is processed by the right EP.

A configuration consists in two operations which are the adding and the removing of a BC. Adding or removing a BC reduces to adding or removing the EP that encapsulates it. The replacing operation is not considered here because a replacing consists in a removing and then an adding. In addition to these operations on EP, the runtime platform should also performs operations on the conduits: in cases of dynamic configuration, it is indeed necessary to connect and disconnect the conduit concerned. All these operations both on EP and on conduit are

performed by the runtime platform according to application description graphs showing old and new configuration [1].

We detail now how adding components is performed. First, the platform detects a QoS violation in the configuration which is running and decides to add a BC. Then, it performs the following five actions :

- (1) the place where to insert the new EP (containing the BC) is identified ;
- (2) the platform disconnects the conduits at this place ;
- (3) it adds the new EP and connects the disconnected conduits (action (2)) at its input ;
- (4) then, it creates the necessary conduits in order to connect the output of the added EP to the rest of the configuration ;
- (5) the conduits created in action (4) are now connected to the new EP.

We detail now how a removing of components is performed. First, at a given time, the platform decides [22] that a component must be removed from the application. Then, it performs the following five actions :

- (1) the platform determines which BC must be removed ;
- (2) the platform disconnects all the conduits in input and output of the EP which contains the concerned BC;
- (3) the platform removes the conduits that were connected at the output of this EP ;
- (4) the EP is now removed from the application ;
- (5) the conduits that were connected at input of the removed EP are now connected in input of others EP according to description graphs.

These two operations can require flow duplication in certain cases. For example, a data flow must be connect to several EP. So, it is necessary to propose a solution to address this kind of problem. Two solutions are possible:

- to use a special component which receives at input one conduit and provides at its output n conduits which are exactly the same than the input data flow;
- to allow to connect a conduit to several EPs, i.e. a conduit can transport data flows towards many destinations (EPs).

For our application model, we choose the first solution because it is necessary to distinguish all the conduits and data flows which circulate within the application. For example, if we want to connect the same conduit with two EPs: one located on the same machine and the other on a remote machine, we must have two conduits in order to perform correctly this operation.

6. Prototype implementation with OSAGAIA

We will now describe a prototype of this kind of application. The aim of this implementation is to test model coherence and development of multimedia applications. Moreover, this prototype must validate the algorithms and mechanisms used to keep synchronization between the multimedia flows. This synchronization must be kept at two levels :

- first, during the transport of multimedia flows within the application even through the network;
- secondly, during the processing of a multimedia flow.

The prototype is shown in "**Fig. 10**". This prototype is implemented with Java/JMF API [20]. JMF is an API for management of time-based media into Java applications.

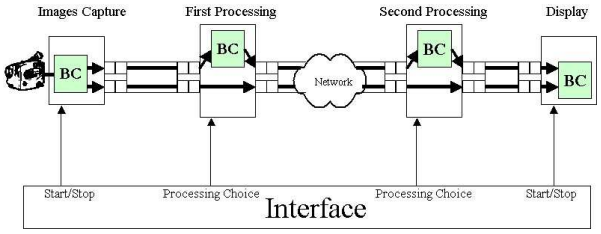

Fig. 10 Architecture of the prototype.

Moreover, it allows to capture, handle, process and render multimedia data flows. For the moment, the network protocol we use is a communication by socket with TCP [16].

This application is composed of four components. We can distinguish a capture component, two processing components and a rendering component.

The capture component provides an image flow captured by a video camera. We use several processing components, for example one which reduces the initial image flow, an other which implements a negative transformation on the grey-levels image, etc. We can choose between these processings with a drop-down list on the interface. This choice can be performed at each side of the network. Finally, the last component is used to display the two flows of this configuration. This component provides two players in order to display each flow on one player. These players allow us to control that the two flows are synchronous like they were at capture time.

The interface simulates in a manual way the behavior of the runtime platform. With this interface

we can start and stop the capture, start and stop the display and dynamically change the two processing components.

The principle of this prototype is easy to understand. The video camera provides image flow. Then, this flow is duplicated in order to obtain two image flows. We apply processing only on the first flow like shown in "**Fig. 10**" and the second image flow is not processed. This allows us to see that the two flows are synchronous during the path from capture to display including network transfer within the prototype, even if only the first one has been processed.

7. Conclusions and Future Work

Currently, complexity of the development of software is increasing. In addition to handle a large amount of various data, applications are actually distributed through the Internet. In the past, applications used to be built with textual data and fixed images. Now, these data (discrete data) are mixed with animated images and sound samples. These applications are known as multimedia.

The software components approach is a solution for the development of this kind of applications. Many component models are developed in both research and industry. These models allow development of software components in order to compose (composition is a property of the software components approach) applications. These models enable to take into account what is called non-functional properties. These properties are known as services given by the model. Thus, a developer of applications can only concentrate on the development of software components by using the provided services. Services are for example transaction, persistence, security, etc.

When we are interested by multimedia domain these non-functional properties are not sufficient. They must be extended in order to consider the characteristics of multimedia data. That is why, we propose a component model which proposes to integrate some non-functional properties about multimedia. We solve the problem of inter-flow synchronization that arises when one considers temporal constraints between different multimedia flows (e.g. images and sound of a video). In this article, we propose algorithms and principles of inter-flow synchronization. Principles used by OSAGAIA are based on the time-stamping of each sample of each multimedia flow. This temporal constraint is an absolute constraint because each sample is linked with a unique time stamp.

This principle is used in the two elements of OSAGAIA : the conduit and the EP. Each element takes into consideration the synchronization that may exist between several flows. The conduit is used to transport multimedia flows in synchronous way. It can be distributed through the network. It is the distribution unit of our model. The EP is a runtime environment for a BC. It implements the mechanisms in order to handle multimedia data, i.e. the non-functional properties. The BC implements the functional properties, i.e. the multimedia processing. In this way our model provides a clear separation between non-functional and functional properties.

We develop a prototype that validates the mechanisms used in this model. This application uses Java and JMF API and implements a simple distributed application. We simulate the runtime platform with an interface to change dynamically processing components.

In this article, we concentrated on multimedia flows. We must specify others types of flows (e.g. events, text, messages, etc.) which are continuous too but not regular in order to integrate this kind of data into our model. That means integrating asynchronous data flows and mixing them with synchronous ones in order to keep temporal relationship between them.

We also want to complete the prototype in order to use others network protocols like RTP for example [24]. This will permit us to collect performance measures allowing to compare various implementation solutions.

8. Acknowledgements

This research is supported by the Conseil Régional d'Aquitaine (Aquitaine Regional Council - France).

9. References

[1] Laplace S., Dalmau M., Roose P., "A formal method for assening quality of service in distributed multimedia applications", *16th International Conference on Software & Systems Engineering and their applications ICSSEA'03*, Paris, France, December 2-4 2003.

[2] Szyperski C., *Component Software – Beyond Object-Oriented Programming*, New-York, Addison-Wesley Publishing Co., 1999.

[3] Roose P., Dalmau M., Luthon F., "A distributed Architecture for Cooperative and Adaptive Multimedia Applications", 26th International Computer Software and Applications Conference COMPSAC'02, IEEE Computer Society Press, ISBN: 0-7965-1727-7, Oxford, England, August 26-29 2002, pp. 444-449.

[4] Sun Microsystems, Enterprise Java Beans Specification 2.1 Final Release, 2003. http://java.sun.com/products/ejb/docs.html

[5] Microsoft Corporation, COM Component Object Model Specification 0.9, October 1999. http://www.microsoft.com/com/resources/comdocs.asp
[6] Bruneton E., Coupaye T., Stefani J-B., The Fractal Component Model 2.0-3, February 2004.
[7] OMG, Streams for CORBA Components, Request For Proposal, OMG document, 2003. http://www.omg.org/cgi-bin/doc?mars/2003-06-11
[8] Exposito E., Specification and implementation of a QoS oriented transport protocol for multimedia applications, PhD Thesis, Institut National Polytechnique de Toulouse, December 17th 2003.
[9] Yavatkar R., "MCP: A Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications", *In Proceedings of IEEE 12th International Conference on Distributed Computing Systems ICDCS'92*, Yokohama, Japan, June 9-12 1992.
[10] Demeure I., Leboucher L., Rivierre N., Singhoff F., "Modèle et plate-forme pour le support d'applications multimédias réparties", *1ère Conférence Française sur les Systèmes d'Exploitation CFSE'99*, Rennes, France, June 8-11 1999.
[11] ISO/IEC JTC1/SC29/WG12, Coding of audio, picture, multimedia and hypermedia information, MHEG Working Group S.5, June 1999.
[12] Luther Arch C., *Principles of digital audio and video*, Boston, The Artech House audiovisual library, 1997.
[13] Hafid A., von Bochmann G., Dssouli R., "Distributed Multimedia Application and Quality of Service: A Review", *Electronic Journal on Networks and Distributed Processing*, n°6, 1998.
[14] Paradinas P., Etat de l'art des modèles de composants, Projet Cesure RNRT n°98, 2000.
[15] Layaida O., Atallah S-B., Hagimont D., "Reconfiguration-based QoS Management in Multimedia Streaming Applications", *In Proceedings of the 30th IEEE/EUROMICRO Conference, Rennes, France, 2004.*
[16] Comer D., Internetworking With TCP/IP – Volume 1: Principles Protocols, and Architecture, Prentice Hall, 2000.
[17] Godfrey M-D., Hendry D-F., "The Computer as von Neumann Planned It", *IEEE Annals of the History of Computing*, vol. 15, n° 1, p. 11-21, January-March 1993.
[18] Villalobos J., Components Federation: a Software Architecture for Composing by Coordination, PhD Thesis, Université Joseph Fourier de Grenoble, July 15th 2003.
[19] OMG, Unified Modeling Language Specification, OMG document, March 2003. http://www.omg.org/technology/documents/formal/uml.htm
[20] Sun Microsystems, Java Media Framework API Guide, November 1999. http://java.sun.com/products/java-media/jmf/2.1.1/guide/
[21] Kon F., Campbell R., Nahrstedt K., "Using Dynamic Configuration to Manage A Scalable Multimedia Distribution System", *Computer Communication Journal (Special Issue on QoS-Sensitive Distributed Systems and Applications)*, Vol. 24, pp. 105-123, Elsevier Science Publisher, January 2001.
[22] Roose P., Dalmau M., Luthon F., Laplace S., "Gestion de la Qualité de Service par Reconfiguration Dynamique dans les Applications Interactives Multimédia", *Journées ALP (GDR du CNRS) – ACM/SIGOPS – Systèmes à composants adaptables et extensibles*, Ed. INRIA, ISBN: 2-7261-1229-3-17, Grenoble, France, October 17-18 2002.
[23] Bouix E., Un modèle de composants multimédia adapté à la circulation des flux de données – Synchronisation des flux par conduits, Mémoire de DEA, UFR Sciences et Techniques de l'Université du Maine, Le Mans, France, September 2003.
[24] Schulzrinne H., Casner S., Frederick R., Jacobson V., RTP: A Transport Protocol for Real-Time Applications, RFC 1889, 1996.