A Multimedia Oriented Component Model.

Emmanuel Bouix, Marc Dalmau, Philippe Roose, Franck Luthon

A Multimedia Oriented Component Model

Emmanuel Bouix, Marc Dalmau, Philippe Roose, 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

Handling the inter-flow synchronization (e.g. images and sound of a video) in distributed multimedia applications seems to be fundamental. In this paper, we propose a new component model, named OSAGAIA[*]*, in order to tackle synchronization between multimedia flows from the source (e.g. media capture) to the destination (e.g. media player). This model is made of two entities. The first one is called "elementary processor" and is used as a runtime environment (container) for multimedia components. The second one is called "conduit" and is used to transport synchronous multimedia flows within the application. To ensure the synchronization within both entities, we sample multimedia flows in order to constitute temporal units and synchronous slices. We test this mechanism with a distributed prototype implemented with Java language using JMF (Java Media Framework) API and TCP/IP as the network protocol. This prototype shows that synchronization is kept between a non-processed and a processed image flow.*

1. Introduction

Our work deals with distributed multimedia applications through the Internet. Due to network characteristics, these applications require a high flexibility to provide and maintain a correct Quality of Service (QoS). They need to adapt themselves in real-time to both end-users and runtime environment requirements [1]. Considering that during runtime these requirements can evolve, this adaptation must be managed dynamically. We choose to provide this flexibility by using software component approach [2]. Thus, by adding, removing or replacing components in real-time, applications will be adapted according to these requirements.

[*] OSAGAIA means software component in Basque Language

Our previous work proposes a software architecture for this kind of application [3]. It is divided into two parts: the application and a runtime platform. The application part is composed of components connected to each other by multimedia flows. The runtime platform has a supervision role: it detects critical situations and tries to solve them by adapting the composition of the application to the current requirements. Both parts are, of course, distributed.

Multimedia applications deal with several types of data such as images, audio, sub-title text, etc. These data often exist in the form of continuous flows since they consist in a continuous sequence of finite size samples with strict temporal dependencies at two levels:

- between samples of the same flow (rate): this is known as Intra-Flow Synchronization ;
- between samples of several flows: this is known as Inter-Flow Synchronization. [12]

In this paper, we focus only on this last point and propose a new component model which takes care of this kind of synchronization. A prototype shows that this mechanism works well.

2. Context and Motivations

Academic (e.g. Fractal [4]) and commercial (e.g. EJB [5], COM [6]) component models allow developers to design and implement applications. Thus, development consists in the selection and assembly of pre-existing software components. These models provide some non-functional properties such as persistence, transaction or security, etc. However, there is a lack when the interest is focused on particular domains. So, these non-functional properties need to be extended or new models need to be created. For example, the PECOS model [7] has been developed for a specific class of embedded systems known as "field devices". In this way, specific characteristics of multimedia need to be considered.

Researches in the multimedia domain try to take in consideration the characteristics of multimedia data.

The Presentation Processing Engine (PPE) framework [8] simplifies the development of multimedia components and enables them to dynamically adapt the quality of the presentation to the resources in heterogeneous environments. Exposito in his PhD [9], proposes to design a new QoS oriented transport protocol aimed at providing a large set of transport mechanisms to efficiently satisfy application requirements using available resources and network services. Singhoff [10] defines a data flow graph model to specify both flows of a multimedia system and temporal constraints associated to these flows. Then, several algorithms are provided to translate these specifications into scheduling information (e.g. processor allocation).

Our research consists in providing a new component model for the multimedia domain. This model must consider the specific characteristics of multimedia data in order to compose multimedia applications. This should be achieved in the form of services provided by the model for developers (non-functional implementation). For example, OSAGAIA model ensures inter-flow synchronization which means that developers don't have to implement this functionality. Thus, they can focus their development task on the application (functional implementation). To illustrate our purpose, we will take the example of a 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 one composite audio signal. Such a configuration (shown in "**Fig. 1**") uses four capture devices connected to the same machine and implements two processing components (compositing process and audio mixing). At capture time, all flows are synchronous. When implementing this application with an existing model, e.g. EJB [5], the major problem is the loss of synchronization between flows. Indeed, these flows 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 more 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 semantics of the data. This problem is known as inter-flow desynchronization. OSAGAIA solves this problem by implementing it as a non-functional property [11] (concern independent from the business logic of the application).

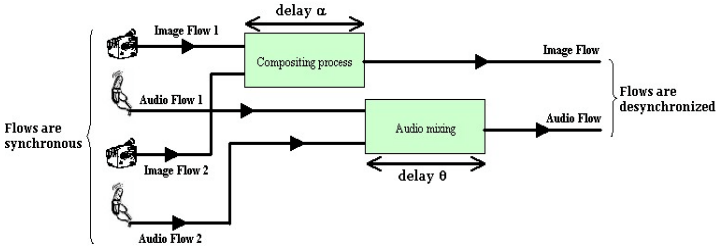

Figure 1. Example of configuration with existing models

3. OSAGAIA Multimedia Components Model

The OSAGAIA model is made of two entities which handle inter-flow synchronization. The first one is the conduit that allows the transport of synchronous multimedia flows within the application. It can be distributed through the Internet. The second one is the Elementary Processor (EP) that provides a runtime environment for a Business Component (BC). The BC encapsulates a particular multimedia processing, i.e. the functional implementation [11]. For instance, a video capture BC implements the necessary mechanism to provide this capture.

3.1. Inter-flow Synchronization Mechanism

Inter-flow synchronization is known as temporal constraints between several flows (e.g. the sound and the image of a video). More precisely, these constraints are defined between the samples of each flow (e.g. one image corresponds to several sound samples in a video). So, it seems necessary to identify the samples of each flow in a unique way. To do this, a time-stamp is associated to each sample on each flow at acquisition or creation time. We name this mechanism the flows time-stamping. The principle is illustrated in "**Fig. 2**".

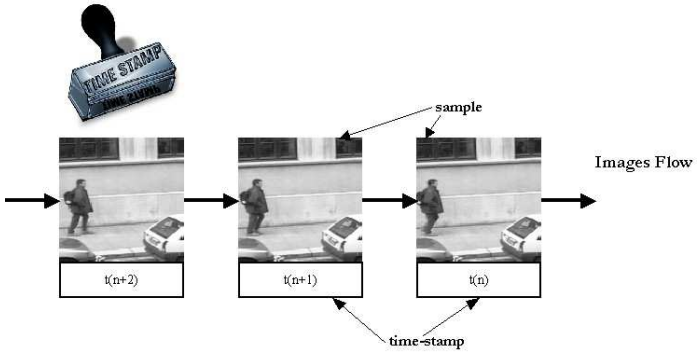

Figure 2. Time-stamping of an image flow

The real value of this time-stamp is not important because it will only be used to associate different flow samples which were created at the same moment. As a time-stamp, we use the clock system. The couple formed by a sample and a time-stamp is called a Temporal Unit (TU).

Thus, we can express the temporal constraints binding the TUs of several synchronous flows. So, we can transfer for each flow an information quantity which corresponds to a same time interval. For example, to transfer images and sound flows, we can for each image transfer the audio samples associated to the time-stamps which are equal or lower to the current image time-stamp and greater than the time-stamp of the previous image. Such a set of TUs of different synchronous flows is called a synchronous slice. Thus, a succession of synchronous slices constitutes a group of synchronous flows. They will be group into a conduit in order to be transported.

3.2. Input/Output Connections: the Ports

To connect both entities of the model, each one must have at least one port as shown in "**Fig. 3**". Ports are the means by which multimedia flows pass from EP to Conduit, and conversely. Ports accept TUs as input or provide TUs as output. Indeed, they are categorized by the direction of multimedia flows as either Input ports (which receive flows from previous entity) or Output ports (which produce flows to next entity). A group of ports connected to the same conduit will contain a synchronous slice. Their size is adapted to the size of the synchronous slice. This technique is known as dynamic buffering because the size of ports can evolve during runtime.

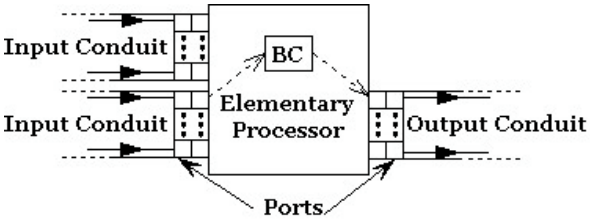

Figure 3. Way to connect Conduits and EP in OSAGAIA

The port is the structural unit of connection between both entities of the model (connectable element). Output ports of one entity can only be connected with input ports of others ones.

3.3. The Business Component (BC)

BC implements a particular media processing (functional implementation). 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 EP (see section 3.5).

The BC is data driven, that means its processing is linked to incoming data.

We describe now the general way to implement a BC. First, the BC must get data to process. To do this, it executes a reading operation in an input port of the EP (which encapsulates it) in order to get one or more TUs to process. Then, it can apply on data (i.e. data part of the TU) its own processing and write the processed data in an output port of the EP by executing a writing operation.

Because the runtime platform supervises the application part by removing/adding components, the life cycle of the BC [13] is composed of four states. On one hand, the first two states (Configured and Connectable) represent an activity where BC does not operate (initialized and ready to start). On the other hand, the two other states (Connected and Disconnectable) represent an activity where the BC applies its processing. This life cycle allows the runtime platform to know the activity of all the BCs which are used by the application at a given time. Thus, if needed, the platform can add, remove or replace a BC without disturbing the execution of the whole application. This is the key point to address QoS management.

3.4. The Conduit

The conduit is the entity used 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 3.2). Each port is connected to a buffer which stores the TUs. The buffers are implemented with a queue data structure.

The "**Fig. 4**" describes the path of data within the conduit. When the conduit is distributed, a client/server approach is used to transfer TUs towards the output buffers for each flow. In case of a local conduit, a producer/consumer approach is used. The transfer is performed with synchronous slices (see section 3.1) by using the time-stamp linked to each sample. First, we search for the TU with the maximum time-stamp in each multimedia flow of the conduit. This time-stamp

becomes the current one. Then, for each flow, we transfer into corresponding output ports, all the TUs which time-stamp is lower or equal to the current one.

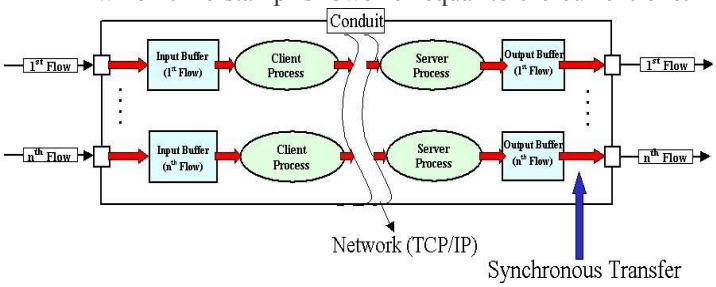

Figure 4. Internal structure of the Conduit with the path of data

Thus, the synchronous slice is reconstituted in output ports. It should be noted that each writing in output ports is signaled by an event which can be used by the connected component.

The conduit encapsulates the network protocol used for remote communication. The synchronous slice is rebuilt by the receptor. "**Fig. 4**" shows an example of a distributed conduit.

The conduit has a Control Unit (CU) which allows it to communicate with the runtime platform (not represented in "**Fig. 4**"). In reconfiguration cases, the conduit is supervised by the platform. The CU reports various information on the behavior of the conduit. This information is interpreted by the platform for future reconfigurations. This evaluate a quantitative QoS-level, e.g. the filling rate of buffers on 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).

3.5. The Elementary Processor (EP)

The EP is a container for the BC. It supports 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 [15]. The EP is composed of an Input Unit (IU), an Output Unit (OU) and a Control Unit (CU) as shown in "**Fig. 5**". The EP is supervisable by the platform (add, remove or replace EPs).

The EP has input/output ports for each multimedia flow entering or outgoing (see section 3.2). These ports allow its connection to conduits. 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).

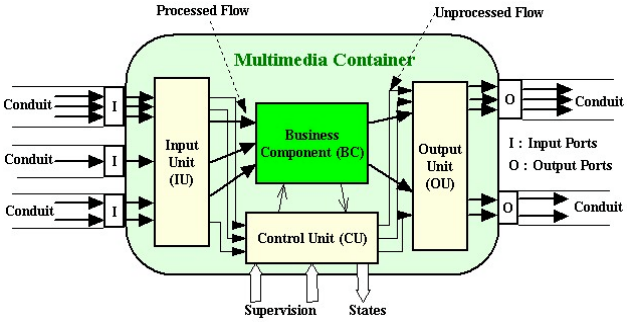

Figure 5. Internal Structure of the EP

The CU manages all the elements of the EP. This unit communicates with events and specific methods. For instance, the BC behavior is controlled by the CU through its methods init(), start() and stop(). CU also manages the data circulation within the EP.

The BC processes one or more multimedia flows. In order to preserve synchronization between processed and non-processed flows, the EP is connected to the conduits transmitting the flows which are 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. Thus, when the BC requests a reading operation, the corresponding TUs are sent to it. During processing, IU stores the whole synchronous slice corresponding to the TUs processed. When processing is achieved, the BC puts the results into the corresponding output ports via the OU. Then, new TUs are requested by the BC with a reading operation. During this time, TUs of the previous synchronous slice are transfered to the output ports via the OU. By this way, a new synchronous slice is now available in the output ports of the EP. It should be noted that each writing in output ports is signaled by an event used by the conduit (connected at output of the EP).

4. Applying the Model

This section shows how OSAGAIA solves the problems described in section 2. We apply it to a TV news application (see description in section 2).

On one hand, each component of this application is now encapsulated into an EP. On the other hand, the multimedia flows are encapsulated within conduits. "**Fig. 6**" shows the configuration with the OSAGAIA model.

When the two image flows are processed by the first component of the configuration (compositing process),

the two audio flows pass through the EP synchronously with the flow provided 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).

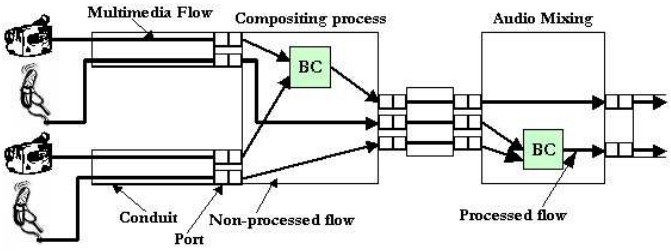

Figure 6. The TV news application with OSAGAIA

5. Prototype implementation with OSAGAIA

We develop a prototype in order to test model coherence and development of multimedia applications with OSAGAIA. Moreover, this prototype must validate the algorithms and mechanisms used to keep synchronization between several multimedia flows. This synchronization must be kept at two levels:

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

The prototype is shown in "**Fig. 7**". It is implemented with Java/JMF API [16]. JMF is used to manage time-based media into Java applications.

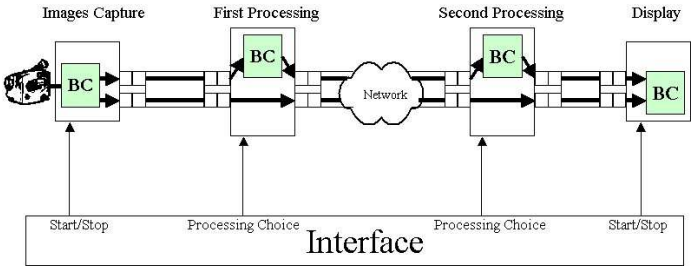

Figure 7. Prototype Architecture

Moreover, it allows to capture, handle, process and render multimedia data flows. For the moment, the network protocol that we use is a communication by sockets with TCP [14].

This prototype is composed of four components: a capture component, two processing components and a rendering component. As we need to start with two synchronous flows, the capture component provides two identical image flows captured by a video camera. We give the choice to end-users between several processing components. For instance, one which reduces the size of the initial image (spatial subrampling), an other which implements a negative transformation, etc. These choices can be performed at each side of the network ("**Fig. 7**") with a drop-down list ("**Fig. 8**") on the interface. As we can see in "**Fig. 7**", one of the two image flows is processed before and after network transport and the other one is used as a reference flow. Finally, the last component provides two video players in order to display the two flows of this configuration. These players allow to control that the two flows are displayed in a synchronous way like they were at capture time.

Figure 8. Drop-down list and video players used to display image flows

The graphic interface simulates the platform behavior in a manual way. With this interface, we can start/stop both capture and display and dynamically change the processing components with the drop-down list ("**Fig. 8**").

The principle of this prototype is easy to understand. The video camera provides one image flow. This flow is thereafter duplicated in order to obtain two image flows. We apply processing only on the first flow like shown in "**Fig. 7**". This allows to verify if 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 ("**Fig. 8**").

6. Conclusions and Future Works

Currently, complexity in the development of software is increasing. In addition to handle a large amount of various data, applications are actually distributed. In the past, applications managed textual data and fixed images. Now, these data are mixed with animated images and sound samples.

The software components approach is a good solution to develop this kind of application. Many

component models are designed in both research and industry. These models allow development of software components in order to compose applications. They enable to take into account what is called non-functional properties (services given by the model). Thus, developers can only concentrate on the development of specialized software components by using these services.

When we are interested in multimedia domain, these services are not sufficient. The specific characteristics of multimedia data have to be taken into account. That is why we propose a new component model which considers and maintains the inter-flow synchronization by time-stamping each sample of each flow and constituting synchronous slices of data. Thus, the temporal constraints are considered and respected.

This principle is applied in the two elements which constitute the OSAGAIA model: the conduit and the EP. The conduit is used to transport multimedia flows in a synchronous way. It can be distributed. The EP is used as a runtime environment for a BC. It implements mechanisms in order to handle multimedia data, i.e. non-functional properties. The BC implements a particular multimedia processing, i.e. functional properties. In this way, our model provides a clear separation between non-functional and functional properties.

A prototype has been developed in order to validate the mechanisms and principles exposed here. This configuration uses Java language and JMF API. It is distributed. We simulate a part of the behavior of the runtime platform with a graphic interface which allows for instance to change dynamically some processing components which is a first step towards QoS management in multimedia applications.

In this paper, we concentrated our approach on multimedia flows. However, other kinds of data can circulate in applications (e.g. events, text, messages, etc.). So, it is necessary to specify these kind of flows which seem to be continuous too but not regular. That means to integrate not regular synchronous data flows and to mix them with regular ones.

Further work will consist in completing the prototype by using and testing other network protocols (RTP, UDP). This will permit us to collect performance measures allowing to compare various implementation solutions.

7. References

[1] Laplace S., Dalmau M., Roose P., "A formal method for assessing 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*, Addison-Wesley, November 2002.

[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] Bruneton E., Coupaye T., Stefani J-B., The Fractal Component Model 2.0-3, February 2004.

[5] Sun Microsystems, Enterprise Java Beans Specification 2.1 Final Release, 2003.

[6] Microsoft Corporation, COM Component Object Model Specification 0.9, October 1999.

[7] Genssler T. and al., PECOS in a Nutshell, The PECOS Consortium, September 16th 2002.

[8] Posnak E. J., Lavender R. G., Vin H. M., "An Adaptive Framework for Developing Multimedia Software Components", *Communications of the ACM*, Vol. 40, No. 10, October 1997.

[9] Exposito E., Specification and implementation of a QoS oriented transport protocol for multimedia applications, PhD Thesis, Institut National Polytechnique de Toulouse, December 17th 2003.

[10] Singhoff F., Spécification temporelle modulaire et support pour les applications multimédias réparties, PhD Thesis, Ecole Nationale Supérieure des Télécommunications, Paris, December 14th 1999.

[11] Bruneton E., Riveill M., JavaPod : une plate-forme à composants adaptable et extensible, INRIA research report, January 2000.

[12] ISO/IEC JTC1/SC29/WG12, Coding of audio, picture, multimedia and hypermedia information, MHEG Working Group S.5, June 1999.

[13] Bouix E., Un modèle de composants multimédia adapté à la circulation des flux de données – Synchronisation des flux par conduits, Master Thesis, UFR Sciences et Techniques, Université du Maine, Le Mans, France, September 2003.

[14] Comer D., Internetworking With TCP/IP – Volume 1: Principles Protocols, and Architecture, Prentice Hall, 2000.

[15] 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.

[16] Sun Microsystems, Java Media Framework API Guide, November 1999.