AN ARCHITECTURE FOR STREAMING CONTROL IN DISTRIBUTED MULTIMEDIA SYSTEMS

R. Canonico, D. Cotroneo, S. D'Antonio, S. Russo, and G. Ventre

Università di Napoli Federico II
Dipartimento di Informatica e Sistemistica
Via Claudio, 21 – 80125 Napoli (Italy)

e-mail: {roberto.canonico, cotroneo, saldanto, sterusso, giorgio}@unina.it

ABSTRACT

The development of distributed multimedia systems can benefit from the availability of a flexible infrastructure able to support the interoperation between components, and capable of dynamically adapt itself to the system configuration. CORBA-compliant distributed multimedia applications permit a high level of accessibility, decentralization and interoperability with other components. However, a large number of non-CORBA-compliant multimedia client applications are already available and commercialized. These applications have been designed according to a protocol-centric interoperation scheme. In this paper we present an innovative architecture for streaming control which allows client applications implementing the standard RTSP multimedia streaming protocol access a Multimedia Storage Server which provides a set of services through a CORBA interface. Our proposed scheme is particularly suitable in the case of proxy-based scenarios, where clients do not directly interact with the storage server, but receive streams from an intermediate caching element.

1 INTRODUCTION

Multimedia Storage Servers play a fundamental role in the forthcoming scenario of a world-wide distributed infrastructure enabling the provision of multimedia services to end-users. Due to the peculiar requirements imposed by multimedia applications to distributed systems, the design of most such servers has been primarily marked by the necessity of achieving reliable and predictable performance with maximum efficiency, thus leading to the adoption of proprietary solutions in their architecture. Further requirements for multimedia service provision over world-wide systems are high flexibility and extensibility. These issues have been addressed by adopting standard middleware architectures, such as the Common Object Request Broker Architecture (CORBA), defined by the Object Management Group (OMG). CORBA provides an object-oriented infrastructure that allows object to communicate, regardless of the specific platforms and techniques used to implement these objects [1]. The use of CORBA as a communication middleware enhances application flexibility and portability, by automating common network programming tasks, such as service location and object activation. CORBA provides the basic mechanisms for remote object invocation, through the Object Request Broker (ORB), as well as a set of services for object management, e.g. Naming Services, Transaction Service and Event Service [2]. The interface of a CORBA object is defined in a standard definition language, the Interface Definition Language (IDL). The advantages of an interface-based design are flexibility, extensibility and pluggability.

The OMG group is currently standardizing a growing number of common services. The Telecommunication Workgroup of OMG has defined in [3] a standard IDL interface to be implemented in CORBA-based multimedia servers. This interface

defines a set of services for multimedia stream control.

On the other hand, several companies have supported the definition of standard protocols for client-server interaction. These protocols can be roughly classified in three areas: data transport, command and control communication, and network/application signalling. This kind of interaction does not conform to an object-oriented paradigm. However, a large number of commercial applications already implement different standard 'command and control' protocols [8] (e.g. RTSP defined by IETF [9], and DSM-CC defined by ISO/IEC [10]). As for the transport of media data from the server to the client, either standard (e.g. RTP [11]) or proprietary solutions (e.g. Real Networks' RDT) can be found in commercial products.

This paper presents a new scheme which enhances the accessibility and flexibility of the services provided by a CORBA-compliant Multimedia Storage Server by allowing such services to be accessed also through standard streaming protocols, and in particular via the RTSP protocol. Hence, our architecture allows to access the services not only by a CORBA-compliant client, but also by non-CORBA-compliant commercial applications. We believe that with this approach we can achieve interoperability and still keeping the advantages provided by CORBA. We present a prototype of the proposed architecture, that allows the access of a cluster-based Multimedia Storage Server (MuSA) [4] by RTSP clients.

The rest of the paper is organized as follows. In section 2 we describe the architecture of MuSA, a Multimedia Storage Server which provides a CORBA service interface for streams control and resource management. In section 3 we describe a scheme to access the MuSA services by a non-CORBA client application, by means of the IETF-standard RTSP protocol. In section 4 we discuss the interoperability issues and the multi-personality capability made possible by adopting our proxy-based architecture. In Section 5 we conclude the paper by discussing the rationale of our work.

2 MuSA ARCHITECTURE AND SERVICES

In this paper we illustrate a scheme to enhance the accessibility of a CORBA-based Multimedia Storage Server through standard streaming protocols. To substantiate our scheme we have implemented it in MuSA, a real server that we have developed at University of Napoli, in the framework of the MOSAICO national research project [5].

MuSA is a cluster-based architecture for highly scalable multimedia servers. A MuSA server is a mix of hardware and software components. The hardware architecture consists of a cluster of commodity PCs, interconnected by a high performance network (HPN). The software architecture, instead, is made of a set of functional modules, which communicate over the internal server interconnection via message passing, according to the MPI standard [6]. The MuSA server prototype on which we have developed our work is made of Symmetric Multi Processor PCs (SMPs), equipped with 2 Pentium-II CPUs, 512 MB RAM, and Ultra-2 Wide SCSI disks. The interconnection network, i.e. the network through which the nodes of the cluster communicate, is Myrinet [7].

The peculiarity of the MuSA server architecture stems from the functional decomposition which originated its design. The identification of functional modules was guided by the observation that the streaming activity, which is the most demanding for a Multimedia Storage Server, involves interaction of two different subsystems: storage devices and network interface. To preserve the system efficiency, data retrieval from the storage subsystem and media quanta transmission to clients are carried out by two different modules in MuSA. Content management and interaction with clients for control of active streams are also distinct activities, which are managed by other modules. The overall architecture of MuSA is shown in Fig.1. From a functional point of view, it is possible to distinguish two different parts in MuSA:

- server front-end, consisting of the MDDB, SFE and GW modules;
- server back-end, consisting of the SS, ACM and DS modules.

These two parts are briefly described in the following two subsections.

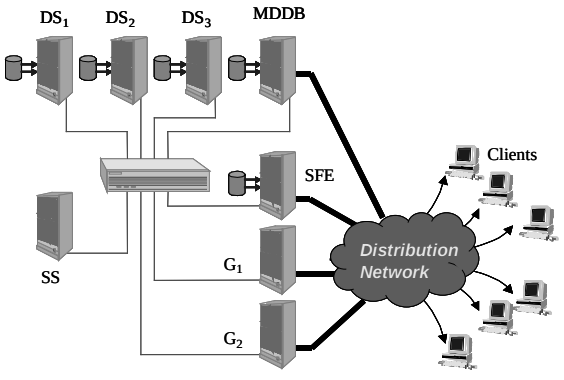

Figure 1. Architecture of the MuSA server.

2.1 MuSA server front-end

The server front-end is made of those modules that directly interact with the client application. Hence, these modules implement a standard service interface.

The *Metadata Database* (MDDB) module provides clients with the full catalogue of available documents. A user can select the document to be transmitted through a web-like user interface. Once the user has selected the desired document, a Document Description file is transmitted to the client application.

The *Server Front End* (SFE) is a CORBA module which offers a set of services to access the server resources by client applications. It also provides management and configuration services to coordinate the server activity in a distributed scenario. The SFE module is responsible of management and control of active streaming sessions. It provides a subset of the OMG Telecommunication Workgroup standard interface in order to make it accessible from any CORBA-client which implements the standard interface. The VCR-like methods used to control document playback were designed after the CORBA standard IDL interface [3]. However, other management services are provided in order to dynamically configure the server resources (e.g. the number of active Disk Servers, or document allocation).

A *Gateway* (G) module receives chunks of multimedia documents from Disk Servers and transmit them as streams to clients. A single MuSA server can be configured with several G modules, instantiated on different nodes of the cluster. If the G modules implement the same transport protocol, the server load can be balanced among them. Another scenario, with multiple Gateway modules implementing different transport protocols, is also possible. In this latter case, the choice of the transport protocol to be adopted depends on the characteristics of the distribution network and/or the characteristics of the client application. When the access network is IP-based, either the standard RTP transport protocol or a proprietary transport protocol for continuous media (e.g RealNetworks' RDP) or both could be implemented. In a LAN scenario, where IP packets do not cross routers, even a raw UDP encapsulation could be adopted (this is the case of our prototype). Finally, if the distribution network is an ATM network, a native AAL5 transport may be implemented.

2.2 MuSA server back-end

The server back-end comprises modules that do not have a direct interaction with clients. A custom design intended to maximize efficiency has been adopted for these modules.

The *Disk Server* (DS) module performs physical access to the local storage subsystem and transfers data to the Gateway module. The DS module does not provide any timing control, but it relies on the SS module for isochronous data pumping.

The *Server Scheduler* (SS) module is the heart of a MuSA server. Its main role is the orchestration of other nodes' activities. The SS maintains the status of client sessions and periodically sends commands to other modules to guarantee regular delivery of continuous data. It is also responsible of assigning a stream_ID to each streaming session.

The *Admission Control Module* (ACM) performs the admission control test, i.e. it decides if enough resources are available in the server to accept a new service request, and gives directives to the SS about the most suitable DS and GW modules to be assigned to each streaming session.

3 A PROXY-BASED ARCHITECTURE TO SUPPORT THE RTSP PROTOCOL

To date, several multimedia client applications and Multimedia SDKs (e.g. Real Player, CISCO IP/TV Viewer, Apple QuickTime) are not CORBA-compliant, but they adopt some standard streaming protocol such as RTSP/RTP or DSM-CC.

The MuSA server was adopted as a case-study to investigate the possibility of enhancing the interoperability of a CORBA-based multimedia server through a standard command and control protocol, namely RTSP [9]. RTSP, *Real Time Streaming Protocol*, is an application-level client-server protocol which enables controlled delivery of streamed multimedia data over IP networks, between a Multimedia Storage Server and clients. It provides "VCR-style" remote control functionality for audio and video streams, like pause, fast forward, reverse, and absolute positioning. A client application can use RTSP to control a stream which may be sent via a separate protocol, independent of the control channel. For example, RTSP control may occur on a TCP connection, while data flows via UDP.

In RTSP, each presentation and media stream is identified by an RTSP URL (*Universal Resource Locator*). The overall presentation and the properties of the media are defined in a presentation description file, which may include the encoding, language, RTSP URLs, destination address, port, and other parameters. Such a description file can be retrieved by the client application from a web server (via the HTTP GET method). RTSP is a text-based protocol. Client and server communicate via

messages, according to a request/reply scheme (see Figure 2).

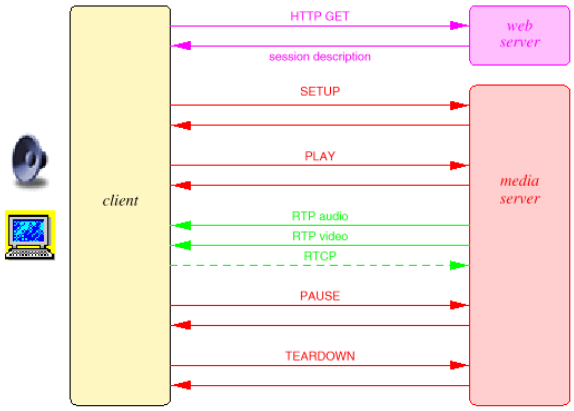

Figure 2. RTSP protocol messages.

In order to improve the flexibility of MuSA, by making it accessible from any commercial multimedia client which uses the RTSP protocol, we present a proxy-based architecture based on a component named RTSP_proxy. The aim of this component is to map the RTSP protocol logic into a corresponding sequence of methods provided by the SFE CORBA-interface. Figure 3 depicts the overall architecture, which allows the MuSA server to be accessed by both CORBA clients and RTSP clients. RTSP clients access the server through the proxy component, which holds their sessions status, and translates each RTSP message into the corresponding service provided by the SFE CORBA module. Since the MuSA frontend implements a subset of the standard OMG interface for the control of Audio/Video flows, its services are also accessible from any CORBA-compliant client which implements the standard interface.

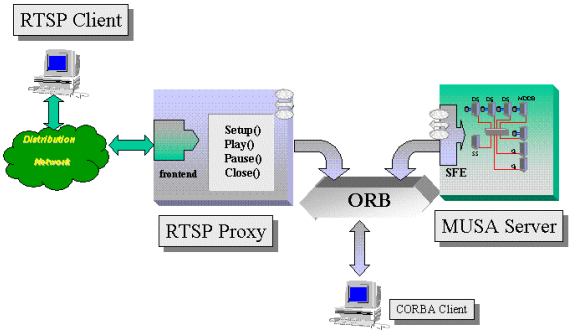

Figure 3. The RTSP proxy architecture.

It is worth noting that the RTSP proxy acts as a middle tier only for the command and control tasks, while media delivery is carried out directly from the Gateway modules and the client, via the selected transport protocol (e.g. RTP).

At startup the RTSP_proxy needs a number of configuration values, stored in the file RTSP_proxy.cfg. This file contains the following information:

- **doc_ID:** RTSP presentation identifier;
- **fileID:** identifier of the media file belonging to the presentation;
- **doc_path:** RTSP URL of the presentation;
- **media_type:** media type (video, audio, etc…);

The RTSP_proxy also maintains an archive of document description files, encoded in the SDP description format [10]. Hence, each document is associated to an sdpfile. All these values are organized in a URL_t_LIST object (see Figure 5). By means of this object, the RTSP_proxy keeps track of all the resources available on the server.

RTSP_proxy.cfg

```
     doc_ID    doc_path          media_type fileID

DOC= AB00003   music/songs.mp3     -----     MuSA.000
DOC= AA00001   inet/routers.mpg    audio     MuSA.001
DOC= AA00001   inet/routers.mpg    video     MuSA.002
DOC= AA00002   cmc/customer.mpg    video     MuSA.003
DOC= AA00003   cmc/1000kbs.mpg     video     MuSA.004
DOC= AA00003   cmc/1000kbs.mpg     audio     MuSA.005
DOC= AB00001   ecom/market.avi     video     MuSA.006
DOC= AB00002   ecom/epsilon.avi    video     MuSA.007
```

Figure 4. The RTSP_proxy.cfg file.

As we stated earlier, the main task of the *RTSP_proxy* is to map the RTSP logic onto the CORBA service interface of MuSA. This task is accomplished by taking into account the difference of the concept of 'resource' between these two environments. In particular, in the RTSP context, a 'resource' is conceived as a whole multimedia presentation, which can be composed of multiple media streams. Some RTSP commands address the single media (e.g. the SETUP command), while others address the presentation as a whole (e.g. the PLAY command). The MuSA service interface, instead, considers as a 'resource' a single media file. Methods are provided to deal with the single file entity. For instance, Figure 4 shows that the *RTSP_proxy* associates two distinct MuSA resources (MuSA.001 and MuSA.002) to the same RTSP presentation (AA00001).

The *RTSP_proxy* component has been designed using the OO (*Object Oriented*) methodology. The *RTSP_proxy* class diagram is depicted in Figure 5.

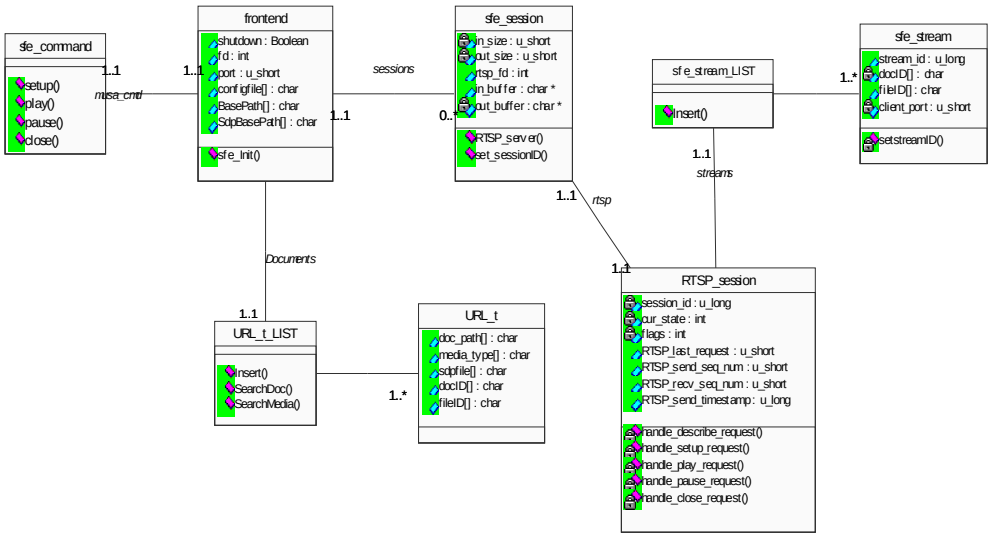

Figure 5. Class diagram of the RTSP_proxy. **component.**

The communication between the client and the *RTSP_proxy* is performed through a TCP connection. The frontend object is responsible for the TCP connection management, by waiting for TCP requests from clients on a well known port (port 554/TCP). Once an incoming request is accepted, the frontend object creates an instance of the sfe_session class. In turn, the sfe_session object creates an instance of the RTSP_session class. The RTSP_session object is responsible of handling the RTSP messages issued by clients for the entire duration of an RTSP session. Each time the user sends an RTSP command, the RTSP_session object maps it into a proper sequence of CORBA sfe_command interface methods. The *RTSP_proxy* component is connected to the MuSA server by means of the ORB (*Object Request Broker*). Once an RTSP command is recognized, the frontend object invokes the correspondent method on a remote sfe_command object, located on the cluster where MuSA resides. For this reason we created a local proxy (stub) on the *RTSP_proxy* which delivers requests to the remote object. The sfe_command skeleton object executes the method, by translating it in a number of MuSA-specific internal operations. sfe_command stub and skeleton objects are generated from the SFE IDL interface, which provides a subset of services defined in the standard document issued by the OMG Telecommunication Workgroup [3]. The scheme presented in Figure 3 has been implemented in C++ and integrated in MuSA using Orbix, a commercial CORBA-compliant ORB developed by IONA Technologies.

The object interaction in a setup scenario is described in the sequence diagram depicted in Fig. 6.

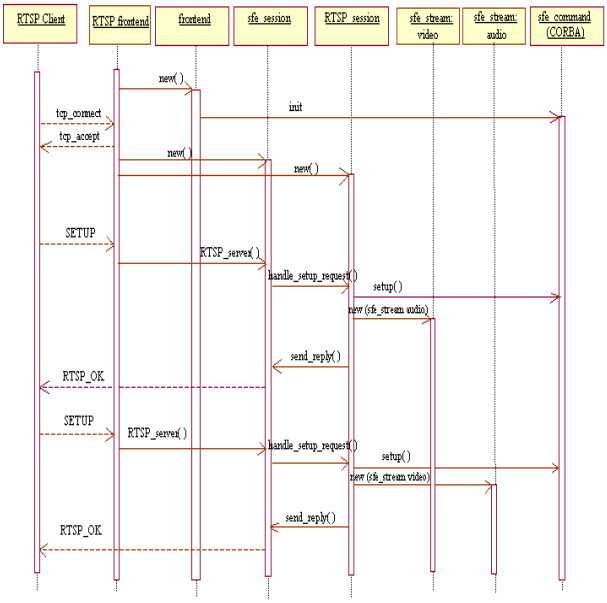

Figure 6. Sequence diagram in the SETUP scenario.

As Figure 6 shows, once the client has transmitted the RTSP SETUP command, the RTSP_session object handles the setup request (*handle_setup_request* method), by extracting from the message, all the information it needs for the instantiation of a new *sfe_stream* object. It is worth noting that Session_IDs are not generated by the RTSP_frontend object, but they are returned from the

MuSA SFE, since they are to be unique for the entire system, and the *RTSP_proxy* is not aware of the whole MuSA server status. Finally, Figure 6 indicates that two SETUP commands are needed to establish an audio and video presentation.

Figure 7 depicts the sequence diagram in the PLAY scenario, with regard to the same presentation considered in Figure 6. Notice that, the RTSP PLAY command involves two distinct play() invocations, one for the audio stream and the other for the video stream.

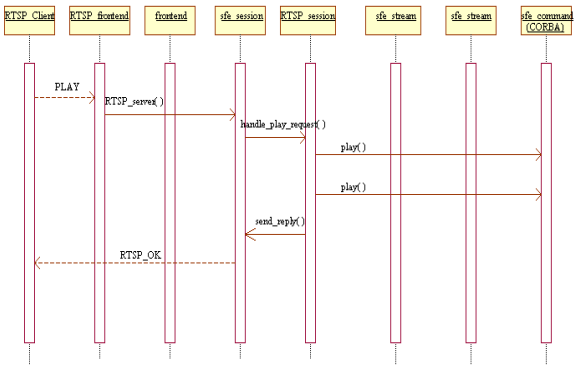

Figure 7. Sequence diagram in the PLAY scenario.

The interface depicted in Figure 3 refers only to VCR-like control services, since the RTSP protocol is intended to provide only this kind of interaction with the server. However, the MuSA SFE module also implements management services (such as document uploading and internal resource configuration). For these services, we believe that a CORBA interface is more suitable.

We claim that the proxy-based architecture presented is general enough to be adopted in different scenarios, to allow the integration between standard "Command and Control" protocols and CORBA-based servers.

4 INTEROPERABILITY ISSUES

The large number of non-CORBA commercial client applications justifies the introduction of an RTSP proxy in the MuSA architecture. This choice has a further positive aspect in that it leads to lightweight applications at the client side that do not require a CORBA middleware infrastructure. In some operational scenarios, this choice has a crucial impact on the application performance. This is particularly the case for applications running on mobile computing devices.

Despite the fact that standard protocols had been designed in order to build interoperable applications, interoperability between client and server products from different vendors is still far from being achieved, due to the fact that commercial applications usually extend the protocol mechanisms to implement their own functionalities. The architecture presented in this paper, which is based on a neutral CORBA-based inner Service Interface, integrated with a proxy element, allows the server to achieve multi-personality capabilities.

5 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a proxy-based architecture, which enhances CORBA-based multimedia servers in order to support standard streaming-protocols. In particular, we have presented an implementation of this scheme which allows commercial client applications interact with a CORBA Multimedia Storage Server through the standard RTSP protocol. We believe that our scheme is sufficiently general to be adopted in different scenarios, where other communication protocols (e.g. DSM/CC) and/or other service interfaces are used. Hence, our scheme can be extended into a general architecture, which enables the communication between a protocol-centric scheme and service-centric software architectures. Our approach recognizes the usefulness of implementing CORBA services in modern distributed servers, in order to achieve reusability, portability, high flexibility, platform and location independence and web-based accessibility.

REFERENCES

[1] Object Management Group. "The Common Object Request Broker: Architecture and Specification. 2nd edition", OMG, July1995.

[2] Object Management Group. "CORBA Services: Common Services Specification. 95-3-31 Edition", OMG, March 1995.

[3] OMG, Object Management Group. "CORBAtelecoms: Telecommunications Domain Specifications - Control and Management of Audio/Video Streams - Version 1.0", June 1998.
Document available at: http://www.omg.org

[4] R. Canonico. "A Cluster-Based Architecture for Scalable Multimedia Storage Servers", Ph.D. Thesis. University of Napoli Federico II, Italy, November 1999.

[5] R. Canonico, G. Capuozzo, G. Iannello, and G. Ventre. "MuSA: a scalable multimedia server based on clusters of SMPs", Proc. WCBC '99, Int. Workshop on Cluster-Based Computing, Rhodes (Greece), ACM Press, June 1999.

[6] Message Passing Interface Forum. "Document for standard message-passing interface", Tech.

Rep. CS-93-214, Univ. of Tennessee, Nov. 1993.
Project web site: http://www.mpi-forum.org

[7] N.J. Boden, D. Cohen, R.E. Felderman, A.K. Kalawik, C.L. Seitz, J.N. Seizovic, and W.K. Su. "Myrinet – a gigabit-per-second local-area network", IEEE Micro, vol. 15 no. 1, pp. 29-36, Feb. 1995.

[8] C.Severance. "Standardizing Real-Time Streaming Protocols", IEEE Computer, July 1998.

[9] H. Schulzrinne, A. Rao, and R. Lanphier. "Real Time Streaming Protocol (RTSP)", Internet Engineering Task Force RFC 2326, April 1998.

[10] Digital Storage Media Command and Control International Standard, ISO/IEC JTC 1.29.13.06 (14496-6).

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications", Internet Engineering Task Force RFC 1889, January 1996.