Design for an Integrated Streaming Framework*

Technical Report CS-99-30

Chan-Gyun Jeong[1], Hyung-Ill Kim[1], Young-Rae Hong[1], Eak-Jin Lim[1], Sungyoung Lee[1], Jongwon Lee[1], Byeong-Soo Jeong[1], Doug-Young Suh[1], Kyoung-Don Kang[2], John A. Stankovic[2], and Sang H. Son[2]

[1] Dept. of Computer Engineering, Kyung Hee University
[2] Dept. of Computer Science, University of Virginia

Abstract

This paper describes a design for an integrated streaming framework that can provide an environment to develop multimedia streaming applications under heterogeneous distributed systems. Our streaming framework was designed to extend the limits of existing streaming systems while offering easy interaction with other streaming systems, to support for diverse media formats and networks, and to provide high level programming environment for streaming application developers. The framework is also compatible with global real-time multimedia DBMS (BeeHive) so that streaming media is efficiently retrieved, stored, and serviced using database systems.

Keywords: streaming, multimedia, audiovisual, real-time, database

1. Introduction

Streaming is a technology for multimedia communications that makes it possible to deliver and display multimedia (such as audio and video) data in real-time. By using streaming technology, users can access multimedia contents immediately upon demand without waiting for the whole file to download. It is well suited to the transmission of video data that has real-time characteristics and requires high bandwidth communication. It might be the only practical solution possible for live video streams.

Even though numerous streaming systems have been designed and implemented, most systems do not consider interoperability with other streaming systems and do not support diverse media formats and network interfaces. Thus they lack flexibility, extendibility, and network transparency. The increasing number of coding and compression standards has led to a situation where heterogeneity is a growing problem. If a server uses many different coding formats, clients must be able to decode these formats. In order to relieve streaming applications from interoperability and heterogeneity concerns, a streaming framework should provide diverse audio and video CODEC and network interfaces.

In this paper, we present an approach to designing an integrated streaming framework which can overcome the limitations of existing streaming systems while providing diverse audio and video CODEC and the ability to run adaptively on different operating systems and networks. This paper proceeds as follows. Section 2 provides an overview of existing streaming frameworks that are similar to our system. We describe the system architecture of our integrated streaming framework in Section 3 and finally states our conclusion in Section 4.

2. Related Work

This section briefly describes existing streaming systems that are related with our work. Most of the systems mentioned below are the programming environments that can efficiently manipulate real-time multimedia streams like audio and video.

2.1 CMT (Continuous Media Toolkit)

CMT is a multimedia programming toolkit developed at MIT, which provides a programming environment for rapid development of continuous media applications [1]. The overall system architecture of CMT consists of three layers, Application Code Layer, CMT Library Layer, and Resource Layer as shown in Figure 1. At the top level, the application code layer represents the programming environment

* Supported in part by contract IJRP-9803-6 from the Ministry of Information and Communication of Korea

provided to application developers. The middle layer is the CMT library layer that provides the Tcl interpreter for continuous media objects and a variety of services. The bottom layer is the hardware and operating system resources layer which handles the CPU, network interface, memory, I/O devices, and other hardware. CMT has the difficulty of not being easily extended since its internal structure is very complicated and flat; furthermore, it does not follow the object-oriented programming paradigm.

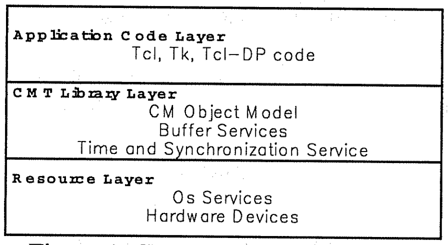


Figure 1. CMT System Architecture

2.2 VuSystem

VuSystem is a programming environment developed at MIT, which facilitates the development of compute-intensive multimedia applications that combine intelligent media processing with traditional capture and display [2]. In other multimedia applications including on-line encyclopedia and video-conferencing systems, the computer acts as a mediator between the application author and user, in the case of on-line encyclopedia, or between two users in the case of video-conferencing applications. In contrast, VuSystem is compute-intensive multimedia system where the computer is an active participant in the processing of audio and video data.

As shown in Figure 2, VuSystem is divided into two parts, *in-band* and *out-of-band* for the purpose of optimizing each partition separately. The *in-band* partition is implemented as a set of C++ classes that comprise the essential modules to process audio and video data, including a source, a sink, a filter, or some other module. The *out-band* partition is programmed in the Tcl, an interpreted script language, while performing event handling and other higher-order functions of a program. However, VuSystem does not provide RTP and Multicast and its user interface has very limited functionality. Furthermore, since VuSystem does not support media compression, it cannot guarantee QoS in the environment of heterogeneous networks having different bandwidths.

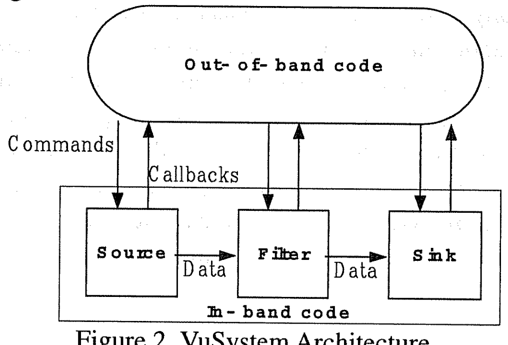
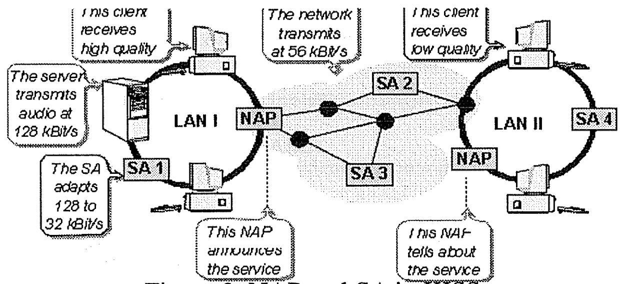


Figure 2. VuSystem Architecture

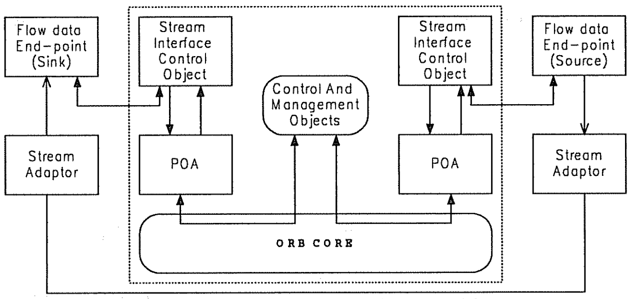
2.3 KISS (Communication Infrastructure for Streaming Services)

KISS is a communication infrastructure for streaming services that allows the transparent integration of network service applications and adapts real-time content streams to network conditions and/or individual client requirements [3]. As shown in Figure 3, KISS provides application-transparent network services that allow user applications to forget about networking-related problems. These services come with the implementation of NAP (Network Access Point) which acts as a mediator between end-system applications and the service network. Although KISS provides network transparency for streaming application developers, it has the drawback of supporting only audio streams such as PCM and MPEG-1 audio layer 3 and it suffers from the performance overhead of NAP.


This client receives high quality
The network transmits at 56 kBit/s
This client receives low quality
The server transmits audio at 128 kBit/s
LAN I
NAP
SA 2
LAN II
SA 4
NAP
The SA adapts 128 to 32 kBit/s
SA 1
SA 3
This NAP announces the service
This NAP tells about the service
Figure 3. NAP and SA in KISS

2.4 CORBA A/V Streaming Service on TAO (The Ace Orb)

To facilitate the development of standard-based distributed multimedia streaming applications, the OMG (Object Management Group) has defined a CORBA-based specification for the control and management of A/V streams [7], based on the CORBA reference model [5]. Figure 4 shows the OMG A/V streaming architecture, which is represented as a flow between two flow data endpoints. One endpoint acts as a source of the data and the other endpoint acts as a sink. In the OMG streaming architecture, the control and signaling operations pass through the GIOP/IIOP-path of the ORB, demarcated by the dashed box. By contrast, the data stream uses out-of-band streams, which can be implemented using protocols that are more suitable for multimedia streaming than IIOP.


Flow data End-point (Sink)
Stream Interface Control Object
Control And Management Objects
Stream Interface Control Object
Flow data End-point (Source)
Stream Adaptor
POA
POA
Stream Adaptor
ORB CORE
Figure 4. OMG A/V Streaming Architecture

Washington University has developed the first freely available implementation of the OMG A/V streaming model using TAO [6], which is a real-time CORBA ORB that has been ported to most OS platforms [4]. However, the CORBA A/V streaming service based on TAO does not implement the OMG specification completely. It implements a simple MPEG video player that has session control functions only. Thus, it cannot be used as a streaming framework.

3. Integrated Streaming Framework

In this section, we describe an approach to designing an integrated streaming framework that can provide flexibility, extendibility, and adaptability. Our streaming framework aims to support diverse audio and video CODEC and adaptively run on different operating systems and network environments. As shown in Figure 5, the overall system architecture of our streaming framework consists of two parts, Streaming Applications and ISSA (Integrated Streaming Service Architecture) which comprises a Content Manager, a Session Manager, a Transport Manager, a Resource Manager, a Media Manager, and Gateway Modules. MOA (Media Object Architecture) resides between them. ISSA performs the essential functions for multimedia streaming services and MOA provides an API (Application Programming Interface) as object-oriented wrapper classes which can be easily used by streaming application developers. VOD (Video On Demand) server and client programs developed by using MOA can be one example of the Streaming Applications layer.

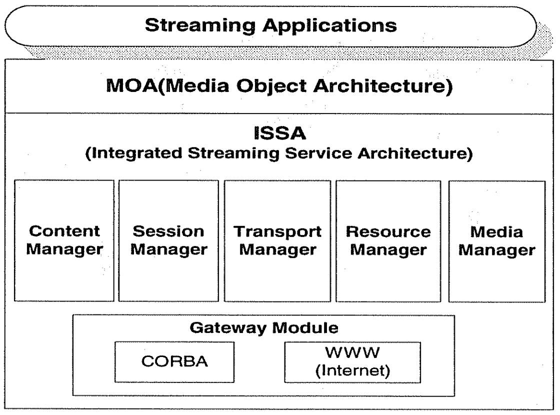


Figure 5. Architecture of Integrated Streaming Framework

3.1 Transport Manager

The Transport Manager performs packetization and depacketization of multimedia data represented by diverse media formats and is responsible for transmitting such packetized media data in real time through the diverse network environments. These functions are accomplished by implementing RTP (Real-time Transport Protocol) [8] that was developed by IETF (Internet Engineering Task Force) in 1996 (RFC 1889) as an application level protocol for multimedia transport. Figure 6 shows the network interface of Transport Manager.

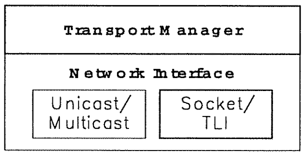


Figure 6. Network Interface of Transport Manager

In order to provide network transparency, the Transport Manager sends and receives multimedia data through the network interface, which can operate independently of network conditions. The Network interface, a kind of network wrapper interface, can support IP Unicast/Multicast and Socket/TLI while working independently of lower layer network interfaces. Figure 7 describes the transmission protocol stack of Transport Manager and Session Manager that is explained in the next subsection.

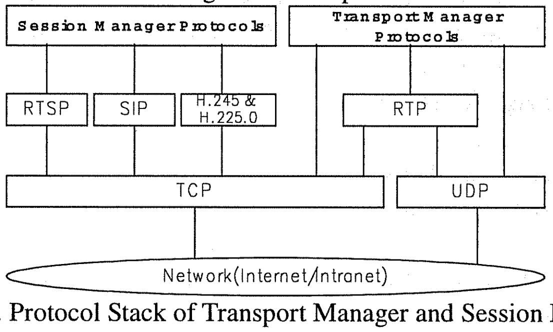

Session Manager Protocols
Transport Manager Protocols
RTSP
SIP
H.245 & H.225.0
RTP
TCP
UDP
Network(Internet/Intranet)

Figure 7. Protocol Stack of Transport Manager and Session Manager

3.2 Session Manager

The Session Manager performs session control functions between streaming application programs, such as session creation/destruction. It is also responsible for streaming media control, such as play-back and stop. These functions are provided by implementing RTSP (Real-Time Streaming Protocol) [9] developed by IETF (RFC 2326) in 1998. RTSP, as an application level session protocol, provides a framework to control on-demand audio/video transmission such as VCR operations. Figure 8 represents a procedure for session establishment of RTSP between session managers of server and client. After session establishment, session control messages, such as PLAY, PAUSE, TEARDOWN, can be transmitted.

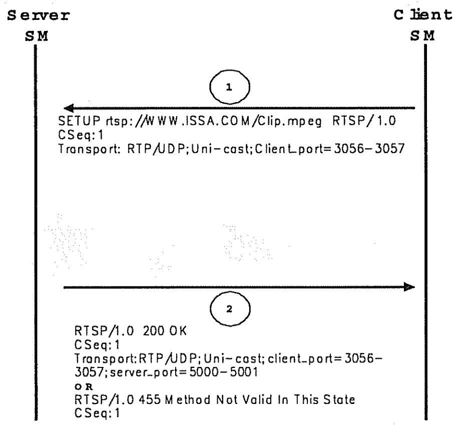

Server SM
Client SM
1
SETUP rtsp://WWW.ISSA.COM/Clip.mpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP;Uni-cast;ClientLport=3056-3057
2
RTSP/1.0 200 OK
CSeq: 1
Transport:RTP/UDP; Uni-cast; client_port=3056-3057; server_port=5000-5001
OR
RTSP/1.0 455 Method Not Valid In This State
CSeq: 1

Figure 8. Procedure of Session Establishment in RTSP

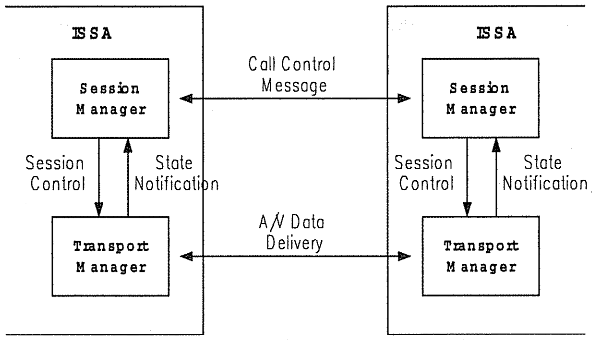


Figure 9. Interaction between Session Manager and Transport Manager

Figure 9 describes the interaction between Session Manager and Transport Manager. When the Session Manager receives a session control message (such as PLAY, PAUSE, and TEARDOWN), it changes the transmission session state according to the control message. The Transport Manager notifies the current state of the transmission session when the state of transmission session changes automatically.

3.3 Media Manager

The Media Manager provides several functions related with media data processing, media file I/O through file systems or DBMS, A/V CODEC to encode/decode diverse media data, and control of audio/video devices. As shown in Figure 10, the Media Manager is divided into two parts, media source and media sink. The Media Source is responsible for transmitting media data to the transport manager. Several media sources, such as media data from DBMS, media files, and live media from camera, are considered as one media source by high level abstraction. The Media Sink is responsible for presenting media data through the appropriate audio/video devices as soon as it receives media data from the transport manager. A/V CODEC provides several media CODEC (e.g., MPEG-1, H.263, G.711, and G.723.1) to the media source and media sink.

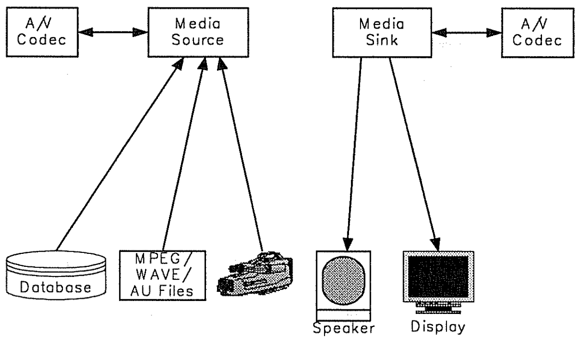


Figure 10. The Structure of Media Manager

3.4 Content Manager

The Content Manager is a component for managing media data such as audio and video. It facilitates interaction with DBMS in order to retrieve and store media data. Figure 11 shows the data flow that processes media data through the DBMS wrapper. A DBMS wrapper is required to interact with different DBMS. Media data retrieved through DBMS can be directly transmitted to the transport manager or transmitted after being transformed into a different media format by A/V CODEC of the media manager.

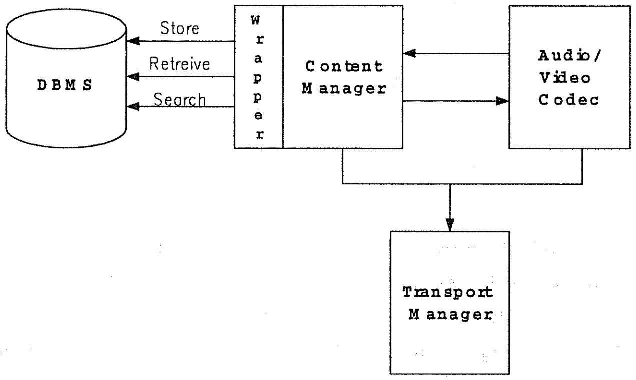


Figure 11. Connection between Content Manager and DBMS

3.5 Resource Manager

The Resource Manager provides QoS related functions, including QoS specifications, mapping, monitoring, memory buffer management, and thread scheduling. The Resource Manager controls QoS statically or dynamically. Static QoS control is achieved during the session establishment of an application level. Dynamic QoS control means that QoS is dynamically adapted during the media data transmission by considering monitored resource information [10, 11].

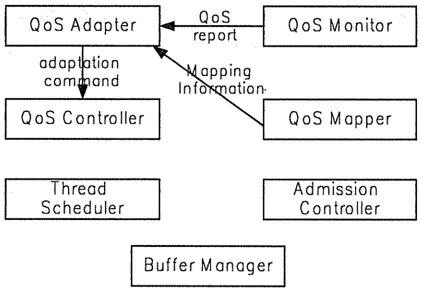


Figure 12. The Structure of Resource Manager

As shown in Figure 12, the Resource Manager consists of several modules: QoS Monitor which traces QoS information of resources, QoS Adapter which performs QoS adaptation according to the monitored QoS information, QoS Controller which controls QoS change for each session, QoS Mapper which provides mapping information about QoS parameters, Thread Scheduler which schedules threads in the stream framework, Admission Controller which decides the admission of new session requests, and Buffer Manager which performs memory buffer allocation and de-allocation.

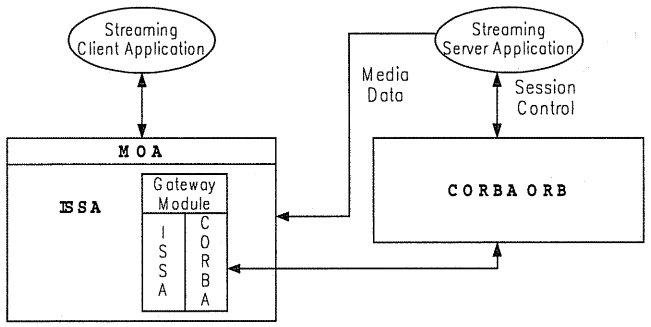


Figure 13. Interworking between ISSA and CORBA Streaming Service

3.6 Gateway Module

The Gateway Module is a component of ISSA that is intended to allow interworking with other streaming frameworks such as the CORBA A/V Streaming Service and to support streaming service in the WWW (World Wide Web) environment. Figure 13 shows interworking between streaming server program on CORBA and streaming client programs on an integrated streaming framework. As the diagram shows, only session control information is transformed by the Gateway Module of ISSA and transmitted through CORBA ORB. Media data is directly transmitted from the streaming server on CORBA to ISSA and displayed by the client program, since transformation overheads might be a problem for real-time media transmission.

3.7 MOA (Media Object Architecture)

MOA provides a high level programming interface for Streaming Application Layers in an Integrated Streaming Framework [12]. The complex API functions that are provided by several components of ISSA, are simplified as class objects of C++ or Java by MOA. Such simplified APIs help users to easily develop streaming application programs.

3.8 Interface Design between BeeHive and ISSA

Due to the well-known advantages of a database system compared to a file system, we use a multimedia system with database service support. BeeHive is the distributed real-time multimedia database system developed in University of Virginia. Since BeeHive supports real-time multimedia transactions in a distributed environment, it is a good match to our system objective in providing real-time multimedia streaming with database services.

In this section we present the interface between ISSA (Integrated Streaming Service Architecture) and BeeHive to show how they work together. ISSA is a streaming framework for easier multimedia streaming application development. It comprises a set of library functions used for developing streaming applications that form client-server architecture. It supports RTP (Real-Time Protocol) and RTSP (Real-Time Streaming Protocol) for media transmission protocol between client and server. In this project BeeHive works as a multimedia server for ISSA, it supports canned transactions to support real-time requirements. The transactions should be pre-analyzed and planned to meet its timing requirements. In order to access database transactions of BeeHive, RPC (Remote Procedure Call) interface is designed/developed between BeeHive and ISSA where BeeHive works as the multimedia server and ISSA works as the client. In this project we design and develop a real-time movie streaming system as to show its practical applicability.

BeeHive supports four transactions to manage movie objects in a coherent manner. Since it is object-oriented database system not any other method except specified in the movie object can access these transactions. It provides four transactions for the movie class: Read, Write, Find, and Play. Since we have already found out the major problems for the integration of BeeHive and ISSA we can easily extend the database system to support more transactions in the future.

A user can search the multimedia database to find movies fitting his/her interest using the Find transaction. Using the Read transaction, user can get the attributes about a movie such as the title, director, etc. A user can modify attributes of a movie such as the rating of movie by the Write transaction. Using the Play transaction, a user can play back the movie across the network.

Interoperable interface model between ISSA and BeeHive consists of Transaction Interface and Streaming Interface. Transaction Interface deals with a series of functions which ISSA client requests for transactions of BeeHive. Streaming Interface provides a mechanism to stream media between ISSA client and server. Transaction Interface is composed of GUI part that handles transaction requested from end-user, RTTP (Real-Time Transaction Protocol) part which deals with transaction processing between ISSA client and server, and RPC part which is an interface between ISSA server and BeeHive. Streaming Interface consists of a part that relates with media display of ISSA client, a part that controls RTSP and RTP session between ISSA client and server, and a part that handles media stream transmission between ISSA server and BeeHive. All functions previously mentioned are contained in BeeHive Connector that resides in Content Manager of ISSA server.

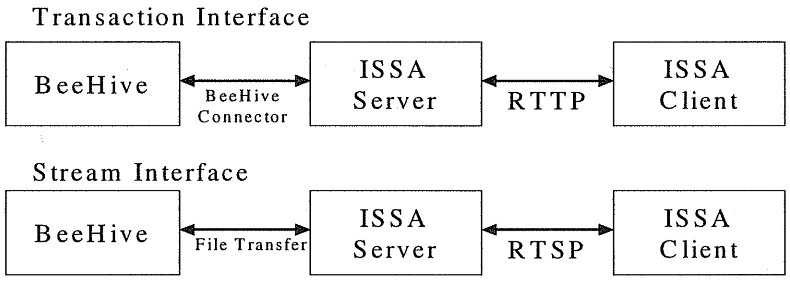


Figure 14. Transaction and Stream Interfaces between ISSA and BeeHive

3.8.1 Transaction Interface

In order to utilize BeeHive transaction in ISSA environment, we add BeeHive transaction interface to Content Manager of ISSA. Content Manager of ISSA provides database interface with BeeHive and processes media source request from ISSA client. We use RPC mechanism for interaction with BeeHive and devise RTTP (Real-Time Transaction Protocol) for transaction processing between ISSA client and server. RTTP is a protocol that handles transaction requests for BeeHive in ISSA environment. But it is not directly related with BeeHive. The syntax of RTTP is similar to HTTP. RTTP is easily extended, even though it handles only four kinds of transactions right now. Figure 15 represents the structure of RTTP protocol.

Request:

[TransactionName] rttp://[HostName] [RTTP]/[VersionNo]
[ParameterName]:[Value]
...

Response:

BHTP/[VersionNo] [Status] [Response Phrase]
[ParameterName]:[Value]
...

Figure 15. RTTP Request and Response structure

Based on the above structure, we implement four transactions, Read, Write, Fine, and Play. For example, Read transaction has the format of request and response as follows:

```
READ rttp://host RTTP/0.1
Object:32414
Attribute:Title

RTTP/0.1 200        OK
Content-length:

Title:Terminator II
Rating:Good
```

3.8.2 Streaming Interface

For media streaming interface, when BeeHive transmit media file name through RPC interface after storing media data into file, Content Manager of ISSA reads the media file and streams the media data to ISSA client. The structure of Streaming Interface is as follows:

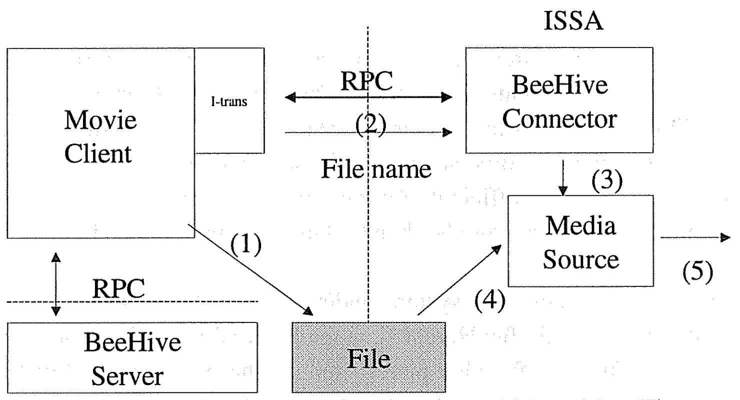


Figure 16. Streaming Interface between ISSA and BeeHive

3.8.3 Transaction Flow

Transactions between ISSA and BeeHive are classified into two cases, i.e. one requires media streaming service, and the other does not require media streaming. Read, Write, and Find transactions do not require media streaming, but Play transaction requires media streaming service. In the case of no media streaming, only RTTP protocol is need to process transactions. But in the other case (which requires media streaming service), RTTP protocol and RTSP protocol are simultaneously used.

3.8.4 Integrated Architecture

The interoperable architecture that works through transaction interface and streaming interface mentioned by now can be represented as follows:

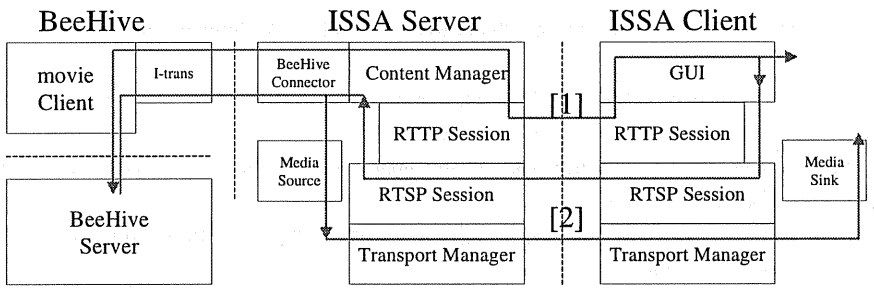


Figure 17. Integration Architecture

3.8.5 Basic Implementation

In order to implement the interoperability between ISSA and BeeHive, 1) we firstly port ISSA libraries on Solaris, 2) implement BeeHive connector based on RPC mechanism, and finally make an experimental application that can show the interoperability between them. As an application, we implement a small size VOD server and Video client. We develop server by using g++ on Solaris. Video client is developed by using Visual C++ 6.0 on Windows NT. The implemented modules are as follows:

- ♦ ISSA library for Solaris (porting part)
  - ISSA VideoServer for Solaris
  - ISSA VideoClient for Windows NT
- ♦ ISSA and BeeHive Integration
  - ISSA BHVideoServer for Solaris
  - ISSA BHVideoClient for Windows NT
  - BeeHive Connector module (based on RPC)
  - BeeHive Video Object Client

4. Conclusions

This paper describes an integrated streaming framework that integrates heterogeneous environments and work easily together with other streaming systems. Our streaming framework gives more robustness, flexibility, and extendibility than existing streaming systems since it is designed according to an object-oriented paradigm. It also supports diverse media formats and heterogeneous network environments. Simplified APIs by MOA provide efficient programming environment for the streaming application developer. Our streaming framework can also handle large amounts of media data through interworking with DBMS.

In the future, we plan to implement our system under diverse OS platforms, such as UNIX and MS-Windows, and to integrate it with BeeHive[13], a global real-time database system which is under development at the U. of Virginia. We also plan to reduce the performance overhead of our system as much as possible since we believe that performance is a major factor to guarantee QoS in multimedia systems.

5. References

[1] K. Mayer-Patel, and L. A. Rowe, "Design and Performance of the Berkeley Continuous Media Toolkit", in Multimedia Computing and Networking 1997, in Proc. SPIE 3020, pp 194-206, 1997.

[2] C. J. Lindblad, and D. L. Tennenhouse, "The VuSystem: A Programming System for Compute-Intensive Multimedia", in IEEE Journal of Selected Areas in Communications, 1996.

[3] K. Jonas, M. Kretschmer, and J. Modeker, "Get a KISS - Communication Infrastructure for Streaming Services in a Heterogeneous Environment", in Proc. of ACM Multimedia '98, Bristol, UK, pp. 401-410, 1998.

[4] S. Mungee, N. Surendran, and D. C. Schmidt, "The Design and Performance of a CORBA Audio/Video Streaming Service", in Proc. of the 32st Hawaii International Conference on System Systems(HICSS), Hawaii, January 1999.

[5] Object Management Group, The Common Object Request Broker: Architecture and Specification Revision 2.2, February 1998.

[6] D. Schmidt, D. Levine, and S. Mungee, "The Design and Performance of Real-time Object Request Brokers", Computer Communications, vol. 21, pp.294-324, April 1998.

[7] Object Management Group, Control and Management of A/V Streams specification, OMG Document telecom/97-05-07 ed., October 1997.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, IETF RFC 1889, January 1996.

[9] H. Schulzrinne, A. Rao, and R. Lanphier, Real-Time Streaming Protocol (RTSP), IETF RFC 2326, April 1998.

[10] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A Survey of QoS Architectures", ACM/Springer Verlag Multimedia Systems Journal , Special Issue on QoS Architecture, Vol. 6 No. 3, pp. 138-151, May 1998.

[11] S. N. Bhatti and G. Knight, "Enabling QoS adaptation decisions for Internet applications", Journal of Computer Networks, Vol. 31, No. 7, pp. 669-692, March 1999.

[12] Hyung-Ill Kim and Sungyoung Lee, "Design of Media Object Architecture to Support Multimedia QoS", In Proc. of Korean Information Science Society 98, pp. 699-701, April 1998.

[13] J. Stankovic, S. Son and J. Liebeherr, "BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications", In Proc. of Second Workshop on Active Real-Time Databases, Lake Como, Italy, September 1997.