Building Scalable and High Efficient Java Multimedia Collaboration

Wenjun Wu, Tao Huang, Geoffrey Fox
Community Grids Computing Laboratory, Indiana University, USA

{wewu ,taohuang, gcf }@indiana.edu

ABSTRACT

Java Media Framework (JMF) is platform-independent multimedia programming framework which enables easy and fast development of collaborative applications. This paper describes our work on building a high efficient multimedia collaboration system using JMF. We introduce a new rendering approach to optimize the performance of JMF and add screen capturing capability as well as new codecs. Based on this enhanced framework, a high efficient and platform-independent conferencing client named Global-MMCS AVPortlet is developed. The performance evaluation shows that it outperforms other well-known video collaboration tools.

KEYWORDS: JMF, RTP, Multimedia Programming, Render, Screen capture

1. INTRODUCTION

Collaboration and videoconferencing systems have become very important applications in the Internet. Since there are so many different technical solutions to multimedia collaboration, the issues of the interoperability and platform-independency are emerging to build a general collaboration environment. And they should be addressed in a unified and service-oriented framework.

Java as a cross-platform programming language is a very promising candidate enabling developers to build such a collaborative multimedia systems. It has an elegant multimedia programing framework named Java Media Framework (JMF) [1], which provides a unified architecture, messaging protocol and a Java API for accessing underlying media frameworks, managing the acquisition, processing, and delivery of time-based media data. By exploiting the advantages of the Java platform, JMF delivers the promise of "Write Once, Run Anywhere" to multimedia developers. Although JMF has been used in some research projects on tele-collaboration, these research works are limited in building a scalable and high efficient Java multimedia collaboration system. Because they are still either based on traditional client-server or multicast communication architecture, and make few improvement in the performance of JMF.

To build a Service-Oriented multimedia collaboration system, we proposed XGSP (XML based General Session Protocol) [2] as a common, interoperable, Web-Serviced based framework. XGSP uses a unified, scalable, robust "overlay" network is to support audiovisual and data group communication over heterogeneous networking environments. Based on this framework, we have developed a prototype system called Global Multimedia Collaboration System (Global-MMCS) [3] to support scalable web-service based interoperable collaborations. Global-MMCS integrates various services including videoconferencing, instant messaging and streaming, and supports multiple videoconferencing technologies such as H.323, SIP and Access Grid clients [4].

JMF is the key building block for the implementation of Global-MMCS. The media services including video, audio mixing, snapshot generation, are all developed using JMF library. Furthermore, we also built our own audiovisual collaboration tool named GlobalMMCS AVPortlet, to fully make use of the services provided by Global-MMCS. The tool which can run on multiple desktop platforms such as windows, Linux and Mac OSX, integrates audio and video collaboration together (compared to Access Grid with separate audio and video tool) and supports the screen capture service with MPEG-4 DivX codec.

From the experience of implementing Global-MMCS, we realize that the performance optimization of JMF has to be made especially for the conferencing application. The poor implementation of JMF may lead to unacceptable end user QoS. This paper provides a novel approach to the application of JMF in the real-time conferencing by extending its function, optimizing the performance and enhancing the communication capability with the publish/subscribe overlay network service. This approach enables us to have an integrated platform-independent desktop conferencing system. The paper is organized as follows. Section 2 compares JMF to other multimedia programming framework and describes our work on extending Java Media Framework. Section 3 and 4 describe the enhancement of JMF and design of this collaboration tool. Section 5 presents the result of performance test results. Finally, Section 6 gives the conclusion.

2. Problem Statement and Related work

Researches on multimedia programming frameworks began as early as 90s and produced a lot of software packages. Among them there are well-known frameworks such as TCL/TK [5] based package, DirectShow [6] and JMF. This section presents advantage of JMF over other frameworks and explores the research issues which have not been addressed by related JMF-based researches.

2.1 JMF and other multimedia programming frameworks

JMF provides a common cross-platform Java API, especially the JMF RTP APIs, for developers to build videoconferencing systems. Many researches have been done about the multimedia programming frameworks including the early efforts like CMT [7], VIC [8], VuSystem [9] and mature industry solutions like Microsoft DirectShow. CMT, VuSystem and the MBone tools, though each developed independently, all converged on the same basic architecture, which is split into low-overhead control functionality implemented in a scripting language like Tcl and performance-critical data handling implemented in a compiled language like C or C++. Therefore these tools can run in both UNIX and Windows. In contrast, DirectShow is a C++ COM API that only enables Windows applications to control a wide variety of multimedia devices and make media processing based on Windows codec devices and media formats.

All of these frameworks share the similar multimedia filter pipeline architecture because of two crucial factors in building the multimedia applications. One is the performance issue and the other is how to support a wide variety of media formats and devices in different platforms. There are many standard codecs including video: h.261/h.263/mpeg1/mpeg2/mpeg4 and audio: ulaw/alaw/gsm/g.723/g.729. Moreover, each operating system has it own programming API in multimedia capture and rendering as well as codecs management.

The filter pipeline architecture can support high efficient media processing. High-volume multimedia data is typically generated by a source filter objects and piped through one or more filter objects. Eventually, the media reaches a sink filter object and is consumed. The source usually is capture device or network receiving protocol; filters can be color space converters, compressors / decompressors, packetizers, and the like, while the sink might be a render or network transmission protocol. Each filter can run as a thread and make its process concurrently, which can ensure the high throughput of streaming workflow. Furthermore such a software architecture is easy for further extension and maintenance, which is also very critical to the development of multimedia systems because of the diversity in media formats and codecs.

Although the VuSystem literature cleanly articulated the filter pipeline in TCL/TK multimedia framework research, most of them such as VIC don't follow filter design pattern in their implementation. In contrast, both JMF and DirectShow not only define their clear filter APIs but also build the reference implementation based on media filter pattern.

In our opinion, the portability of multimedia application is not quite straightforward because multimedia application usually needs a lot of native codes. First of all, multimedia processing usually consumes a lot of CPU and other hardware resources, especially media capture, codec compression and decompression and media render. Secondly, operation systems have their own media packages which have different frameworks and APIs, which makes it a non-trivial task to design multimedia extensions that are portable across multiple flavors of UNIX, Windows and the Macintosh. In fact, most multimedia programming frameworks available today do not provide this level of cross-platform portability. For example, VIC can run on UNIX, Windows but has some problems in Mac OS, and DirectShow only works in Windows.

Therefore a portable multimedia framework must define a good platform-independent part to cover the heterogeneous native multimedia platforms. This separation strategy has been proven quite useful because it cleanly leverages the capabilities of the underlying operating system and divides the burden of design, maintenance and extension. And the advantage of Java over TCL makes it possible for JMF to include the platform-independent part as much as possible in the whole media process workflow. For example, some bulk-data operations like data pushing can be implemented in Java but not in TCL. Given this fact, JMF is more flexible and powerful than TCL media framework.

Some people may have the concern about the Java performance for multimedia processing in videoconferencing. It is true that the current JMF implementation is not as good as VIC multimedia package. And obviously it is a critical problem for using JMF in videoconferencing development. However, through careful performance tuning and optimization, JMF performance can be comparable to TCL/TK based system. We will discuss it in Section 3.1 in detail.

2.2 Related Researches based on JMF

There are many JMF-based collaboration systems, for example Java Collaborative Environment (JCE) [10] from National Institute of Standards and Technology (NIST) and Java Enabled Telecollaboration System (JETS) [11] from University of Ottawa. Based on multicast communication, JCE introduces video multiplexers and audio mixers to address the issue of conferencing scalability and has a Java-based GUI interface to integrate all the video windows created by JMF into one frame, instead of having the windows scattered all over the desktop. JETS is based on client-server framework that permits sharing of Java applets and applications. JETS 2000, the latest version of JETS, also offers video-conferencing using JMF.

All these works didn't make any quantitative performance measurement of JMF, or investigate the performance issues of JMF for the conferencing application and make optimization. Furthermore, they built Java tele-collaboration based on either multicast or client-server framework, which restricts their scalability. Research community has reach the consensus that tele-collaboration applications need a scalable, robust and QoS-aware "overlay" network for multimedia group communication over heterogeneous networking environments. Actually these problems hinder the widely application of JMF in the real-time conferencing. This paper addresses the critical issues by optimizing the performance, extending its function, and enhancing the communication capability with the publish/subscribe overlay network service.

3. Enhanced Java Media Framework

This section introduces our work on JMF, including how to improve JMF performance by exposing the interfaces of the JMF filters and rewriting a faster one, how to add the state-of-art codecs in JMF and screen capture, and how to extend the JMF to the Mac-OS platform.

3.1 Video Rendering

Video rendering, quite different from ordinary bitmap operation, is a high-throughput task involving large amount data movement between main memory and display memory. In most modern operation systems, the video rendering procedure has similar three steps: initialization, bitmap copying and blittering. During the initialization, a screen surface and an off-screen surface are created based on the window handle. The second step copies the video bitmap data into the off-screen surface. Finally a Blit service which can supported directly by display hardware, is called to move the bitmap data from the off-screen surface to the screen surface

Java has its AWT package for rendering. But to support fast native rendering, the developer must use **AWT Native DrawSurface** [12] and override the paint method to direct drawing operations to a native rendering library which then queries the Java VM to determine the information it needs in order to render. JMF already includes Windows native draw (DirectDraw) and UNIX native draw (X11). We developed the JMF rendering interface for Mac OSX. QuickDraw Port of CoCoView is used to access video surfaces and Decompressor Method is used to blit image into the QuickDraw Port.

In multi-party videoconferencing, most CPU overhead caused by client comes from video rendering, especially if a client has to display multiple video streams. And since at most time motions in video streams coming from the meeting scenes are relatively small, each video frame only contains some changed pixel blocks that need to be rendered. If the client only pushes these blocks into the video surface, the amount of data copy is definitely reduced. Since video rendering is a high-throughput task, this optimization strategy will remove a lot of unnecessary overhead and improve the performance.

Unfortunately, current JMF 2.0 reference implementation makes its optimization impossible. JMF filter pipeline is composed of filter Modules and linking Connectors. Each Module has either InputConnecor or OutputConnector. The OutputConnector of the upstream Module connects to the InputConnector of the downstream. There are two kinds of streaming protocols between a OutputConnector and InputConnector. One is **Safe protocol**, which means both the downstream Module and the upstream Module run in separate threads. The "safe" protocol introduces an extra copy since the upstream module needs to copy data into the intermediate circular buffer and the downstream Module reads the buffer as follows. In JMF2.0, BasicCodec Module and BasicSource Module run in the same thread, while BasicRender Module runs in the other thread. The other is **Push protocol**, which means that the upstream Module and downstream Module share the same thread. The upstream Module loops on its "*process*" method and invokes the **writeReport ()** method after it finishes one frame. Consequently, this method triggers the next call of the *"process"* method in the downstream Module running in the current thread. When this downstream Module finishes its "process", it calls the next downstream Module, recursively.

Obviously, since the Push protocol has no extra copy like the safe protocol, rendering should take the second protocol for the performance reason. But JMF 2.0 uses the first protocol and makes the copy operation over the whole video frame no matter how many macro blocks are actually updated. We believe the reason is that JMF designers regard rendering as a time consuming job and want to separate from the decoder processing module. This design strategy turns out insufficient. Since in the modern desktop machines, hardware improvement like AGP graphic interface enables the high throughput rendering which needed by conferencing application, the rendering operation should be able to keep up with the codec decompression operation. This paper proposes a "Direct Write Through" approach to address the issue.

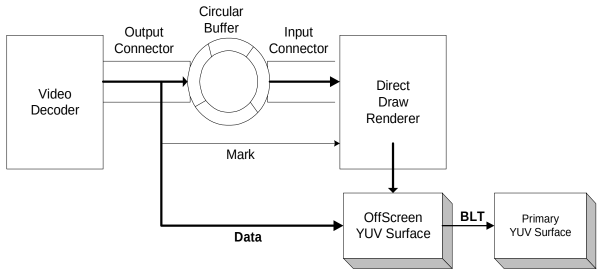


Figure 1. Direct-Write-Through optimization in JMF rendering

Figure 1 displays the connectors and filters in the rendering data path of a conferencing client. In the current JMF implementation, the Video decoder copies all the pixels in a decoded frame into the circular buffer shared by the output and input connectors. Through the input connector, the DirectDraw renderer transfers the frame from the buffer to the offscreen surface and then blit the video frame into the screen surface. "Direct-Write-Through" allows the OutputConnector of the decoder to directly move the changed pixel blocks to the offscreen surface so that the extra copying between connectors is removed. Each time the decoder finishes the copy, the DirectDraw Renderer will do Blit to fresh the screen display.

In some platforms, there is no such hardware video surface available and it also needs some video color conversion filter such as transformation from YUV to RGB in the rendering process pipeline. For example, JMF needs to use YUV-to-RGB filter to make the format conversion on Linux because Linux rendering depends upon XLib which can only support RGB bitmap. In such a case, the "Safe" protocol has to be applied to match the speeds of the decoder and renderer. Therefore, the OutputConnector of the decoder should passes the "Pixel Block Mark" down to the transformation and renderer filters so that these downstream filters could read it and make whatever optimization they can do to reduce the bitmap copying based on the mark.

3.2 Video Capturing

The task of Video capture is to grab the video frame from the video capture device in a constant interval of frame rate and push the data to the downstream codec filter for compressing. JMF creates a video data source to abstract the real capture services which usually have different APIs in modern operating systems.

We added Mac OS video capturing into JMF package. Apple's QuickTime architecture [13] is the primary support for video-based applications on Mac OS X. QuickTime has two types of video capture API: the low level API is called Video Digitizer and the high level one is Sequence Grabber of QuickTime. Since the Video Digitizer is not quite stable and dependent of underlying device, the high level API seems to be a better choice in terms of portability. Unfortunately QuickTime's model of real-time video input is based around the recording of the input stream to disk, and providing screen based video previews to assist this. This model is not quite suitable for the video capture in video conferencing. Therefore the Sequence Grabber API has to be adapted to fit in JMF video capture interface.

Beside a video capture devices like a Web camera, the desktop is also can be regarded as a video data source especially the window playing a movie. The feature is necessary to support remote desktop sharing. We built JMF screen capture which can create a JMF DataSource from the data in the framebuffer in the operating systems. The screen DateSource should have such a form of URL: screen://screen-number, in which the screen number means the number for multiple screens in the system. Users have to specify the co-ordinates < left, top, width, height > of the desktop area mapped by the screen DateSource.

The screen DateSource thread grabs the bitmap from the target region of the framebuffer, and copies it into the off-screen buffer, and pushes the data like JMF PushBufferDatasource. The rate of the copying framebuffer is determined by the frame rate of the video stream. Before the screen data source is pushed into video codecs for compressing, RGB-to-YUV conversion is usually needed because the bitmap in the screen frame is usually RGB rather than YUV.

Screen capture can be regarded as a reverse operation to video rendering since it grabs bitmap from the framebuffer. Through the graphic API provided by operation systems, we can get the pointer to the framebuffer and copy the bitmap into the buffer in the application memory space. Note that the low level graphic API is preferred to get the best of performance. For example, we should use DirectDraw API instead of GDI in windows because DirectDraw allows the programmer to access the framebuffer directly.

3.3 Add New Codec

JMF 2.0 package only supports H.261 decoder, H263 and JPEG. We added H.261 encoder and MPEG-4 video codec (DivX[14]) based on JMF codec interface. In addition, new RTP format and payload for MPEG-4 video are also added through JMF RTPManager.

4. Global-MMCS AVPortlet

On the basic services of JMF, we develop a custom conferencing client named Global-MMCS AVPortlet, which uses publish/subscribe for multiparty audiovisual collaboration. Although publish/subscribe was mostly used for large-scale event notification and information dissemination, it is also perfect candidate for scalable conferencing media distribution networks. A RTP videoconference usually has three components: RTP video stream set (RVS), RTP audio stream set (RAS) and participating RTP endpoints (RES). A multimedia RTP stream either from RVS or RAS, is regarded as a "topic" and each RTP packet from this stream as an "event" for this topic. The sender of this stream (publisher) can "publish" RTP events to this topic. Other endpoints (subscribers) need to subscribe to this topic in order to receive the stream.

In addition to the semantic adequacy in supporting the conferencing multicast communication, the publish/subscribe broker network (NaradaBrokering) has very important features in terms of network engineering. First of all, it provides software overlay solution to the deployment issue of hardware multicast. Secondly, it enables RTP communication through NAT and firewall barrier. It also offers the similar service to single-source-multicast that only forwards the subscribed streams to reduce the network traffic.

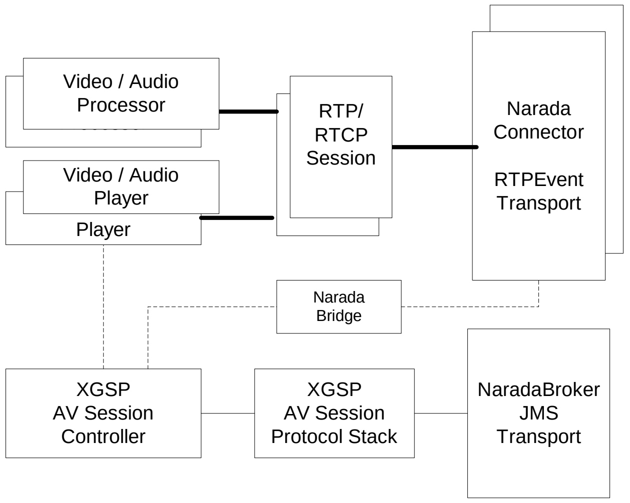


Figure 2. Global-MMCS AVPortlet

Figure 2 displays the internal architecture of GlobalMMCS AVPortlet. The upper half is the RTP media processing path discussed in Section 4.1. The bottom half is the control components presented in Section 4.2.

4.1 Extending JMF RTP Transport over Publish/Subscribe Overlay

In order to be transported in the NaradaBrokering overlay, RTP/RTCP packets have to be wrapped into RTPEvents with a short extra header to simplify the topic-based routing service. JMF RTPConnector is extended to support this encapsulation at the side of client. RTPConnector class abstracts the underlying transport mechanism for RTP control and data from the RTPManager. An implementation of the RTPConnector must be created and handed over to RTPManager during initialization. The RTPManager will then use it to handle the sending and receiving of the data and control packets. We implement an RTPConnector subclass named NaradaConnector**,** which acts as a transport layer for JMF applications to talk to NaradaBrokering network. It can send and receive RTPEvents over various transport protocols including UDP, TCP and HTTP. In addition, a factory class NaradaBridge is also introduced to create NaradaConnectors with the subscription of RTP topics and deliver the outbound RTP events of the connectors to brokers. As an overlay networking on Internet, NaradaBrokering can use the available transport channels through the firewall and NAT boundaries. By extending the JMF RTP transport over this overlay, Global-MMCS AVPortlet can run behind firewalls and NATs.

4.2 XGSP Audiovisual Session Management

To support different audiovisual application endpoints having their own signaling procedures, XGSP framework provides a common XML based signaling protocol for them. H.323 (H.225, H.245) and SIP signaling protocols have to be translated into the XGSP A/V signaling protocol and vice versa. The complete discussion for using the XGSP goes beyond the scope of this paper. One of the reasons for using XGSP for the GlobalMMCS AVPortlet is because of underlying media communication overlay – NaradaBrokering. To support the topic based publish/subscribe media transport, there is no way to use the standard conferencing signaling framework like H.323 or SIP. Or we can say XGSP enables the customized clients to make full use of services provided by NaradaBrokering and Global-MMCS.

Besides of the RTP events for the stream topic, some control events indicating the status of each stream topic have to be defined. They include five major events: *NewStreamEvent*, *ByeEvent*, *TimeOutEvent*, *Active-to-Passive*, *Passive-to-Active*. Separate audio video stream lists are maintained by using these control events. To visualize the video stream list, GlobalMMCS AVPortlet requires the thumbnail service of media servers to display a JPEG picture along with the metadata of each video stream. Although a multicast conferencing client such as VIC can generate up-to-date thumbnails by receiving and decoding all the video streams in a multicast session, it causes a lot of network traffic if there are many streams in the multicast session. For example, in the lobby session of Access Grid, nearly fifty H.261 streams can be seen at most time. It demands at least 10Mbps wide-area network connection for such a medium-size group of multicast video session [15]. In contrast, our approach can save a lot of network resources. For the JPEG thumbnails, we can just update in one frame per minute which will allow enough visual awareness for users to select the video streams. Conservatively, it can be estimated that such a solution only cost less than 100 kbps even in the 50-stream video session. (Assume each JPEG image of the streams have the size of 20 KB. Then totally we have 1000 KB to be transmitted in one minute, which means 130kbps average.) Therefore even DSL user with 1Mbps can attend Access Grid meetings through Global-MMCS.

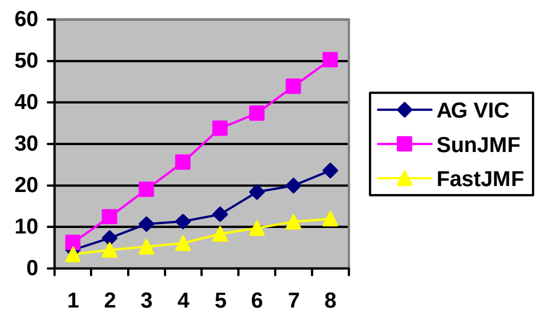
The client sends its subscription to NaradaBrokering for video stream selection. Once the subscription is established as a pair of < client identification, media stream topic >, the overlay will route the media stream packets to the client. This subscription reduces the traffic to receivers because they only need to handle the incoming streaming to their interests.

5. Performance Evaluation

As mentioned in Section 3.1, the most CPU consuming part in medium-scale videoconference is the video rendering. In the following test, we measure the rendering performance of VIC, JMF 2.0 reference implementation and the optimized JMF 2.0. It shows that through careful performance tuning and optimization, JMF performance can be comparable to TCL/TK based system.

Testing video streams are captured from the same desktop of the sending machine, and multicasted to the receiver machine which runs Access Grid VIC, two JMF clients based on Sun's JMF implementation and our improved version. The receiver machine is the XP desktop with the configuration of P4 2.0 GHz, 1.0 GRAM. And the configuration of its display device is GeForce2 MX/MX 400 with On-Board memory 64MB. The sender machine with 4 CPUs is powerful enough to pump up to 8 testing video streams.

We have two test scenarios with different source streams. In the first test one, the CIF-size still image of the desktop area is sent to the clients. Each stream is encoded in H.261, and has average bandwidth 20kbps. In the second test scenario, the CIF-size video sequence from a 30-second movie with a lot of motions is streamed to the clients. Each stream is encoded in H.261, and has average bandwidth 500kbps. The CPU overhead caused by these clients running in these tests is compared in the figure 3.



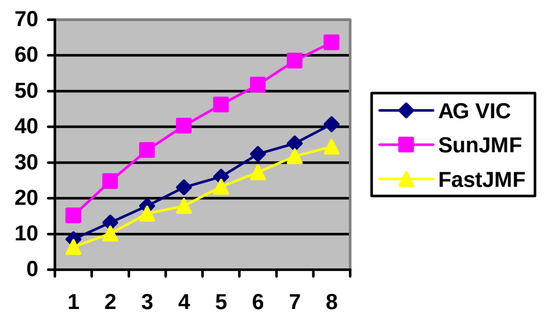


Figure 3 Video Rendering performance (top: still desktop, bottom: movie sequence)

From the result, we can see the CPU overhead is roughly in linear increase when the number of stream rises. We can imagine an empirical equation like OverHead = A*X + B, where X: the number of rendered stream; A: normalized incremental step associated with the bandwidth or frame-per-second of the video stream; B: the constant depends upon the machine. Since the traffic bandwidth for the still-image video is much less than the motion movie, the rendering overhead is also less. And in both cases, the optimized JMF implementation is much faster than Sun JMF, and even better than VIC. In Test 1, the average incremental step (A in the empirical equation) for VIC is about 3 %(2.8%), for Sun JMF is 6% (6.3%) and for FastJMF is 1%. In Test 2, the average incremental step for VIC is about 4%, for Sun JMF is 7% and for FastJMF is 2%.

In both cases, the incremental step of fastJMF is much less than SunJMF because of "Direct-Write-Through" Strategy. However, in the first scenario, the benefit of the optimization is much obvious than the second one since the movie stream demands more bitmap copying in fastJMF than the still desktop stream. Notice that AccessGrid VIC has also better performance than SunJMF since it implements "Direct-Write" in the native C/C++ code. The rendering performance of our fastJMF even outperforms AG VIC because we are using video memory for bitmap copying while AG VIC uses main memory.

6. Conclusion

This paper introduces our effort to support a high-efficient multimedia collaboration using JMF. It provide a new approach of applying JMF in the real-time conferencing by extending its function, optimizing the performance and enhancing the communication capability with the publish/subscribe overlay network service.

We make comprehensive analysis about the advantage of JMF as multimedia programming framework. From the experience of implementing the whole system, we realize that the performance optimization of JMF has to be made especially for the conferencing application. Therefore "Direct-Write-Through" optimization is introduced to improve the video rendering performance. And it achieves almost two timers faster than the original version, and as fast as the VIC version of Access Grid. Furthermore, Screen capture function was built in JMF to extend the video data source and upgrade the collaboration capability. And the latest codec like DivX MPEG-4 was integrated into JMF package. JMF RTP transport layer was extended to support publish/subscribe software multicast. Based on all the above work, a flexible and high-efficient audiovisual collaboration tool based on Java Media Framework was developed.

REFERENCES

[1] Gordon R. and Tally S., Essential JMF -- JMF Java Media Framework, 1st edition, Prentice Hall, 1998.

[2] Wu W., Fox G. C, Bulut H., Uyar A., Altay H., "Design and Implementation of A Collaboration Web-services system", Journal of Neural, Parallel & Scientific Computations, Volume 12, 2004.

[3] Global Multimedia Collaboration System, www.globalmmcs.org

[4] Access Grid, www.accessgrid.org

[5] Ousterhout J., Tcl and the Tk Toolkit. Addison-Wesley Publishing Company, 1994.

[6] Microsoft DirectX (version 8.0): Microsoft DirectShow, Online Documentation: http://msdn.microsoft.com/directx/.

[7] L. Rowe and B. Smith. "A Continuous Media Player." Network and Operating Systems Support for Digital Audio and Video. Third Int'l Workshop Proceedings, 1992.

[8] McCanne S. and Jacoboson V., "VIC: A Flexible Framework for Packet Video", ACM Multimedia 1995.

[9] Lindblad C.J., Wetherall D. J., Tennenhouse D.L., "The VuSystem: A Programming System for Visual Processing of Digital Video", ACM Multimedia, San Francisco, California, 1994. pp 307-314.

[10] Abdel-Wahab H., Kim O., et al, "Java-based Multimedia Collaboration and Application Sharing Environment", *Colloque Francophone sur l'Ingenierie des Protocols* , CFIP'99, Nancy, France, April 26 - 29, 1999

[11] de Oliveira J. C., Hosseini M., Shirmohammadi S., Malric F., Nourian S., Saddik A.E. and Georganas N. D., "Java Multimedia Telecollaboration", IEEE Multimedia, pp18-29, Vol.10, No. 3, July-Sept 2003.

[12] AWT Native DrawSurface, http://java.sun.com/j2se/1.3/docs/guide/awt/AWT_Native_Interface.html

[13] QuickTime, www.apple.com/quicktime/

[14] DivX, www.divx.com

[15] Access Grid Node Minimum Requirements, http://www.accessgrid.org/agdp/guide/min-req/1.0/min-req-1-0.pdf