Design and operation of a Java based distributed media processing system

Balázs Bámer
Computer Integrated Manufacturing Laboratory
Computer and Automation Research Institute of the Hungarian Academy of Sciences
Kende u. 13-17, H-1111 Budapest
Hungary
bamer@sztaki.hu

Abstract – **In this paper we present a *mu*lti-user *mu*ltitask distributed audio and video capturing, processing and recording *s*ystem (MUMUS). The MUMUS system is implemented in Java using Sun Microsystems, Inc.'s Java Media Framework (JMF) to assure portability and easy extension. The software includes fault tolerance routines to limit effects of communication, configuration and resource faults and keep smooth operation of intact tasks and parts. Control is decentralized with different embeddable interfaces and provides flexible management of different tasks. With additional codecs and some functions, this system would form a cheap, high-performance real-time distributed video (audio) processing application.**

I. INTRODUCTION

Today there is an increasing demand on video transfer and processing applications. The application area is very wide: security and surveillance, conferencing, process observation, call center archiving, movie streaming, etc. all require various live media management solutions. PCs are available for reasonable price, and their current CPU, memory and disk performance enables live video stream processing without any special hardware. We tried to develop a general and cheap solution to cover such areas.

A. Aims

The aims of developing MUMUS (means *bogey* in Hungarian) were:

- Audio and video capturing, transfer and recording – main aim was video stream handling, but JMF supports audio as well, so not difficult to include
- Platform-independence – currently Linux (x86), MS Windows (x86) and Sun Solaris (UltraSparc) are supported
- Circular buffer – transfer a (live) video stream with a custom time delay to allow inspection or recording of events before an alert
- Regular media modules – other functions provided by interconnectable modules like file reader, split stream, capture video or audio, player, file writer, etc.
- Distributed – virtually unlimited number of computers can take part in media processing, a task can use an arbitrary subset of these
- Multi-user – different users (using different controllers) can run their tasks if the necessary resources are available
- Multitask – different tasks run independently
- Standard and widespread format for configuration and task description
- Cheap – no special hardware needed, even 1GHz PCs with webcams and 10Mbps network are suitable for some tasks, based on free software
- Fault tolerance – limit effects of communication, configuration and resource faults and keep smooth operation of intact tasks and parts
- Extensible by user – allow users to implement their own codecs, filters or functions
- Embeddable controller – user can directly control MUMUS from inside his application
- Bi-directional video transmission – allow developing e.g. video conference application
- Support some common stream and file formats
- Stream interchange – streams can be sent to and received from other applications

B. Overview of existing systems

First we searched the Internet for video transmission and recording systems providing circular buffer. Table 1. summarizes our results.

Most of these products are intended for security and observation purposes. Some uses custom hardware (with or without a built-in camera) to generate network video stream instead of PCs (marked with "-" in the seventh column). We cannot use these systems (IPCom, [VCS], [AUVIDEA], VIP-AV, SecureLink, Visare), since our aim is to have a cheap system, which uses hardware available in an average office. Bi-directional video transmission was also essential, so the systems not providing it are also rejected: VideoSavant, DVRLite, CamStream, InputStream-based DataSource for the JMF (well, this is not a complete application, but a JMF extension), Scrubba, [IteL]. We have very little information about Merlot. It is a project created to investigate and increase efficiency of networked image delivery rather than a video transmission and processing application. It is still in initial implementation stages.

We found two systems fairly close to our aims, RBNB Data Turbine and NetMeeting. The first one is written in Java and is a general data server using novel database approach. However, video stream management is only partially implemented, and writing the rest would mean almost totally implementing MUMUS. NetMeeting is a simple, user-friendly and robust technology for video conferencing. It works well under Microsoft Windows variants, but for Linux the work is not ready yet. Moreover, it cannot share the same API, so common application development would be difficult.

It would be a mistake to forget about systems without a circular buffer. However, Internet search yielded dozens of home video recording and media player applications, but

there is only one system, which more or less meets our requirements, so we gave up searching. This system is being developed in the VideoLAN project. It is a real multi-platform media streaming system, providing lots of converters and filters. Inspecting the architecture we found that multitask operation would be difficult to implement in its existing framework. These circumstances convinced us to develop our application.

Table 1. Existing video processing solutions containing circular buffer

Product	Extensible by user	Platform	Embeddable controller	Distributed operation	Bi-directional video transmission	(Existing HW) / (SW only)	Existing network	Remark
VideoSavant	SDK	WINNT+	Python/ SDK	restricted	-	+	+	
DVRLite	+	?	?	-	-	+	n/a	
CamStream	source	Linux	source	-	-	+	n/a	
(inputstream)[1]	source	Linux/Win/ Sun	source	-	-	+	n/a	
Merlot	?	?	?	+	?	?	+	
IPCom	-	Windows	?	+	-	-	+	
[VCS][2]	SDK	Win2K+	SDK	+	+	-	+	
Scrubba	-	Beos	-	-	-	+	n/a	
[AUVIDEA][2]	-	Win2K+	-	-	-	-	n/a	
RBNB Data Turbine	SDK	Linux/Win/ Sun	SDK	+	SDK/JMF	+	+	lot of JMF work
VIP-AV	-	?	?	+	-	-	+	
[IteL][2]	?	?	?	?	-	?	+	
SecureLink	?	Win2K+	?	+	-	-	?	
Visare	-	?	-	+	?	-	radio	
NetMeeting	SDK	Windows, alternative for Linux	SDK	?	+	+	+	totally different APIs

[1]Full name is "InputStream-based DataSource for the JMF"
[2]Company names

II. DESIGN

A. Platform and technology

Multiple platform application and network-based media processing made the choice of programming language almost evident: Java is designed to meet the first requirement. Its language-level multithreading, supplied network routines and additional free libraries are unbeatable. We found two free Java-based media streaming and processing libraries, which are offered on multiple platforms: Apple Computer, Inc.'s QuickTime for Java [1] and Java Media Framework [2]. QuickTime for Java is available under Mac OS (X) and Windows, but not under Linux, which becomes more and more important in Hungary, so the choice of JMF was evident. JMF is a very flexible system, since almost every component can be re-implemented to support new formats, hardware or provide new functions, and is designed for multitask operation. It uses the industry-standard Real Time Protocol (RTP) for streaming, which is also the base of H323. JMF supports live transmission, media recording and playback, synchronization, standard Java sound capture and video capture using the underlying OS' standard video interface (e. g. Video for Linux, the old Video for Windows, Video Compression Manager). JMF supports easy access to raw data, extension with almost every part (codec, media content, custom processing or effect filter, renderer) and comprehensive diagnostics. These features showed it totally suitable for our purposes.

B. Architecture and operation

Every computer possibly participating in the media processing runs the processor program named *skeleton* in one instance. Each skeleton can accept and run arbitrary many *tasks* to perform the requested tasks. The task consists of interconnected *modules* providing the atomic functions of the system. As the processing can be distributed, the task prescribes which skeleton should perform each function (provided by modules, see later), i.e. the place of modules in the task. The example in Fig. 1. contains three computers. The first one captures audio and video and transmits them to the second one, which displays both. The video stream is split in the first one and in turn transmitted to the last one in low resolution and is archived.

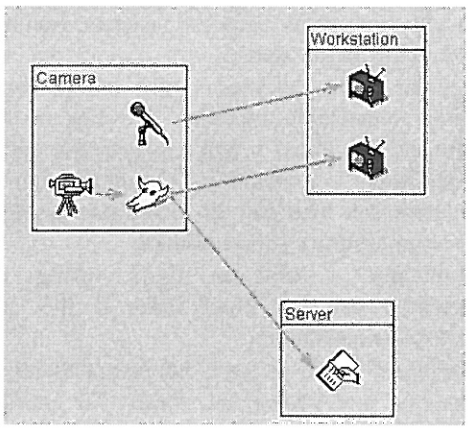

Fig. 1. Production process observation

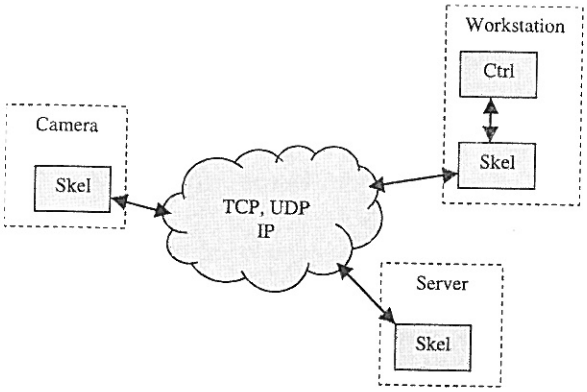

Fig. 2. Computers and roles participating in process observation task

Task *descriptions* (XML format, a dedicated graphic editor helps creating them) are sent to skeletons by other components of MUMUS named *controllers*. These have two types: one can be embedded into a Java application; the other is a standalone command-line program for testing. The recipient skeleton is the *boss* of that task, and manages other participating skeletons (*slaves*, if any) to do what the sender controller (and no other) wants them to do. This controller sends the boss any command concerning this task, and the boss sends back status/error messages. Fig. 2. shows the participating computers according to their roles in this task (one skeleton and the interested controller runs on the same computer).

The life of a task can be described by the state-transition graph in Fig. 3.

The states are:

- Dawning – the description is sent to the boss, which forwards it to the slaves, and each begins to construct the modules needed. The success is signed to the boss, which tells the controller if all ready. If timeout occurs, destruction of task is initiated.
- Living – the task does what it should, and it or its modules can perform commands
- Dying – either an explicit command, or discovering of some error initiates destruction of this task, and the boss signs the end if all modules are dead.

In each state, description-level, resource-related or internal JMF errors are caught and cause destruction of the task (on all skeletons), and the controller is notified.

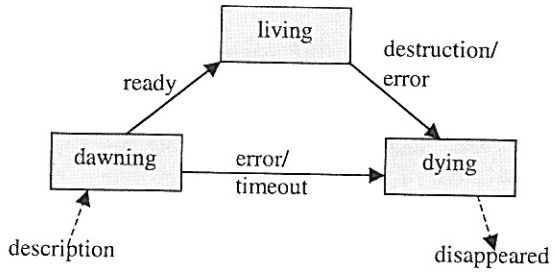

Fig. 3. Task state transition graph

The available module types for tasks are:

- *Capture* – this can be either audio, or video. All main parameters are adjustable. Has one output.
- *Player* – this renders the stream put to its single input. Output is screen window or sound card.
- *Reader* – reads the contents of a media file and sends an audio, a video or both streams depending on the content and the settings. Has one or two outputs.
- *Writer* – starts writing a media file, writes it and finalizes. Type depends on the settings, accepts audio or video stream or both. The file can be remote one on a DAV server. Has one or two inputs.
- *Transmitter* – transmits the stream via an RTP connection to the specified unicast or multicast IP address: source for outside streaming. Has one input.
- *Receiver* – receives an RTP stream from the specified unicast or multicast IP address: sink for outside streaming. Has one output.
- *Monoscope* – generates audio or video stream with adjustable parameters for test purposes. Has one output.
- *Splitter* – splits a single input stream to arbitrary many outputs. One remains the input, the other ones are clones.
- *Circular buffer* – contains a disk-based circular buffer of preset length and sends the input video stream with 1s<T<length delay on its output. The playback movie clip can be sent to a DAV server.
- *Transcoder* – changes resolution, sample rate, etc.
- Other modules – motion detection, various filtering algorithms, subtitling can be easily developed, or code already exist in public domain for JMF [3].

Each module can perform stream format conversion on its output. It can be compulsory when sending RTP stream on network. However, some codecs are missing, or their availability depend on OS configuration or third-party software.

Streams can be mono or stereo audio streams or video streams. Mixed or multiplexed media streams are not used. They are always directed from one prescribed module to an other. A stream has no duplex or semi-duplex operation. Format of intra-computer streams is arbitrary (raw or compressed), but ones sent through network must be RTP-based. Outside streams have explicit port numbers in the description (taken from a compile-time defined interval) so the user takes the responsibility to prevent collision. In this case the second task claiming that port fails. Network streams inside MUMUS use dynamically allocated ports (taken from an other compile-time defined interval). Negotiation between the interested skeletons is initiated to find a free port. All ports are marked as reserved through the whole life of the owner task.

Controllers and skeletons communicate with messages. These can be

- *Commands* – which have three scopes:
  - Skeleton-wide: the only one of this sort causes the target skeleton to destruct all of its tasks and exit.
  - Task-wide: create, destruct, start or stop a task. Stopping means pausing stream and processing activity, starting resumes it.
  - Module-wide: start and stop file recording.
- *Replies* – sent to report an event, state change or errors.

Communication medium can be Java RMI or JGROUPS based. Both have advantages and drawbacks. They cannot be mixed in a given set of skeletons/ controllers, since each application uses either one, not both.

The asynchronous nature of message-based communication and the event-based notification mechanism of JMF required a central queue and processing loop in the skeleton. Having all events and messages in one queue greatly simplifies handling synchronization problems.

Fault tolerance is implemented in different levels.

- Streams – JMF uses UDP in RTP as media carrier for performance reasons. UDP is unreliable protocol, which required implementation of error handling and recovery algorithms in RTP. This implementation is moderately flexible, handles low-bit rate or unreliable connections with significant performance loss and can recover from temporary network failures.
- Messages – although both RMI (TCP based) and JGROUPS [4] are reliable, message delivery cannot be guaranteed (in case of temporary network failure). However, connection loss between different skeletons in a task should be avoided, since resource leak (occupied ports, memory, capture devices) and data loss could occur. For this reason, each skeleton uses a message resend mechanism with per task message ordering to guarantee that each message is delivered and each task receives its messages in correct order.

III. OBSERVATIONS

A. Performance

MUMUS performance depends on three factors:

- Computer performance – 1 GHz PC with 128 Mbytes RAM is the minimum for video applications; HDD speed is not so crucial.
- JMF, codecs and OS efficiency – as currently all processing is performed by underlying code, its quality influences performance very much. Some codecs are available only in Java version, but all can be replaced by native ones. Initializing different modules can take extremely long and this time depends on OS internal states. For example, the same webcam takes dozens of seconds to initialize on one computer while only some on the other. MUMUS' task consists of 90% organizing.
- Network – lower speed networks (e.g. 384/64 kbps xDSL) seemed to sweat JMF implementation of RTP. Connection was hard to establish and achieved bit rate / frame rate was far lower than theoretical possible. However, on a 10 Mbps LAN it is totally stable.

B. Applications

MUMUS was developed for mainly the following application areas:

- Versatile media playback/transmit/recording system – this is more-or-less ready, and could be used in the following examples:
  - General-purpose surveillance system with archiving – not probed
  - Videoconference system – this is a multi-language system with interpreters and at most one speaker any time, whose speech is translated to the requested languages. Everyone who speaks transmits the voice using IP multicast, and every listener listens in his preferred language. The system works well for the few computers with a hack (restarting MUMUS at every change), but long initializations are very annoying.
  - Production process monitoring and event inspecting with circular buffer – this was tried in a fluorescent lamp production factory. The task was quite simple: one computer at the production line with a capture connected to a circular buffer, whose output was transmitted (inside the task) to a player running on a computer in the control room. The circular buffer sent each movie clip to a DAV server. The task runs flawlessly as long as the lock problem of the JPEG codec doesn't arise (see later).
- Real-time distributed media processing system – This would require an efficient and loss less video codec for RTP transmission. Motion-JPEG could be used with care (high-quality settings with very few encoding/decoding stages). We also made experiments with HuffYUV [5], but due to various software problems, we couldn't try it. The processing algorithms depend on the task, so they are missing.

During development JMF turned out to be far from perfect, although the concept is nice and straightforward. Sun doesn't seem to make real efforts to eliminate these problems (see also the JMF mailing list [6]), but time will tell. We found the following bugs (which, unfortunately, make the most important parts useless):

- During releasing an RTP connection, the port remains occupied if the address was multicast, and no other multicast address can be used on that port.
- H263 stream transmission can fail sooner or later if the stream is split.
- Use of circular buffer seems to harm the JPEG codec's lock (should not be possible in Java) so that

the underlying native JPEG library gets called in wrong state and decoding fails, even the whole virtual machine can abort.

- We couldn't use HuffYUV format (not known to JMF), although it fits Windows' VCM system.
- The fact that JMF uses Video for Windows creates additional problems. VfW seems to be not very stable, which means unaffordable long delays during opening a camera what may occur several months after install. The camera even may stop functioning, which may or may not be resolved by system restart.

IV. CONCLUSION

MUMUS fulfils our initial goals (such as platform independence, multi-user, multitask, fault tolerance and extensibility), although there is still much to do. It is easy to use for experiments, configuring for specific real purposes or extending with additional modules. Unfortunately, the above problems make complex applications extremely hard or currently impossible to develop. Neither of these problems could be foreseen at the beginning of development. JMF source code is partially available, so some bugs could be fixed, but that would require more effort than we currently are able to make. Smaller applications (like programmable video recording, small streaming server) can run flawlessly, and MUMUS could run special processing tasks even in these circumstances.

V. ACKNOWLEDGEMENT

The research project was partly supported by the National Committee for Technological Development (OMFB) under the NKFP Digital Factory contract.

VI. REFERENCES

[1] http://developer.apple.com/quicktime/qtjava/index.html
[2] http://java.sun.com/products/java-media/jmf/index.jsp
[3] http://www.cs.uno.edu/~krzeszut/programming/motion-detection/
[4] http://www.jgroups.org/javagroupsnew/docs/index.html
[5] http://neuron2.net/www.math.berkeley.edu/benrg/huffyuv.html
[6] http://swjscmail1.java.sun.com/cgi-bin/wa?A0=jmf-interest

Table 2. URLs of solutions described in Table 1.

VideoSavant	http://www.ioindustries.com/videosavant/vspro.html
DVRLite	http://www.s3group.com/digital_consumer/digital_terminals/dvrlite/
CamStream	http://www.smcc.demon.nl/camstream/
(inputstream)	http://www.extollit.com/isdsjmf.php
Merlot	http://www.llnl.gov/icc/sdd/img/merlot.shtml
IPCom	http://www.cbceurope.com/ganz/pdfs/ipcom.pdf
[VCS]	www.vcs.com
Scrubba	http://209.197.100.159/scrubba/
[AUVIDEA]	http://www.auvidea.com/
RBNB	http://rbnb.creare.com/rbnb
VIP-AV	http://www.industrialenet.com/Downloads/VIP/Presentations/VIP%20Presentation.pdf
[IteL]	http://www.itel-co.com/products/remote.htm
SecureLink	http://www.dvtel.com/pdf/8_18_03/EtherReach_Products_User_Manual.pdf
Visare	http://www.wavelengthsolutions.co.uk/support/Products/manuals/Visare%20Veb/VISARE%20USER%20GUIDE%201.2.PDF