A LOW BANDWIDTH SOLUTION FOR TELEMEDICINE APPLICATION EVENT SYNCHRONISATION USING JMF

N. Seixas*, H. Pereira*, P. Carvalho*, J. Henriques*, M. Antunes**

*Centre for Informatics and Systems, University of Coimbra, Coimbra, Portugal
** Centre of Cardio-thoracic Surgery of the University Hospital of Coimbra, Coimbra, Portugal

{nseixas, hmanuel, carvalho, jh}@dei.uc.pt, antunes.cct.huc@sapo.pt

Abstract: Telemedicine is becoming an essential tool in supporting the delivery of medical care and knowledge. In many situations a store-and-forward strategy may be applied in developing telemedicine services to enable discussion among clinical experts. However, a perfectly synchronized distributed environment has to be guaranteed during on-line medical discussion in order to avoid potentially critical misunderstandings. To promote dissemination of telemedicine services, it is desirable that cheap consumer level infra-structures are applied to implement these solutions. This is a major challenge for today's distributed object synchronization mechanisms, due to the limited bandwidth and best effort service provided by the internet.
In this paper, an extension to JMF (Java Media Framework) is presented to enable the synchronous exchange of application events and voice streams in low bandwidth networks. The strategy employed is built upon a simple event dictionary and an event registration procedure. Practical tests have shown that this mechanism enables adequate setup and synchronization of distributed medical environments with an overhead of 1Kbps.

Introduction

Recent advances in computing and data transmission technologies, compression technology, high bandwidth storage devices and communication networks enable real time multimedia services over the Internet. Telehealth services, such as telemonitoring and teleconsultation are some of the most demanding and appealing in this context. Telemedicine is regarded as a significant breakthrough in medical care with major potential, social and economic impacts. Besides facilitating the prompt access to specialized medical care (e.g. in remote or underdeveloped regions), there is a great expectation for these solutions to be the pillars for new preventative healthcare policies, as well as relevant tools to support autonomy and quality of life. This is particularly important in developed countries where accentuated aging of the population is observed. In these countries, major reduction in health expenditure is expected to be achieved from adopting telehealth solutions. These savings are expected to stem from avoided dislocations of patients and professionals, timely detection of disease development (which usually enables more cost effective and safer treatments), as well as avoided or shorter hospitalization periods.

An important application of telemedicine is the access to a second expert opinion [1], which is particularly relevant at several clinical specialty areas, with significant shortage of specialists. Furthermore, disease prevalence and incidence in some of these areas are increasing, which puts further pressure on the health provision system.
In order to accomplish a good telemedicine service, a good quality of service (QoS) [2] is required in the transmission of real time data. Inappropriate QoS is usually perceived when a remote application does not perform as expected, either by lack of accurate feedback or by excessive response latency. In the context of telemedicine applications, the perceived QoS is mainly conditioned by the response latency as well as by the achieved synchronization of the distributed environment, i.e., by the achieved synchronization of audio-visual data channels and user actions. The latter is of paramount importance to achieve an unambiguous and effective collaborative working environment. It is observed that a certain limit of latency is acceptable, as long as accurate synchronization among critical data channels is achieved (for instance, if two physicians are examining some patient data – e.g. an X-ray – in order to avoid misunderstanding, it is fundamental that all mouse actions match voice instructions during reporting).
Currently, the Internet provides best effort service and is limited by its bandwidth, delay and loss. Furthermore, due to network and systems heterogeneity, the difficulty to transmit real time data in an efficient and flexible way is increasing.
At present, there are few available mechanisms able to grant the necessary QoS for real time distributed applications. Currently, QoS research and definition is mainly concentrated at the network level defining policies for scheduling strategies, queue management, admission control and resource reservation (see, for example, IntServ and DiffServ [2]). These solutions usually require dedicated networks or special service contracts at the provider level. In contrast, best effort networks do not deliver the performance required for a wide range of interactive and multimedia applications that exhibit demanding latency and bandwidth requirements: the behavior of a data stream in the presence of congestion is completely unpredictable, having no guarantee that a critical application will perform correctly [3]. Under best effort IPv4 networks, QoS solutions are mainly achieved at the application

ISSN: 1727-1983

level. Namely, in order to increase QoS new and more fault tolerant coding techniques, such as the H264, have been recently introduced. Other mechanisms required to successfully implement telemedicine services over best effort networks are fault tolerant and low bandwidth event synchronization schemes. There are several distributed object synchronization mechanisms available over heterogeneous environments. The most widely applied distributed object paradigms are the Distributed Component Object Model (DCOM) from Microsoft [4], the Common Object Request Broker Architecture (CORBA) [5] from OMG, and the Java/Remote Method Invocation (Java/RMI) [6] from *JavaSoft*. However, it turns out that these mechanisms induce very high latencies due to the large amount of exchanged data, as well as their complexity in updating remote events.
In this paper, an alternative solution is proposed using the Java Media Framework (JMF) [7], a flexible API developed both by Sun Microsystems and IBM Corporation for real time multimedia applications, which is gaining increased interest by the software development community. In the proposed solution, synchronization between the voice data stream and the application event stream is achieved by merging both streams. In order to keep the bandwidth requirements low, a dictionary based strategy is applied to encode registered events for transmission. Furthermore, to grant some level of fault tolerance, a retransmission mechanism is applied. This solution was successfully integrated and tested using HeartBit - a second opinion telemedicine application where accurate synchronization between the voice stream and application events (e.g. mouse position, clicks, etc.) is of paramount importance. Using this environment, three types of tests were conducted: (i) using a LAN with limited available bandwidth, (ii) using ADSL connections of limited upload bandwidth (128kbps) from different providers and (iii) an ordinary internet connection between Portugal and Mozambique (both had a maximum theoretical upload limit bandwidth of 128kbps).
In the next section the synchronization mechanisms as well as the main implementation details are introduced. Section 3 describes some of the achieved results in the aforementioned tests. Finally, in section 4, some main conclusions are presented.

The Synchronization Mechanism

The Conceptual Solution

The proposed solution is based upon a three layer architecture. This provides the necessary abstraction in order to enable its adaptation to different systems and environment definitions.
The three layers are inspired on the OSI System Interconnection model. Namely, they are equivalent to the OSI Interaction Layer, Session Layer and Network Layer. The Interaction Layer defines the presentation interface to the application while collecting and answering all the actions performed by the user. The Session Layer is responsible for storing all the definitions required for connection management and the processing of the events, i.e. it is responsible for sending the events to the local and the remote stack of execution. It is in this layer that almost the whole mechanism takes place, as will be detailed later.

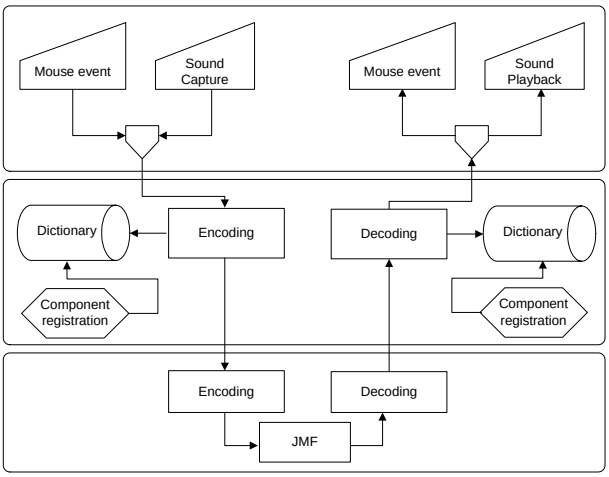

Figure 1: Synchronization mechanism overview

Finally, the Network Layer provides all the necessary services to interact with the network, and, therefore, with the remote application, i.e., its main function is to grant the transmission as well as the reception of the issued and encoded events through the JMF API.
The conceptual solution proposed herein is depicted in *Figure 1*. As it can be observed, the solution is composed by the following main stages at the sender side (at the receiver, the inverse operations are performed): event capturing, component registration, event coding and event transmission.

Event Capturing

Two types of events are considered in the proposed mechanism: (i) private events, i.e. events that must not be sent to the remote applications (e.g. local management mouse clicks), and (ii) public events, i.e., events that must be transmitted to keep the distributed working environment synchronized (e.g. starting video reproduction button click).
In order to capture and to distinguish among these events, two strategies may be applied: (i) to have a dedicated thread that permanently monitors and analyses the event stack of each client in order to identify newly issued public events and (ii) to extend the visual component classes with adequate overloading of behaviors. The latter exhibits lower implementation complexity and computational overhead, although it implies the extension of all utilized public visual component classes. Fortunately, in most graphical user interface API this can be minimized by extending the base class (e.g. the *JComponent* class in JAVA). For these components the ordinary event capturing and dispatching mechanism has to be changed. Namely, once an event is captured it has to be cloned. One copy of the *event* object is treated as usual, i.e., it is sent to the event stack of the local client. The second copy of the *event* object is sent to the session layer of the

ISSN: 1727-1983

mechanism that will encode and integrate it properly into a valid JMF data stream in order to allow its transmission to the remote application. In Java, this can be easily achieved by overloading the appropriate event listeners.
When the remote JMF's data stream receives an event, the application decodes the received array of bytes and creates a compatible event message (object) that is sent to the local event stack.

Component Registration

In order to be able to distinguish between public and private events, as well as to filter valid event messages at decoding a registration procedure is followed. Each component is added with a public/private tag, a unique component ID and its (*x*,*y*) coordinates. The public/private tag indicates if the visual component is to be sent or not to the remote application. The component ID has two important functions: (i) it uniquely identifies the visual component that issued the event and (ii) it may serve to efficiently index a data structure where all the active components are registered. Regarding the registration mechanism, it is performed whenever a visual component is created. This operation is composed by two steps: (i) setting up the component unique ID, and (ii) placing this object's reference in the active components data structure. When a visual component is disposed (i.e. whenever a particular component is no longer required due to, for example, a panel change), its reference is removed from the active components data structure. Whenever an event message is received, the decoder uses this data structure to verify if the component is active in order to further process the message. If the component is not registered it is discarded by the decoder.

Event encoding and decoding

JMF is built upon the principle of independent data sources which may be multiplexed into one data source in order to achieve a common time-base. Otherwise, these data sources are transmitted asynchronously. Hence, to achieve the required synchronization between event and other data streams, a specific JMF data-source has to be extended for the events. One possible solution to build the event data stream could be using object serialization. However, this strategy has two major pitfalls: (i) in a JAVA environment it requires compatible versions of the JVM at both application end-points and (ii) in actual tests it was verified that an *event* JAVA object does require an overhead of approximately 100kbps of available bandwidth to be transmitted in real time, i.e., to avoid event lost due to packet dropping. As JMF is built upon UDP packets, the later is a major constrain in general propose IPv4 networks, since it would occupy a major portion of the available bandwidth leading to severe QoS deterioration (sound quality, video quality and event delivery) due to packet losses.
To drastically reduce this overhead, a smaller message description mechanism was introduced. This message has to encode all the important information about the events to allow its unambiguous decoding at the remote site. To perform the correspondence between event object type and its ID, a dictionary lookup is performed. Furthermore, each message is composed by the concatenation of several context dependant elements using a system independent encoding and transfer syntax (encoding length and format, little or big endian syntax, etc.). In a pure JAVA platform this may be achieved by simply using elementary data types. Otherwise, several encoding and transfer syntaxes are available (e.g. the encoding and transfer syntax used in DICOM). Figure 2 depicts a typical message structure for a mouse event.

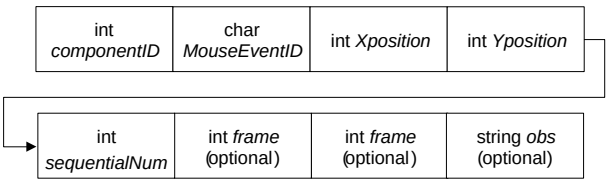

Figure 2: Message structure.

In this example, the component ID is applied to identify the exact visual component that issued the event. The *MouseEventId*, represented by a char, together with the next twp elements, *Xposition* and *Yposition* are used to represent the type of event issued. It is of note that (*x*, *y*) coordinates sent in this type of messages must not represent absolute screen or even component coordinates. These have to be encoded using a virtual coordinate system, which has to be independent of the size of the visual component as well as screen resolution, so that, when the event is performed in the remote application, the positioning can be correct, even if the remote visual component exhibits different geometrical properties (size, position, etc.).
Depending on the type of event, there may be several other event dependant fields, as well as optional fields. For instance, in order to keep the reproduction of multi-framed clinical exams synchronized, the number of each visualized *frame* may be sent in a specific event message. The actual structure of each event message is inferred from the message ID fields and from the dictionary.
At the remote application client, once an event message is received, the inverse flow of operations is performed to build a regular event object for the local platform. This object is then placed on the application's event message stack.

Event management and transmission

JMF data streams are transmitted using the RTP protocol. This may lead to packet losses under network congestion, since it uses UDP packets transmission. From the user perspective this may induce some critical and annoying situations. It is well known that the human auditory perception is able to adequately reconstruct lost audio packets from context. However, losing an event may completely compromise the collaborative working environment (e.g. missing to move the mouse pointer to a specific region of an X-ray during reporting). Therefore, some fault tolerance mechanism is required

ISSN: 1727-1983

at this stage. Actually, there are several solutions for this problem. One could implement a packet reception acknowledge mechanism, similar to TCP. This solution would, however, induce a considerable computational overhead, while doubling, at least, the latency. Another solution can be the periodic repetition, up to a maximum number of times (typically this should be limited to some milliseconds), of the issued events. In order to avoid the duplication of these events, they have to be filtered at the receiver. This filtering mechanism consists in saving the last event received for each visual component at the remote application side and to compare it against similar events previously received (up to given time-out). Assuming that packet losses are *iid*, it is observed that the probability of lost decreases by the exponent *n*, where *n* is the number of repetitions. Hence, using this mechanism it is possible to control the probability of losing an event. Of course, this solution leads to a linear increase of the required data to be transmitted. However, given the small data overhead induced by the described dictionary message encoding approach, it is observed that this solution is quite acceptable for most type of events. Since, the applied data packets are UDP, packet ordering is not guaranteed at reception. Hence, in these messages a message number is sent. Each event source has its independent counter, which is only incremented when a new event is issued by the source. The receiver will ignore all messages whose message number is less or equal to the last received event message of the same source (to allow counter reset a threshold may be applied).
Exceptions to the described strategy are events issued by mouse movements. Usually, it is observed that these events tend to be issued in very small time periods in bursts. Hence, the described procedure could induce some latency. To solve this, during mouse movements, events are send in 25 millisecond intervals to the remote application. To achieve the necessary mouse positioning accuracy when the mouse movement stops, an event message that encodes the current mouse position is periodically sent to the remote application, by using an independent thread. This avoids the aforementioned drawbacks, while granting the necessary mouse positioning accuracy required for most telemedicine applications. At the receiver, before inserting the newly arrived mouse move event, the event stack is checked and similar events waiting to be dispatched are removed if they are contiguous and their (*x*,*y*) correspond to less than a predefined horizontal or vertical translation.

JMF Integration

As was already stated, the synchronization mechanism is built upon the JMF API. This API is responsible for the management of connections and real time data streams, and natively uses independent *datasources* for sound and video. To have a common time base between one of these sources and the event data stream, it is necessary to extend a specific *datasource*. This process enables JMF to multiplex both streams into one data stream at the sender and to perform the inverse operation at the receiver, as depicted in Figure 3.

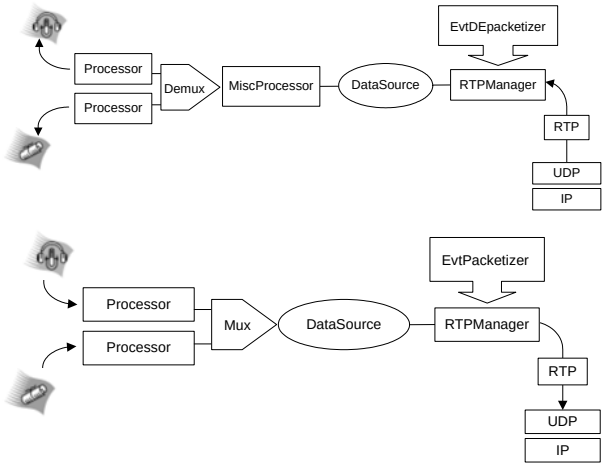

Figure 3: JMF (top) receiver model; (bottom) sender model.

To achieve this, a new JMF compatible codec for the event data stream has to be defined and registered in JMF. For this goal, two classes need to be implemented: *EvtPacketizer* and *EvtDepacketizer*. The former is responsible for encoding the data received from the capture device into the assigned encoding format. The latter class is responsible for the decoding, performing the reverse process. These classes implement the JMF *Codec* interface and overload its *process* method, responsible for encoding and decoding the received packets. In the current implementation, the only action taken in this process is to transfer the stream into the data buffer (encoding) and to perform the inverse operation during decoding. The *EvtPacketizer* and *EvtDepacketizer* classes are responsible for carrying information about the new codec and supported formats, which are registered by the built-in JMF object *RTPManager*, through the *addFormat* method. This manager class is responsible for setting up the RTP session variables, like local and remote addresses. Associated to the session there is a stream that allows the communication between the two endpoints. Whenever new data arrives through this data stream, an event is triggered to the corresponding listener. In order to process the incoming data, *SyncReceiver* and *SyncSender* classes have to be implemented as *RTPManager*'s session listeners. These classes must overload the *update* method, responsible for *datasource* association to the session stream.
In order to enable the *SyncSender* class to handle and to process the received data (events or sound), it is necessary to declare it as a controller listener, by overloading the *controllerUpdate* method.
When the *RTPManager* receives data from the incoming data streams, a *datasource* has to be associated in order to receive and to manipulate (stop and start) the data using the *SyncReceiver*, through the associated *Processor*.
When receiving data, a single *datasource* is de-multiplexed into several different *datasources*, each one being accessed by the correspondent *Processor* (e.g. events and sound). When sending events and sound, the two Processors feed one *MiscProcessor* that will merge

ISSN: 1727-1983

them into one single *datasource*, as depicted in Figure 3 (bottom).

Experimental Results

The described synchronization mechanism was tested, integrated into a telemedicine application specifically developed to enable distributed real time clinical exams discussion and reporting between to clinical experts (see Figure 4).

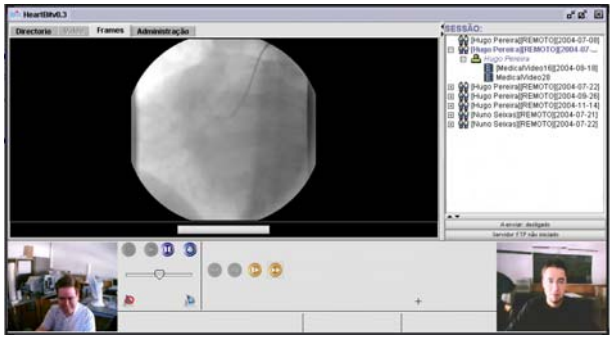

Figure 4: HeartBit screenshot.

In this type of applications it is fundamental to keep the voice stream perfectly synchronized with mouse movements as well as with application control commands (e.g. playing or stepping through a multi-frame exam). This application presents at each client two mouse pointers, one remote and one local. To test mouse position accuracy and its synchronization with respect to the audio stream a test image with several well identified landmarks was used, as it can be seen in Figure 5.

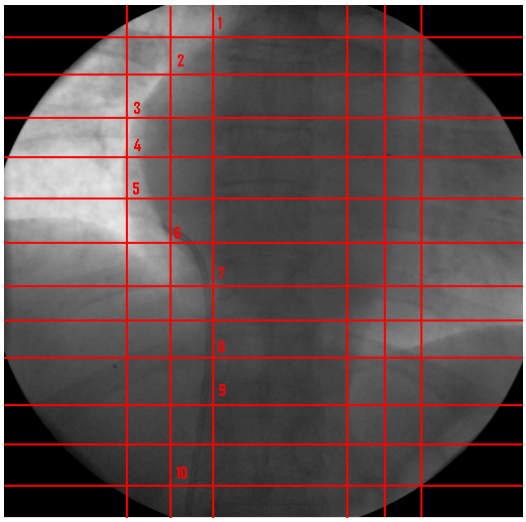

Figure 5: Test image with landmarks.

During the performed tests, the remote user moved his mouse pointer through the landmarks (using sequential and non-sequential mouse movements) on the image while stating the landmark number under his mouse pointer. Synchronization between the audio stream and the mouse position was evaluated qualitatively by the remote user. A similar setup was applied for assessing the synchronization of stepping operations through multi-frame exams. During all these tests an audio stream and a video stream were transmitted between each client application. The applied codecs were, respectively, the G.723 and the H.263 (QCIF at 5fps), which are part of the JMF implementation.

Regarding the network setup, three types of networks were applied: (i) a LAN (laboratory of Centre for Informatics and Systems) with limited bandwidth (the bandwidth was artificially limited using a commercial Bandwidth Controller [11]), (ii) a WAN through ADSL connections of limited upload bandwidth (128kbps) from different providers and (iii) an ordinary WAN intercontinental internet connection between Portugal and Mozambique. Each of these tests have been performed at several levels of network congestion by choosing different timings for the performed tests (no actual network congestion measurements were performed; congestion was assessed using ping response latency).

The achieved results are reported in Table 1.

Table 1: Result table that present the several synchronization testing stages. (*) Adapted by bit rate controller.

Test	*Used bandwidth (kbps)*			*Quality*
	Video	*Sound*	*Events*	
LAN@ 128Kbps	80	40	1	Good sound quality, synchronized with the remote events
LAN@ 81Kbps	40*	40	1	Acceptable sound quality, synchronized with the remote events
Regional WAN	80	40	1	Acceptable sound quality, synchronized with the remote events
Regional WAN	OFF	40	1	Good sound quality, synchronized with the remote events.
IntCont. WAN	80	40	1	In Mozambique: acceptable sound quality, events synchronized but with some marginal losses. In Portugal: not understandable sound, unable to synchronize events.
IntCont. WAN	OFF	40	1	In Mozambique: good sound quality, events synchronized. In Portugal: Sound not perceivable, unable to synchronize events.

As it can be observed from the tests performed using the LAN of the laboratory with bandwidth strangling, the application performed as expected up to a limit of 81kbps of available bandwidth. If the video data stream was turned off, similar results were achieved up to a limit of 41 kbps. In each case it was observed that the required bandwidth for sending the synchronization events was 1Kbps. This is a significant result compared to 100kbps of required bandwidth, if ordinary event object serialization is applied (it should be noted that

ISSN: 1727-1983

currently most internet connections do not provide this bandwidth).
The performed tests WAN based on two commercial 128 kbps (upload) ADSL connections in the same country (provided by two different providers) provided similar results, i.e. good sound quality was achieved with a perfectly synchronized environment. Only few events were lost during these tests (no actual measurement was performed).
As was already mentioned, a third series of tests were carried out in an intercontinental operation scenario using a regular internet connection. Namely, several sessions between Portugal and Mozambique were conducted. During all these sessions it was observed that sound and events initiated at the Portuguese side arrived fully synchronized at Mozambique and that events loss was insignificant. Regarding the video quality, a significant packet loss was noticeable (an image without reconstruction errors was only obtained from I frames; almost all predictive frames induced noticeable reconstruction errors in several blocks). It should be noted that the H263 is very sensitive to packet losses; this problem has only been addressed in the new H264 where multiple references for predictive blocks are available. In this context the H264 could solve the aforementioned problem with the added advantage of being a much more optimized codec with equivalent video quality at much lower bit rates.
Regarding the data flow from Mozambique to Portugal, a very high packet loss rate was verified. It was observed that almost none of the transmitted video frames could be reconstructed at the receiver without significant error. Furthermore, the sound was almost imperceptible due to jitter and high packet loss. Finally, most synchronization events were lost. The differences between the two flows are now under analysis by the African network provider. The explanation for this behavior may come from observed latency times. In fact, when connecting to Mozambique, this measure increases rapidly, causing a massive packet drop in the UDP protocol.

Conclusions and Future Work

This paper introduces a mechanism to synchronize events for real time distributed applications in low bandwidth and best effort networks. Although this mechanism was build upon the JMF API it is easily adaptable for other commercial and non-commercial videoconference APIs operating in heterogeneous environments.
Several tests conducted under real-life conditions have shown that the proposed mechanism is able to achieve the required synchronization, even for events that occur with high frequency, with a communication overhead of 1kbps. Furthermore, although it may not be absolutely guaranteed that the sent events actually arrive at the destination, a retransmission mechanism was defined to minimize event losses. This is a topic that has to be further researched. One possible solution could be the incorporation of an acknowledge mechanism to signal the event source that the event message has been delivered successfully. This could be used by the event source to signal the local user that a given command was successful or unsuccessful at destination. However, this may not be the typical TCP mechanism, since once an event is triggered locally it must produce immediate effect in order to keep it synchronized with the voice and video streams.

Acknowledgements

This project was partially financed by POSI of the Portuguese Foundation for Science and Technology and the European Union FEDER. The authors would like to express their acknowledgement to VisaBeira, S.A., and to the Instituto do Coração de Moçambique for providing the necessary technical support and resources for the intercontinental tests. In this context, we would like to acknowledge the direct involvement of Eng. José Luís Nogueira, Eng. David Rodrigues and Eng. Jorge Cunha from VisaBeira, S. A..

References

[1] BARBOSA, A. (2001): '*HealthNet: Um sistema integrado de apoio ao telediagnóstico e à segunda opinião médica*', Master Thesis, Center of Informatics, Pernambuco Federal University.

[2] PEREIRA, H., CURADO, M., CARVALHO, P. (2005): '*QoS in real time data transmission*', Department of Informatics Engineering, University of Coimbra.

[3] WU, D., HOU, Y. T., AHU, W., ZHANG, Y., PEHA J. (2001): '*Streaming Video over the Internet: Approaches and Directions*', IEEE Transactions On Circuits and Systems for Video Technology, vol. 11, N.3, pp. 283-300.

[4] WANG, Y., DAMANI, O., LEE, W. (1997) '*Reliability and Availability Issues in Distributed Component Object Model (DCOM)*', AT&T Labs-Research and Univ. of Texas at Austin and New York University.

[5] WANG, L., RODRIGUEZ-TOMÉ, P., REDACHI, N. (2000): '*Accessing and distributing EMBL data using CORBA (common object request broker architecture)*', Genome Biology - research 0010.1-0010.10; 2000.

[6] WALDO, J. (1998): '*Remote procedure calls and Java Remote Method Invocation*', Sun Microsystems.

[7] SUN MICROSYSTEMS, INC. (1999): '*JMF 2.0 API Guide - Working with Time-Based Media*', http://java.sun.com/products/javamedia/jmf/2.1.1/guide/JMFTBM.html

[8] ZHANG, J., STAHL, J, HUANG, H., ZHOU, X., LOU, S., SONG, K. (2000): '*Real-Time Teleconsultation with High-Resolution and Large-Volume Medical Images for Collaborative Healthcare*', IEEE Transaction on Information Technology in Biomedicine, vol. 4, No.2, pp. 178-185.

[9] BOUAZIZI, I., GUNE, M. (2003): '*A Framework for Transmitting Video over Wireless Networks*', Department of Computer Science, University of Aachen.

ISSN: 1727-1983

[10] WOLF, S., PINSON, M., CERMAK, G., TWEEDY, E (1997): '*Objective and Subjective Measures of MPEG Video Quality*', Institute for Telecommunication Sciences.

[11] http://www.bandwidthcontroller.com.

ISSN: 1727-1983