An Adaptable Architecture for Mobile Streaming Applications

Mabel Vazquez-Briseno [1] and Pierre Vincent [2],
[1] GET-INT, RST Department, Evry, France / UABC, Ensenada, Mexico
[2] MobKit, Lille, France

Summary

Many emerging mobile applications and services require playback of streaming media. In this paper we describe an adaptable system architecture to implement mobile streaming services. The main components of this architecture are the streaming server, the multicast proxy and the mobile client. The main novelty of our approach lies on the client which is designed to fit most mobile devices. The streaming servers and client are all compliant with 3GPP standards, and therefore use the Session Description Protocol (SDP), Real Time Streaming Protocol (RTSP), and Realtime Transport Protocol (RTP), as well as the Adaptive Multi-Rate (AMR) audio media standard.

Key words:
Mobile computing,, multimedia, software, streaming.

1. Introduction

The rapid growth of mobile communications systems has made possible to provide mobile users with new services and applications. Among these, streaming is one of the most appealing and interesting services. Streaming refers to the ability of an application to play synchronized media streams like audio and video in a continuous way while those streams are being transmitted to the client over a data network.

On the other hand mobile applications provide a new set of design challenges for application designers. Concerning the implementation of mobile streaming applications, terminal heterogeneity is a major challenge. Since mobile terminals have a wide range of different capabilities it is not probable that all of them will be able to support all proprietary Internet streaming formats and protocols. A common standardized format is then required to guarantee the creation of compatible solutions. The Third Generation Partnership Project (3GPP) has standardized streaming services and specifies both protocols and codecs [1]. As protocols the 3GPP defines the Real-time Streaming Protocol (RTSP) and Session Description Protocol (SDP) for session setup and control, and the Real-time Transfer Protocol (RTP) for transporting real-time media such as video, speech, and audio. The standardization process has also selected individual codecs on the basis of both compression efficiency and complexity, among these MPEG-4 for video and AMR (Adaptive Multi-Rate) for audio.

Another important issue related to the development of mobile applications is that mobile devices have limited resources, for that reason client applications must be adequate to the mobile environment. Several enabling technologies to construct mobile applications have been introduced in recent years. Among these technologies, Java 2 Micro Edition (J2ME) [2] is one of the most popular, due to the fact that the majority of mobile devices support it, thus guarantees compatibility.

Taking into account these aspects we have designed a generic architecture in order to facilitate developers work. Our architecture implements the server and client side according to the 3GPP standards. There are several mobile applications that implement streaming but the main idea of our approach is to provide a ready-to-use streaming system adapted to the specific requirements of each developer and that fits most mobile devices. J2ME is used to develop the client side and to implement the RTSP protocol on mobile phones, even if they formerly lack of support for this protocol. The architecture also considers the possibility of multicast streaming using one mobile phone to send a media stream to several ones.

In the following section we explain the architecture components and their implementation. Section III demonstrates the utility of our approach by building and testing a working example. Section IV provides the conclusions and future work.

2. Architecture Description

The main purpose of our approach is to facilitate the work of mobile applications developers by providing a set of reusable software elements. These elements can be easily adapted to different applications that require the streaming principle. To achieve this goal we conceived a generic architecture that can fit diverse mobile streaming

Manuscript received September 5, 2007
Manuscript revised September 20, 2007

applications. Fig.1 depicts the main components of this architecture.
In frame A of Fig. 1 we show the components of a basic streaming system. In the simplest case a mobile client communicates only with a RTSP server to get a particular file. The media that can be streamed consists of AMR audio files. It is also possible to add a PHP server in order to store and administrate media files received from the clients. HTTP connection is then required to send files from the mobile phone to the server. Furthermore the PHP server can be used to provide security to the system by maintaining a list of authorized users, called members, into a MySQL database. In this case the user will require authenticating before been able to start a RTSP session.

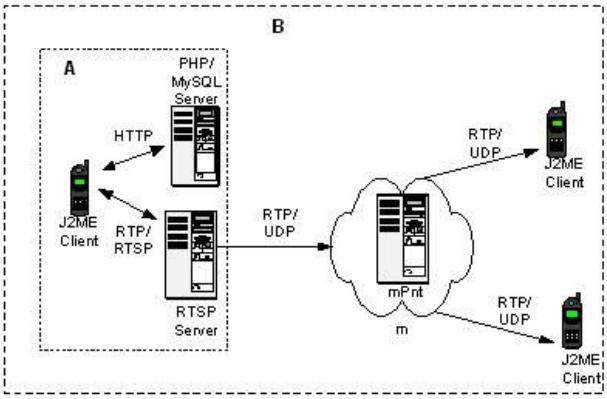

Fig. 1 Architecture components

Frame B of Fig. 1 shows a system that supports also multicast streaming. In this case one mobile phone starts a RTSP session with the server, indicating in the SETUP event that it is a multicast session. A multicast session is established using the Multicast Proxy Network Platform (MPNT), which was developed with Java programming [3]. The element *m,* in Fig. 1, may consist of a single MPNT proxy or a MPNT network, which interconnects several proxies. All mobiles connected as clients to this multicast session will receive the streamed data, but it will be controlled only by the mobile that started the RTSP session.

Depending on the mobile application requirements the developer may add or remove system components. Among them the J2ME client is the most flexible one. The idea is to have a client that contains only the functionalities required by the application. For example, if the application does not comprise multicast streaming or member's authentication, then the client will have neither the J2ME classes required to support multicast nor the ones required to support HTTP connection. The system created in this case will consist only of the J2ME client and the RTSP server.
The components included in the generic architecture are described in the following sections.

2.1 RTSP Server

RTSP is an application-level protocol created for the purpose of controlling the delivery of data with real-time properties [4]. It provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. The RTSP server in our architecture is written in Java SDK 1.5. It implements the basic RTSP mechanisms to control the streamed data, but once the session is established, media streams are sent using RTP over UDP as transport mechanism as depicted in Fig. 2. A RTSP session is started when the client sends a Describe/Setup event containing all the session information into the Session Description Protocol (SDP). The Setup event causes the server to allocate the resources for the stream and to determine if this is a unicast or multicast session. In the case of a unicast session the streamed data is sent directly to the mobile client. On the other hand, if this is a multicast session, the streamed data will be sent to a MPNT proxy. Once the data is being streamed the mobile client can control it using the RTSP events as Play, Pause and Stop that are implemented in the server. All RTSP requests and responses between the server and the clients are carried out using Socket connections. RTP/UDP delivery is done using a Datagram connection. The server sends one AMR frame in each RTP packet. An AMR audio frame consists of 1-byte header, and several bytes of audio data. The entire frame is fed into the AMR decoder. Each frame represents 20 ms of speech encoded with an AMR mode (0-8), for AMR mode 2 the bit rate is 5.9 kbps, and each frame is 16 bytes long. The J2ME client receives each frame into a buffer and reconstructs smaller AMR files in order to play them, as explained in section 2.4.

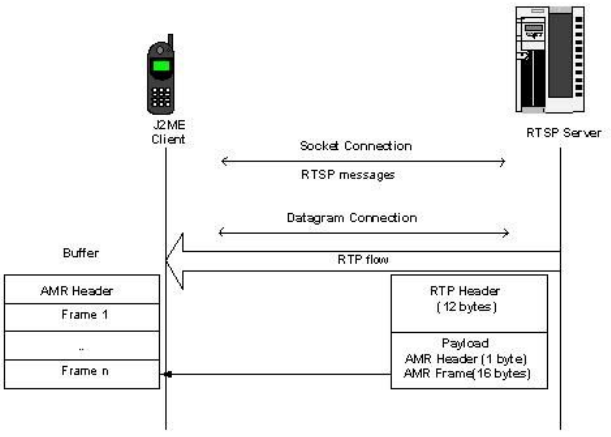

Fig. 2 J2ME client and RTSP server communication

2.2 MPNT Platform

The MPNT (Multicast Proxies NeTwork) platform is an overlay architecture conceived at GET-INT to provide multicast access to unicast-only users or multicast clients. It is written in Java, thus it is totally compatible with the components of our generic streaming architecture. Unlike multicast IP, MPNT does not intervene at network layer but at both transport and application layers. MPNT architectures are composed of a set of relaying nodes (called mPnt proxies), interconnected as an overlay network. Each node provides access for a subset of the protocol stack given by Fig. 3. Providing an access means launching a listening server with specific transport and application layer. Examples of accesses can be [RTP||UDP], [RTP||UDP/IP multicast], or [SIP||TCP].

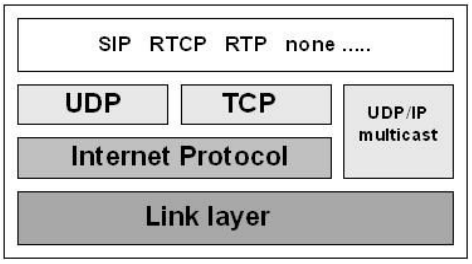

Fig. 3 MPNT protocol stack

As explained before, a mobile streaming application establishes a session with the RTSP server to allocate resources and control data, but actual media is streamed using RTP/UDP. If the application requires multicast communication, MPNT has to be incorporated to the mobile streaming architecture providing the RTP over UDP or RTP over UDP/IP multicast services.

A mPnt proxy is made up of two entities: a set of forwarding servers, and a managing tool for launching and stopping these servers, making interconnections, etc. MPNT architecture can be used in standalone mode or jointly with multicast IP sessions. Each mPnt can be connected to several multicast sessions; each one of them will be associated to a user group (SIP, RTP, RTCP or "none"). It is also possible to interconnect several mPnts using different topologies [5] in order to ensure load scalability and provide more flexibility for conceiving architectures. To avoid routing cost a single-path interconnection model was adopted. In this case each interconnection consists of creating two channels: one for management, and another for data. Concerning the tables, each mPnt has got two: one containing connected mPnts and their access path, and another one for available servers on the mPnt network. The control channel is used also to detect interconnection failures and remedy them. The mechanism used is a dynamic recovery algorithm that reconfigures the architecture after a failure [5]. In local use, meaning multicast applications restricted to a LAN, MPNT duplicates messages so it is not as efficient as multicast IP, but in a WAN context it can be more efficient because it is based in tunnelling but at application layer, the overhead is then smaller than IP overhead. Another advantage is flow multiplexing; a single channel is used to carry all session's messages.

2.3 PHP/MySQL Server

The PHP Server was implemented using PHP version 5.0, Apache Server version 2.0 and MySQL version 5.0. It consists of PHP scripts that accomplish authentication functions as well as communication with the J2ME clients. All server/clients communication is done using HTTP. A mobile streaming application may require this server to allow the clients to store media files that will be streamed later. The server can also be used to add security to the system by maintaining a database with the records of the authorized users allowed to access the streaming system. In order to add this server to a mobile application the J2ME clients must have the HTTP connection module as explained in the next section.

2.4 J2ME Client

J2ME supports programming on mobile devices. The architecture of J2ME consists of: configurations and profiles [6]. The first one defines the minimum set of java core classes required by the virtual machine to work. The second one adds additional functionalities for specific devices. The Mobile Information Device Profile (MIDP) is the profile corresponding to mobile phones. The applications developed using the MIDP profile are called MIDlets. A J2ME-application suite is stored on a JAR file. The size of this file is important because most mobile phones have a memory size allocated for MIDP applications not greater than 64k.

The J2ME client or MIDlet in our architecture is a very flexible component. The idea is to construct it using an applications generator framework that we have previously conceived [7]. This framework consists of a set of Java classes and PHP scripts; it uses Velocity templates [8] to merge the information provided from the developer with predefined modules already included in the framework's library. We have now extended this framework to create a J2ME application that supports streaming. A mobile

streaming client is then implemented by adding several reusable elements and determining its attributes in order to satisfy the particular mobile application task. Each element consists of a set of J2ME classes that implement a specific function. They are described in the next sections.

2.4.1 RTSP_Client

This element provides a template to implement the communication with the RTSP server. It establishes a socket communication using TCP and sends the Describe, Setup, Play, Pause and Stop events.

Once the RTSP session is established, another class, called Datagram_Listener, is in charge of receiving and playing the streamed media. This class is then the one that implements the streaming function. It extracts data from the RTP packets and plays the media. To play media the J2ME clients use their Mobile Media API (MMAPI) capabilities. MMAPI is an extension to J2ME. It supports time-based multimedia on small wireless devices [9].

Unfortunately J2ME and MMAPI do not support playback of streaming media. The player in MMAPI requires a whole AMR audio file to be able to play it. To resolve this aspect it was necessary to reconstruct new smaller files from the original one. The server streams one AMR frame from the whole file in each RTP packet, the client then waits to have a specified amount of AMR frames to construct a file that is then played. While this small file is been played the server continues sending RTP packets, for this reason two buffers and two players where used. When an AMR file is ready, the first player starts to play it and the second buffer starts to receive the AMR frames, then when the second buffer is filled, the second player starts to play this buffer and the first buffer starts to receive the AMR frames. This process is done during all the transmission as depicted in Fig. 4.

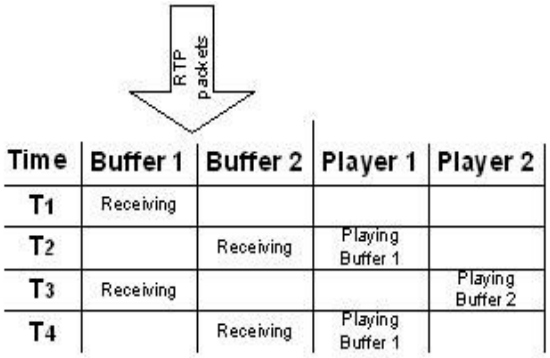

Fig. 4 Streaming in the J2ME client

2.4.2 Voice_Record

This template allows the user to record a voice file that can be sent to the PHP server using HTTP. To generate a new audio file the phone uses the MMAPI audio capture feature. Most mobile phones have only the capacity to capture WAV audio, but recent models can also capture AMR audio, which is a most compressed and efficient format. Audio files can be captured using WAV or AMR, but if these files are stored in the server to be streamed later, they are converted to AMR format.

2.4.3 Http_Connection

Even thought the HTTP protocol is not compulsorily required in a streaming application, we have included it in order to allow the client to communicate with the PHP server. The Post and Get methods have been predefined and can be easily adapted adding the information concerning to the HTTP server URL and the data to be sent.

2.4.4 MPNT Connection:

This element starts a RTP/UDP connection with a mPnt proxy through its IP address. It is possible to add a multicast client or server or both to a mobile application. A multicast client starts an UDP connection with the corresponding mPnt proxy and waits until it gets information. The Datagram_Listener class is reused in this component to play RTP packets once they are received. A multicast server is a mobile that is able to stream a file that is stored in a remote server to several multicast clients. The mobile acting as multicast server sends RTSP events to the RTSP server using the multicast option into the SDP protocol. In this case the RTSP server sends the RTP datagrams to the mPnt proxy. All multicast clients will receive data but only the mobile working as multicast server can control the Pause, Play and Stop events.

3. Tests and Results

In order to test the system we implemented a mobile streaming application with all the components shown in

Fig. 1. The mobile J2ME application created using the framework consists of two MIDlets: Voice Messages and Multicast Audio Streaming. Both MIDlets share the basic streaming classes defined previously and communicate with the same servers. The J2ME client application structure is showed in Fig. 5.

The Voice Messages MIDlet's goal is to exchange voice messages among mobile members already registered in the server's database. Voice messages are captured in the phone and then are sent to the PHP server. They are stored in the space of safeguard corresponding to the addressee member. To get the messages a client connects to the PHP server and gets a list of hers/his messages. To listen to a message the mobile client has to connect to the RTSP server and send the information corresponding to the desired file. After the session is established the voice message is streamed.

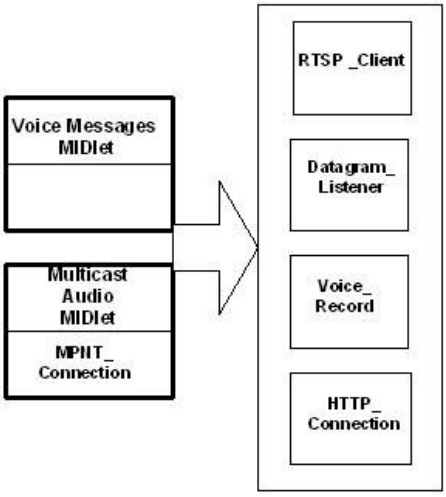

Fig. 5. Shared classes in a J2ME application

The Multicast Audio Streaming MIDlet implements the multicast client and server functions. As a client it is able to connect to a MPNT proxy and be added to a list of listeners. As a server it is able to start a multicast RTSP session to stream an audio file already stored in the mobile or to record a new audio file. In order to stream a file to a multicast group it must first send the file to the PHP server.

All servers were installed on the same PC and we use a Sony-Ericsson W810 mobile phone to capture and send AMR voice files to the PHP server. The mobile phone was also used to start a multicast session. The streamed data was sent to several phones emulators installed on other PCs. We use the Sprint simulator with emulators for the Samsung A900, A920, A940 as well as Sanyo MM-7500 and 9000. We also use the Siemens SL65 emulator. The mobile application works fine on the mobile phone as well as on the emulators. The change between buffers and players is not perceptible, there is only a little delay before starting to play the media that corresponds to the time required to fill the first buffer.

The J2ME JAR file containing all the streaming classes included in the framework is 61.4 k. It could be possible to have a smaller client with fewer functions, for example only the RTSP client, or multicast client. There are many possibilities to construct applications with smaller JAR sizes and not much to have greater ones thus is possible to affirm that the generated MIDlet suite will always fit into most mobile phones.

4. Conclusion

Streaming on mobile devices is a huge challenge for developers. Those devices have different and restricted capabilities. The newest ones support the RTSP protocol, but not all of them do. On the other hand most mobile devices are J2ME-enabled. In this paper we presented a solution to implement streaming mobile applications using RTSP even if the mobile phone does not formerly support it. We propose an architecture that includes all the required components to implement most mobile streaming applications. These components can be added to a streaming system and adapted if it is needed. The most flexible one is the J2ME client; it can be easily created by adding some predefined modules that perform the main functions required to support streaming on a mobile phone. We demonstrate the utility of our architecture by creating an application with all the possible components. Even with all the functionalities the JAR file of the MIDlet suite generated is a normal size application that can be installed without problem on almost any mobile phone.

We consider that the architecture presented can be adapted easily to fit specific needs of a streaming application that could be for example: a music sharing system, an electronic library system, a mobile learning system, among others.

As future work we plan to add other streamed media like MPEG-4 video and MP3 audio. For this it will be only necessary to add the corresponding modules to the RTSP server and the J2ME client, without any modification on the other components.

References

[1] 3GPP, TSGS-SA, Transparent end-to-end Packet Switched Streaming Service(PSS). Protocols and codecs.(Release 7), TS26.234 v.7.1.0 (2006-12).
[2] Java 2 Platform, Micro Edition- J2ME, Official Site, http://java.sun.com/javame/index.jsp.
[3] Sbata K., Vincent P., Benaini R. and Naja N. MPNT: An architecture for multimedia applications. IWCSE Workshop, part of the 2nd IEEE ISSPIT, December 2002.
[4] H. Schulzrinne, A. Rao, R. Lanphier and M. Westerlund, "Real Time Streaming Protocol. RFC 2326, March 2003.
[5] Karim Sbata, Redouane Benaini and Pierre Vincent, Comparative study of MPNT topological models, IEEE Proceedings of the 2005 Systems Communications (ICW'05).
[6] Muchow, J., Core J2ME Technology & MIDP, The SUN Microsystems Press, Prentice Hall, London, 1st. Edition, 2002.
[7] Vazquez M., Vincent P., A Modular and Extendable Framework for Mobile Applications Generation. WSEAS Transactions on Computers, Oct. 2006, Issue 10, Vol.5, ISSN: 1109-2750.
[8] Naccarato, G., Template-Based Code Generation with Apache Velocity. http://www.onjava.com., 2004.
[9] Mobile Media API 1.2. Sun. http://java.sun.com/products/mmapi/ .

Mabel Vazquez-Briseno
[Mabel.vasquez_briseno@int-edu.eu, mabelvb@uabc.mx] received the M.Sc degree in Electronics and Telecommunications from CICESE Research Center, Mexico, in 2001. She has been a lecturer of several B.Sc and M.Sc computers and networks courses at the Autonomous University of Baja California (UABC), Mexico, since 2002. Currently she is a Ph.D student at the Institut National des Télécommunications (GET-INT) in Evry, France. Her research interests include computer networks, mobile computing and protocols.

Pierre Vincent
[Pierre.Vincent@mobkit.com] Received the Ph.D degree in Computer Science from the University of Paris 6 Jussieu in 1988. He was assistant professor at GET-INT from 1986 to 1999. Then, he participated to the creation of ENIC, a computer and network faculty based in Lille where he worked from 1990 to 1999, as head of the Computer and Network department. At ENIC, he developed Teleteaching facilities and was also in charge for international relations. Back at GET-INT, from 1999 to 2005, he was the head of Software and Network Department, where he managed the development of communication services. At the present time, he works for MobKit, a software company that he has created to develop on-line services for mobile phones and audio conferencing His research interests include networks, computer programming and software for Internet.