JAVANETPHONE: A JAVA CLIENT FOR IP TELEPHONY APPLICATIONS IN AN MGCP FRAMEWORK

Susanna Spinsante[†] *(Student Member IEEE), Franco Chiaraluce*[†], *Ennio Gambi*[†], *Aldo Vespasiani*[‡], *Alessio Perotti*[‡]

[†] Dipartimento di Elettronica, Intelligenza artificiale e Telecomunicazioni, UniversitàPolitecnica delle Marche
Via Brecce Bianche, I-60131, Ancona, Italy
phone: + (39) 071 2204894, fax: + (39) 071 2204835, email: s.spinsante,f.chiaraluce,e.gambi@univpm.it
web: www.deit.univpm.it
[‡] Selta Telematica S.p.A.
Via Nazionale-Km. 404,500, I-64019 Tortoreto Lido, Teramo, Italy
avespasiani, aperotti@seltatel.it
web: www.selta.net

ABSTRACT

Many software applications have been developed in recent years, to exploit the convergence between voice and data networks and the availability to the final user of high speed and low cost connections. These software applications operate in real-time communication scenarios and, at least in principle, should ensure portability and platform independence. In this paper a software IP telephony application completely written in Java language and called JavaNetPhone is presented. The aim of such a communication tool is to operate in a business LAN environment, based on the Media Gateway Control Protocol framework, to extend the already available telephony facilities. At the authors' best knowledge, other similar applications exist, but they adopt different programming languages and sometimes are not platform-independent. The prototype functionalities evaluation show that the application can give acceptable voice performance, and further developments are being pursued to provide the adoption of wireless communication technologies, such as BlueTooth and WiFi, according to the increasing demand for user's mobility.

1. INTRODUCTION

In recent years, many software applications have been developed in order to integrate conventional telephone services with the new communication facilities exploiting the Internet Protocol (IP) paradigm. The nature of IP telephony is intrinsically software-oriented, thus making it possible to design telecommunication applications suitable for desktops or handheld PCs. These applications face the problems related to real-time communications and, at least in principle, should ensure portability and platform independence. In this paper we present a platform independent IP telephony application, completely developed by means of the Java language, called JavaNetPhone (JNP), which is able to inter-operate in a pre-existent Media Gateway Control Protocol (MGCP) [1] framework. The object of this tool is to extend the telephony facilities already present in a business LAN, and provided by means of hardware VoIP (Voice over IP) phones and a VoIP gateway. The gateway is a traditional PBX (Private Branch eXchange) equipped with a VoIP board to perform H.323 and SIP (Session Initiation Protocol) gateway functions towards the external domains, and to act as a Call Agent (CA), by managing connections and signalling exchanges among the MGCP phones within the LAN. The JNP has been entirely developed in Java: many other applications for IP telephony exist in different programming languages but often they are not completely portable. Furthermore, many of them are based on implementations of the SIP or H.323 signalling stacks, and consequently cannot be applied in different environments. The MGCP centralized architecture fits well a traditional PBX functional scheme; moreover, its packet format allows an easy encapsulation of proprietary commands inside the MGCP envelopes, to provide advanced PBX services to the terminals connected to the LAN. The prototype testing and evaluation showed that the application can give acceptable voice performances, and further developments are being pursued, exploiting BlueTooth and Wi-Fi connections, in order to widen the LAN telephony scenario by adding mobile communication devices.

2. VOIP PROTOCOLS AND MGCP COMMANDS

There are a number of software VoIP applications and products commercially available and developed by means of different programming languages; they can be basically divided into two major groups, one based on H.323 protocol suite and another one based on SIP (Session Initiation Protocol). H.323, a protocol suite defined by ITU-T for voice transmission over Internet, also provides mechanisms for video communication and data collaboration: it is an "umbrella" specification, including various other ITU standards. SIP is the IETF standard for multimedia conferencing over IP. It is a peer-to-peer, ASCII-based, application-layer control protocol that can be used to establish, maintain and terminate calls between two or more endpoints. The complexity of SIP is adequate to IP networks, as it is an HTTP-like protocol; H.323 complexity is higher, as it uses several different protocols. Finally, the SIP architecture is modular, as many functions and services reside in separate protocols, while the H.323 architecture is quite monolithic, as its components provide a mix of services. The JNP presented in this paper does not make reference to any of these two protocol specifications:

it is based on a modified version of the MGCP signalling. This protocol was initially adopted inside the business LAN, the target application scenario, to interface the PBX and the hardware IP phones. As the software IP telephony application should interact with the pre-existent PBX and IP phones like any other hardware device, it has been necessary to adopt the same communication protocol. Since the beginning, the MGCP packets were used by the VoIP Gateway to encapsulate commands and signalling information for the hardware IP phones: in order to allow the PBX to identify the JNP, we have adopted the same commands and signalling information format, to make the IP telephony application work correctly. The MGCP is used to manage telephony gateways by means of external call control elements called media gateway controllers or Call Agents. A telephony gateway is a network element that provides conversion between the audio signals, carried on telephone circuits, and data packets, carried over the Internet or over other packet networks. MGCP assumes a call control architecture where the call control intelligence is handled by the Call Agents, which have to synchronize to send commands to the gateways under their control. MGCP is, in essence, a master/slave protocol, where the gateways are expected to execute commands sent by the Call Agents. The MGCP interface is implemented as a set of transactions composed of a command and a mandatory response. There are eight types of commands: *CreateConnection*, *ModifyConnection*, *DeleteConnection*, *NotificationRequest*, *Notify*, *AuditEndPoint*, *AuditConnection* and *RestartInProgress*. The first four commands are sent by the Call Agent to a gateway. The first command creates a connection between two endpoints, using UDP (User Datagram Protocol) to define the receiver capabilities of the participating endpoints. The other three commands modify the properties of a connection, terminate a connection, collecting statistics on its execution, and request the media gateway to send notifications on the occurrence of specified events in an endpoint, respectively. The *Notify* command is sent by the gateway to the Call Agent, to inform it about the events occurred. The gateway may also send a *DeleteConnection*. The Call Agent can send either the *Audit* commands to the gateway, in order to determine the status of an endpoint or retrieve the parameters related to a connection. Finally, the gateway may send a *RestartInProgress* command to the Call Agent to notify that an endpoint or group of endpoints are in, or out of, service. All commands are composed of a command header, optionally followed by a session description. All responses are composed of a response header, optionally followed by a session description. Headers and session descriptions are encoded as a set of text lines, separated by a carriage return and line feed character. Headers and session descriptions are separated by an empty line. MGCP uses a transaction identifier to correlate commands and responses; header and command lines are composed of strings of printable ASCII characters, separated by white spaces. In detail, the JNP has been developed in order to use not the whole set of MGCP commands, but only four of them: *Notification Request* (NTRQ), to encapsulate commands coming from the Traffic Handler, inside the VoIP board, and directed to the IP phones, *Notify* (NTFY), to encapsulate the IP phone signalling information for the Traffic Handler, *AuditEndPoint* (AUEP) used for diagnostic purposes, *RestartInProgress* (RSIP), which is used by the IP phones to ask for a connection with the VoIP Gateway and sent by the Gateway to force the disconnection of the terminals, if an error event occurs. The overall architecture of the system is strongly centralized: all operations are managed by the software module within the VoIP Gateway board, the Call Agent, which tells the IP phones the actions to be performed.

3. THE IP TELEPHONY FRAMEWORK

In order to implement the MGCP specifications, JNP has been developed on a strongly modular basis, which allows an efficient software management and a simplified updating process; moreover, this structure, which is depicted in Fig.1, fits very well the object-oriented nature of Java. Four of the five software modules are devoted to signalling and user interaction management; the fifth has to handle the audio resources and the bidirectional RTP (Real Time Protocol) transmissions between the software application and the DSP on board of the VoIP Gateway. The Network Stack Interface (NSI) acts as an interface for the signalling exchange between the Java application and the VoIP Gateway, in order to correctly receive and transmit the UDP datagrams on the network. The MSPU (MGCP Signaling Protocol Unit) module interprets the MGCP messages coming from the CA through the NSI, generates the messages to be transmitted and sets different timers necessary to control some functions, like those related to diagnostic operations. The UIU (User Interface Unit) executes the remote commands coming from the CA and generates the commands due to the user's actions on the graphical interface; the GUI (Graphic User Interface) allows a user friendly interaction with the JNP. Both the GUI and UIU also interface a relational DataBase (DB) by means of a Java wrapper for Access databases. The DB, which is managed by SQL queries, is used to implement all services related to the user's contacts management, like the Address Book and the Call Register. The RTP unit handles all the acquisition and processing operations related to the voice samples, by means of the G.711 A-law, G.729 and G.723 audio codecs, managing the vocal sessions and the transmission of RTP packets. The connection among the GUI and the RTP modules, introduced to allow the user to control both the speakers and the microphone settings, involves the linking between the GUI and the Java classes which implement audio and ring management, denoted as *MainVoiceEngine* and *Ring*. These two classes are created by the *JavaNetPhone* class; then, the GUI is instantiated, and the two classes are passed as parameters to its constructor. With the aim of making the whole application as much efficient and time and CPU low consuming as possible, its software

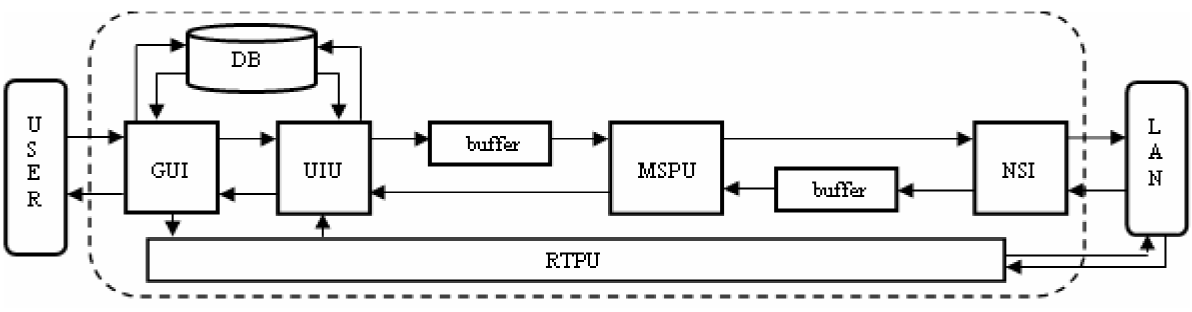

Fig. 1: The JNP modular structure

architecture has been significantly optimized. In the NSI a single thread is used to listen to the UDP socket; two threads are used in the RTP module to allow full duplex communications. Finally, two threads are used to manage incoming and outgoing data through the buffers connected to the MSPU module. All the other elements have been designed by means of Java methods and classes, avoiding any infinite cycle which could make the JNP heavily resource-consuming and eventually incompatible with other applications concurrently running on the same machine. As any other communication device connected to the PBX managing the IP telephony service on the LAN, the JNP acts on the basis of a finite states machine [2], depicted in Fig. 2, which is controlled by the Call Agent, in the sense that the transitions among the three admitted states (*Connected*, *Connected Waiting for ACK* and *Not-Connected*) depend on the response of the application to the commands coming from the Call Agent, or on local timers' expiration. When the JNP is running, the MSPU starts from the *Not Connected* state; in order to establish a connection with the Call Agent, an explicit query is sent, by the generation of an RSIP packet. The Call Agent will identify the JNP, and will give it the possibility of establishing a connection, by means of a series of parameters that the user can set by acting on a configuration panel available through the GUI. If the MSPU receives an ACK (positive response), it stores a number of parameters sent by the Call Agent and passes into the *Connected* state; otherwise, it waits 30 seconds and retransmits another request for connection. There is not a limit on the number of connection requests that can be sent. In order to maintain the connection active and the state *Connected*, the MSPU module must periodically receive an AUEP command from the Call Agent; if the AUEP command is not received after a maximum time delay (Tmax), the state becomes *Not Connected*. When connected, the JNP receives a number of commands encapsulated inside NTRQ packets necessary to handle the call. Their processing is left to the UIU module. Only when the MSPU module is in the *Connected* or *Connected Waiting for ACK* states, the application processes the NTRQ packets; otherwise, all the information coming from the Call Agent is ignored. When the MSPU is connected, it can deliver all signalling information due to the actions performed by the user to the Call Agent, enclosed in NTFY packets. If the user performs a call, the NTFY packets generated by the keyboard buttons are sent to the Call Agent. The MSPU passes into the *Connected Waiting for ACK* state, until the ACK arrives from the Call Agent, after 1 second. If the ACK arrives, the state becomes *Connected*, otherwise the NTFY packets are sent again, up to 10 times; if the maximum number of NTFY transmissions is reached, without a positive response from the Call Agent, the state becomes *Not Connected* and the MSPU has to re-negotiate a new connection.

4. THE IP TELEPHONY FUNCTIONS IN JNP

The functionalities implemented in JNP can be easily located on its GUI, which is shown in Fig. 3. Besides starting and terminating a phone call, the hold and retrieve options are also available. The VoIP Gateway can provide the hold/retrieve functions for either incoming and outgoing calls, managing the activation or deactivation of the audio circuits, holding the suspended calls, transmitting the proper tone to the held user and to the one who requested the transmission. On the other hand, the IP terminal has to send the appropriate bytes and perform the commands sent by the CA. The hold/retrieve functions in the JNP are performed by the Java classes built to manage the hook panel and the UIU module. By acting on the GUI, the user updates the Java method which is listening to the events related to the hook panel; the Java interface *VNIKeyMapping.class* associates the user's actions on the keyboard to the suitable sequence of bytes to be transmitted to the CA. All settings and parameters which can be configured by the user are located on a configuration panel. In order to allow the connection between the JNP and the CA, the network parameters must be set: the IP address of the CA and the RTP port number. Every time one of these parameters is changed, the updated value is saved in a configuration file, used to ask for a new connection. The configuration panel is implemented by extending the properties of the class *javax.swing.JFrame*; by means of the *ActionListener* interface, the values input by the user are read and stored in a file. The user can then configure the speaker device and the ringer device; this way it is possible, for instance, to handle a phone call on a USB or Bluetooth headset, while having the phone ring on the PC speakers. The speakers' volume can be set manually, while the choice of the microphone in use (desktop or wearable) automatically determines the amplification of the captured voice signal. In fact, while the user can easily check his speakers'

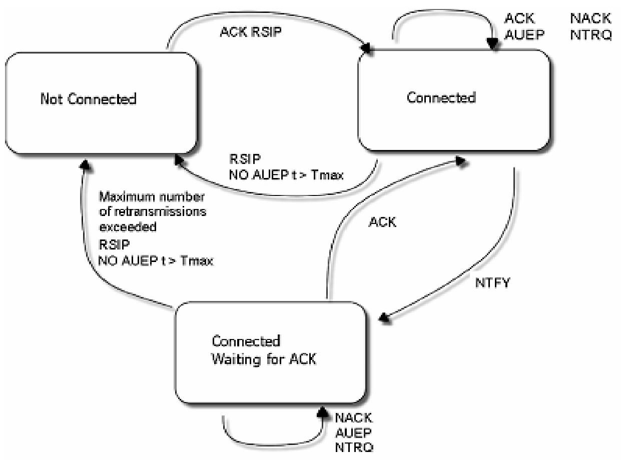

Fig. 2: The JNP finite states machine

volume, it could be very difficult and maybe ineffective trying to adjust the proper microphone settings on the basis of the subjective feedback information coming from the remote end user. Moreover, the volume of the transmitted voice signal can strongly depend on the underlying audio settings of the Operating System in use. This is the reason why an Automatic Gain Control (AGC) algorithm has been implemented, according to the type of microphone adopted, which automatically sets the coefficients used to amplify the voice samples coming from the microphone, before they are inserted in the RTP packets' payload. The GUI module invokes the constructor method for the class *VolumeSpeaker.class*, which manages the outgoing audio, deriving all its properties from the class *JPanel* in the *javax.swing* package. All events of the type *ChangeEvent* and *ActionEvent*, generated by the user's actions, are listened to and processed by the class *GUI.class*, by means of its reference which is also an argument of the constructor method for *VolumeSpeaker.class*. Also a Voice Activity Detection (VAD) scheme has been implemented in order to distinguish the speaker's voice from the background noise and avoid packets transmission when not necessary. Under the assumption that the JNP running on a given machine will be probably used again with the same audio hardware equipments, the state variables associated to the microphone selection and the gain are initiated to the same values they had in the last session. During the first execution of JNP, all these options are set to default values; then they must be properly configured by the user. If one of these configuration parameters is changed by the user, the Java audio classes of the type *MainVoiceEngine* are re-initialized. The Address Book and the Call Register provide the user some advanced services typically available on mobile phones, as the opportunity to store contacts' details (name, phone number) and to quickly activate a call, to check the lists of all the calls lost, performed and answered, to clean them or search for a specific one. All these functions are performed by making reference to a relational DB, developed by means of Microsoft Access ®, which has to be loaded on the PC running JNP.

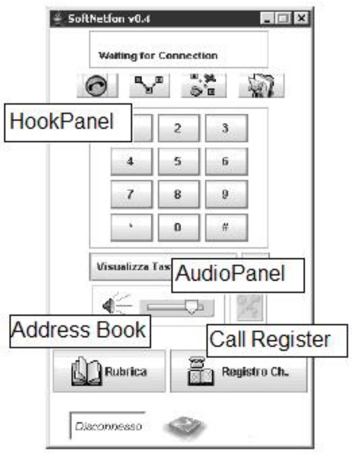

Fig. 3: The JNP GUI

5. PROTOTYPE EVALUATION

Java has emerged as one of the most popular programming languages for Internet applications, being platform independent, simple to use and able to natively support multithreading and multitasking. JNP has been entirely developed in Java, by means of the J2SDK 1.5.0: it can run on any platform having a JVM (Java Virtual Machine) installed. We have tested JNP on different hardware and O.S.s (e.g. Pentium III 1000 MHz, 192 MB RAM); even if many factors, such as the CPU clock and the hardware facilities, can affect the performance, the results obtained are quite positive. Tests have shown that when the call is established, 10%-20% of the CPU resources are used, so that other software tools available to the user on the same PC can be effectively run at the same time. In real-time communications, two main metrics are considered: the call set-up time and the media delay time. We have tested our application in a network consisting of a PBX equipped with the VoIP board, some PCs running JNP, other PCs online, hardware IP phones, digital and PSTN telephones. In all our tests, the operations related to signalling exchange have been correctly executed with no any additional call set-up time besides the one required by the MGCP, according to which the connection is enabled by the CA after a random starting delay. The control functions related to the speakers and the microphone make the audio performance of the JNP almost uniform, when tested over different hardware platforms and equipments; all the possible configurations between holding and active terminals have been tested on the network and properly managed.

REFERENCES

[1]F. Andreasen, B. Foster, "The Media Gateway Control Protocol (MGCP) Version 1", *RFC3435*, January 2003.
[2]S. Spinsante, E. Gambi, A. Perotti, A. Vespasiani "A Java Based VoIP Application: Extending LAN Telephony Facilities in a MGCP Framework", *Proc. 2004 IEEE 6th Workshop on Multimedia Signal Processing*, Siena, Italy, 29 September – 1 October 2004.