Adding High Level VoIP Facilities to the Unicon Language

Ziad Al-Sharif
Department of Computer Science
New Mexico State University

Clinton Jeffery
Department of Computer Science
New Mexico State University

Abstract

Many programming languages have built-in functions for socket programming and data communications. These days, the Internet is widely used for voice communications that employ a Voice over Internet Protocol (VoIP). A programming language with built-in VoIP functions can accelerate the development of voice enabled applications. Most programmers are familiar with manipulating files; making VoIP connections as simple to create and use as local files simplifies VoIP programming and its applications. This paper describes augmenting the Unicon programming language with very high level VoIP facilities.

1. Introduction

With the rise in popularity of digital voice communications, many collaborative applications are adding support for voice. One domain where voice communications can greatly enhance applications is the area of Collaborative Virtual Environments (CVEs). If 3D graphics make a CVE feel like a place, voice communication helps that place feel real and makes it more useful. Programming a VoIP application in a common language such as C/C++ or Java requires mastery of many low-level details. This prevents programmers from building such applications unless they have the background knowledge and the time. A very high level language that directly supports voice communications can accelerate the adoption of VoIP and the range of applications in which VoIP is supported.

This paper describes a set of VoIP facilities developed for the Unicon programming language. A small number of high-level VoIP functions is designed on top of a more complex set of underlying VoIP libraries. The paper also describes how the VoIP facilities fit in the Unicon's VM and runtime system, and their usability. Unicon's VoIP functions are built on top of an open source cross platform library called JVOIPLIB [1, 2], a C++ library that provides flexible VoIP services on both Linux and MS Windows. Unicon's VoIP facilities are very high level supporting peer-to-peer, one-to-many, and many-to-many VoIP sessions.

2. Overview of the Unicon Language

Unicon [3] is the unified extended dialect of Icon [4]. It is an open source, cross platform, high level, goal directed, general purpose application language. Unicon's control flow has a flavor of logic programming supported by the idea of failure or success, and backtracking during expression evaluation. Unicon's semantics is expression based; all code is evaluated in terms of expressions that may fail or succeed. Unicon emphasizes ease of programming, with a rich set of dynamic and polymorphic built-in structure data types. Unicon programmers do not have to worry about pointers and the garbage collector manages space as necessary.

Unicon modernizes the Icon language, providing numerous facilities that are ubiquitous in modern applications such as objects, classes and packages, networking, accessing file systems, execution monitoring and visualization, and a high level interface to SQL database servers. Unicon programs have a high level of portability; programmers can take a program written for UNIX platform and run it on MS Windows with almost no modifications [5, 6].

3. Design

The main design considerations for Unicon's VoIP facilities are:

- Provide an easy to use VoIP interface at a very high level of abstraction.
- Reduce the time (compared with mainstream languages) that a programmer must spend in order to learn, write and maintain VoIP applications.
- Keep the Unicon VM as small as possible.
- Minimize the complexity of the runtime system.

The new VoIP facilities were kept consistent with the high level syntax and the semantics of the rest of the Unicon language by hiding the low level details within the Unicon runtime system. Only one built-in function, `VAttrib()`, was added to the Unicon language, and it is similar to an existing Unicon graphics function WAttrib(). Two existing functions, open() and close(), were extended to support VoIP session opening and closing respectively. Extending these familiar functions and overloading them with new jobs simplifies the tasks of programmers learning and using the VoIP interface.

3.1. Adding VoIP Built-in Functions to Unicon

No programming language is complete, so most provide means for extension. Unicon supports a mechanism for accessing C functions [7]. However calling a C function from Unicon has numerous disadvantages compared with calling a function that is built in to the VM. Integrating voice support into the language allows a higher semantic level than the C calling interface, and produces distributions and executables that are more portable. Perhaps equally important, built-in functions are documented in primary language references. Adding VoIP facilities to the Unicon runtime system required an understanding of the Unicon VM source code structures and the tools used to build the VM.

3.2. Unicon's VoIP Functions

Unicon's VoIP interface is a modest extension of the file data type. The function open() with mode "v" for voice, opens a voice session at a specific port and returns a handle to that session. It takes the port number and additional optional parameters that allow the programmer to specify multiple destinations. This function fails if the sound device is reserved by another program or if it cannot open a socket for the RTP protocol.

To close a voice session, the function close(x) is passed a voice session handle. The close(x) function checks the type of the passed parameter. If close(x) is passed a voice session handle, it terminates the session and releases all resources related to that session.

In order to make a meaningful voice connection, a program adds destinations representing other users' voice sessions. The following Unicon program opens a voice session at port 4500 and establishes a voice connection with a destination at the time the session is opened.

```
# A Unicon program opens a voice session at port 4500
# and makes a voice connection to the host addressed by
# 128.123.64.48 that has an opened voice session
# at port 5000
procedure main()
   local vsession
   vsession :=open("4500","v","jef:128.123.64.48:5000")
   write("Voice session is opened, Press Enter to close:")
   read()
   close(\vsession)
end
```

Unicon VoIP sessions can add and drop destinations, and change the voice session settings on the fly using the VAttrib() function. Changing the voice session settings on the fly adds a very important feature to the Unicon's VoIP APIs. Users do not have to close the voice session and lose all the connections just to change one attribute such as the bandwidth level. VAttrib() also allows users to perform queries about those who are listening to the current voice session especially in the case of a multicasting situation or n-user conferencing. VAttrib() takes two parameters: the voice session handle and a string of attributes.

The attribute "cast+=" adds new destinations into the voice session. A destination consists of three parts separated by colons: (1) a user nickname, (2) the machine address, and (3) the base port. The "cast+=" attribute supports many destinations separated by commas as illustrated in the following example.

```
VAttrib(voice,"cast+=ziad:localhost:4500")
VAttrib(voice,"cast+=jeffery:unicon.cs.nmsu.edu:5000,_
                              erick:128.123.64.48:4500")
```

The attribute "cast-=" drops existing destinations from the opened voice session. Destinations can be identified by the user nickname, by the machine address, or by both the user nickname and the machine address. For example:

```
VAttrib(voice, "cast-=ziad,jeffery")
VAttrib(voice, "cast-=localhost:4500,_
                      unicon.cs.nmsu.edu:5000")
VAttrib(V,"cast-=ziad:localhost:4500, jeffery")
```

The following Unicon program opens a voice session at port 4500, makes a voice connection to the destination 128.123.63.40 at port 5000. The program then uses the VAttrib() function to add a new destination unicon.cs.nmsu.edu at port 4500. After that it drops the voice session with the destination 128.123.63.40.

```
procedure main()
   local voice
   voice :=open("4500","v","jef:128.123.64.40:5000")
   write("A Voice session is opened.")
   read()
   VAttrib(voice, "cast+=Zy:unicon.cs.nmsu.edu:4500")
   read()
   VAttrib(voice, "cast-=128.123.64.40:5000")
   read()
   close(\VSession)
end
```

The "bandwidth=" attribute selects an appropriate compression technique according to the available bandwidth level. Unicon defines three levels of bandwidth defined in Table 1.

“bandwidth=”	Employed compression:
high	No compression is used.
medium	DPCM
low	Silence Suppression

Table 1. The Unicon bandwidth attributes

The attribute “compression=” enables Unicon programmers to select a specific compression technique. Compression techniques can be changed during the voice session. The supported compression techniques are provided in Table 2 in section 3.4 below. The following example demonstrates changing the bandwidth level.

```
procedure main()
   local VSession, technique
   VSession :=open("4500","v","jef:128.123.64.48:5000")
   # Default bandwidth has no compression
   read();  VAttrib(VSession, “bandwidth=medium”)
   # Compression technique can be set up manually
   write(“Select the right compression:_
           PCM, Mu-law, DPCM, GSM, LPC, Suppression”)
   technique := read()
   VAttrib(VSession,“compression=” || technique )
   read();
   close(\VSession)
end
```

The VAttrib() function also provides a mechanism for performing queries about those who are listening to the voice session at any time. VAttrib() with the attribute “cast” returns a string of the current session listeners; with the attribute “castlist” it returns the information about each listener as an entry in a Unicon list; with the attribute “castnames” it returns the listeners' names; with the attribute “castaddresses” it returns the listeners addresses with their base ports. This flexibility allows the programmer to process the listener information in the shape that is suitable for their own use. The following example demonstrates some of the queries that can be performed on a voice session.

```
procedure main()
   local voice, dest1, dest2, listeners
   dest1 := "ziad:zorro.cs.nmsu.edu:4500"
   dest2 := "erick:128.123.64.225:5000"
   voice := open("5000","V",_
                 "Dr.J:unicon.cs.nmsu.edu:5000")
   write("The voice session is opened.")
   VAttrib(voice,"cast+=" || dest1 || "," || dest2)
   write("Drop Erick out of  the voice session:")
   read()
   VAttrib(voice,"cast-=erick")
   write("To get the listeners as a string:")
   read()
   listeners := VAttrib(voice, "cast")
   read()
   listeners := VAttrib(voice,"castlist")
   every write(! Listeners)
   write("To get the listeners names as a list:" )
   read()
   listeners := VAttrib(voice,"castnames")
   every write(! Listeners)
   write("To close the voice session: Press Enter:")
   read()
   close(voice)
end
```

The complete VAttrib() syntax is shown in Figure 1.

```
VAttrib(Session handle, “ attribute ” )

attribute :: =  cast
              | castlist
              | castnames
              | castaddresses
              | cast+= destinations
              | cast-= destinations
              | bandwidth= bandwidth_type
              | compression= compression_type

destinations ::= destination
               | destination,destinations

destination ::= nickname: machine :port

machine ::= hostname
          |  IP address

bandwidth_type ::=  high
                  | medium
                  | low

compression_type ::=  PCM
                    | Mu-law
                    | DPCM
                    | GSM
                    | LPC
                    | Suppression
```

Figure 1. The VAttrib() syntax

3.3. Usability of Unicon's VoIP Facilities

Unicon's voice facilities support peer-to-peer, one-to-many, and many-to-many VoIP connections. A voice session is a listening session if it is opened without any destinations added to it; an opened voice session receives voice data and plays it on the local speakers. Adding a destination to an existing voice session creates a unidirectional voice connection from the opened voice session to the added destination. Adding a destination allows the user to send voice data to that destination and it is up to the destination to figure out what was sent. In order for the destination to play the sender's voice data, the destination must have an opened voice session at some port number and the sender connects to the destination on that port number.

A simplex voice connection can be useful in some situations such as broadcasting a lecture. However, a simplex connection is not sufficient for a phone-like voice conversation; a phone connection is a full duplex connection based on a connection-oriented protocol. Full duplex provides the ability to communicate in both directions simultaneously. Unicon's VoIP facilities provide full duplex voice conversations by having each client as a destination in the other's voice session. However, this requires previous knowledge of each client machine address and their port number. Employing a server as a session manager enables those VoIP facilities to provide full duplex connection-oriented voice sessions.

3.4. Required Bandwidth

In order for VoIP to satisfy most users as an alternative to traditional phones, quality of service must be achieved. One important factor that affects the quality of service is the available bandwidth. In local area networks, the bandwidth is much more than is needed. In a wide area network, the bandwidth cannot be controlled and problems may arise because of the delays. If the delays become too large, the voice conversation is unpleasant. Reducing the required bandwidth through a compression technique addresses the problem at a cost of added latency due to the compression and decompression time [1, 2].

In order to cover most of the situations that may occur, Unicon's VoIP supports different bandwidths through a set of compression (or encoding) techniques. Compression reduces the required bandwidth. However, the bandwidth is assumed to be high and no compression is used by default at all. In this case, voice data is just raw pulse code modulation (PCM) data, and the required bandwidth is about 10KB/s (kilobyte/second) for each direction, about 20KB/s for a bidirectional voice connection. If the available bandwidth is low, the bandwidth can be reduced by using a very high rate compression technique such as linear predictive coding or silence suppression. Silence suppression produces a variable amount of traffic by omitting the voice packets that have no sound in them or have sound that cannot be heard by human ears.

The supported compression techniques at 8000 Hz sampling rate, 8-bit sampling encoding, and 20ms sampling interval are listed in Table 2. Their bandwidths are measured on Linux machines by a simple console-based live bandwidth monitor.

Compression Technique	**Bandwidth** (KB/s)
PCM (no compression)	about 10
Mu-law	about 10
DPCM	0 – 7.5
GSM	3 – 4.5
LPC	2.5 – 3.5
Silence Suppression	0 – 10

Table 2. Compression techniques and their bandwidths

4. Implementation

Unicon’s VoIP APIs are built on top of JVOIPLIB. The Unicon VM and runtime system source code is written in C, while JVOIPLIB is a C++ library. JVOIPLIB provides a sophisticated interface with many low level details that complicate VoIP applications. In order to make JVOIPLIB usable within the Unicon VM, it was necessary to determine the exact C++ functions and data structures needed, and write a set of wrapper functions that can be used by a C program. A mechanism to make the C++ classes and functions usable from the Unicon C source code was needed. The solution to this problem takes advantage of the link compatibility between C and C++ in the wrapper functions [8, 9, 10].

Unicon is supported with tools that formalize any addition of new features. One of those tools is the Run-Time implementation Language (RTL). RTL is a superset of C that is used to define operators, built-in functions, and keywords. The RTL interface defines what an operation will look like, specifies the type checking and conversion needed for arguments to the operation, and presents the overall structure of the operation's implementation. RTL also provides C extensions that assist the runtime system C code manipulate, construct, and return Unicon values [11].

4.1. The Supporting Unicon VM Data Structure

After adding the new built-in functions to Unicon’s runtime system, an additional data structure was developed to represent voice sessions within the Unicon VM. The wrapper functions that manipulate this voice session structure and call the underlying C++ code were placed in an interface library and linked with JVOIPLIB into the C code in the Unicon VM. In order to create and maintain VoIP sessions in Unicon, the data structure in Figure 2 was defined in the VM to support both Linux and MS Windows.

```
struct VOIPSession{
  JVOIPSession                    Session;
  JVOIPSessionParams              sessparms;
  JVOIPRTPTransmissionParams      rtpparms;

  #ifndef WIN32
    JVOIPSoundcardParams          sndinparam;
    JVOIPSoundcardParams          sndoutparam;
  #endif

  int                             MaxList;
  int                             InList;
  char                            **ListenerList;
};

typedef struct VOIPSession    VSESSION, *PVSESSION;
```

Figure 2. The Unicon's VM VoIP data structure

The structure in Figure 2 is used by the RTL language and the Unicon VM source code. It combines the important components for a VoIP session within the Unicon VM.

4.2. A C Language API for JVOIPLIB

As mentioned above, a set of wrapper functions provide an interface between C in the Unicon's runtime system and JVOIPLIB C++ code. The main goal of the wrapper functions is to hide the low level details of JVOIPLIB and make it accessible from a C program. They focus on creating, setting up, and destroying VoIP sessions, and establishing and dropping connections. The wrapper function prototypes are defined as follows:

```
PVSESSION CreateVS(char Port [5] ,char Dests[]);
```

This function is responsible for allocating and creating an instance object of the VSESSION struct in the heap, and establishing a voice session on the specified port. Its parameters are the base port followed by a string of destinations. The destinations are parsed internally and saved in the listeners list.

```
void SetVAttribs( PVSESSION VSession , char attribs[]);
```

This function is responsible for changing the voice session settings such as changing the compression technique. It takes the VoIP session handle as the first parameter, followed by an attribute.

```
int Cast( PVSESSION VSession ,char Destinations[]);
```

This function is used to cast to any number of destinations.

```
int DropCast ( PVSESSION VSession ,char Dests[]);
```

This function stops casting to one or more of the existing destinations.

```
void CloseVoiceSession( PVSESSION VSession );
```

This function is responsible for closing the voice session and returning the allocated memory for the VSESSION struct back to the heap.

5. Related Work

Unicon's VoIP facilities are a simplified subset of voice API's used in other languages. The simplicity and convenience gives Unicon an advantage over other languages in VoIP prototyping since programmers do not have to spend much time and effort in order to learn or use them. This section evaluates Unicon against the C++ VoIP API on which it is built, as well as a mainstream Java VoIP API.

5.1. JVOIPLIB

The JVOIPLIB library directly depends on two other libraries JThread and JRTPLIB, which are a thread library and a Real Time Protocol (RTP) library respectively. The RTP protocol is based on RFC 3550. These three libraries can be used in a C++ program that declares a session object, a session parameters object, and an RTP object. Through the C++ class based APIs of those objects the voice session can be established and managed.

In contrast, a Unicon programmer does not have to know anything about the several underlying classes in the JVOIPLIB, JRTPLIB, and JThread libraries, how they operate with each other, or what methods need to be called for a successful voice session. The following very simple C++ example was taken from the JVOIPLIB distribution [1, 2]; it opens a voice session with one destination, which is the local host. This program uses JVOIPLIB and JRTPLIB directly and JThread implicitly.

```
#include "jvoipsession.h"
#include "jvoiprtptransmission.h"
#include <stdio.h>
#include <arpa/inet.h>
#include <netinet/in.h>

int main(void)
{
    JVOIPSession session;
    JVOIPSessionParams params;
    JVOIPRTPTransmissionParams rtpparams;
    unsigned long ip;

    ip = inet_addr("127.0.0.1");
    ip = ntohl(ip);

    rtpparams.SetAcceptOwnPackets(true);
    params.SetTransmissionParams(&rtpparams);

    session.Create(params);
    session.AddDestination(ip,5000);
    fgetc(stdin);

    //IMPORTANT: The Destroy method MUST be called.
    session.Destroy();
    return 0;
}
```

The following is an equivalent Unicon program that does the same job as the preceding C++ program.

```
procedure main()
    local voice
    voice := open(“5000”, “v”, “myself:127.0.0.1:5000”)
    read()
    close(voice)
end
```

5.2 Java SIP API's

The Java programming language has a package called *javax.microedition.sip*. This package implements the Session Initiation Protocol (SIP) which is based on RFC 2543 and RFC 3261. SIP is defined to be a request-response protocol just like the HTTP protocol. SIP is mainly used in establishing and controlling multimedia communication sessions on the IP networks. A multimedia session can be very simple as in bidirectional voice communication or very complex as in n-user collaborative video conferencing. The SIP Java package has eight interfaces and four classes, making it comparable or slightly more complex than JVOIPLIB to learn and use. The SIP APIs for J2ME are designed mainly to be used by applications that need to implement a SIP user-agent client and a SIP user-agent server. A mobile device may act as a SIP client sending requests to SIP servers, and also as a SIP server accepting requests from SIP clients. Reference [12] provides some examples explaining the use of SIP and the SIP MIDlet. J2ME does not come with the standard Java distribution and it is mainly for small devices with limited memory.

6. Conclusions and Future Work

Unicon's VoIP interface is a minimal addition that is consistent with the rest of the language. These considerations keep the VM small and reduce the time a programmer must spend in order to learn how to write VoIP applications. Unicon was extended to directly support VoIP specifically in order to support this feature within collaborative virtual environments.

Unicon's VoIP support is demonstrably simpler than comparable facilities in mainstream languages. It works by means of three flexible built-in functions that provide very high level voice communication semantics alongside other very high level I/O facilities. The Unicon VoIP facilities are peer-to-peer and have no need for a running server, even though a server can be utilized for management in the case of n-user sessions. Unicon provides VoIP with adequate quality of service. In the worst case, the required bandwidth is about 10KB/s for each connection; this worst case occurs when no compression is employed. Supporting session recording in a compressed audio file format such as Ogg Vorbis or MP3 is one of the most important tasks for future work.

Acknowledgments

This work was supported in part by NSF award DUE-0402572.

References

[1] Jori Liesenborgs, Wim Lammotte, and Frank Van Reeth, "Voice over IP with JVOIPLIB and JRTPLIB", IEEE 26th Annual Conference on Local Computer Networks, 2001, pp.346-347.

[2] Jori Liesenborgs, Voice over IP in networked virtual environments, PhD thesis, 2000. http://lumumba.luc.ac.be/jori/thesis/thesis.html.

[3] The Unicon programming language. http://unicon.org

[4] The Icon programming language. http://www.cs.arizona.edu/icon

[5] Ralph E. Griswold, Gregg M. Townsend, and Clinton L. Jeffery, Graphics Programming in Icon, Peer-to-Peer Communication Inc., 1998.

[6] Jeffery, Mohamed, Parlett, and Pereda, Programming with Unicon. September 2004. http://unicon.sourceforge.net/book/ub.pdf.

[7] Gregg M. Townsend and Ralph E. Griswold, Calling C functions from version 9 of Icon, Technical report, March 1996. http://www.cs.arizona.edu/icon/docs/ipd240.htm.

[8] Stephen Clamage, Mixing C and C++ code in the same program, Technical report, July 2003. http://developers.sun.com/prodtech/cc/compilers index.html

[9] Giri Mandalika, Mixed-language programming and external linkage, Technical report, March 2005. http://developers.sun.com/prodtech/cc/compilers index.html.

[10] Using C++ code in a C program. http://gcc.gnu.org/ml/gcc-help/2004-07/msg00107.html

[11] Kenneth Walker, The runtime implementation language for Icon, Technical report, Department of Computer Science, The University of Arizona, June 28 1994. http://www.cs.arizona.edu/icon/ftp/doc/ipd261.pdf.

[12] Qusay H. Mahmoud, Getting Started with SIP API for J2ME (JSR 180), November 2004. http://developers.sun.com/techtopics/mobility/apis/articles/sip/