Software Engineering Sequence
SE1/E&CE451/CS445/CS645/SE463
SE2/E&CE452/CS446/CS646/SE464
SE3/E&CE453/CS447/CS647/SE465
Hardware Interface Description

David R. Cheriton School of Computer Science and
Department of Electrical and Computer Engineering
University of Waterloo, Waterloo, ON, Canada, N2L 3G1

Last revision: September 5, 2006

1 Introduction

This document describes the hardware interface provided to you. This interface is sometimes referred to as an Application Program Interface (API).

2 The Interface

Conceptually, the Phone Subsystem is embedded code that the runs on the phone's hardware. In reality, your Phone Subsystem consists of a UNIX process running on `commando.math` that uses an API to monitor and control a Nortel i2004 IPPhone.

For those using C++, a complete implementation of the interface can be found in `/u/cs446/VoIP/public/interface`.
It consists of a static library, `libcs446VoIP.a`, and the header files (in the 'facade' directory):

`PhoneInterface.h`: Main interface class. Provides methods to allocate, monitor and control a phone, such as `ReceivePhoneEvent()`, `StartRinging()`, and `HandsetOn()`.

`Event.h`: Phone events, off-hook, key presses, and so on. returned by the `PhoneInterface` are represented by the Event class.

`RingTone.h`: Contains enumerations of the values for ring tones and cadences.

`Digits.h`: An enumeration of the values for the buttons on the phone.

Some simple examples of this API can be found in `/u/cs446/VoIP/public/examples`.

Originally, only C++ was to be supported, but for other languages to be used, the underlying XML interface is being exposed. If you wish to use a language other that C++, you will need to read appendices B and C.

3 Additional Information

If you are having trouble using the interface, you can use the "state of the world" tool found in `/u/cs446/VoIP/public/SOTWGui`.
This tool displays all the phones and student connections that the Interface Server knows about, as well as the states that they are in.

If a phone stops responding, just pull out the power cord from the phone, wait a moment, and plug it back in. If that does not help, there may be a problem with the Nortel equipment or our interface to it. This equipment is fairly resilient. If a critical failure occurs, the hardware should fix itself within half an hour. If it doesn't, contact a TA or your professor and they'll fix the problem as soon as possible.

Finally, if you would like to use a feature of the IPPhones that is not currently available through the given interface, tell a TA what you would like to do. There is a small chance that someone can extend the API to suit your needs.

4 Appendix

Appendix A discusses the Nortel hardware and our interface to it.

If you plan to use a languages other than C++ and Tcl/Tk, then you must read appendix B. You will also need to read appendix C on the XML API.

A Under the Hood

In their typical configuration, the IPPhones connect to a device called a Signalling Server (SS). Communication between the SS and the phones is done via sockets and a proprietary protocol. The Signalling Server also connects to a device called a Call Manager (CM). It is the Signalling Server's responsibility to translate from the IPPhone/SS protocol to the SS/CM protocol, and vice versa. Also the SS/CM protocol is proprietary. It is the Call Manager's responsibility to monitor and control the phones.

In our system, you will be implementing a distributed version of the Call Manager. However, to avoid the hassles of learning the proprietary SS/CM protocol, we add an additional step between the phones and the Call Manager. This additional step is embodied in the Interface Server, which translates from our own XML-based protocol to the SS/CM protocol and vice versa.

The Interface Server is a process running on `commando.math`. You can access the Interface Server directly using a socket or through the provided C++ API.

B Programming Languages

In the past, the front-end was written in Tcl/Tk and the back-end was done in C or C++. However, you have the option to use any language you like. Different languages provide different benefits, be it IDEs,

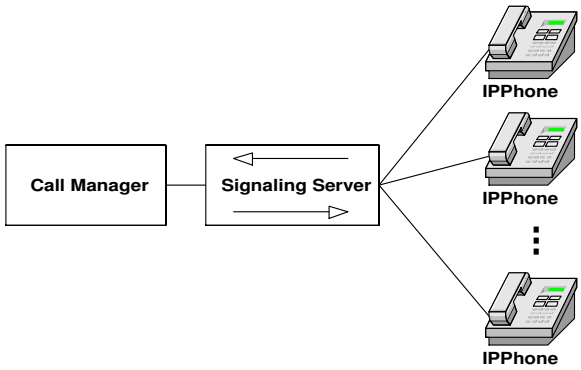

Figure 1: Typical Nortel Hardware Configuration

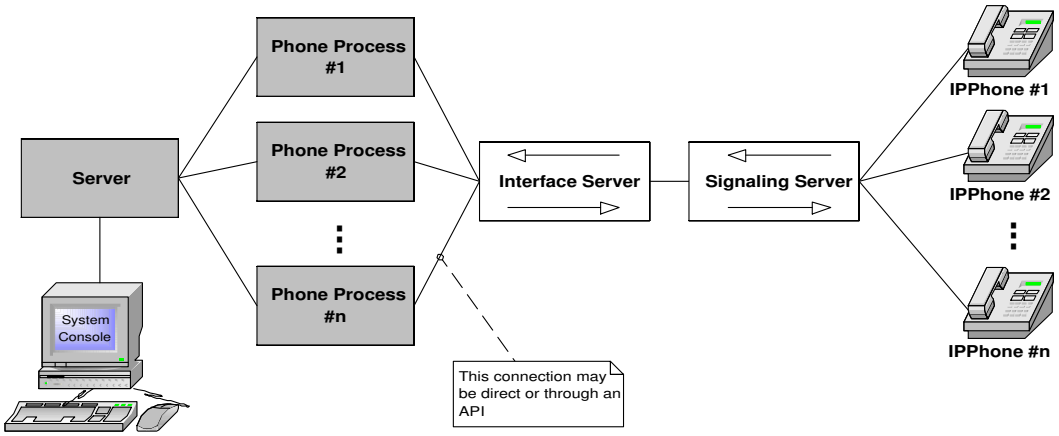

Figure 2: Our Hardware Configuration. The shaded areas correspond to components you will write.

debuggers, or available libraries. If you use a language other than C or C++, make sure it has sockets and multi-threading. It must also be able to interface with your GUI.

Before committing to a language, other than C++ or Tcl/Tk, discuss it with your professor. He or she may inform you of some additional constraints on languages. For example, some of the tools used for automated testing in SE3 require that Tcl/Tk be used for your GUI. You should determine if similar or adequate tools are available for your language. The TAs may be able to help with this choice; otherwise you may need to do an excessive amount of work in SE3.

C XML API

Beneath the C++ interface is a socket-based interface that communicates with a server using XML. The socket protocol is TCP, and the IP address and port of this server can be found in `PhoneInterface.h`. Once connected to the server, you may send or receive messages at any time. So, you'll likely need to spawn a thread to listen at all times. Finally, to disconnect from the server, simply close the socket.

If you choose to use this XML–socket-level interface, it would be wise to encapsulate it in a class.

The following subsections describe the XML messages used at this level of the interface.

C.1 Connecting

C.1.1 AcquireResource

```
<AcquireResource>
   <Resource>phoneIP</Resource>
   <Name>phoneName</Name>
</AcquireResource>
```

To connect to the interface server you need to use this message. This message must be sent before any other communication with this phone. You cannot use any of the other messages until this message is successfully sent. `phoneIP` is the IP address of the phone you wish to use. `phoneName` is the name you may give the phone; this `phoneName` shows up in the "state of the world" application. Possible responses are:

- ResourceAcquired
- Error

C.1.2 ResourceAcquired

```
<ResourceAcquired>
   <Resource>phoneIP</Resource>
</ResourceAcquired>
```

Request was a success and you can start using the phone at the IP address `phoneIP`, which is the same as the requested `phoneIP`.

C.1.3 Error

```
<Error>
   <ErrorDescription>description</ErrorDescription>
</Error>
```

Couldn't get phone for the reason given in `description`.

C.2 Requests

C.2.1 HandsetOn

```
<HandsetOn/>
```

The speaker and microphone on the handset needs to be explicitly turned on. Without the handset, you can't hear tones being played or a connected audio path.

C.2.2 HandsetOff

```
<HandsetOff/>
```

Turns off handset speaker and microphone.

C.2.3 LampOn

```
<LampOn/>
```

Turn on lamp on top of phone.

C.2.4 LampOff

```
<LampOff/>
```

Turn off lamp on top of phone.

C.2.5 StartRinging

```
<StartRinging/>
```

Start phone ringing. Phone will ring on and off continuously. There is no need to tell the phone to ring every second; the phone does it for you.

C.2.6 StopRinging

```
<StopRinging/>
```

Stop the phone from ringing.

C.2.7 PlayTone

```
<PlayTone>
   <Tone>tone</Tone>
   <Cadence>cadence</Cadence>
</PlayTone>
```

Play the tone `tone` with cadence `cadence`. `tone` and `cadence` are integral values between `0` and `255`. A tone value of `255` stops playing the current tone. You are welcome to experiment with different values for the tones and cadences. Some of the most useful values can be found in the C++ file `RingTone.h`.

There are some subtleties to playing tones on the phones. They don't like being told to keep playing different tones without first stopping the tone currently being played. If you have not explicitly stopped the current tone by sending `255`, the current tone continues to play. This holds even for a silent tone, such as the tone `0`. Of course, the stop-playing-current-tone value `255` cannot be turned off. If you want silence, you should explicitly turn off the tone being played. A phone may stop responding entirely and need to be reset, if it is constantly told to play different tones

C.2.8 DisplayString

```
<DisplayString>
   <String>lineStr</String>
   <lineNum>lineNum</lineNum>
   <linePos>linePos</linePos>
</DisplayString>
```

Display string `lineStr` on line `lineNum` starting at position `linePos`. The command places the display cursor on the given line at the given position and display the given string. The line number, `lineNum`, may be a value between `0` and `2`, and the line position, `linePos`, may be a value between `0` and `25`. A string that goes past the end of the current line with wrap to the next line.

To clear a line simply send spaces as the string.

Unfortunately there is no way to position the cursor without also writing a character.

C.2.9 AppendString

```
<AppendString>
   <String>lineStr</String>
</AppendString>
```

Append string `lineStr` at the current cursor position.

C.2.10 StartAudioSend

```
<StartAudioSend>
   <DestDevice>phoneIP</DestDevice>
</StartAudioSend>
```

Start sending audio from the current phone to the phone at the IP address `phoneIP`. To make a voice call, both phones must start sending to each other and receiving from each other. The phone handset must be turned on for this to be useful.

C.2.11 StartAudioReceive

```
<StartAudioReceive>
   <DestDevice>phoneIP</DestDevice>
</StartAudioRecieve>
```

Allow receiving audio on the current phone from the phone at the IP address `phoneIP`. To make a voice call, both phones must start sending to each other and receiving from each other. The phone handset must be turned on for this to be useful.

C.2.12 StopAudioSend

```
<StopAudioSend>
   <DestDevice>phoneIP</DestDevice>
</StopAudioSend>
```

Stop sending audio from the current phone to the phone at the IP address `phoneIP`.

C.2.13 StopAudioReceive

```
<StopAudioReceive>
   <DestDevice>phoneIP</DestDevice>
</StopAudioReceive>
```

Stop sending receiving audio on the current phone from the phone at the IP address `phoneIP`.

C.3 Received Events

C.3.1 OnHook

```
<OnHook/>
```

Phone was placed off-hook.

C.3.2 OffHook

```
<OffHook/>
```

Phone was placed on-hook.

C.3.3 DigitPressed

```
<DigitPressed>
   <Value>digit</Value>
</DigitPressed>
```

The button `digit` was pressed. Values of the different buttons can be found in the C++ file `Digits.h`.

C.3.4 DigitReleased

```
<DigitReleased>
   <Value>digit</Value>
</DigitReleased>
```

The button `digit` was released. Values of the different buttons can be found in the C++ file `Digits.h`.

C.4 Testing Messages

C.4.1 TestOnHook

```
<TestOnHook/>
```

Tells the interface to immediately send back a `<OnHook/>` message if the phone is on hook; otherwise there is no reply.

C.4.2 TestOffHook

```
<TestOffHook/>
```

Tells the interface to immediately send back a `<OffHook/>` message if the phone is off hook; otherwise there is no reply.

C.4.3 TestDigitPressed

```
<TestDigitPressed>
   <Digit>digit</Digit>
</TestDigitPressed>
```

Tells the interface to immediately send back a `<DigitPressed>` message. The button `digit` appears to be pressed. Values of the different buttons can be found in the C++ file `Digits.h`.

C.4.4 TestDigitReleased

```
<TestDigitReleased>
   <Digit>digit</Digit>
</TestDigitReleased>
```

Tells the interface to immediately send back a `<DigitReleased>` message. The button `digit` appears to be released. Values of the different buttons can be found in the C++ file `Digits.h`.