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

School of Computer Science and Department of Electrical and Computer Engineering
University of Waterloo, Waterloo, ON, Canada, N2L 3G1
Last revision: January 14th 2004

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 runs on the phone's hardware. For the purposes of the course project, your Phone Subsystem consists 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 (this API) 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. For example, `ReceivePhoneEvent()`, `StartRinging()`, `HandsetOn()`, etc.
`Event.h`	Phone events, off-hook, key presses, etc. 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 has been exposed. If you wish to use a language other than C++ you will need to read the Programming Languages and XML API sections on pages 6 and 6, respectively. Also, an unsupported Java interface may be found in the project directory. This Java interface was written by a team from a previous term and is provided as-is, with no guarantee about its efficiency or correctness[1].

1Although to be fair, it does seem to work well.

3 Additional Information

To assist in debugging your system, 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 state they are in. When a phone stops responding, the most likely cause is that a student’s process failed to release the phone. This failure can happen if a process or thread crashes and the crashed process or thread is not killed, or if a process is simply left running. It’s important to remember that the phones are shared, and if you’re going to leave your terminal for a while, then remember to release any phones you’re using.

If a phone stop responding, just pull the power cord out from the phone, wait a moment, and plug it back in. Alternatively, most phones will have a button on the power cord which will cut the power to the phone when pressed. If the phone still does not respond, then 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 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.

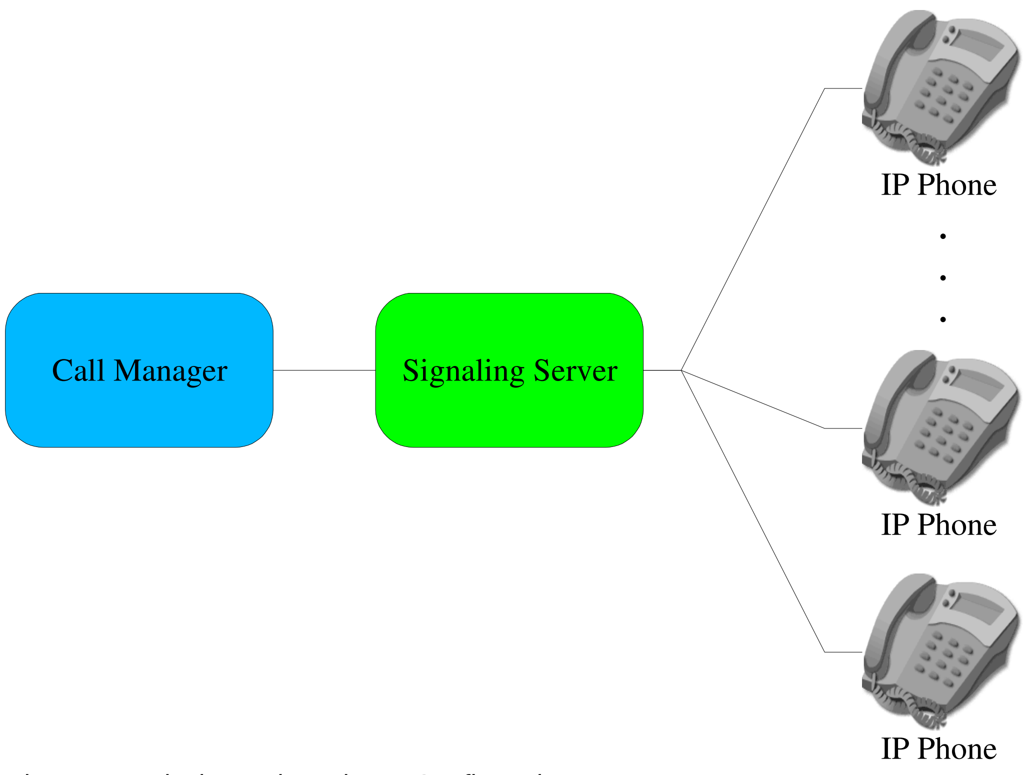

Figure 1: Typical Nortel Hardware Configuration

In your 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.

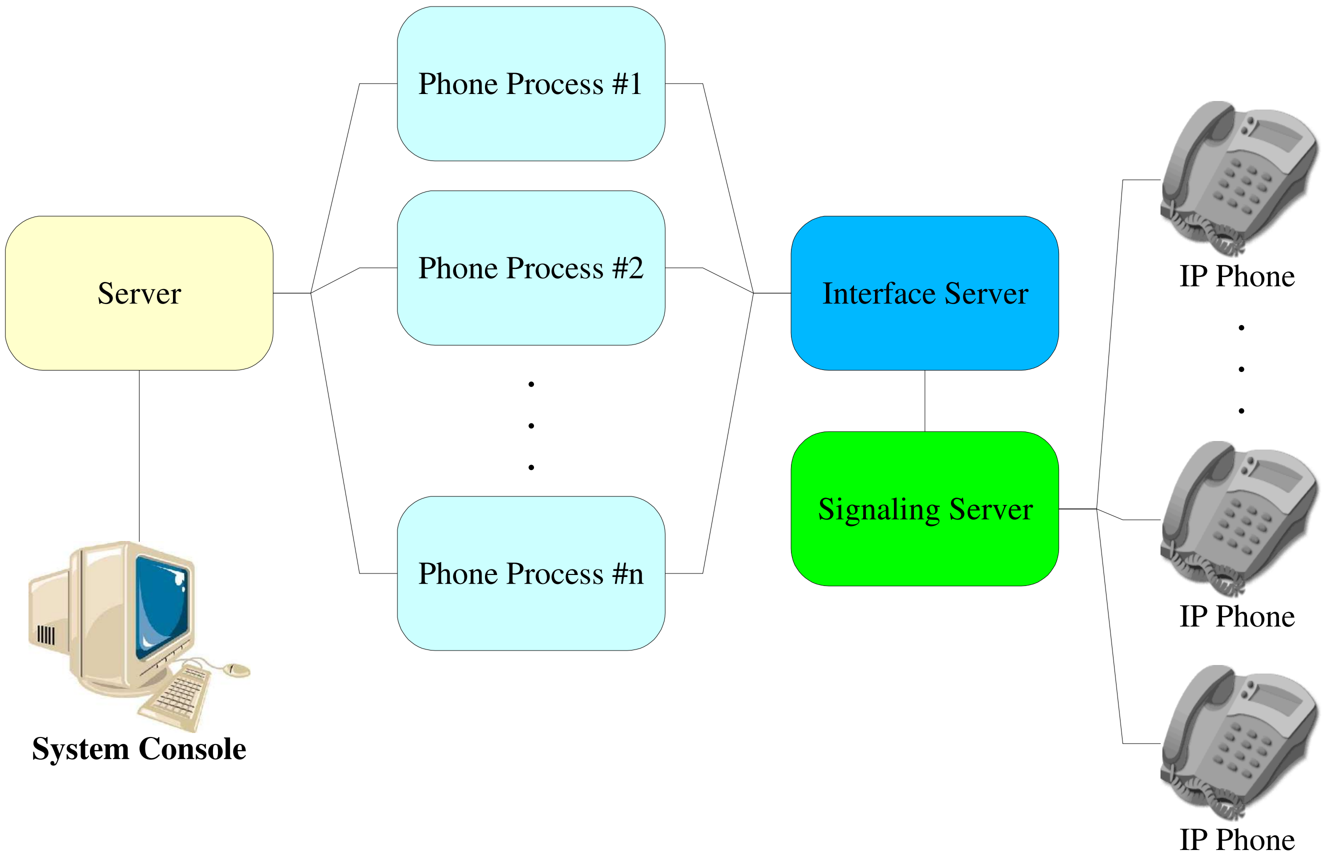

Figure 2: Our Hardware Configuration. The Server(s) and Phone Processes are implemented by each team.

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.

The system should be viewed as described in the *Overview of the Course Project* document. Conceptually, a phone process behaves as if it were running on an actual phone. Since no new code can run on a phone, the levels of abstraction described above fake it.

Each phone should be viewed as having an operating system that runs a phone process. The phone process may have multiple threads. However, since a phone process conceptually runs on a phone, it can communicate with other processes, be they phone processes or other servers, only using sockets. Therefore a phone process *must* be a separate operating system process[2] and not just one of many threads running in a single process. Your architecture and design documents should adhere to this view.

2 To clarify, by “process” we mean a separate executable like a shell command, as opposed to a thread, which are threads of control within a process (recall your concurrency class). A process runs in a separate memory space. A thread runs in shared memory space along with other threads of the same process, enabling all the threads of a process to share variables. For our purposes of this project, light-weight and heavy-weight threads are indistinguishable.

The diagrams of Figures 1 and 2 illustrate this conceptual view. Your documents are not to talk about the interface server or signalling server. For the most part, you may pretend that these servers do not exist. A number of decisions were made by the course planners to keep the project realistic but without too much tedious realism. The circuitous way in which a phone process connects to a phones is a result of these decisions and other constraints.

5 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, debuggers or available libraries. If you use a language other that C/C++ or Java, 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/C++, Java, Tcl/Tk, discuss it with your professor or TA. He or she may inform you of some additional constraints on the chosen language(s). The TAs may be able to help with this choice; otherwise you may need to do an excessive amount of work in the ECE453/CS447. We strongly recommend looking over the ECE453/CS447 assignment specifications before choosing an alternate language to C/C++, Java, or Tcl/Tk.

It's worth studying the XML API section, even if you choose to use one of the provided APIs.

6 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 to 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.

6.1 Connecting

6.1.1 AcquireResource

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

To connect to the interface server you need to use this message. 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

6.1.2 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`.

6.1.3 Error

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

Couldn't get phone for the reason given in `description`. It's almost always going to be the case that if acquiring a resource fails, it's due to it currently being in use by another process. If you're having trouble, it's suggested that you use the "State of the World" tool.

6.2 Requests

These messages are the requests sent by your phone processes to the interface process. They're the ones which affect the phone.

6.2.1 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.

6.2.2 HandsetOff

```
<HandsetOff/>
```

Turns off handset speaker and microphone.

6.2.3 LampOn

```
<LampOn/>
```

Turn on lamp on top of phone.

6.2.4 LampOff

```
<LampOff/>
```

Turn off lamp on top of phone.

6.2.5 StartRinging

```
<StartRinging/>
```

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

6.2.6 StopRinging

```
<StopRinging/>
```

Stop the phone from ringing.

6.2.7 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

6.2.8 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.

6.2.9 AppendString

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

Append string `lineStr` at the current cursor position.

6.2.10 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.

6.2.11 StartAudioReceive

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

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.

6.2.12 StopAudioSend

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

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

6.2.13 StopAudioReceive

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

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

6.3 Received Events

These messages are sent by the interface process to your phone processes. Using the C++ interface, they're represented as Event objects.

6.3.1 OnHook

```
<OnHook/>
```

Phone was placed on-hook.

6.3.2 OffHook

```
<OffHook/>
```

Phone was placed off-hook.

6.3.3 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`.

6.3.4 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`.

6.4 Testing Messages

The following test messages are meant for unit testing. You can use them to write automated scripts that simulate phone events. What they are not used for is the "Automatic Hardware Fault Detection", as described in the overview document. They have absolutely nothing to do with fault detection, and can't be used for it without invariably introducing errors and/or convoluted and complicated code into your phone processing logic.

Generally speaking, if you're not writing unit tests, don't use these messages.

6.4.1 TestOnHook

```
<TestOnHook/>
```

Tells the interface to immediately send back a `<OnHook/>` message.

6.4.2 TestOffHook

```
<TestOffHook/>
```

Tells the interface to immediately send back a `<OffHook/>` message.

6.4.3 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`.

6.4.4 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`.

7 Frequently Asked Questions

Q1) I get a message like "`fatal: libccgnu2-1.0.so.0: open failed: No such file or directory`".

A1) Your phone process code links against at least three shared libraries. When executing it, the system needs to know where to find them. The easiest solution is to append to your LD_LIBRARY_PATH shell variable the location of these libraries, currently found in: `/u/cs446/VoIP/public/libs/lib`

Q2) The phones aren't working. I can't get any work done. Can I have an extension?

A2) No. Even if the phones aren't working it's still possible to continue working using a fake interface found in `/u/cs446/VoIP/public/FakeInterfaces/`. These are simple java applications which will simulate a phone connected to the interface server. Instead of connecting to the interface server, have your phone process connect to a running fake interface. Once it connects, a simple GUI with buttons representing a phone will appear. Your code won't be able to tell it's using the fake interface, allowing you to develop even when the phones aren't working.

It's also encouraged to try develop at home using the fake interface. You'll still need the physical phones to test the proper voice paths are being created and tones set correctly and so forth, but 90% of your development could in theory be done away from the lab.

If the fake interface doesn't quite do what you need it to do, you're encouraged to modify it as you like (the source is provided). Alternately you could try writing your own version, and if you like donate it to the course.

Q3) The commando server is really, really slow. Can anything be done?

A3) Unfortunately commando does tend to slow down with a lot of people using it. To help alieviate the problem try running webbrowsers, Eclipse, etc on one of the CPU servers (do a "`hostselect cpu`" for a randomly selected one). Likewise you should be able to run most of your code, apart from the phone processes, on any server.

Q4) Can we use product X?

A4) Generally yes. You're free to install any technologies you want to use, including webservers, compilers, databases, etc. Everything must work in the lab environment for your demo, so don't assume you can run something at home. You won't be given any extra disk quota so you'll need to figure out amongst your group members how to distribute your project and its supporting tools.

Q5) Phone X is broken/doesn’t work!

A5) Let us know and we’ll try to get MFCF to do something about it. We can’t fix any problems we don’t know about.

Q6) The Unix shell responded with an error message I haven’t seen before. What does it mean?

A6) Go to www.google.com (or whatever your favourite search engine is) and do a search for the error. Chances are Google will give you a faster answer than the TAs. If Google can’t help you, then ask a TA.