Introduction

1.1 OVERVIEW

omniPresence is a software that is meant to aid in the herculean task of automating the industry. omniPresence has been used as a means to make attendance in organizations in an automated, organized and protected manner.
Let us assume a hypothetical situation to justify the usability of omniPresence.

Mr. X works in PQR Constructions. His payment is done on the basis of hours of work done. His reporting time is 8:00 am. Let us suppose, intentionally or otherwise, he reaches office at 9:00 am one day. He has quite a number of ways to get his attendance marked if the system is manual. In this case, the company bears a loss of a whole hour. The same is true for all the employees of the firm. The loss incurred is also proportional to time delay. Try as hard as he might, the owner of the firm will not be able to enforce totally protected and secure measures. The immediate, fool-proof solution that the owner of the firm can use here is some kind of tracking device for his employees so that their movements in the workplace can be monitored and actions can be taken accordingly.

The technology available for this situation comes to mind automatically—Radio Frequency Identification (RF ID). However, RF IDs involve huge hardware costs. omniPresence is a low cost alternative to RF IDs and can be used successfully as a miniature tracking device and thus could be an answered prayer of the owner of PQR Constructions mentioned above.

omniPresence uses basic computing hardware and general software to create a powerful combination to detect devices on a network. Detection is done via Bluetooth technology. As soon as the Bluetooth enabled devices are detected, attendance is marked. omniPresence gives tremendous attention to security. Just to ensure that the system is protected always, without any intrusion, security is enforced by mechanism.

Thus, we see that this system can be used for automated detection in organizations and if implemented on a large scale, is omnipresent. Hence, the name. Now let us delve into the technologies that have been used in omniPresence.

1.2 BLUETOOTH

omniPresence works on Bluetooth. Bluetooth is this system's working backbone. Bluetooth is a wireless protocol for exchanging data over short distances from fixed and mobile devices, creating personal area networks (PANs). It was developed by Ericsson for short range communication. It was originally conceived as a wireless alternative to RS232 data cables. It can connect several devices, overcoming problems of synchronization.

Bluetooth uses a radio technology called frequency-hopping spread spectrum, which chops up the data being sent and transmits chunks of it on up to 79 frequencies. In its basic mode, the modulation is Gaussian frequency-shift keying (GFSK). It can achieve a gross data rate of 1 Mbps.

Bluetooth provides a way to connect and exchange information between devices such as mobile phones, telephones, laptops, personal computers, printers, Global Positioning System (GPS) receivers, digital cameras, and video game consoles through a secure, globally unlicensed Industrial, Scientific, and Medical (ISM) 2.4 GHz short-range radio frequency bandwidth.

The Bluetooth specifications are developed and licensed by the Bluetooth Special Interest Group (SIG). The Bluetooth SIG consists of companies in the areas of telecommunication, computing, networking, and consumer electronics. Usually, it has ranges between 1 meter, 10 meters and 100 meters based on low cost transceiver microchips in each device. The use of radio frequency implies that the communicating devices need not be in line of sight.

Bluetooth devices are categorized as under:

Class	Max. Permitted Power	Range
Class 1	100 mW (20 dBm)	~100 meters
Class 2	2.5 mW (4 dBm)	~10 meters
Class 3	1 mW (0 dBm)	~1 meters

Here, we encounter a new unit, dBm. A dBm is a standard unit for measuring levels of power in relation to a 1 milliwatt reference signal.

dBm is similar to dB, or decibel, except that where dB is relative to the power of the input signal, dBm always relates to a 1 milliwatt signal.

In other words, dB is a relative measurement and dBm is an absolute measurement.

A minus sign before the dBm indicates a loss and a plus sign or no sign at all before the dBm indicates a gain.

1.2.1 THE dBm FORMULA

The dBm value of a signal can be expressed utilizing the following algorithm:

$$\boxed{\text{db} = \log_{10} \text{Signal Power} / .001 \text{ watt}}$$

Talking about the different classes of Bluetooth devices, in most cases the effective range of class 2 devices is extended if they connect to a class 1 transceiver, compared to a pure class 2 network. This is accomplished by the higher sensitivity and transmission power of Class 1 devices.

1.2.2 BLUETOOTH MAC ADDRESS

This is a 48 bit (6 bytes) identifier of a Bluetooth device. The core first three bytes of the address are assigned to the manufacturer by the standard of IEEE and the last three bytes are freely allocated by the manufacturer.

For instance, a Sony Ericsson phone may have the Bluetooth MAC address as 00:OA:D9:EB:66:C7. Here the first three bytes are 00:OA:D9 which are assigned by the IEEE and the last three bytes are EB:66:C7 which are assigned by manufacturer randomly.

1.2.3 WHAT HAPPENS DURING BLUETOOTH COMMUNICATION ?

Bluetooth communication is based on enquiries generated by Bluetooth enabled devices. During enquiry, devices generate an enquiry hopping (channel changing) sequence. This enquiry hopping sequence is derived from the local device's clock and chosen enquiry access code. This hopping sequence covers 32 channels subset of the available 79 Bluetooth channels. Once a device generates an enquiry hopping sequence, it broadcasts enquiry messages as sequential switches to each channel defined in the hopping sequence.

Discoverable devices will periodically enter the enquiry scan sub-state. In this sub-state devices hop according to the enquiry scan hopping sequence which is also based on the enquiry access code and the local clock.

If the device performing the enquiry scan receives the enquiry message, it enters the enquiry response sub-state and replies with an enquiry response message. The enquiry response includes the remote device address and clock, both of which are needed to establish a Bluetooth connection.

After obtaining and selecting a remote device's Bluetooth address, the local device enters the paging sub-state to establish a connection with the remote device. In the paging sub-state, the local device generates a hopping sequence based on the remote device's address and estimated current clock. The paging device then repeatedly sends page messages as it hops through the generated sequence of channels. If a device allows other devices to connect to it, it will periodically enter the page scan sub-state. A hopping sequence is generated based on the local address and clock. When the remote slave receives a page packet, it responds to the local master device with a page response packet.

Upon receiving the response packet, the master sends a FHS (Frequency Hopping Sync) packet to the slave. The FHS packet includes the master's Bluetooth address and clock. Once the slave receives the FHS, it sends an acknowledgement to the master.

When the master receives the acknowledgement, it generates a new hopping sequence from its own address and its own clock. The slave then uses the master's address and clock to generate its own hopping sequence which is identical to the one generated by the master. The identical hopping sequences allow the devices to hop on common channels while being connected.

Once this process (paging process) is complete, the devices move to the connection state. The master sends a poll packet to the slave verifying that the transition from the page hopping sequence to the new hopping sequence is successful. If all the conditions are satisfied, i.e., if it is successful then the devices would continue with FHS in a pseudo-random order based on the master device's address and clock.

1.3 BLUETOOTH STACK

A Bluetooth stack refers to an implementation of the Bluetooth protocol stack.

Bluetooth stacks can be roughly divided into two:

1. General-purpose implementations that are written with emphasis on feature-richness and flexibility, usually for desktop computers. Support for additional Bluetooth profiles can typically be added through drivers.
2. Embedded system implementations intended for use in devices where resources are limited and demands are lower, such as Bluetooth peripheral devices.

Generally, only a single stack can be used at any time: switching usually requires uninstalling the current stack, although a trace of previous stacks remains in the registry. However, there are some cases where two stacks can be used on the same computer, each using their own separate Bluetooth radio hardware.

1.3.1 GENERAL – PURPOSE IMPLEMENTATIONS

1.3.1.1 WINDOWS

The following are majorly used bluetooth stacks on the Microsoft Windows Platform. All of the bluetooth stacks are propreitory and one has to acquire license in order to use it or program it. Each stack here is resposible for handling a unique device or radio hardware.

- **Widcomm**

Widcomm was the first Bluetooth stack for the Windows operating system. The stack was initially developed by a company named Widcomm Inc., which was acquired by Broadcom Corporation in April 2004. Broadcom continues to license the stack for inclusion with many Bluetooth-powered end-user devices.

An API is available for interacting with the stack from a custom application. For developers there is also a utility named BTServer Spy Lite bundled with the stack (some vendor-tied versions excluded) which monitors Bluetooth activity on the stack at a very low level - although the category and level of trace is configurable. This stack also allows use of RFCOMM without creating a virtual serial port in the operating system.

- **Microsoft Windows Stack**

The Microsoft Windows Bluetooth stack only supports external or integrated Bluetooth dongles attached through USB. It does not support Bluetooth radio connections over PCI, I²C, serial, PC Card or other interfaces.

Windows XP includes a built-in Bluetooth stack starting with the Service Pack 2 update, released on 6th August 2004.

Prior to this, Microsoft released a QFE of its Bluetooth stack for Windows XP Service Pack 1 labeled as QFE323183. Microsoft only released this directly to third-party companies and did not directly release it to the public. The third-party companies were then allowed to release the QFE as part of their own Bluetooth device's software installation. Microsoft no longer supports this QFE.

Windows Vista also includes a built-in Bluetooth stack which is an expansion over the Windows XP Bluetooth stack. In addition to supporting more Bluetooth profiles than Windows XP Service Pack 2, it also supports third-party driver development which enables third-parties to add support for additional Bluetooth Profiles. This was lacking in the Windows XP Service Pack 2 built-in Bluetooth stack, which only allowed application development on top of the Microsoft Bluetooth stack, which some observers felt slowed the adoption of the Microsoft Windows Bluetooth stack. This stack does however provide RFCOMM support using sockets instead of virtual serial ports.

The Windows XP and Windows Vista Bluetooth stack supports the following Bluetooth profiles natively: PAN, SPP, DUN, HID, HCRP Windows 7 also supports audio related profiles-HFP, HSP, A2DP and AVRCP natively.

Microsoft has not released an official Bluetooth stack for older Windows versions, such as Windows 2000 or Windows Me.

- **EtherMind Stack**

EtherMind is a Bluetooth protocol stack from MindTree's for embedded devices and host platforms.) A non-disclosure agreement is required to obtain the API documentation.

- Toshiba Stack

Toshiba has created its own Bluetooth stack for use on Microsoft Windows. Toshiba licenses their stack to other original equipment manufacturers (OEM) and has shipped with some Fujitsu Siemens, ASUS, Dell and Sony laptops. A non-disclosure agreement must be signed to obtain the API. The Toshiba stack is also available with certain non-OEM Bluetooth accessories such as USB Bluetooth dongles and PCMCIA cards from various vendors.

The Toshiba stack supports one of the more comprehensive list of Bluetooth profiles including: SPP, DUN, FAX, LAP, OPP, FTP, HID, HCRP, PAN, BIP, HSP, HFP (including Skype support), A2DP, AVRCP, GAVDP.

- **BlueSoleil**

BlueSoleil is a product of IVT Corporation, which produces stacks for embedded devices and desktop systems. The stack is available in both standard and VOIP versions. It supports the profiles DUN, FAX, HFP, HSP, LAP, OBEX, OPP, PAN SPP, AV, BIP, FTP, GAP, HID, SDAP, and SYNC.

1.3.1.2 LINUX

The Linux operating system currently has two widespread Bluetooth stack implementations:

- BlueZ included with the official Linux kernel distributions, initially developed by Qualcomm.
- Affix, developed by Nokia Research Center.

- **BlueZ**

BlueZ is the official Bluetooth stack for Linux and is used in Google's Android OS (1). Its goal is to make an implementation of the Bluetooth wireless standards specifications for Linux. As of 2006, the BlueZ stack supports all core Bluetooth protocols and layers. It was initially developed by Qualcomm, and is available for Linux kernel versions 2.4.6 and up.

- **Affix**

Affix is a Bluetooth Protocol Stack for Linux developed by Nokia Research Center in Helsinki and released under GPL. Affix supports core Bluetooth protocols like HCI, L2CAP 1.1, L2CAP 1.2, RFCOMM, SDP and various Bluetooth profiles.

1.4 BLUEZ

In addition to the basic stack, the Bluez-utils and Bluez-firmware packages contain low level utilities such as dfutool which can interrogate the Bluetooth adapter chipset to determine whether its firmware can be upgraded.

hidd is the Bluetooth human interface device (HID) daemon.

The Bluetooth protocol stack is split into two parts:

1. A Controller Stack: It contains the timing critical radio interface. It is generally implemented in a low cost silicon device containing the Bluetooth radio and micro-processor. Protocols of controller stack are:
   - ACL (Asynchronous Connection less Link)
   - SCO (Synchronous Connection Oriented link)
   - LMP (Link Management Protocol)
   - HCI (Host/Controller Interface)

2. <u>A Host Stack: </u>It deals with high level data. It is generally implemented as a part of the operating system. Protocols in the host stack are:

    - L2CAP (Logical Link Control and Adaptation Protocol)
    - BNEP (Bluetooth Network Encapsulation)
    - RFCOMM (Radio Frequency Communication)
    - SDP (Service Discovery Protocol)
    - AVCTP (Audio/Visual Control Transport Protocol)
    - AVDTP (Audio/Visual Data Transport Protocol)
    - OBEX (Object Exchange)

1.4.1 THE CONTROLLER STACK

In omniPesence, the controller stack protocols have been maneuvered. Let us look into the details of the protocols of the controller stack:

<u>ACL (Asynchronous Connection Less Link)</u>

It refers to the normal type of radio links used for general data packets using a polling Time division Multiple Access scheme to arbitrate access. It can carry several types of packets. Packets are distinguished n the basis of several parameters such as length, forward error connection, modulation etc. ACL is usually connection oriented. Acknowledgement based transfer is done.

<u>SCO (Synchronous Connection Oriented)</u>

It refers to the type of radio link used for voice data. An SCO link is a set of received time slots on an existing ACL link. There is no option of re-transmission available here but forward error detection can be optionally re-applied.

eSCO(enhanced SCO) is a variant that uses re-transmission within a limited time frame to achieve reliability.

LMP (Link Management Protocol)

It is used to establish a control on the radio link. It is implemented on the controller stack.

HCI (Host Controller Interface)

It is the standardized communication between the host stack (eg a PC or a mobile phone operating system) and the controller stack (bt IC).

There are several HCI transport layer standards, each using a different hardware interface to transfer the same command, event and data packets. The most commonly used ones are USB (in PCs) and UART (in PDAs, phones etc).

1.4.2 THE HOST STACK

Now let us see the basics of the host stack. This part of the software are normally given in the API's of all well known bluetooth stacks. The Operating system uses these protocols to actually send data and recieve them.

L2CAP (Logical Link Control and Adaptation Protocol)

It is used to multiplex multiple logical connections between two devices using different higher level protocols. It provides segmentation and reassembly of on-air packets. In basic mode, L2CAP provides reliable sequenced packets with a payload configurable up to 64kB, with 672 bytes as the minimum mandatory supported size. In retransmission & flow control modes, L2CAP can be configured for reliable or isochronous data per channel by configuring the number of retransmissions and flush timeout.

The EL2CAP specification adds an additional "enhanced mode" to the core specification, which is an improved version of retransmission & flow control modes.

BNEP (Bluetooth Network Encapsulation)

BNEP is used to transfer another protocol stack's data via an L2CAP channel. Its main purpose is the transmission of IP packets in the Personal Area Networking Profile. BNEP performs a similar function to SNAP in Wireless LAN.

RFCOMM (Radio Frequency Communication)

Radio frequency communications (RFCOMM) is the cable replacement protocol used to create a virtual serial data stream. RFCOMM provides for binary data transport and emulates EIA-232 (formerly RS-232) control signals over the Bluetooth baseband layer.

RFCOMM provides a simple reliable data stream to the user, similar to TCP. It is used directly by many telephony related profiles as a carrier for AT commands, as well as being a transport layer for OBEX over Bluetooth.

Many Bluetooth applications use RFCOMM because of its widespread support and publicly available API on most operating systems. Additionally, applications that used a serial port to communicate can be quickly ported to use RFCOMM.

SDP (Service Discovery Protocol)

It is used to allow devices to discover what services each other support, and what parameters to use to connect to them. For example, when connecting a mobile phone to a Bluetooth headset, SDP will be used to determine which Bluetooth profiles are supported by the headset (Headset Profile, Hands Free Profile, Advanced Audio Distribution Profile etc) and the protocol multiplexer settings needed to connect to each of them. Each service is identified by a Universally Unique Identifier (UUID), with official services (Bluetooth profiles) assigned a short form UUID (16 bits rather than the full 128).

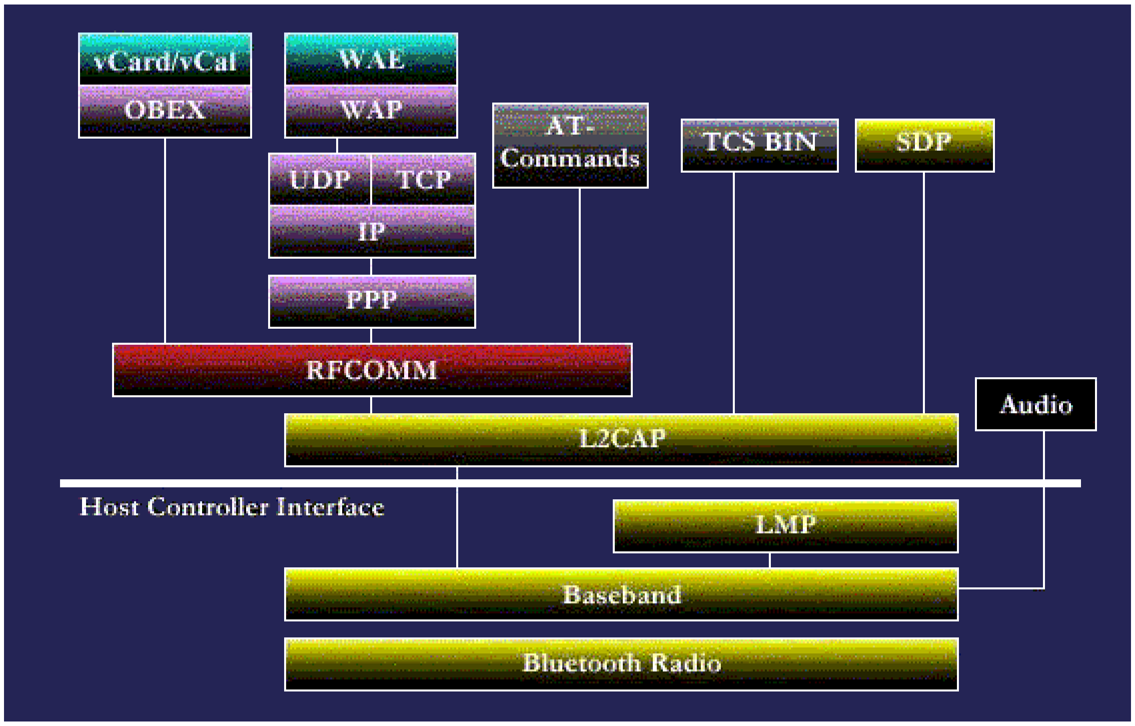
AVCTP (Audio/Visual Control Transport Protocol)

It is used by the remote control profile to transfer AV/C commands over an L2CAP channel. The music control buttons on a stereo headset use this protocol to control the music player.

AVDTP (Audio/Visual Data Transport Protocol)

It is used by the advanced audio distribution profile to stream music to stereo headsets over an L2CAP channel. It is intended to be used by video distribution profile.

The following figure illustrates the Bluetooth stack effectively:



1.5 EMBEDDED IMPLEMENTATIONS:

These implementations are normally embedded into a devices ROM or Operating system. They mightnot support all kinds of protocols but definately supports the protocols the device is made for. e.g. A bluetooth enabled Keyboard or a mobile device. We can site an example of a bluetooth enabled Digital camera designed to use only RFCOMM to send pictures from the camera to a PC or a mobile device. In this case the RFCOMM protocol software is embedded into the ROM of the digital camera. Such Implementations are normally reserved by the technology makers of the device. Few Such implementations are given below.

1.5.1 BlueMagic

BlueMagic 3.0 is Open Interface's (now Qualcomm) highly portable embedded Bluetooth protocol stack which power's Apple's iPhone and Qualcomm-powered devices such as the Motorola RAZR. BlueMagic also ships in products by Logitech, Samsung, LG, Sharp, Sagem, and more. BlueMagic 3.0 was the first fully certified (all protocols and profiles) Bluetooth protocol stack at the 1.1 level.

1.5.2 BlueCore Host Software (BCHS)

CSR's BCHS or BlueCore Host Software provides the upper layers of the Bluetooth protocol stack (above HCI, or optionally RFCOMM) - plus a large library of Profiles - providing a complete system software solution for embedded BlueCore applications. BCHS supports 1.2, 2.0+EDR and 2.1+EDR. Current qualified Profiles available with BCHS: A2DP, AVRCP, PBAP, BIP, BPP, CTP, DUN, FAX, FM API, FTP GAP, GAVDP, GOEP, HCRP, Headset, HF1.5, HID, ICP, JSR82, LAP Message Access Profile, OPP, PAN, SAP, SDAP, SPP, SYNC, SYNC ML.

1.5.3 lwBT

lwBT is an open source lightweight Bluetooth protocol stack for embedded systems. It acts as a network interface for the lwIP protocol stack.

It supports some Bluetooth protocols and layers, such as the H4 and BCSP UART layers. Supported higher layers include: HCI, L2CAP, SDP, BNEP, RFCOMM and PPP. The supported profiles are: PAN (NAP, GN, PANU), LAP, DUN and Serial Port.

lwBT has been ported to the Renesas M16C, used on the Mulle platform, and Atmega AVR line of micro-controllers, and Linux as well as Windows.

1.5.4 Windows CE

Windows CE is Microsoft's embedded operating system, which also supports Bluetooth. However, different stacks can be installed on windows CE devices, including Microsoft, Widcomm, and Toshiba, depending on the embedded device on which the OS is installed.

1.5.5 BlueLet

It is also a product of IVT Corporation. DUN, FAX, HFP, HSP, LAP, OBEX, OPP, PAN and SPP are currently supported.

1.5.6 ClarinoxBlue

Bluetooth host subsystem product of Clarinox Technologies. Support for WinCE, Embedded Linux, eCos, VelOSity, DSP-BIOS, QNX and ThreadX. HCI, L2CAP, RFCOMM, SDP, SDAP, GAP, SPP, DUN, HFP, HSP, OBEX, FTP, AVRCP, A2DP, AVDTP are currently supported.

1.5.7 Symbian Operating System:

Symbian OS is an operating system for mobile phones, which includes a Bluetooth stack. All phones based on Nokia's S60 platform and Sony Ericsson/Motorola's UIQ platform use this stack. The Symbian Bluetooth stack runs in user mode rather than kernel mode, and has public APIs for L2CAP, RFCOMM, SDP, AVRCP, etc. Profiles supported in the OS include GAP, OBEX, SPP, AVRCP, GAVDP, PAN, PBAP Additional profiles supported in the OS + S60 platform combination include A2DP, HSP, HFP1.5, FTP, OPP, BIP, DUN, SIM access, device ID.

1.6 TIME DIVISION MULTIPLEXING:

While discussing ACL, we came across a term called Time Division Multiple Access Schemes. This is also known as Time Division Multiplexing. Let us discuss Time Division Multiplexing or TDM in a little detail.

Time-Division Multiplexing (TDM) is a type of digital or (rarely) analog multiplexing in which two or more signals or bit streams are transferred apparently simultaneously as sub-channels in one communication channel, but are physically taking turns on the channel. The time domain is divided into several recurrent timeslots of fixed length, one for each sub-channel. A sample byte or data block of sub-channel 1 is transmitted during timeslot 1, sub-channel 2 during timeslot 2, etc. One TDM frame consists of one timeslot per sub-channel. After the last sub-channel the cycle starts all over again with a new frame, starting with the second sample, byte or data block from sub-channel 1, etc.

1.6.1 TDM VS. PACKET MODE COMMUNICATION

In its primary form, TDM is used for circuit mode communication with a fixed number of channels and constant bandwidth per channel.

What distinguishes time-division multiplexing from statistical multiplexing such as packet mode communication (also known as statistical time-domain multiplexing) is that the time-slots are recurrent in a fixed order and pre-allocated to the channels, rather than scheduled on a packet-by-packet basis. Statistical time-domain multiplexing resembles, but should not be considered as, time division multiplexing

In dynamic TDMA, a scheduling algorithm dynamically reserves a variable number of timeslots in each frame to variable bit-rate data streams, based on the traffic demand of each data stream. Dynamic TDMA is used in

- HIPERLAN/2
- IEEE 802.16a

1.6.2 TRANSMISSION USING TDM

In circuit switched networks such as the Public Switched Telephone Network (PSTN) there exists the need to transmit multiple subscribers' calls along the same transmission medium. To accomplish this, network designers make use of TDM. TDM allows switches to create channels, also known as tributaries, within a transmission stream. A standard DS0 voice signal has a data bit rate of 64 kbit/s, determined using Nyquist's sampling criterion. TDM takes frames of the voice signals and multiplexes them into a TDM frame which runs at a higher bandwidth. So if the TDM frame consists of n voice frames, the bandwidth will be n*64 kbit/s.

Each voice sample timeslot in the TDM frame is called a channel. In European systems, TDM frames contain 30 digital voice channels, and in American systems, they contain 24 channels. Both standards also contain extra bits (or bit timeslots) for signaling (see Signaling System 7) and synchronization bits.

Multiplexing more than 24 or 30 digital voice channels is called higher order multiplexing. Higher order multiplexing is accomplished by multiplexing the standard TDM frames. For example, a European 120 channel TDM frame is formed by multiplexing four standard 30 channel TDM frames. At each higher order multiplex, four TDM frames from the immediate lower order are combined, creating multiplexes with a bandwidth of n x 64 kbit/s, where n = 120, 480, 1920, etc.

1.6.3 SYNCHRONOUS T.D.M.

Under this we basically need to understand the concept of Synchronous Digital Hierarchy.

1.6.4 SYNCHRONOUS DIGITAL HIERARCHY (SDH)

Plesiochronous Digital Hierarchy (PDH) was developed as a standard for multiplexing higher order frames. PDH created larger numbers of channels by multiplexing the standard Europeans 30 channel TDM frames. This solution worked for a while; however PDH suffered from several inherent drawbacks which ultimately resulted in the development of the Synchronous Digital Hierarchy (SDH). The requirements which drove the development of SDH were as follows:

- **Be synchronous** – All clocks in the system must align with a reference clock.
- **Be service-oriented** – SDH must route traffic from End Exchange to End Exchange without worrying about exchanges in between, where the bandwidth can be reserved at a fixed level for a fixed period of time.
- **Allow** frames of any size to be removed or inserted into an SDH frame of any size.
- Easily **manageable** with the capability of transferring management data across links.
- Provide high levels of **recovery** from faults.
- Provide **high data rates** by multiplexing any size frame, limited only by technology.
- Give **reduced** bit rate **errors**.

SDH has become the primary transmission protocol in most PSTN networks. It was developed to allow streams 1.544 Mbit/s and above to be multiplexed, so as to create larger SDH frames known as Synchronous Transport Modules (STM). The STM-1 frame consists of smaller streams that are multiplexed to create a 155.52 Mbit/s frame. SDH can also multiplex packet based frames such as Ethernet, PPP and ATM.

While SDH is considered to be a transmission protocol (Layer 1 in the OSI Reference Model), it also performs some switching functions, as stated in the third bullet point requirement listed above. The most common SDH Networking functions are as follows:

- **SDH Crossconnect** – The SDH Crossconnect is the SDH version of a Time-Space-Time cross point switch. It connects any channel on any of its inputs to any channel on any of its outputs. The SDH Crossconnect is used in Transit Exchanges, where all inputs and outputs are connected to other exchanges.
- **SDH Add-Drop Multiplexer** – The SDH Add-Drop Multiplexer (ADM) can add or remove any multiplexed frame down to 1.544Mb. Below this level, standard TDM can be performed. SDH ADMs can also perform the task of an SDH Crossconnect and are used in End Exchanges where the channels from subscribers are connected to the core PSTN network.

SDH Network functions are connected using high-speed Optic Fiber. Optic Fiber uses light pulses to transmit data and is therefore extremely fast. Modern optic fiber transmission makes use of Wavelength Division Multiplexing (WDM) where signals transmitted across the fiber are transmitted at different wavelengths, creating additional channels for transmission. This increases the speed and capacity of the link, which in turn reduces both unit and total costs.

1.6.5 STATISTICAL TIME DIVISION MULTIPLEXING

STDM is an advanced version of TDM in which both the address of the terminal and the data itself are transmitted together for better routing. Using STDM allows bandwidth to be split over 1 line. Many college and corporate campuses use this type of TDM to logically distribute bandwidth.

If there is one 10MBit line coming into the building, STDM can be used to provide 178 terminals with a dedicated 56k connection (178 * 56k = 9.96Mb). A more common use however is to only grant the bandwidth when that much is needed. STDM does not reserve a time slot for each terminal, rather it assigns a slot when the terminal is requiring data to be sent or received.

This is also called Asynchronous Time-division Multiplexing (ATDM), in an alternative nomenclature in which "STDM" or "Synchronous Time Division Multiplexing" designates the older method that uses fixed time slots.

1.7 BLUETOOTH DEVICES

A Bluetooth USB dongle with a 100m range.

Bluetooth exists in many products, such as telephones, the Wii, PlayStation 3 and recently in some high definition watches, modems and headsets. The technology is useful when transferring information between two or more devices that are near each other in low-bandwidth situations. Bluetooth is commonly used to transfer sound data with telephones (i.e. with a Bluetooth headset) or byte data with hand-held computers (transferring files).

Bluetooth protocols simplify the discovery and setup of services between devices. Bluetooth devices can advertise all of the services they provide. This makes using services easier because more of the security, network address and permission configuration can be automated than with many other network types.

1.8 COMMUNICATION AND CONNECTION

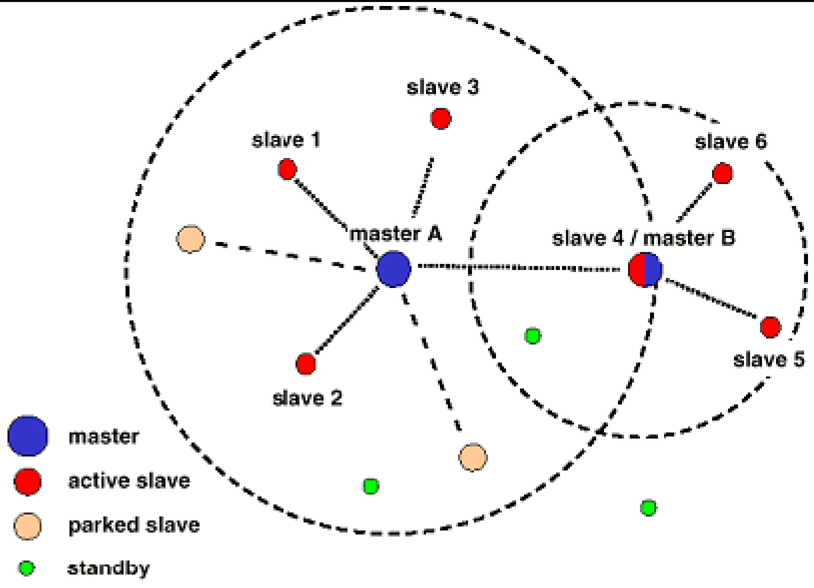
The physical links with a Bluetooth device are created on the basis of masters and slaves. A master can control several slaves (up to 7 in its zone). This forms a small network known as **Piconet**.

A piconet is an ad-hoc computer network, using Bluetooth technology protocols to allow one master device to interconnect with up to seven active devices. Up to 255 further devices can be inactive, or parked, which the master device can bring into active status at any time.

A master is simply the first apparatus connected and it is the master that sets the clock, the frequency jumping sequence and the access code for the link. At any given time, data can be transferred between the master and one other device, however, the devices can switch roles and the slave can become the master at any time. The master switches rapidly from one device to another in a round-robin fashion. (Simultaneous transmission from the master to multiple other devices is possible, but not used much). All the Bluetooth modules of the same piconet use the same frequency jumping sequence and are synchronized with the master's clock. The network becomes saturated when the number of devices increases in the piconet.

The Bluetooth specification allows connecting two or more piconets together to form a scatternet, with some devices acting as a bridge by simultaneously playing the master role in one piconet and the slave role in another.

The following illustrates the diagram of a scatternet:



The transmission diagram of the center of the piconet is based on the principle of Time Division Duplex (TDD).

If we have a look at the whole process as such, here is what happens. The master sends a data packet on the network.

A synchronization word is attached to the head of the data packet which enables the slave to re-synchronize its clock as per the master.

1.8.1 SETTING UP CONNECTIONS

Any Bluetooth device will transmit the following information on demand:

- Device name.
- Device class.
- List of services.
- Technical information, for example, device features, manufacturer, Bluetooth specification used, clock offset.

Any device may perform an inquiry to find other devices to connect to, and any device can be configured to respond to such inquiries. However, if the device trying to connect knows the address of the device, it always responds to direct connection requests and transmits the information shown in the list above if requested. Use of a device's services may require pairing or acceptance by its owner, but the connection itself can be initiated by any device and held until it goes out of range. Some devices can be connected to only one device at a time, and connecting to them prevents them from connecting to other devices and appearing in inquiries until they disconnect from the other device.

Every device has a unique 48-bit address. However these addresses are generally not shown in inquiries. Instead, friendly Bluetooth names are used, which can be set by the user. This name appears when another user scans for devices and in lists of paired devices.

Most phones have the Bluetooth name set to the manufacturer and model of the phone by default. Most phones and laptops show only the Bluetooth names and special programs are required to get additional information about remote devices. This can be confusing as, for example, there could be several phones in range named T610.

1.8.2 PAIRING

Pairs of devices may establish a trusted relationship by learning (by user input) a shared secret known as a passkey. A device that wants to communicate only with a trusted device can cryptographically authenticate the identity of the other device. Trusted devices may also encrypt the data that they exchange over the airwaves so that no one can listen in. The encryption can, however, be turned off, and passkeys are stored on the device file system, not on the Bluetooth chip itself. Since the Bluetooth address is permanent, a pairing is preserved, even if the Bluetooth name is changed. Pairs can be deleted at any time by either device. Devices generally require pairing or prompt the owner before they allow a remote device to use any or most of their services. Some devices, such as mobile phones, usually accept OBEX business cards and notes without any pairing or prompts.

Certain printers and access points allow any device to use their services by default, much like unsecured Wi-Fi networks. Pairing algorithms are sometimes manufacturer-specific for transmitters and receivers used in applications such as music and entertainment.

Bluetooth 2.1 has an optional "touch-to-pair" feature based on Near Field Communication (NFC). By simply bringing two devices into very close range (around 10 cm/4 in), pairing can securely take place without entering a passkey or manual configuration.

1.8.3 AIR INTERFACE

The protocol operates in the license-free ISM band at 2.4-2.4835 GHz. To avoid interfering with other protocols that use the 2.45 GHz band, the Bluetooth protocol divides the band into 79 channels (each 1 MHz wide) and changes channels up to 1600 times per second. Implementations with versions 1.1 and 1.2 reach speeds of 723.1 kbit/s. Version 2.0 implementations feature Bluetooth Enhanced Data Rate (EDR) and reach 2.1 Mbit/s. Technically, version 2.0 devices have a higher power consumption, but the three times faster rate reduces the transmission times, effectively reducing power consumption to half that of 1.x devices (assuming equal traffic load).

1.8.4 SECURITY

Overview:

Bluetooth implements confidentiality, authentication and key derivation with custom algorithms based on the SAFER+ block cipher. In Bluetooth, key generation is generally based on a Bluetooth PIN, which must be entered into both devices. This procedure might be modified if one of the devices has a fixed PIN, e.g. for headsets or similar devices with a restricted user interface. During pairing, an initialization key or master key is generated, using the E22 algorithm. The E0 stream cipher is used for encrypting packets, granting confidentiality and is based on a shared cryptographic secret, namely a previously generated link key or master key. Those keys, used for subsequent encryption of data sent via the air interface, rely on the Bluetooth PIN, which has been entered into one or both devices.

An overview of Bluetooth vulnerabilities exploits has been published by Andreas Becker.

In September 2008, the National Institute of Standards and Technology (NIST) published a Guide to Bluetooth Security that will serve as reference to organization on the security capabilities of Bluetooth and steps for securing Bluetooth technologies effectively. While Bluetooth has its benefits, it is susceptible to denial of service attacks, eavesdropping, man-in-the-middle attacks, message modification, and resource misappropriation. Users/organizations must evaluate their acceptable level of risk and incorporate security into the lifecycle of Bluetooth devices. To help mitigate risks, included in the NIST document are security checklists with guidelines and recommendations for creating and maintaining secure Bluetooth piconets, headsets, and smart card readers.

Bluejacking:

Bluejacking is the sending of either a picture or a message from one user to an unsuspecting user through Bluetooth wireless technology. Common applications are short messages (e.g. "You've just been bluejacked!"), advertisements (e.g. "Eat at Joe's"), and business information. Bluejacking does not involve the removal or alteration of any data from the device.

1.8.5 BLUETOOTH VS. WI-FI IN NETWORKING

Bluetooth and Wi-Fi have many applications in today's offices, homes, and on the move: setting up networks, printing, or transferring presentations and files from PDAs to computers. Both are versions of unlicensed wireless technology. Wi-Fi differs from Bluetooth in that it provides higher throughput and covers greater distances, but requires more expensive hardware and may present higher power consumption.

They use the same frequency range, but employ different modulation techniques. While Bluetooth is a replacement for cabling in a variety of small-scale applications, Wi-Fi is a replacement for cabling for general local area network access.

1.9 THE LINUX OPERATING SYSTEM

Linux is an open source operating system coded intially by Linus Torvalds at university of Helsinki in 1991. Today it is maintained by the Free Software foundation GNU. Linux is not only free to use but is a great development tool too.

Linux initially was made for 386 AT clones but today runs on almost all processors with varying hardware types. Linux can be used as a server as well as a desktop standalonce configuration. It is also being used in embedded devices like Google Android and Ubuntu MID.

Today Torvalds leads the development of the linux kernel. The current release is kernel 2.6.29. Currently the X Window System adds all possibilities for fully graphical desktop as like other operating systems. Two such ,ostly used desktop managers based on the X Window System are KDE and GNOME.

Linux based distributions are intended by developers for interoperability with other operating sytem, although it is by far the most widely used. Free software projects, although developed in a collaborative fashion, are often produced independently of each other. The fact that the software licences explicitely permit redistribution, however, provides a basis for large scale projects that collect the software produced by stand-alone projects and make it available all at once in the form of a linux distribution.

1.9.1 PROGRAMMING IN LINUX

Most linux distributions support dozens of programming languages. The most common collection of utilities for building both linux application and operating system programs is found within the GNU toolchain, which includes the GNU Compiler Collection (GCC) and the GNU Build system. Amongst others, GCC provides compilers for Ada, C,C++, Java and Fortran. The Linux Kernel itself is written to be compiled with GCC. Proprietary compilers for linux include the Intel C++ Compiler and IBM XL C/C++ Compiler.

Two main frameworks for developing graphical application are those of GNOME and KDE. These Projects are based on the GTK+ and Qt Widget Toolkits, respectively, which can also be used independently of the larger framework. Both support a wide variety of languages.

The following are the reasons as to why we selected Linux as our platform of coding.

- Linux has a powerful socket interface (Berkeley Sockets) which is used in almost every kind of networking and communication on Linux, be it ethernet, internet, Bluetooth, infrared or wifi etc.

- Linux has an inbuilt powerful compiler which gives power to program each and every facility of linux.

- Linux Bluetooth stack Bluez works using the powerful socket systems of Linux.

- Linux Build System gives the facility to add debugging information to a program.

- The GNU Debugger (GDB) works on the added debug information of an executable are provides debugging tools and features like backtrace, stack view etc.
- Linux provides API to connect to database, GNOKII and other 3rd party tools via C/C++.

1.10 THE C PROGRAMMING LANGUAGE

C is a general purpose structured programming language. Its Instructions consists of terms that resembles algebraic expressions. C is a high-level structured programming languages. It contains certain additional features, however, that allow it to, be used at a lower level. Thus bridging the gap between the machine language and the more conventional hogh level languages. This flexibility allows C to be used for systems programming (e.g. For writing operating systems) as well as for applications programming (e.g. For writing a program to solve a complicated system of mathematical equation or writing a program to bill customers).

C is characterised by the ability to write very concise source programs due in part to the large number of operators including within the language. It has a relatively small instruction set, though actual implementations include extensive library functions which enhance the basic instructions. Further more, the language encourages users to write additional library functions of their own. Thus, the features and capabilities of the language can be easily be extended by the user.

C Compilers are commonly available for computers of all sizes. The compilers are usually compact and they generate object programs that are small and highly efficient when compared with programs compiled from other high level languages.

Another important characteristic of C is that its program are highly portable, even more so than with other high level languages. The reason for this is that C relegates most computer dependent features to its library functions. Thus, every version of C is accompanied by its own set of library functions, which are written for particular characteristic of host computer. These library functions are relatively standardised, however, and each indivudual library function is generally accessed in the same manner from one version of C to another. Therefore most C programs can be processed on many different computers, with little or no alterations.

1.11 MYSQL DATABASE

MySQL is openSource Database managed by Sun Microsystems. It is a relational Database Management System. The program runs as a server providing multi-user access to a number of databases.

MySQL is owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now a subsidiary of Sun Microsystems which holds the copyright to most of the code-base. The project's source code is available under terms of the GNU General Public License, as well as under a variety of proprietary agreements.

This adheres to the official ANSI pronunciation; SEQUEL was an earlier IBM database language, a predecessor to the SQL language.

MySQL is written in C and C++. The SQL parser uses yacc and a home-brewed lexer, sql_lex.cc(9)

MySQL works on many different system platforms, including AIX, BSDi, FreeBSD, HP-UX, i5/OS, Linux, Mac OS X, NetBSD, Novell NetWare, OpenBSD, OpenSolaris, eComStation , OS/2 Warp, QNX, IRIX, Solaris, Symbian, SunOS, SCO OpenServer, SCO UnixWare, Sanos, Tru64 and Microsoft Windows. A port of MySQL to OpenVMS is also available.(10)

Libraries for accessing MySQL databases are available in all major programming languages with language-specific APIs. In addition, an ODBC interface called MyODBC allows additional programming languages that support the ODBC interface to communicate with a MySQL database, such as ASP or ColdFusion. The MySQL server and official libraries are mostly implemented in ANSI C/ANSI C++.

Using MySQL Database with C/C++.

The MySQL Database provides API to be included and used with Python, C++ etc.

In C, all database access functions are provided with mysql library included under the header "*mysql.h*"

- Firstly, one needs to create a connection variable using *MYSQL * connectionName*.
- Secondly, initialise the connection.
- Connect to the database using *mysql_real_connect()*.
- Parse Query using *mysql_query()*.
- Use the retirned result using a *MYSQL_RES ** variable using the mysql_fetch_row function call.
- Access rowwise using *MYSQL_ROW* variable which is an array of n rows (if n is the number of feilds in the table accessed).
- When done close the connection using *mysql_close()* .

1.12 GIMP TOOL KIT (GTK)

GTK, or The GIMP Toolkit, is a cross-platform widget toolkit for creating graphical user interfaces. It is one of the most popular toolkits for the X Window System, along with Qt.

GTK was initially created for the GNU Image Manipulation Program (GIMP), a raster graphics editor, in 1997 by Spencer Kimball and Peter Mattis, members of the eXperimental Computing Facility (XCF) at UC Berkeley.

Licensed under the LGPL, GTK+ is free software and is part of the GNU Project.

It's called the GIMP toolkit because it was originally written for developing the GNU Image Manipulation Program (GIMP), but GTK has now been used in a large number of software projects, including the GNU Network Object Model Environment (GNOME) project. GTK is built on top of GDK (GIMP Drawing Kit) which is basically a wrapper around the low-level functions for accessing the underlying windowing functions (Xlib in the case of the X windows system), and gdk-pixbuf, a library for client-side image manipulation.

GTK is essentially an object oriented application programmers interface (API). Although written completely in C, it is implemented using the idea of classes and callback functions (pointers to functions).

There is also a third component called GLib which contains a few replacements for some standard calls, as well as some additional functions for handling linked lists, etc. The replacement functions are used to increase GTK's portability, as some of the functions implemented here are not available or are nonstandard on other UNIX s'.

As the last component, GTK uses the Pango library for internationalized text output.

GTK

Developed by	GNOME Foundation
Latest release	2.16.0 / 2009-03-13;
Written in	C
OS	Cross-platform
Type	Widget toolkit
License	GNU Lesser General Public License
Website	www.gtk.org

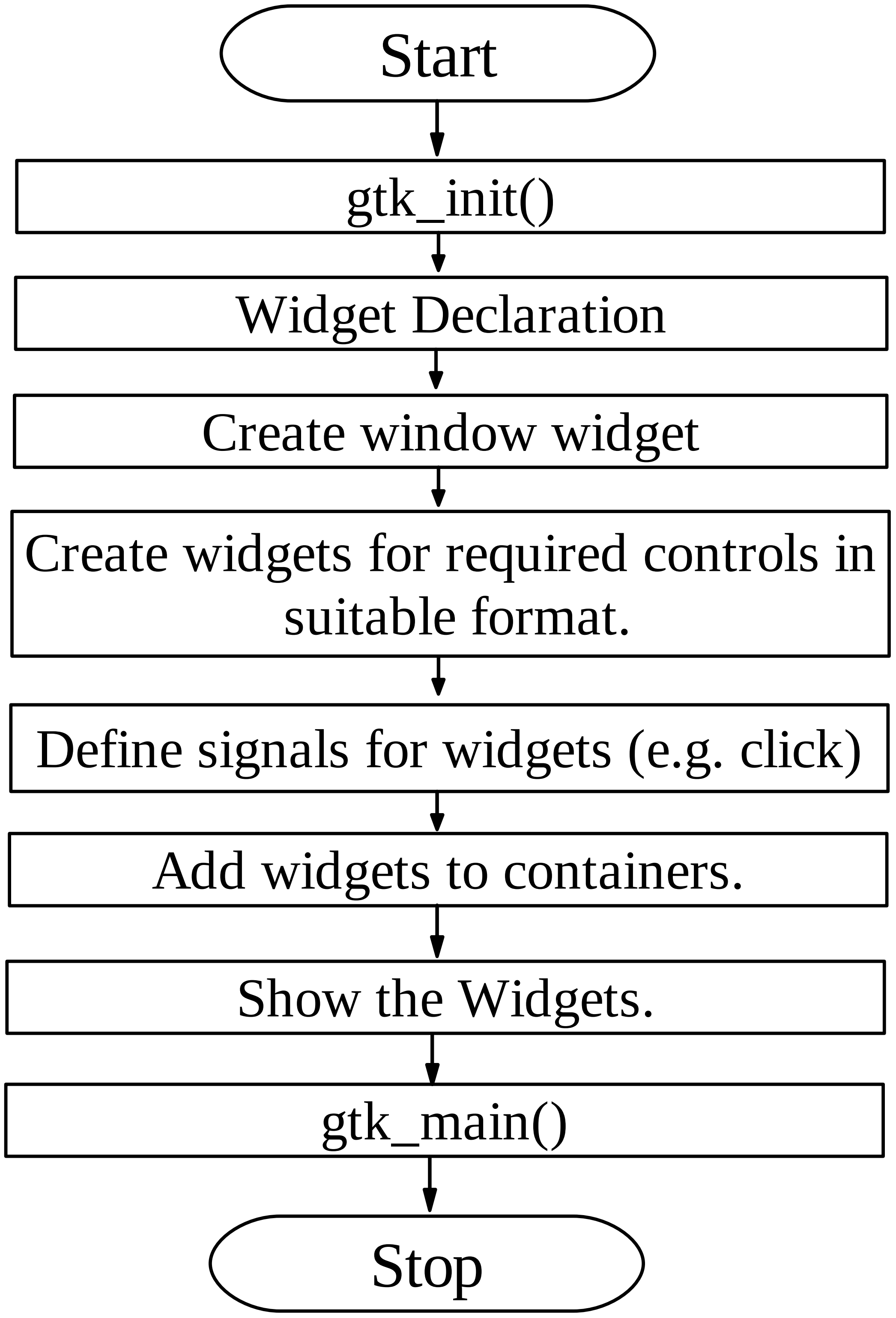
The first thing to do, of course, is download the GTK source and install it. You can always get the latest version from ftp.gtk.org. You can also view other sources of GTK information on http://www.gtk.org/. GTK uses GNU autoconf for configuration. Once untar'd, type ./configure --help to see a list of options.

1.12.1 Programming in GTK

GTK is widget based frontend programming tool, which allows us to program the X.Org graphical framework in Linux using its libraries. Everything in GTK is a dynamic control named a widget. We create, manipulate and destroy widgets to accomplish our tasks.

The gtk libraries are included by using header "gtk/gtk.h". The program must have a gtk_init(), gtk_main() and gtk_main_quit().

The workflow of GTK Graphical Coding is as follows:


Start
gtk_init()
Widget Declaration
Create window widget
Create widgets for required controls in suitable format.
Define signals for widgets (e.g. click)
Add widgets to containers.
Show the Widgets.
gtk_main()
Stop

1.13 SOCKETS

When we are talking of Bluetooth connectivity, it is essential that we also take a look at the basics of sockets. A socket is an end-point of a bi-directional process to process communication flow across an IP based network. Each socket is mapped to an application process or thread. A socket is an interface between an application process or thread and the TCP/IP protocol stack provided by the operating system.

A socket is identified by the operating system as a unique combination of the following:

- Protocol (TCP, UDP or raw IP)
- Local IP address
- Local port number
- Remote IP address (only established for TCP sockets)
- Remote port number (only established for TCP sockets)

The operating system forwards incoming IP data packets by extracting the above socket address information from the IP, UDP and TCP headers.

Communicating local and remote sockets are called socket pairs.

On Unix-like operating systems the netstat-an command line tool lists all currently established sockets. On some systems, netstat-b lists sockets created by application programs.

1.13.1 TYPES OF SOCKETS

There are three main types of sockets. They basically categorised as per connectivity:

- Datagram Sockets, also known as connectionless sockets which us UDP
- Stream Sockets, also known as connection oriented sockets which use TCP or SCTP
- Raw Sockets, also known as Raw IP sockets which are typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are not stripped off, but are accessible to the application. Application examples are ICMP, IGMP and OSPF.

1.13.2 IMPLEMENTATION

Sockets are usually implemented by an API library, such as Berkeley sockets, first introduced in 1983. Most implementations are based on Berkeley sockets, for example Winsock introduced in 1991. Other socket API implementations exist, such as the STREAMS based Transport Layer Interface (TLI).

These are the examples of functions or methods provided by the API library:

- **socket ()**, creates a new socket of a certain socket type, identified by an integer number, and allocates system resources to it.
- **bind ()**, is typically used on the server side, and associates a socket with a socket address structure, i.e., a specified local port number and IP address.
- **listen ()**, is used on the server side, and causes a bound TCP socket to enter the listening state.
- **connect ()**, is used on the client side, and assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection.

- **accept ()**, is used on the server side. It accepts a received incoming attempt to create a new TCP connection from the remote client, and creates a new socket associated with the socket address pair of this connection.
- **send ()** and **receive (),** or **write ()** and **read (),** or **recvfrom ()** and **sendto (),** are used for sending and receiving data to/from a remote client.
- **close (),** causes the system to release resources allocated to the socket. In case of TCP, the connection is terminated.

1.14 GNOKII AND SHORT MESSAGING SERVICE

A few other requirements that will be essential are to be mentioned. A major need is a Nokia mobile phone. This will be needed because this project incorporates a facility of informing employees that their attendance has been marked. omniPresence provides this facility by generating an SMS (Short Messaging Service). This SMS will be generated by the omniPresence software itself. The SMS will be sent by using a Nokia phone in association with the software.

A question arises here as to why should we use only a Nokia phone. The simple reason is that Nokia phones run a service which can be accessed by a Linux operating system by using ASCII Terminal Commands. In other words, we all know that a Nokia phone can be accessed by using Nokia PC Suite. What happens is that in the mobile phone, Symbian (discussed in Bluetooth Stack) runs a server side service which interacts with the PC.

We should also have an idea as to how Linux will communicate with the phone. It occurs through the aid of a device driver. Gnokii is an opensource Nokia Driver Package which runs on Linux.

Delving into the technical aspect a little, Gnokii is a suite of programs for communicating with mobile phones. It was initially only available for Nokia mobile phones, but later extended to support others. So, going by convention, we have used Nokia phones. It is available for Linux, BSD unix, Windows and Mac OS X and as source code.

Gnokii itself is a console tool, but it is used by several GUIs to communicate with phones; for example: Xgnokii [1], Gnocky & Gnome Phone Manager all use Gnokii internally.

It is licenced using the GNU GPL.

Gnokii's features include:

- Use to Activate Nokia network monitor
- Supports sending SMS (with delivery report), picture messages, can send/receive ring tones (as SMS)
- Phone book
- Dial/Receive calls
- Calendar

Gnokii can connect using serial/USB cables, infrared & Bluetooth.

1.15 POSIX THREADS

A thread is a basic usnit of CPU Utilization; it comprises a thread ID, a program counter, a register set, and a stack.

A thread library provides the programmer an API for creating and managing threads. There are two primary ways of implementing a thread library. There are three main threaded libraries in use today.

- POSIX Threads
- Java Threads
- Win32 Threads

In Linux we chose POSIX Threads to multithread our application's Phase1 (described later). Also with programming Language C this is the only threaded library supported with this language.

POSIX Threads, or Pthreads, is a POSIX (IEEE 1003.1c) standard for threads. The standard defines an API for creating and manipulating threads. Pthreads are most commonly used on Unix-like POSIX systems such as Linux, Mac OS X, Solaris and Microsoft Windows.

Pthread is a specification not an implementation, the operating system developers may implement it as per the operating system's design and working.

Pthreads defines a set of C programming language types, functions and constants. It is implemented with a **pthread.h** header and a thread library. Programmers can use Pthreads to create, manipulate and manage threads, as well as synchronize between threads using mutexes and signals. In Pthreads program, separate threads begin execution in a specified function.

1.15.1 IMPLEMENTATION

Lets look at the implementation of a Pthread program. Firstly, a pthread program must include a "*pthread.h*" header file. Secondly, a *pthread_t* declares a identifier for a thread to be created.

After that, each thread is assigned attributes like stack size, scheduling information etc. A *pthread_attr_t* defines an attribute variable which may be assigned to a thread using a function *pthread_attr_init* () which is normally done by the calling thread function (usually main).

Finally we create the thread, and put it to run mode. We may use pthread_exit function to get out of a thread whenever we want.

A Mutex is a serialization technique to prevent concurrency problems from happening (especially deadlocks). When a mutex when is locked, prevents the thread to stop using the variable until when unlocked. A variable or a code is executed between a *mutex_lock* and a *mutex_unlock* function call to serialize it.

S.R.S.

(SOFTWARE REQUIREMENT SPECIFICATION)

Software Requirements Specification

for

omniPresence

Version 0.1

Prepared by

Saurajeet Dutta (BCA/2004/06)

Rahul Bose (BCA/2016/06)

Jyoti Prakash Nath (BCA/2018/06)

Purushottam (BCA/2042/06)

BIRLA INSTITUTE OF TECHNOLOGY, MESRA

EXTENSION CENTRE, LALPUR - 834001

On

January 6, 2008

2.1 INTRODUCTION

2.1.1 Purpose

omniPresence v 0.1 is an RFID (Radio Frequency Identification) System. It uses short range radio service like bluetooth to detect devices in a Piconet and record it in a back end MYSQL database.

This system also processes the recorded device information and use it mark attendances of staffs working in an organization. omniPresence also takes the responsibility to notify the present as well as the absent staffs by using Short Messaging Service facility of a local cellular network.

2.1.2 Document Conventions

The following typographical conventions were followed to write this SRS

HEADING:	FONT:	Chalet Arabic
	SIZE:	20
	FORMATTING:	UPPERCASE, Bold
SUBHEADING:	FONT:	Chalet Arabic
	SIZE:	14
	FORMATTING:	BOLD
SUB SUBHEADING:	FONT:	URW Gothic L
	SIZE:	14
	FORMATTING:	BOLD
TEXT:	FONT:	URW Gothic L
	SIZE:	14
	SIZE:	NORMAL

This document does not use or inherit any third party or higher level software requirements specifications.

2.1.3 Intended Audience & Reading Suggestions

The documentation is intended for developers, project managers, marketing staff, users, testers, and documentation writers or whoever feels it worth reading to know how our software functions.

The rest of the documentation contains the computer or system requirements. Followed by the design and technology of the omniPresence. It is suggested to the readers to read the beginning sections of the documentation viz Chapter 1 to get a brief idea about the existing technologies and to know the foundation of our software.

2.1.4 Project Scope

omniPresence can be used by all kinds of business firms.

<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here. An SRS that specifies the next release of an evolving product should contain its own scope statement as a subset of the long-term strategic product vision.>

2.1.5 References

The Bluetooth SIG Specification –
http://www.bluetooth.com

IEEE Standard 802.15.1-2002 -
standards.ieee.org/announcements/802151app.html

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a

copy of each reference, including title, author, version number, date, and source or location.>

2.2 OVERALL DESCRIPTION

2.2.1 Product Perspective

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

2.2.2 Product Features

<Summarize the major features the product contains or the significant functions that it performs or lets the user perform. Details will be provided in Section 3, so only a high level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or a class diagram, is often effective.>

2.2.3 User Classes and Characteristics

<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the favored user classes from those who are less important to satisfy.>

2.2.4 Operating Environment

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>

2.2.5 Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).>

2.2.6 User Documentation

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.>

2.2.7 Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>

2.3 SYSTEM FEATURES

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

2.3.1 System Feature 1

<Don't really say "System Feature 1." State the feature name in just a few words.>

2.3.1.1 Description and Priority

<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific

priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

2.3.1.2 Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

2.3.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate when necessary information is not yet available.>

<Each requirement *should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

REQ-1:

REQ-2:

2.3.2 System Feature 2 (and so on)

<Divide the system features as following

1. omniPresence Command Central

2. Device Inquiry Module (Phase1)

3. Administration & Maintainence Module (Phase2)

4. Make Attendance Module (Phase2)

5. Backup Module

6. SMS Sending Module.

7. Security Features. (Like rnad and using h/w timer)>

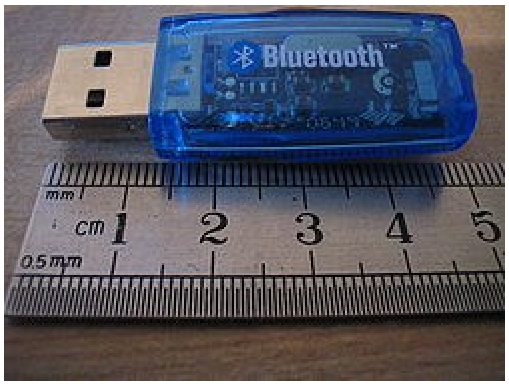
2.4 EXTERNAL INTERFACE REQUIREMENTS

2.4.1 User Interfaces

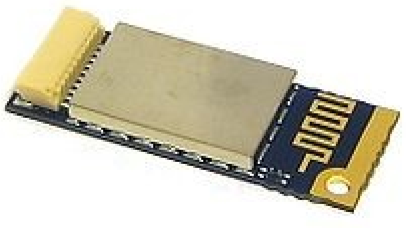
<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

2.4.2 Hardware Interfaces

2.4.2.1 Bluetooth Adapter



A typical Bluetooth USB dongle.



An internal notebook Bluetooth card (14×36×4 mm).

A personal computer must have a Bluetooth adapter in order to communicate with other Bluetooth devices (such as mobile phones, mice and keyboards). While some desktop computers and most recent laptops come with a built-in Bluetooth adapter, others will require an external one in the form of a dongle. The following attributes must be ensured for the bluetooth device for complacency with omniPresence.

- The bluetooth adapter must be compliant to EDR (See bluetooth SIG specification EDR) 2.0 or above, with implementation of embedded HCI controller stack.
- The bluetooth adapter must be IEEE – 802.15.1 – 2002 compliant.
- If an external dongle is used, the dongle must be plug n play supported or a hotplug device.
- The Bluetooth device must be at least class 2 for detection in a 10m piconet. omniPresence recommends a class 2 adapter or above.
- The bluetooth device must support all well known protocols of bluetooth communication (especially BlueZ).

2.4.2.2 Bluetooth Enables Devices to form a piconet

A mobile phone that is Bluetooth enabled is able to pair with many devices. To ensure the broadest support of feature functionality together with legacy device support.

Normally the bluetooth^{TM} enabled devices are characterised by bluetooth trademark logo (like a "B").

2.4.2.3 Nokia Mobile Phone for Sending SMS.

A Symbian S60 v 2 Mobile Phone

A Nokia Mobile Phone is a requirement for sending SMSes over a cellular network. The Nokia Mobile Phone will behave like a SMS Modem which can be used with the help of a PC to send Short Messages by using AT Commands. The following attributes must be maintained for selecting such a mobile phone for its use with omniPresence.

- The Nokia Phone Must be able to connect to a PC via a BLUETOOTH, IRDA or SERIAL connection. I case of serial connections the phone must support connectivity cables like CA-42, CA-53, DKU-2, DKU-5. Etc.

- The phone must run a Symbian^{TM} Series 60 v2 or Series 40 operating system.

- The Mobile phone should be able to connect in a Non Mass Storage Mode and in PC Suite Mode. This mode enables the use of mobile phone features on a PC.
- It must have an equipped, registered and legal cellular connection from a Service Provider of the country where omniPresence will be used. This would imply a Sender Identification Module with enabled Short messaging service for a particular domain.

2.4.3 Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

<You have created this section's layout, just elaborate it . If help needed we are there. Do it a little bit more subjective like the way you did the hardware requirements. Keep Up the good work.>

ELABORATE ALL THE FOLLOWING SUBSECTIONS:

2.4.3.1 Operating System:

A Linux System configured with kernel 2.6.27 or above, with fully configured LSB. (e.g OpenSuse 11.1, fedora)

2.4.3.2 BlueZ Library.

2.4.3.3 MySQL Library

2.4.3.4 GTK Library

2.4.3.5 GNOKII with configured gnokiirc

2.4.4 Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

5. 2.5 OTHER NONFUNCTIONAL REQUIREMENTS

2.5.1 Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

2.5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

2.5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

2.5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability,

maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

2.6 OTHER REQUIREMENTS

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

APPENDIX A: GLOSSARY

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

APPENDIX B: ANALYSIS MODELS

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

APPENDIX C: ISSUES LIST

< This is a dynamic list of the open requirements issues that remain to be resolved, including TBDs, pending decisions, information that is needed, conflicts awaiting resolution, and the like.>

NON FORMATTED SECTION

Now that we have had a look at the basics of all the technologies used in omniPresence we will progress to what really happens in this project.

As a start off omniPresence is divided into three phases- Phase 1, Phase 2 and Phase 3. We will delve into all these three phases individually.

Also, before we look into all the three phases, we just delve into what is christened OPCC. It is a command prompt designed specifically for omniPresence. So, here goes.

omniPresence Command Central (OPCC):

omniPresence involves quite a few functions that will be used frequently to facilitate the smooth functioning of the system. Apart from this, there are queries to be made and other information that anyone, the developer or the user, may be in need of. All of this must be integrated as one and be used in totality. Efficiency and effectiveness must also be ensured. The best means of doing this was to use the command prompt so that the functions that need to be performed could be carried out on the terminal itself.

OPCC is a command prompt that has been generated specifically for omniPresence. It can be used on the terminal if the user has the authorization to do so. The main objective of creating the omniPresence Command Central was to ensure security in the system. In other words, assuming anyone can use the terminal program but not all of them have the authority to query the system about the details collected by omniPresence. In this case, if we were to use any general prompt, security couldn't be guaranteed. However, by using omniPresence Command Central, or rather OPCC, we can guarantee complete security and total avoiding of unauthorized access.

Apart from this, there is the added exclusivity that is imparted to omniPresence by the use of OPCC. OPCC is a special command prompt that is used exclusively for omniPresence. It cannot be used for any other work. So, in a sense, omniPresence is unique in the fact that it uses a "special" command prompt, that is, OPCC. This uniqueness given to the software is an added perk that makes the whole project more special.

Before we have a look at the basics of the OPCC let us take a look at what happens when we use a command line prompt.

We enter the text which we intend to be commands adjacent to the command prompt. The command prompt then processes the commands entered and then verifies the fact that they are valid commands. If the commands are valid, the prompt goes on to process them. If they aren't the prompt goes on to display an error message. If we happen to encounter the former case, i.e., the input by the user is a valid command for the prompt then it is the responsibility of the prompt to execute those commands as specified. The execution of the commands specified in the command prompt is of primary importance because the main objective of the prompt should be to execute valid commands to completion and notify the user if the command entered by him/her is invalid.

The above is a brief description of what a command prompt does. It should be noted that this is what is expected of any command prompt, in this case, omniPresence Command Central. OPCC has been designed in a way that it emulates a terminal program to success. All the functions, even their sequence, of a command prompt, just told about has been established in connection to OPCC. Now, let us take a look into OPCC and its architecture to attempt to understand its functioning.

Looking into the aspects of OPCC in a more detailed way, we divide the whole operation of the OPCC in three different phases:

1. **<u>Menu:</u>**

The Menu part of the OPCC is that which helps the user to get acquainted with the same. It provides the user with two facilities: (i) help and (ii) about. As is suggested by the names, the functions of these two options available to us are obvious.
The help part of OPCC gives us a list of commands that can be executed by the user on the command prompt of omniPresence. The help part of OPCC also gives details about what each command is supposed to do. This description is just a brief line about what the user might expect if he/she runs a particular command on OPCC.
The about part of OPCC gives a brief introduction as to what OPCC actually is. It discusses it's uses and also notifies about the developers of the software.

2. **Command Line Processing:**

From the command line processing part we really take a trip down the workings of the OPCC. Just as in any other command prompt, the user enters any text (intended to be commands, though not necessary) which he/she wants to execute. Here is what happens in OPCC.

As soon as the text is entered, the command line processor scans the text. It checks whether the entered text corresponds to the rules and conventions made for OPCC. It, in other words, identifies whether the user has entered a valid command or not. The system associates a variable number with the operation that the user wants to execute. Let us call this variable SelectedOperation. OPCC has categorized commands in the following way based on the value of SelectedOperation:

Value of SelectedOperation	**Operation**
1	Operation not defined in OPCC.
0,2,3,4,5	User intends to exit.
6,7,8,9,10	User intends to continue whatever task he is doing without any deviation.

11,20,21,22,23,24,25,26	User intends to perform the task that he has specified on the prompt.

The reader must be foxed by the unusual selection of values for SelectedOperation. The fact that the numbers 12 to 19 have been omitted is weird. But the justification is that we, the developers of the software consider the numbers 13 and 17 unlucky. So, in order to play safe and also the fact that we really want this project to be successful made us leave the numbers 12 to 19 and move on to 20 directly.

Now, in the last case, where the value of SelectedOperation is 11, 20, 21, 22, 23, 24, 25 or 26 we find that the OPCC needs to execute some commands and bring about a desired result for the user. This brings us to the last and final phase of the omniPresence Command Central: Registry.

One point that needs to be taken into account is that the command line processing is done in such a way that the commands entered by the user are automatically changed to the upper case. This is done just so as to maintain a fixed set of standards. It makes the whole process more appealing visually.

3. Registry:

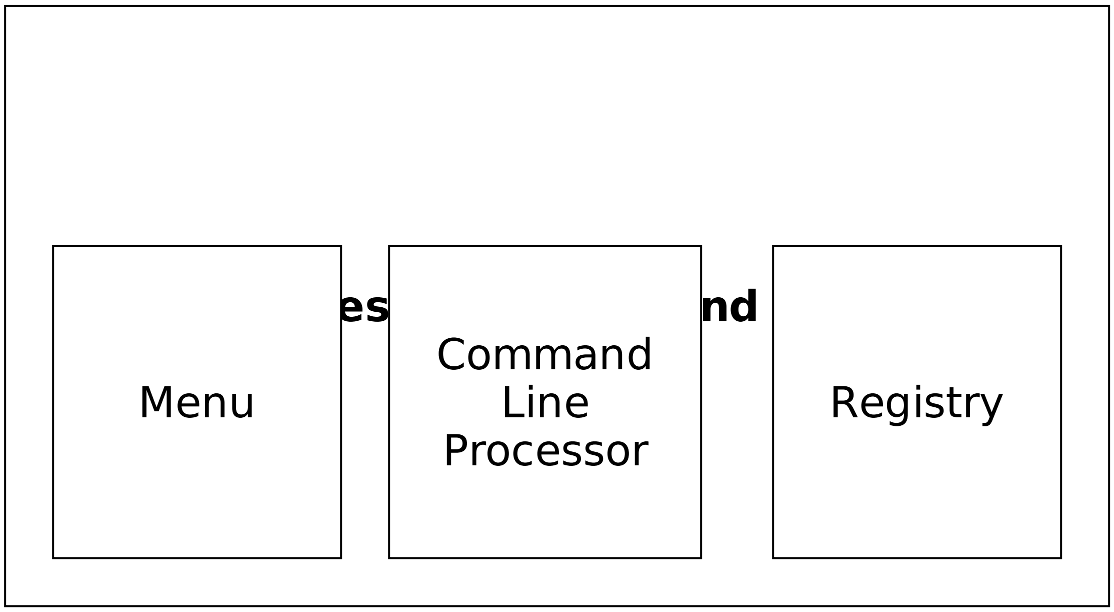
As seen earlier in the section, the OPCC needs to execute commands when the value of SelectedOperation is 11, 20, 21, 22, 23, 24, 25 or 26. These values indicate that the user wants to execute a valid command which is not exit (value of SelectedOperation = 1) and which indicates a deviation from that which is being already followed.

When we come into the registry phase, the SelectedOperation variable is converted to another variable called the regID. It is to be noted here that the value of SelectedOperation, when being converted to regID, can be encrypted or encoded. In other words, it can be changed to satisfy security concerns. However, in this project, we have used the same value for both the variables for the sake of simplicity.

The values of the regID of each operation opted for by the user indicates that the user is requesting for some operation to be executed. It is the responsibility of the Registry section of the OPCC to make sure that the commands requested for are executed to perfection and the desired results are produced. After all, the sole motive of creating the OPCC is to create a terminal program to provide a command prompt so as to aid users to execute commands on it and enable him/her to use the resources available to him/her to the maximum. So, the task of the Registry section of the OPCC is the most important one from the user's point of view because it is the one that will enable the user to access resources and execute desired commands.

So, we see that the OPCC is a special terminal emulating program enabling the users to use the exclusive omniPresence command prompt and get the actions done.

A view of the OPCC can be shown diagrammatically as follows:



OPCC

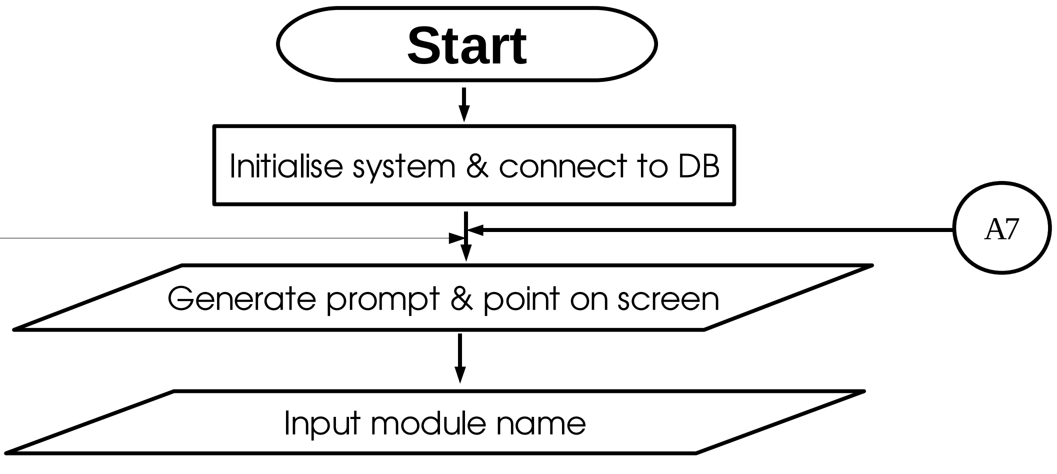
The above is a pictorial representation of the structure of the omnipresence Command Central. We may also depict the functioning of the OPCC in the form of a diagram in the following manner:

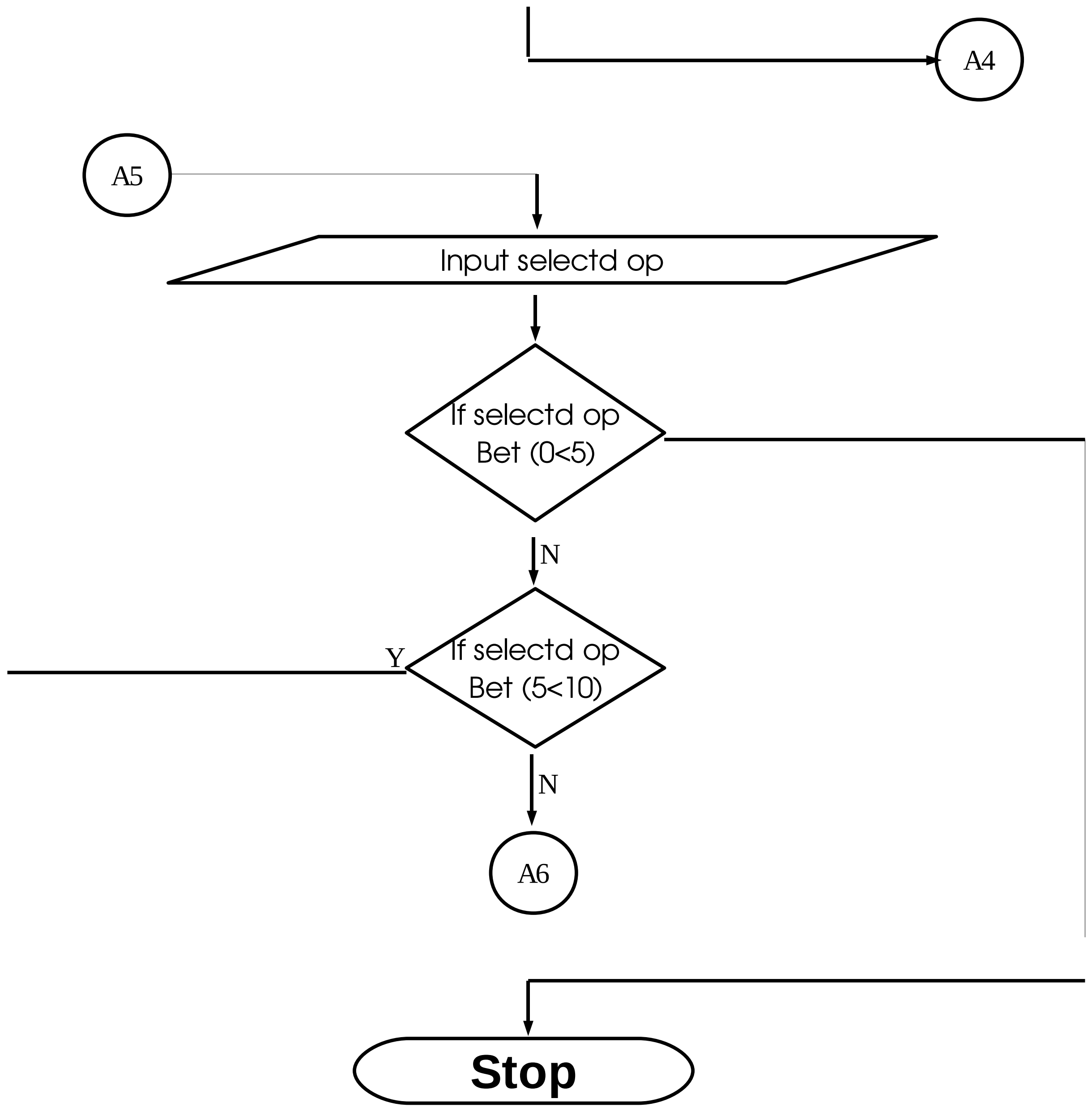
USER ----->

The following is a flowchart representing the OPCC functionings and this is followed by the complexity metrics.

Flowchart for OPCC module

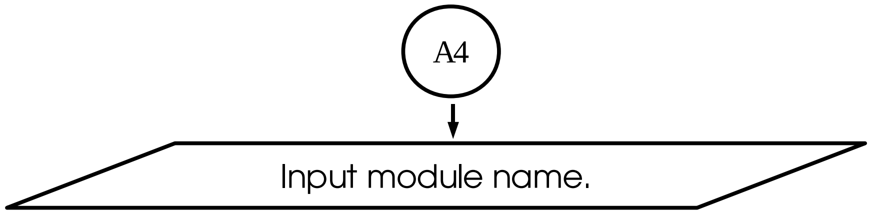
(This module is responsible for controlling different phases and some special features of omniPresence.)

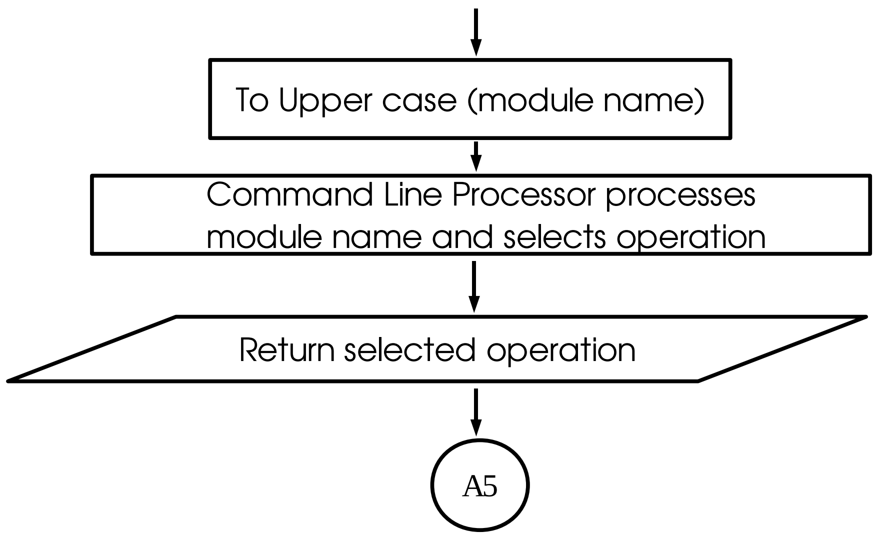

Start
Initialise system & connect to DB
A7
Generate prompt & point on screen
Input module name



Command line Processor Module

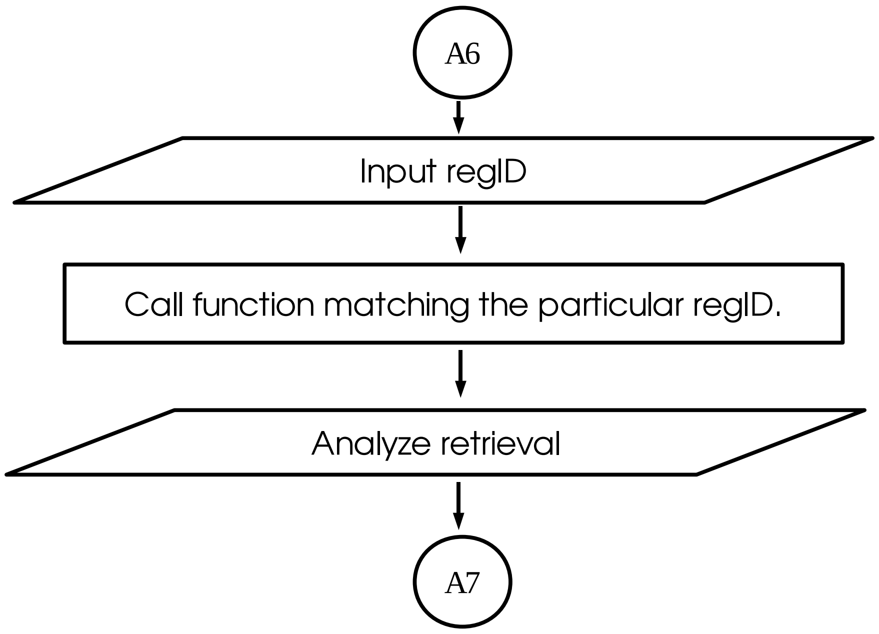
(accepts input from the user and identifies if they are applicable to OPCC or not)




To Upper case (module name)
Command Line Processor processes module name and selects operation
Return selected operation
A5

Registry Module

(brings into action those functions which the user wants to process)



Algorithm for omniPresence Command Central (O.P.C.C.)

```
//INCLUDE SYSTEM COMPONENTS (LIKE C HEADERS)
//INCLUDE CUSTOM COMPONENTS (LIKE P1, P2)
//GLOBAL VARIABLE DECLARATION
//INCLUDE OPCC COMPONENTS
```

```
ALGO OPCC
{       INITIALISE GRAPHICAL SYSTEM
        GENERATE PROMPT AND PARSE VALUES
        LOOP WHILE shutdownOPCC NOT 1
                GENERATE PROMPT AND TAKE INPUT
                SEND INPUT TO COMMAND LINE PROCESSOR
                ANALYSE OUTPUT RETURNED FROM COMMAND LINE PROCESSOR
                        IF (SELECTED OPERATION)
                                SEND TO REGISTRY MODULE
                        ELSE
                                CONTINUE OR EXIT
                        END IF
        END LOOP
        COLLECT AND SHUTDOWN SYSTEM
}

ALGO commandLineProcessor
{       //INPUT
                COMMANDLINE ASCII
        //OUTPUT
                SELECTED OPERATION

        //WORKING
                CONVERT COMMANDLINE TO UPPERCASE
                PERFORM CHECK OF THE COMMANDLINE FROM THE EXISTING FUNCTION LIBRARY
                IF (FOUND)
                        RETURN THE CORRESPONDING SELECTION CODE
                ELSE
                        RETURN NOT FOUND

}
```

Phase 1:

Before we begin with the actual happenings of the project, it is essential that we understand the enormity of the situation that omniPresence caters to. To make the attendance of employees in an organization by Bluetooth we first need to have a mechanism to read up the Bluetooth signals from the employees' devices. The MAC address of ea device needs to be extracted and used for marking of the attendance. Databases need to be maintained which would store the initial feed of data. We also need to maintain relations having details of all MAC addresses detected in a day and finally a table as a final attendance marking place. Security and the need of future reference necessitate the use of different tales. What we must realize is that all this must be done for all the employees year after year without any flaw because attendance is a very important part of making an organization work successfully. This is not an easy task.

Each device, when located, has a structure called enquiry_info associated with it. The structure is as follows:

```
Struct inquiry_info

{

        ….

        …

        long bdaddr;

        ….

        …

}
```

There are quite a number of other elements of enquiry_info which are, however, not mentioned here. This has been done with the sole motive of reducing complexity. The major element is bdaddr which, as is suggested by the name, gives details about the bluetooth device address.

We need to maintain a mechanism to detect and record all the bluetooth devices which are within the range of detection. In order to do so, we have a linked list of type enquiry_info. In this list, one device has a specific node which points to the node specific of the next detected device. In this way one device, after being recorded will lead to the next and so on. As soon as a new bluetooth device is located, a new node is created and added to this list. If we have to devise an algorithm for the same, it would be based on something like the following:

```
If device

{

        Create new node of type inquiry_info;

}
```

Let us take a look into the other necessary details. As we have already seen, we are primarily using HCI for bluetooth detection. The question which arises now is how exactly bluetooth device discovery occurs. HCI makes use of a function called hci_inquiry. hci_inquiry performs a Bluetooth device discovery and returns a list of detected devices and some basic information about them in the variable ii. On error, it returns -1 and sets errno accordingly:

```
        int hci_inquiry (int dev_id, int len, int max_resp, const uint8_t*lap,
        flags);
        inquiry_info **ii;
```

hci_inquiry is one of those few functions that requires the use of a resource number instead of an open socket, so we use the dev_id returned by the hci_get_route. The inquiry lasts for at most 1.28*len seconds and at most max_resp devices will be returned in the output parameter ii, which must be large enough to accommodate max_resp results. We have used a max_resp of at most 255 for a standard 10.24 second enquiry.

Real Timer:

As already mentioned, security is an aspect which has been heavily emphasized on throughout. The real timer is a concept which has been devised for the same.

One can note that the system time can be easily manipulated and a person coming in late can record the detection of his bluetooth device at a time much earlier than what is actual. In order to prevent such a scenario, the real timer comes into play. It interacts directly with the hardware clock, which cannot be manipulated with. When a device is located, or rather detected, the time of detection is matched with the hardware clock. The pre-defined time of "timely arrival" has already been set. If the hardware clock has passed that time, the bluetooth device is not registered. So, a late arriving employee cannot, by any means, get his attendance done.

In Phase 1, before we take a look at all the tables that we have used, we must make a brief tour of MySQL, the relational database management system (RDBMS) that we have used.

To try out MySQL successfully we need to satisfy a few preliminary requirements:

- You need to have the MySQL software installed. (with the WAMP or EasyPHP)
- You need to have a MySQL account so that you can connect to the server.
- You need a database to work with.

The required software includes the MySQL clients and a MySQL server. The client programs must be located on the machine where you'll be working. The server can be located on your machine, although it is not necessary. As long as you have permission to connect to it, the server can be located anywhere. In addition to the MySQL software, you'll need a MySQL account so that the server will allow you to connect and create your sample database and tables.

Following are the tables that are used and which form the base of omniPresence Phase1:

- scanDump: This is the initial table where all the data read initially at the first run of the system will be fed.

It will have the following attributes:

1. sL of type int(11). It will be the serial number of the device being detected. It will be assigned not null with the autoincrement facility on. It will be the Primary Key.
2. mAC of type varchar(19). It will have the blueooth mAC addresses of the devices in question.
3. time of type varchar(8). It will have the time at which the specefied bluetooth device was detected.

- lastDumpSession: This table is meant to be an intermediate table. It holds special importance because it has the function of refining the data collected by the scanDump table. The data collected by the scanDump table in one session is dumped into this table. Further processing is done by the data from this table.

It will have the following attributes:

1. date of type varchar(25). It will have the date of the particular day that the system runs.
2. time of type varchar(10). It will have the time at which the bluetooth device is detected.
3. sL of type varchar(11). It will be the same as scanDump. The serial number of the last device in one session holds importance because all devices whose serial number is less than equal to this number are "dumped" to this attribute of the lastDumpSession.

The utility of this table will be much more vivid and appealing in the later stages of this project.

If we have a look at the user interface, we will use GTK, as already mentioned. If we were to take a closer look at GTK, it is almost like Visual Basic in appearance of output. The coding part of GTK is done only after including the gtk/gtk.h header file. Since the coding in done entirely in C, the regular C header files are also included.

Coming to the interface part again, the objects that are to be present in the interface are called widgets. There are numerous widgets defined and available for use. One can also generate their own widgets and compose widgets from combinations of other widgets.

Examples:

- Push Buttons
- Menu Widget
- Radio Button
- Check Button
- Scroll Bar
- Drawing Area
- Tree View
- Text View
- Text Entry Box

Signals can then be associated with a widget so that an action (i.e. button release) on a Button widget will invoke a callback function to be executed and perform the desired task (i.e. process or display something or both).

Due to some peculiarities of the programming in GTK, we can use only one widget within a window. This limitation is dealt with by using a technique called packaging. Packaging is a technique in which one widget incorporates a number of other widgets within itself and the whole "package" is used as one on the window.

The most popular widget packing technique is the use of box. This is a widget which is used to pack other widgets within itself like a box does. This facility is analogous to the concept of frames in Visual Basic.

To create the user interface, we first must create a user login prompt. This prompt will be made in GTK by using a combination a hboxes and vboxes within a single window. We will be using four hboxes, the first one having a label `User Name' and an entry for the same, the second one having a label `Password' and an entry for the same, the third one containing just a button `Submit' and the fourth `Quit'. The alignment of the widgets one below the other is done inside one vbox.

The user login is thus made completely using the packaging facility of GTK. Only when the user logs in with the correct password, he can use the facilities of omniPresence.

Let us take a look into the various functions involved in using boxes.

Boxes are created using the following function:

gtk_hbox_new() for horizontal boxes and gtk_vbox_new() for vertical boxes. gtk_box_pack_start() and gtk_box_pack_end() are used t place objects inside boxes.

As we have seen, in Phase 1 we need to execute two processes, the first being the process of device discovery via the Bluetooth dongle and the second being the process of running the real timer. Now, we have two options of carrying out Phase 1. We can either run the two processes mentioned above one after the other. However, this is a very obsolete way of programming, which brings us to the second option that we can use.

Before thinking about what our second option is, we need to think about what our needs are. We need to think of a way which can enable us to use both the processes in a concurrent manner. For this we can use the multitasking facility of the operating system. But, in order to make omniPresence available to everyone and to make it more flexible in general, we have used multithreading to incorporate the concurrent execution of both processes of Phase 1.

Now, if we take a closer look at how this multithreading has been implemented, here is what we find. The two processes of Phase 1, i.e, device detection and real timer need to be executed concurrently. The concept of multithreading has been incorporated at the command prompt, from where the whole process or rather work flow of omniPresence will be carried out. A special point to note will be that omniPresence uses a special command prompt, designed specifically for its use. The name of this command prompt is omniPresence Command Central (OPCC). The details of OPCC will be discussed in the following pages. It will be the responsibility of the OPCC to handle the concurrent execution of both the processes.

Another point to note, as is evident from the work flow of Phase 1, is that the processes of device discovery and real timer need to begin together as well as end together. The concept of Mutex has been applied here for the same. It is a facility that ensures complete serializability of the system. No other process will attempt to or rather can use the resources of Phase 1 when it is in execution. It is almost analogous to the system of locking very common in databases. Also, another variable called shutdownRealtime has been introduced. It ensures that as soon as the process of device discovery ends, the real timer will also stop. In other words, as soon as one session of device discovery ends, the variable, shutdownRealtime is triggered to stop the real timer, almost immediately.

Thus, we see that by incorporating the concept of multithreading, Phase 1 has been made better and much more efficient.

Phase 2:

Phase 2 is concerned with functions that will be applicable to the whole system at large. We need to make the whole system efficient and easy to use for the user. The same goes for the operator as well. It will be better for the organization if the operator is at ease. Following are the functions which have been included in Phase 2 to make the whole system harmonious:

- Create_table: This function serves the unique function of creating the attendance sheet table for each month. This is necessary because the attendance sheet table will have one unique identifier (rNad) and 31`attributes for each day, plus one attribute more for the total. To eliminate the task of creating the table each month, this function is introduced to automate the same.
- Prepare: As discussed earlier, holidays and Sundays need to be crossed off from the attendance sheet table. This function is responsible for the same.
- Sanction_leave: This function is used to mark an employee on the attendance sheet if he is on official leave.

Many other functions have been introduced which will be clear during the demo of the project. Phase 2 also incorporates the facility of querying the database for any information that is required from the same.
rNad is a new variable that has been encountered here. It is nothing but a unique random number that is generated for each employee, unknowingly to him/her. It has basically been done for security reasons. It's use will be clear in the later phase of this project.

Another aspect that needs to be looked at is the command prompt designed specially for this project. It has been christened "OmniPresence Command Central", or just OPCC. It is here at the OPCC that all the functions associated with it will be run and made use of. Apart from that querying the database will also be done from here itself.

Now, let us take a look into some of the important functions that have been used in Phase 2 of omniPresence. We will discuss these functions by having a look at a brief description of the functions, followed by their flowcharts, algorithms and the complexity metrics.

Now, we need to understand how the functions of Phase 2 work. Here we have selected a few functions of Phase 2 and explained the basics of their functioning along with the corresponding flowcharts and complexity metrics. This is just so that the user of omniPresence can get a fair picture of the usability levels of the software. Following are the functions or rather modules:

1. User Login module:

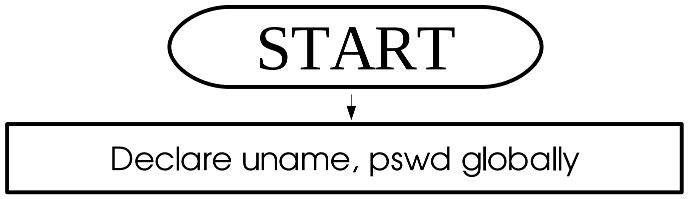
   This module has been designed specifically so that the user can login to the system by giving the correct user name and password. It is responsible for generating the window for the login.

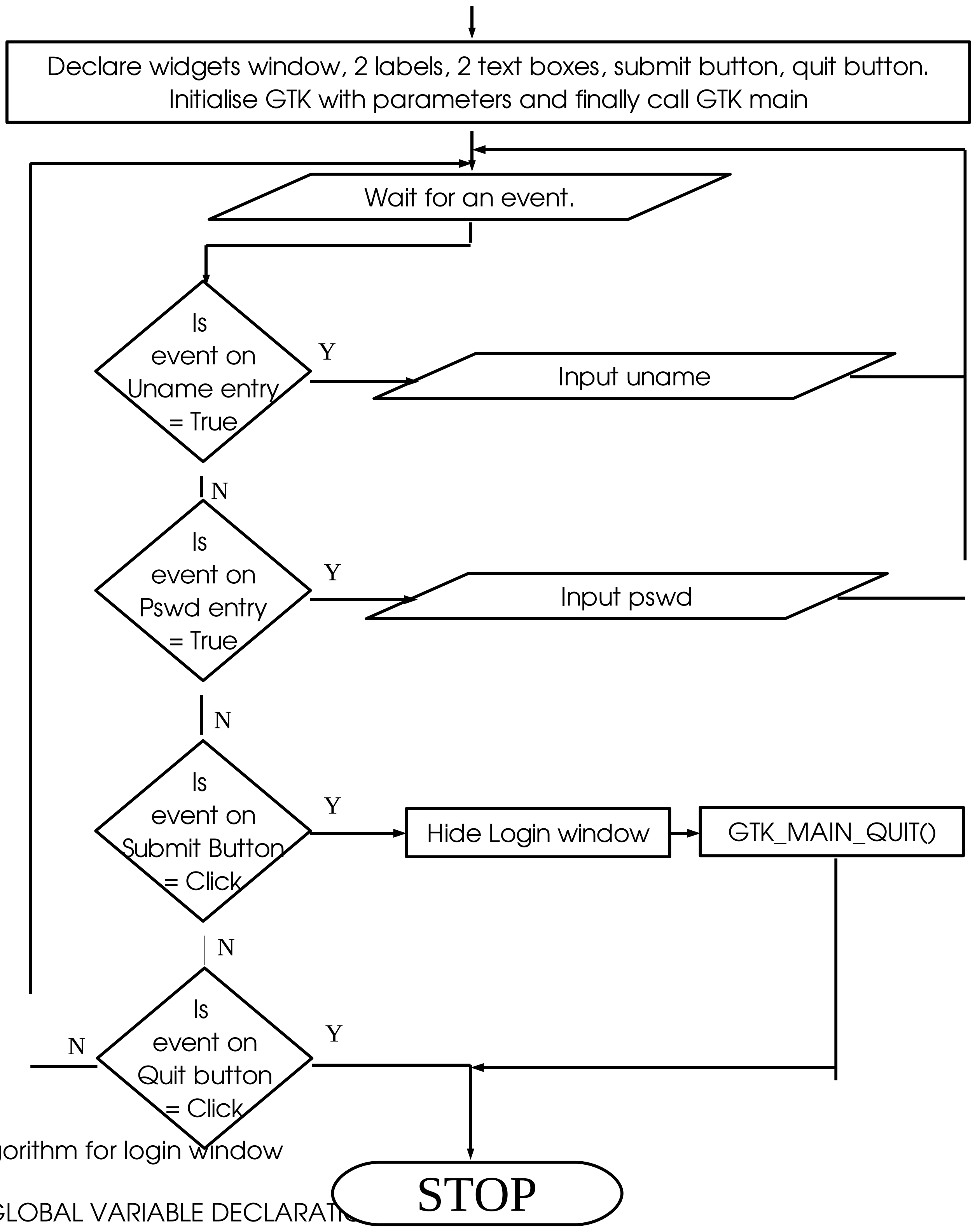
   If we come to the explanation of the module, here the user name and password are declared globally. The text that the user enters for the user name and the password in the respective text boxes in the user prompt window are copied to the corresponding variables. Then the validity of these entries by the user is checked. If all is well, the user is allowed access to the system resources. The user prompt window is generated by the help of GTK using windows and widgets as mentioned in the introduction part.

   Following is the flowchart, the algorithm and the complexity metrics for the user login module:

Flowchart for login module

(This module is responsible for starting graphical window for user login)




Declare widgets window, 2 labels, 2 text boxes, submit button, quit button.
Initialise GTK with parameters and finally call GTK main
Wait for an event.
Is event on Uname entry = True
Y
Input uname
N
Is event on Pswd entry = True
Y
Input pswd
N
Is event on Submit Button = Click
Y
Hide Login window
GTK_MAIN_QUIT()
N
Is event on Quit button = Click
N
Y
STOP

Algorithm for login window

//GLOBAL VARIABLE DECLARATIO

static char uname(14)="",pswd(6)="";

```
const char *uname_entry,*pswd_entry;
int login_retcode = 0;

//GLOBAL FUNCTION DECLARATIONS

submit_uname()
{       GET TEXT FROM THE TEXTBOX AND COPY IT TO THE GLOBAL UNAME VARIABLE
}

submit_pswd()
{       GET TEXT FROM THE TEXTBOX AND COPY IT TO THE GLOBAL PSWD VARIABLE
}

submit()
{       CHECK THE VALIDITY OF THE UNAME AND PASS VAR AND QUIT.
}

exiting_login()
{       QUITTING THE LOGIN WINDOW
}

void login_window_content()
{       DECLARE THE WIDGETS
        CREATE WINDOW
        CONNECT THE WIDGETS WITH SIGNALS
        FORMAT THE WIDGETS IN THE WINDOW AND ADD CONTAINERS
        SHOW WINDOW
}

//MAIN FUNCTION STARTS HERE

int login_window(int argc,char *argv())
{       CALL WINDOW CONTENT AND CALL GTK INIT.
        CALL LOGIN WINDOW CONTENT
        GTK_MAIN();
}
```

Space Complexity

h/w stack

2. Sanction Leave module:

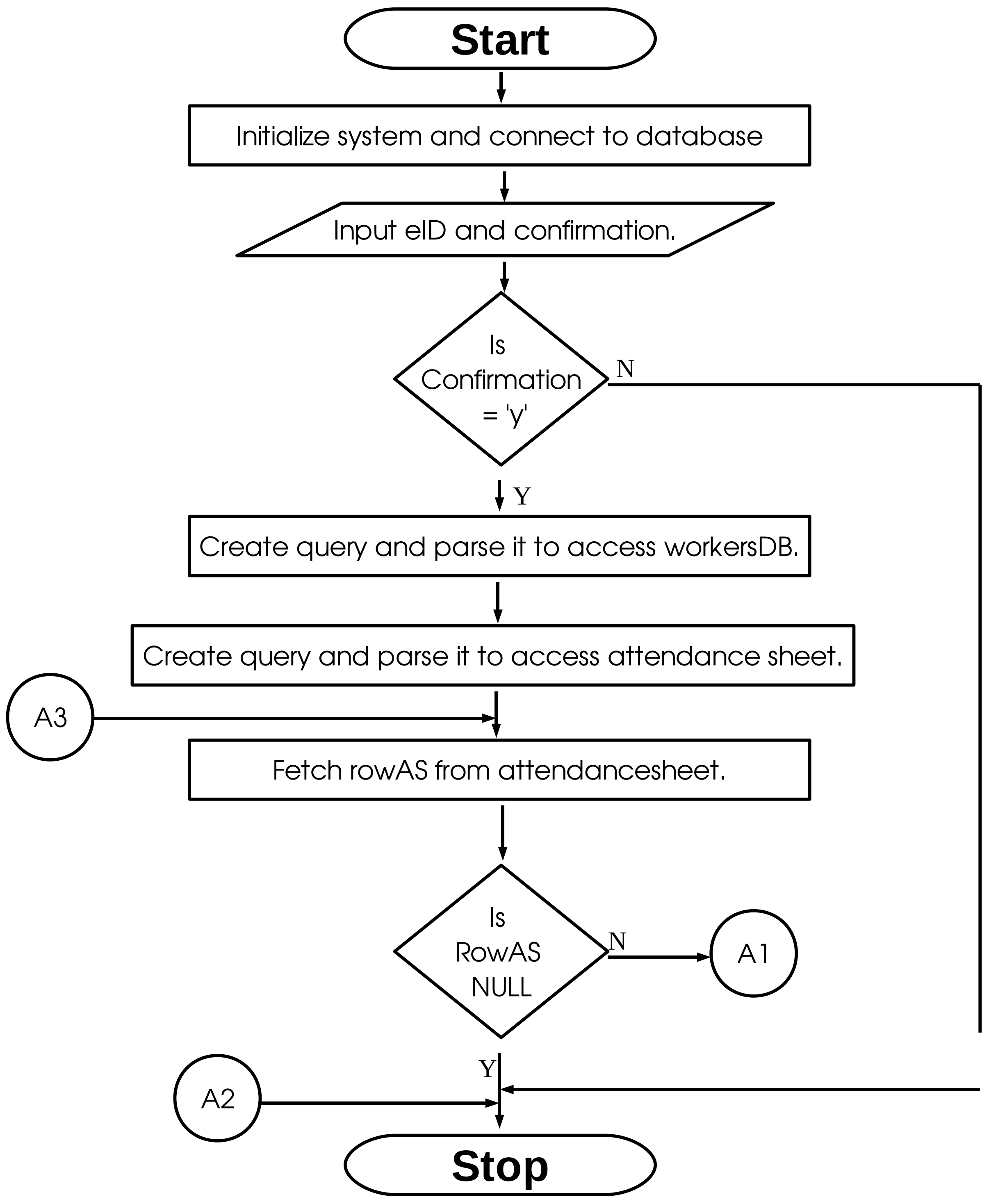
This module is responsible for making arrangements in the system if any employee of the organization has been granted an official leave. This means that for a fixed number of days starting from a particular date, the employee has been granted leave by the officials and he/she is not to be marked absent in the attendance records.

The module for this operation is as follows. Initially we need to connect to the database and input the employee ID for whom the leave is to be granted. After a simple process of confirmation, queries are created and parsed to access the workersDB and attendance sheet databases in the same order. For the rNAD of the empID, the corresponding workersDB entry is fetched. The employee's name is shown to the user for confirmation. After the name is confirmed, the number of leaves and the date for starting the leave is asked for. The query for marking leaves is created and injected into the system.

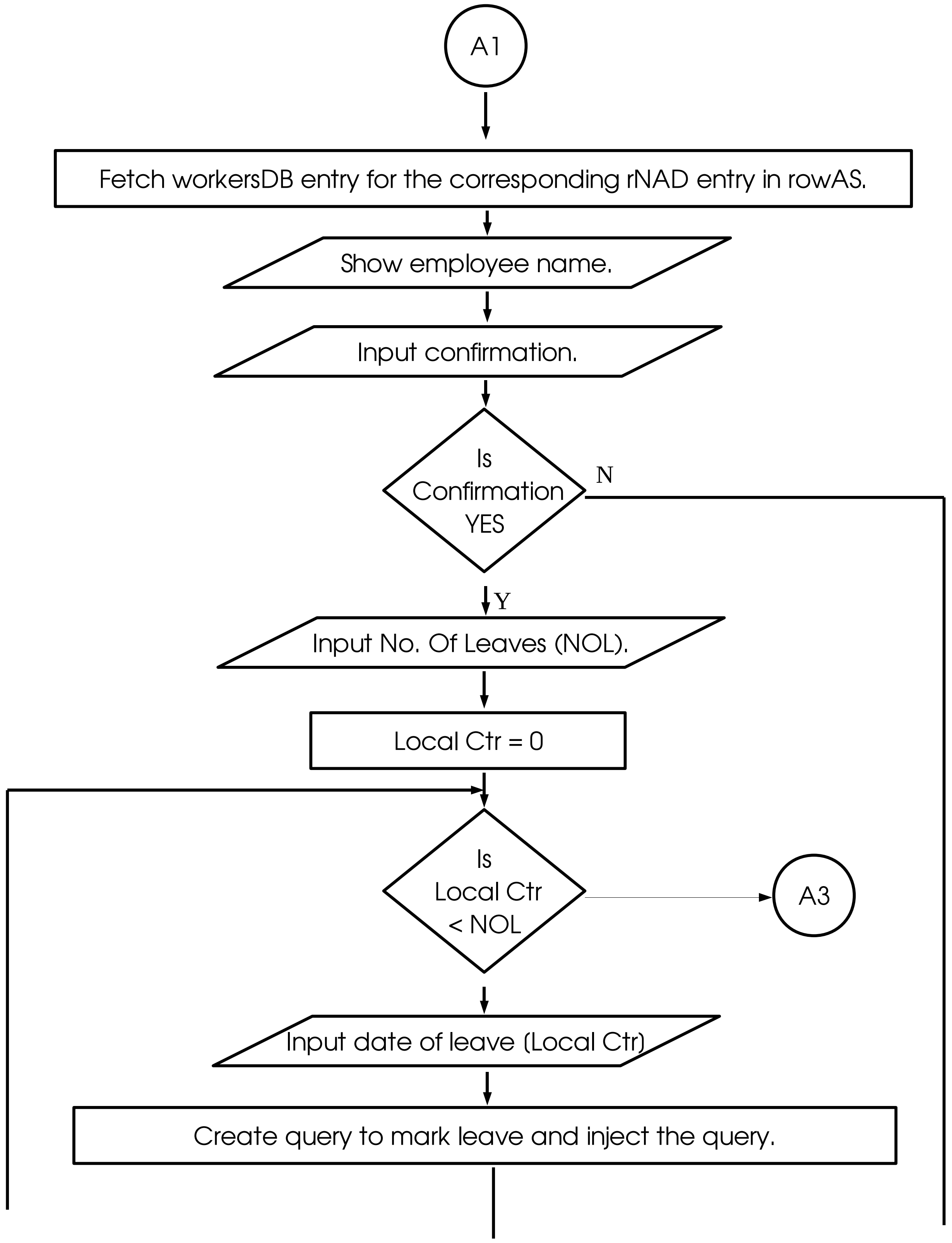
The flowchart, algorithm and complexity metrics for the sanction leave module is as shown below:

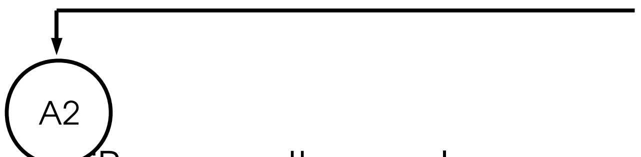
Flowchart for Sanction_leave Module

(This module is responsible for sanctioning the leaves for a particular employee for a given month)


Start
Initialize system and connect to database
Input eID and confirmation.
Is Confirmation = 'y'
N
Y
Create query and parse it to access workersDB.
Create query and parse it to access attendance sheet.
A3
Fetch rowAS from attendancesheet.
Is RowAS NULL
N
A1
Y
A2
Stop

Sanction_leave Module (Continued)


A1
Fetch workersDB entry for the corresponding rNAD entry in rowAS.
Show employee name.
Input confirmation.
Is Confirmation YES
N
Y
Input No. Of Leaves (NOL).
Local Ctr = 0
Is Local Ctr < NOL
A3
Input date of leave (Local Ctr)
Create query to mark leave and inject the query.



3. rnadgen module:

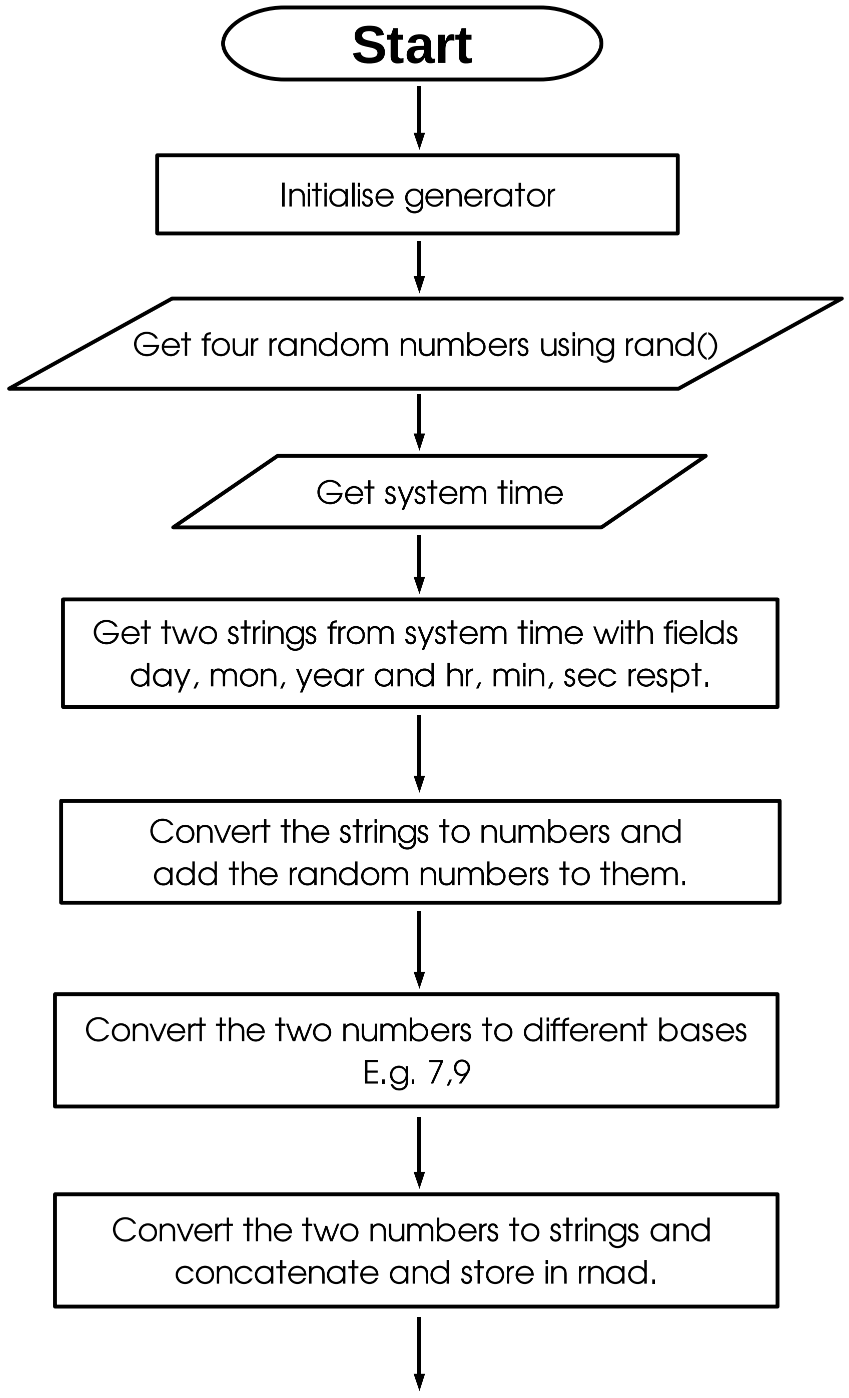
In the system omniPresence, the employees are assigned unique random numbers which are denoted by rNAD. This is done basically for security purposes. Security is enforced by the fact that no one can possibly guess what the rNAD of a particular employee is, it being system generated and not user generated. The module rnadgen is responsible for generating the rNAD of each employee in the organization.

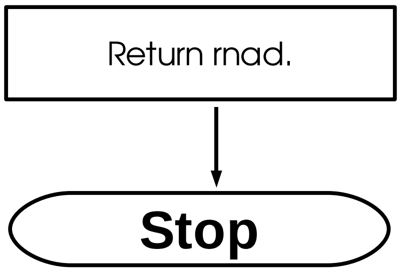
rnadgen has the following work flow. As the first step, the random number generator is initialized. Then four random numbers are generated using the rand() function. Then the system time is retrieved. Then two strings are obtained with fields day, month, year and hour, minute, second respectively. These strings are converted to numbers and are added to the random numbers. The two numbers thus obtained are then converted to a different number base system. These two numbers are then concatenated and stored in the variable rNAD which is then associated (different value on each run) with each employee.

The flowchart, algorithm and complexity metrics for the rnadgen module are as follows:

Flowchart for rnadgen module

(This module is responsible for generation of the rNAD which is associated with each worker in the workersDB.)


Start
Initialise generator
Get four random numbers using rand()
Get system time
Get two strings from system time with fields day, mon, year and hr, min, sec respt.
Convert the strings to numbers and add the random numbers to them.
Convert the two numbers to different bases E.g. 7,9
Convert the two numbers to strings and concatenate and store in rnad.



4. Search Module:

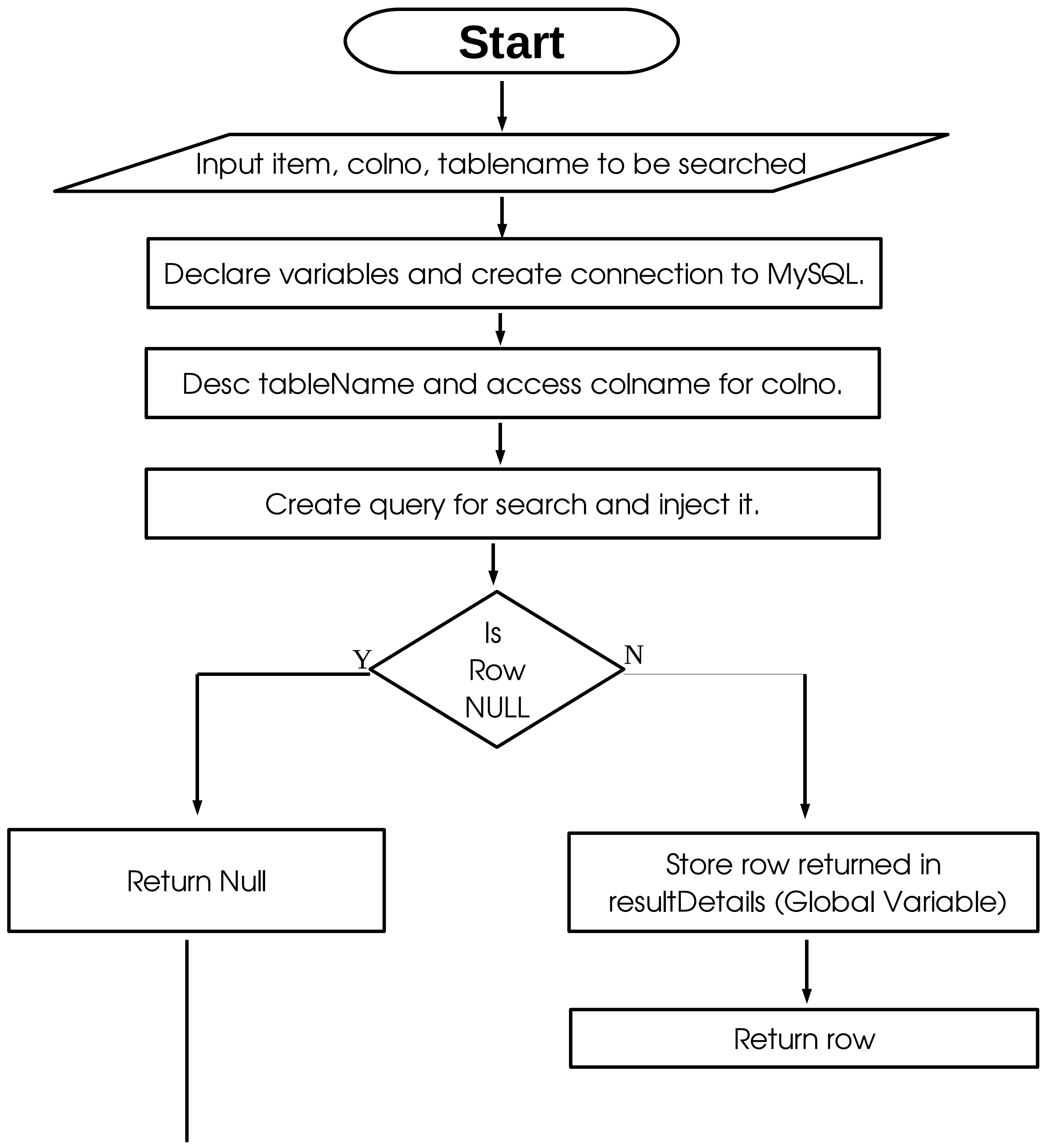
The search module is responsible for finding employees for marking their attendance. It is a supplement of Phase 3 and is responsible for searching among tables for any reason, as the need may be. The search module may be used to search for information regarding the attendance status of an employee, giving details about which days he/she is absent and what his/her attendance stands at.

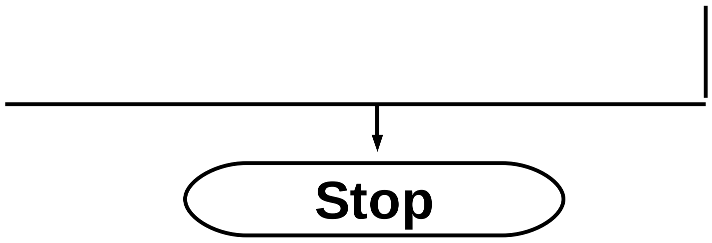
If we take a look at how the search module works, we find the following. As the first step, we input the item, the column number and the table name to be searched. A connection with the database management system, in this case MySQL is established. Next, the query for search is created and injected. The search module then fulfills the responsibility of searching the specified table for the item. Now, if the results of the search module is NULL, then NULL is returned. However, if the search module gets some results, that is a row is found as a result of the search, then the row is stored in the global variable, resultDetails. This is then returned as a result of the search module.

The following encompasses the flowchart, the algorithm and the complexity metrics of the search module:

Flowchart of search module

(Supplement of phase3 module which is responsible for finding the employees for marking attendances to attendance sheet)


Start
Input item, colno, tablename to be searched
Declare variables and create connection to MySQL.
Desc tableName and access colname for colno.
Create query for search and inject it.
Is Row NULL
Y
N
Return Null
Store row returned in resultDetails (Global Variable)
Return row



<u>Sheet creator module:</u>

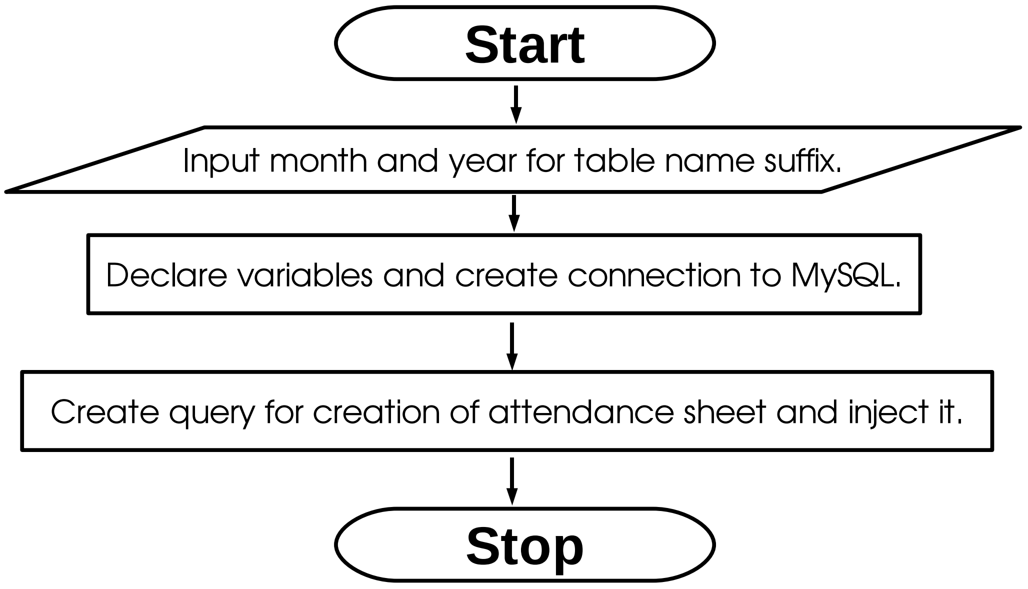
The sheet creator module is a special one because it has been specifically built to aid the user of the software to maintain correct and accurate attendance records without much hassles. The sheet creator is responsible for carrying out the task of creating the tables of attendance for employees each month. This feature is an added bonus because the attendance sheet database for each month is a massive one and would be extremely tedious for the user to create the table over and over again.

If we look into how this function works, we find the following. We initially declare the month and year as the table name suffix. This is done as follows. For instance, if we want to create the table for July 2009, the suffix of the table name will be 0709, for December 2025 the suffix of will be 1225. Then, the appropriate variables are declared and a connection is established with the database, MySQL is established. The query for the designing of attendance sheets is then created and finally injected.

The following depicts the flowchart, the algorithm and complexity metrics of the sheet creator module:

FlowChart for sheetCreater

(Tool for creating new attendance sheets using OmniPresence Command Central)


Start
Input month and year for table name suffix.
Declare variables and create connection to MySQL.
Create query for creation of attendance sheet and inject it.
Stop

Algorithm for sheetCreater

```
Algo attendanceSheetCreater(char month(4),char year(4))
```

```
{       //INPUT
                MONTH YEAR
        //WORKING
                CREATE QUERY
                "CREATE TABLE ATTENDANCESHEET" + MONTH + YEAR
                " ( ATTEND01 VARCHAR,ATTEND02 VARCHAR"
                …..
                INJECT QUERY AND ANALYSE RESULTS
}
```

User add module:

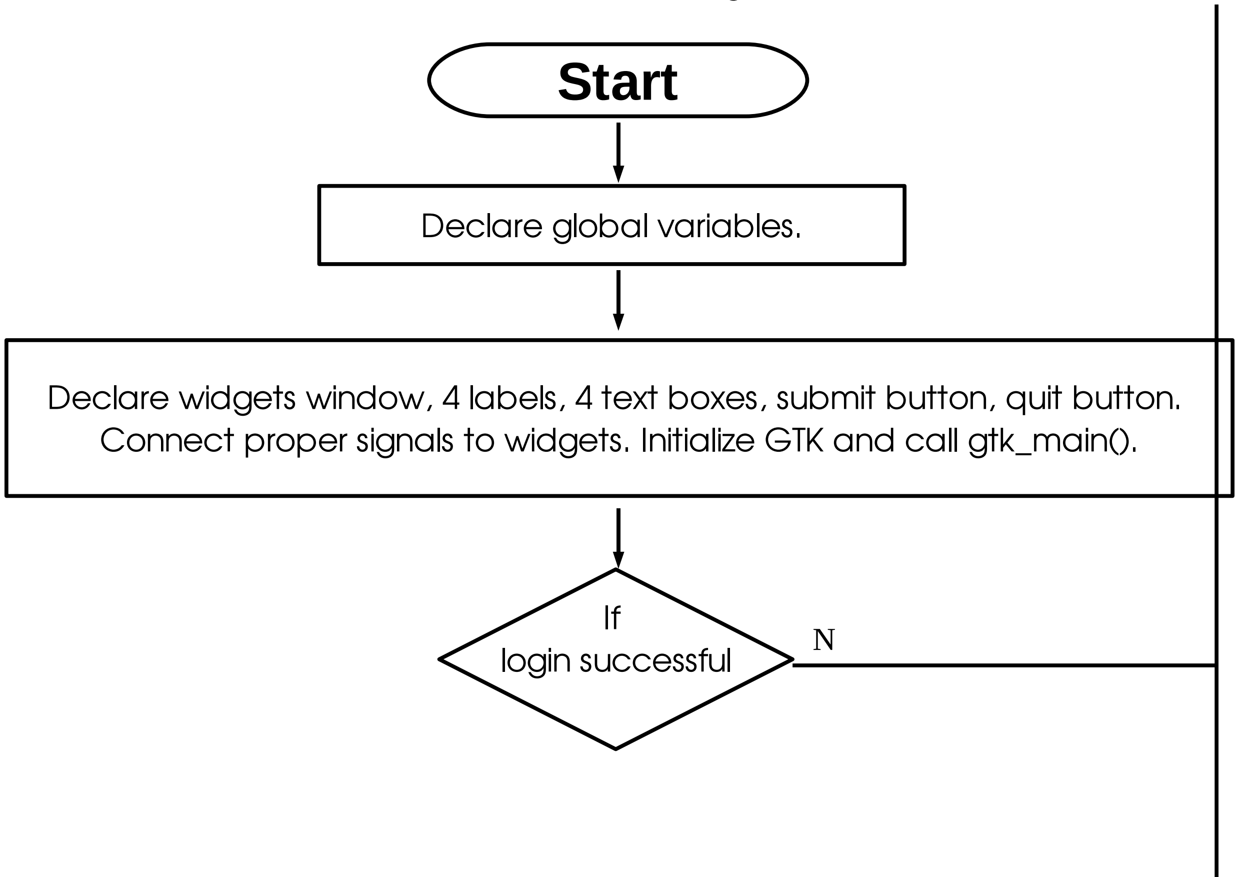
The user add module is an important one because it manages the task of adding new employees to the organization. It is but obvious that the organization will incorporate within itself new employees at some point of time. The module is meant to make amendments in the workersDB module so that the attendance marking functions of omniPresence keep on performing to perfection.

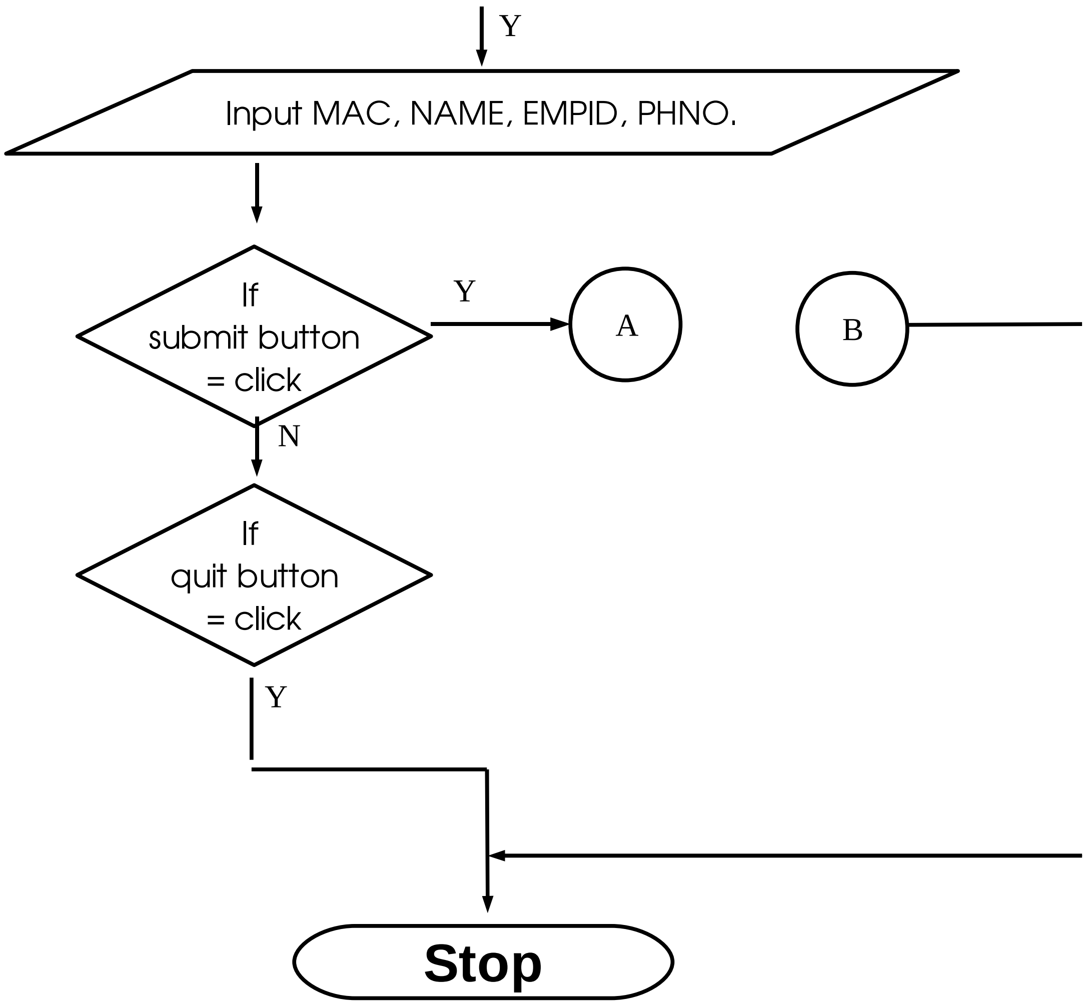
If we delve into how this module works, we find the following. As a first step, the appropriate global variables are declared and using GTK a window is generated for inputting the necessary details of the employees. If the login is successful, the system asks to input the unique Bluetooth MAC address of the employee, his/her name, employee ID and phone number. If submit is clicked, the required variables are declared and a connection is established with MySQL. Then, for the new employee, a unique rNAD is generated using the rnadgen function. Then a query is created for entering all the details of the employee in the workersDB database. Again, a query is generated for entering the rNAD of the employee in the attendance sheet database of the month.

The following depicts the flowchart, the algorithm and the complexity metrics of the user add module:

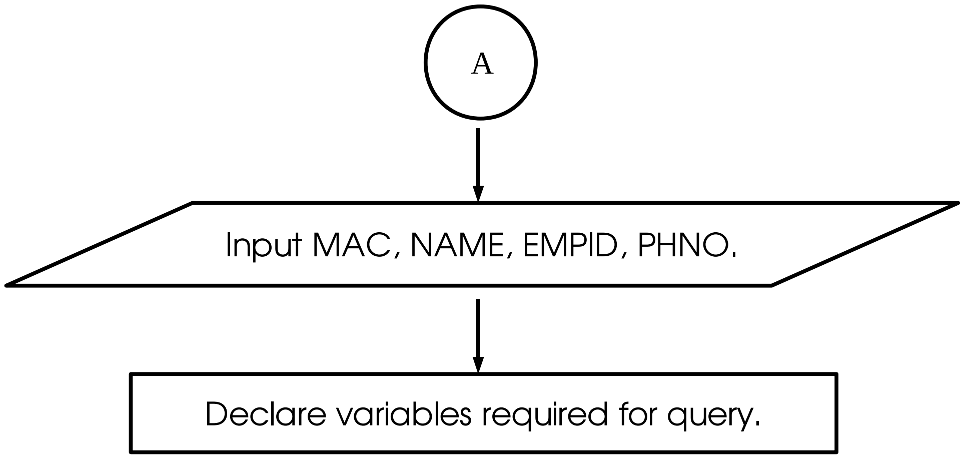
Flowchart for userAdd module

(This module is responsible for making entry in the workersDB.)

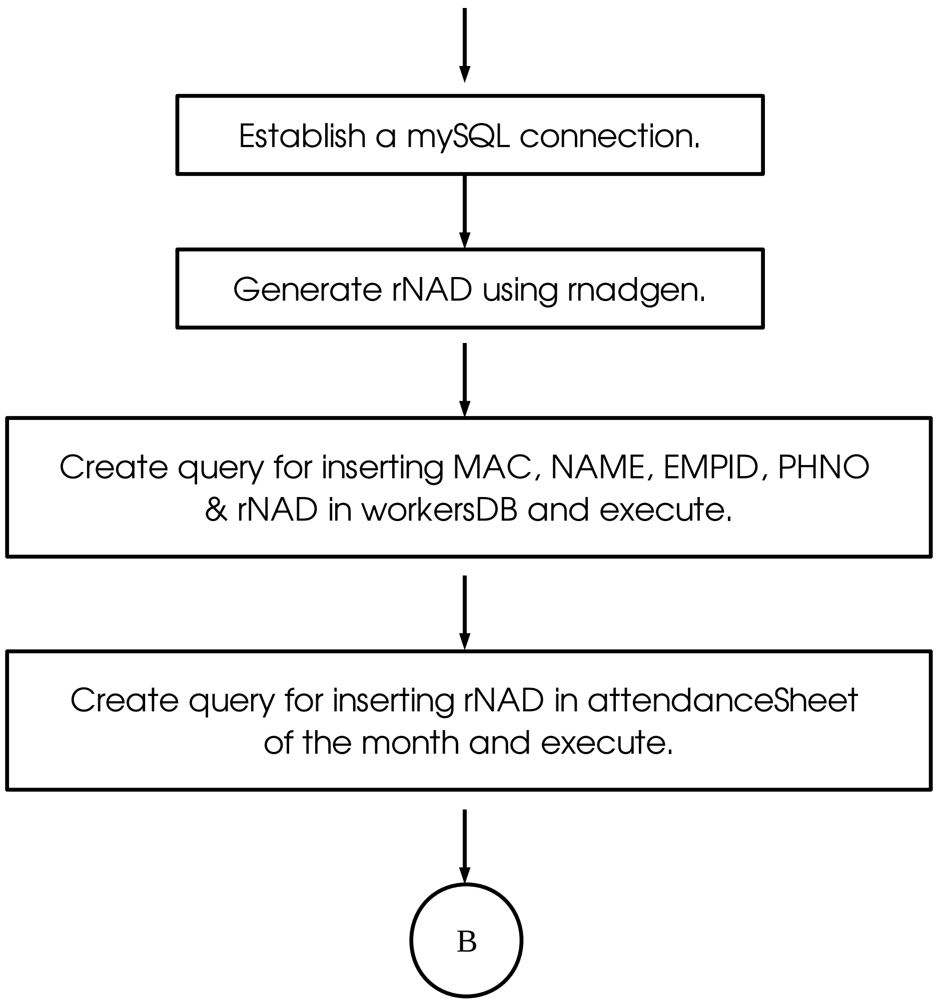

Start
Declare global variables.
Declare widgets window, 4 labels, 4 text boxes, submit button, quit button. Connect proper signals to widgets. Initialize GTK and call gtk_main().
If login successful
N



userAdd module (continued...)



A
Input MAC, NAME, EMPID, PHNO.
Declare variables required for query.


Establish a mySQL connection.
Generate rNAD using rnadgen.
Create query for inserting MAC, NAME, EMPID, PHNO & rNAD in workersDB and execute.
Create query for inserting rNAD in attendanceSheet of the month and execute.
B

Attendance module:

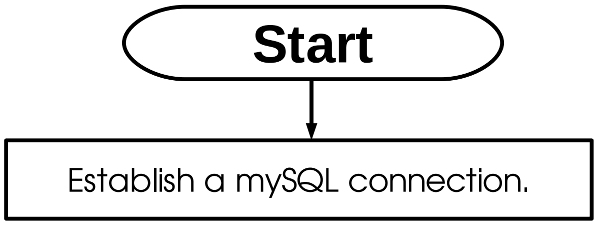
The main objective of omniPresence is to mark attendance of employees in a regulated manner. There needs to be a set of standards which the attendance marking system should follow and attendance should be marked in the same way for all employees at all times. The attendance module undertakes the task of marking attendance of employees in the manner sad above.

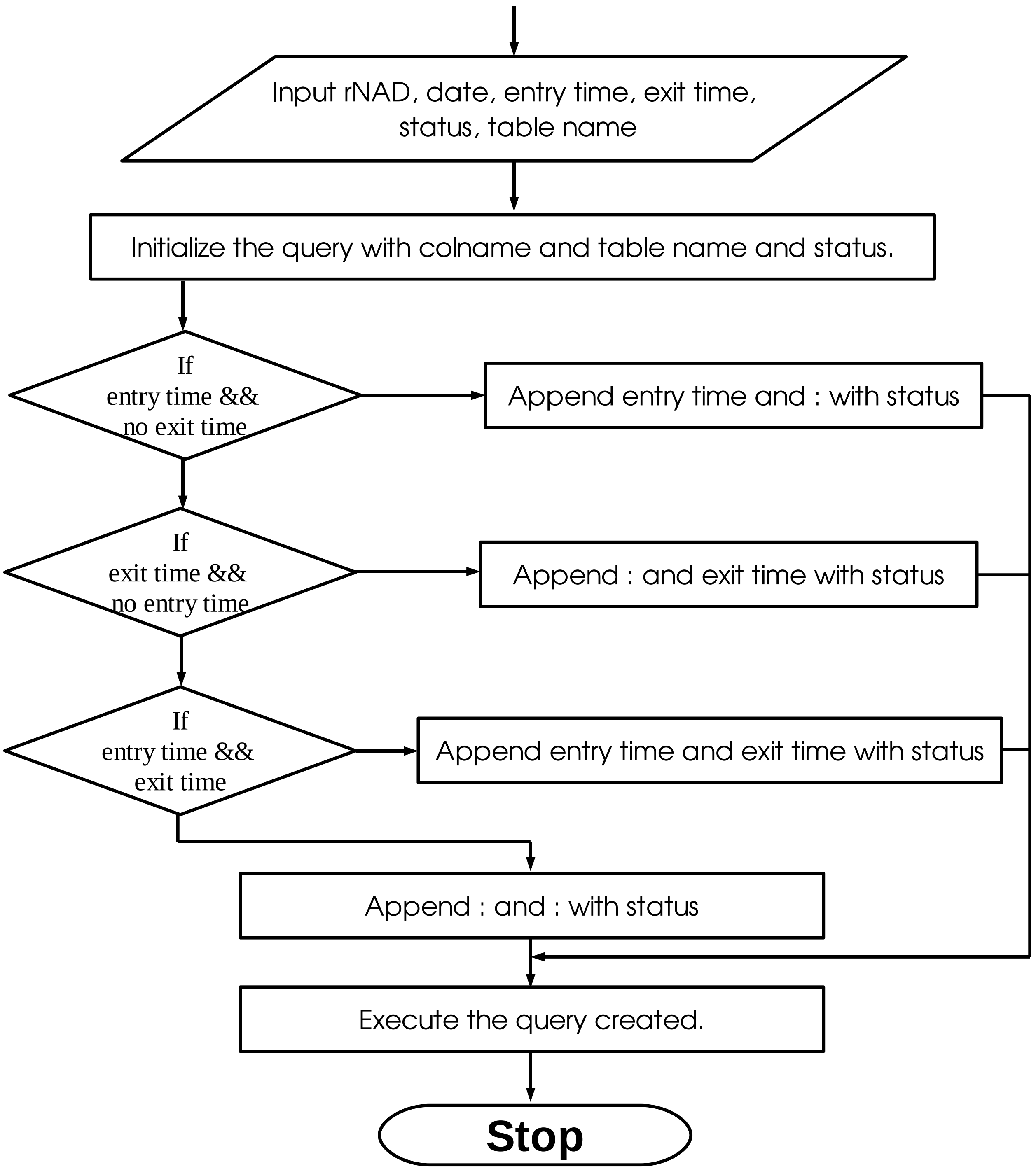
If we delve into the procedural aspect of the module, we find the following. Before understanding how the attendance is marked, we must take a look at the format of marking an employee present. The format is as follows. The present employee is marked 'P:<entry time>:<exit time>'. In the module, firstly a connection is established with MySQL. The rNAD, entry time, exit time, date, status and table name are entered. An appropriate query is initialized with the inputs of entry time, exit time as available. If any field is not available, it can be left blank. After all the requirements are fulfilled, the query is executed to mark the attendance.

The following depicts the flowchart, algorithm and complexity metrics for the attendance module:

Flowchart for attendance module

(This module is responsible for marking attendance.)




Input rNAD, date, entry time, exit time, status, table name
Initialize the query with colname and table name and status.
If entry time && no exit time
Append entry time and : with status
If exit time && no entry time
Append : and exit time with status
If entry time && exit time
Append entry time and exit time with status
Append : and : with status
Execute the query created.
Stop

<u>Insert rNAD module:</u>

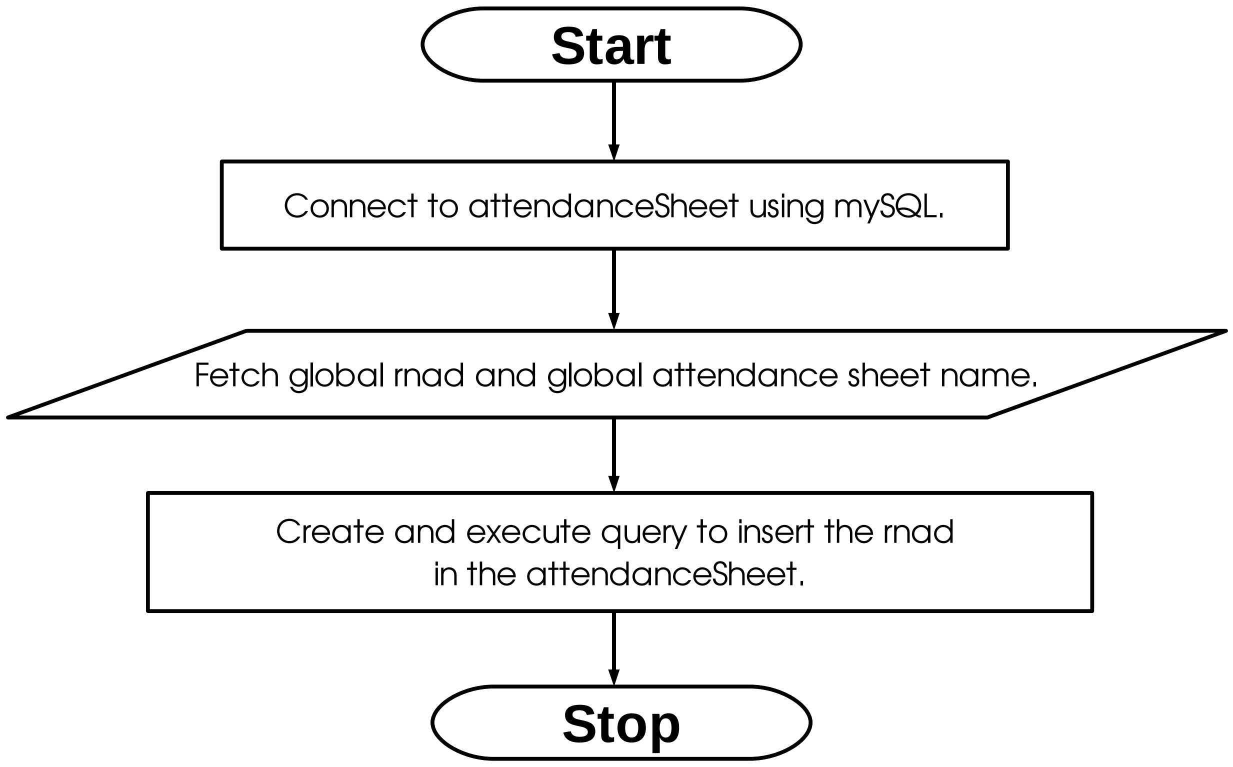
This module is responsible for inserting the rNAD into the attendance sheet database. This is a very important module because in the attendance sheet database, each employee is recognized by the rNAD that is associated to him/her. So, it is of utmost importance that the rNAD that is entered into the attendance sheet table for each month be correct and accurate free of any kind of errors.

If we delve into the what happens in the module, it is quite straightforward and simple. This is how it operates. Initially, as is obvious, a connection is made with the attendance sheet database via MySQL. The global rNAD and the global attendance sheet name (this includes the suffix which has the date and year) are fetched. Finally, a query is executed to insert the rNAD into the attendance sheet.

The following depicts the flowchart, algorithm and complexity metrics for the insert rNAD module:

Flowchart for insertRnad Module

(This module is responsible for inserting rnad to attendanceSheet for new employee.)


Start
Connect to attendanceSheet using mySQL.
Fetch global rnad and global attendance sheet name.
Create and execute query to insert the rnad in the attendanceSheet.
Stop

<u>Guest module:</u>

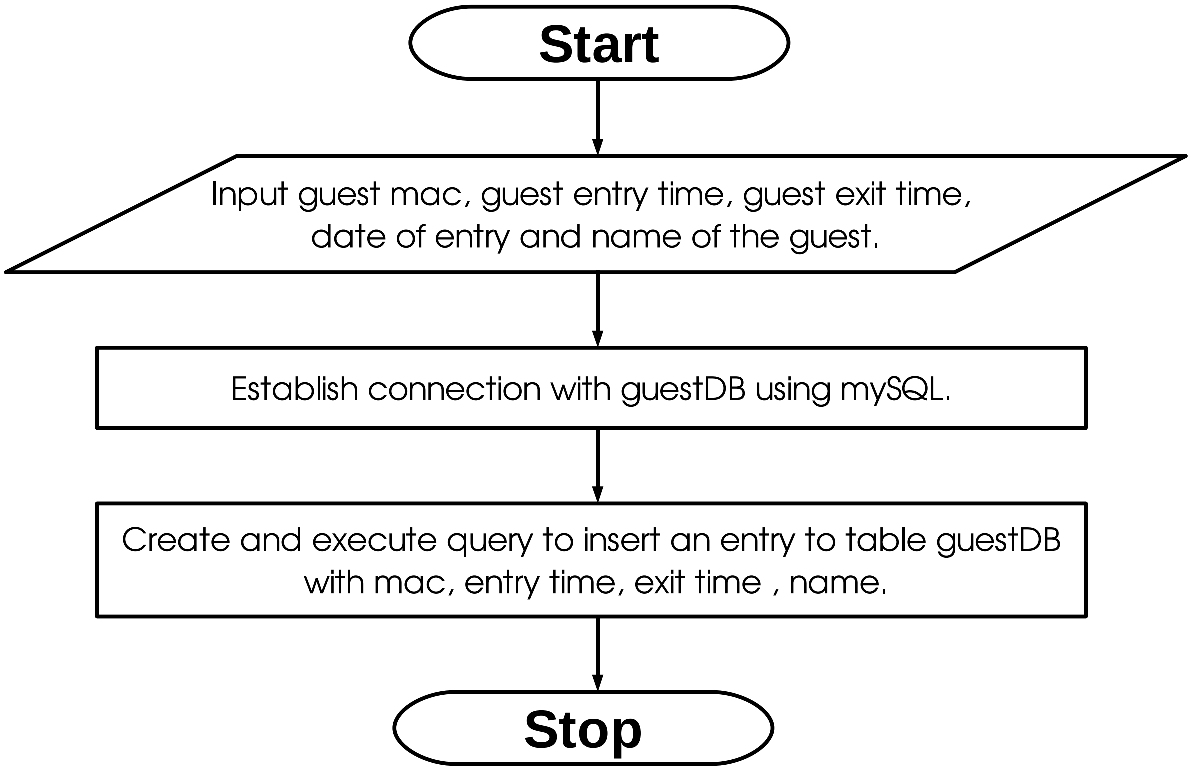
This module is responsible for inserting records in the database meant for guests. It is natural that an organization will have some people entering who are not the employees. In other words, their records won't be mentioned in the workersDB database. So, wen their Bluetooth MAC addresses are received by the system, they won't match with the records in the database. Thus, for the sake of simplicity and to avoid mess ups guests have a separate database alloted to them. This module fills up the GuestDB.

In order to see how this module works, following is a brief description. The module undertakes its task in this way. Firstly, the Guest MAC address, the Guest entry and exit time, the Guest name and the date of detection of the guest acts as input for the system. A connection is established with MySQL and a query is generated for entry of records into the GuestDB.

Following is the flowchart, algorithm and complexity metrics for the Guest module:

Flowchart for guest Module

(This module is responsible for inserting guest data in the guestDB.)


Start
Input guest mac, guest entry time, guest exit time, date of entry and name of the guest.
Establish connection with guestDB using mySQL.
Create and execute query to insert an entry to table guestDB with mac, entry time, exit time , name.
Stop

Sheet prepare module:

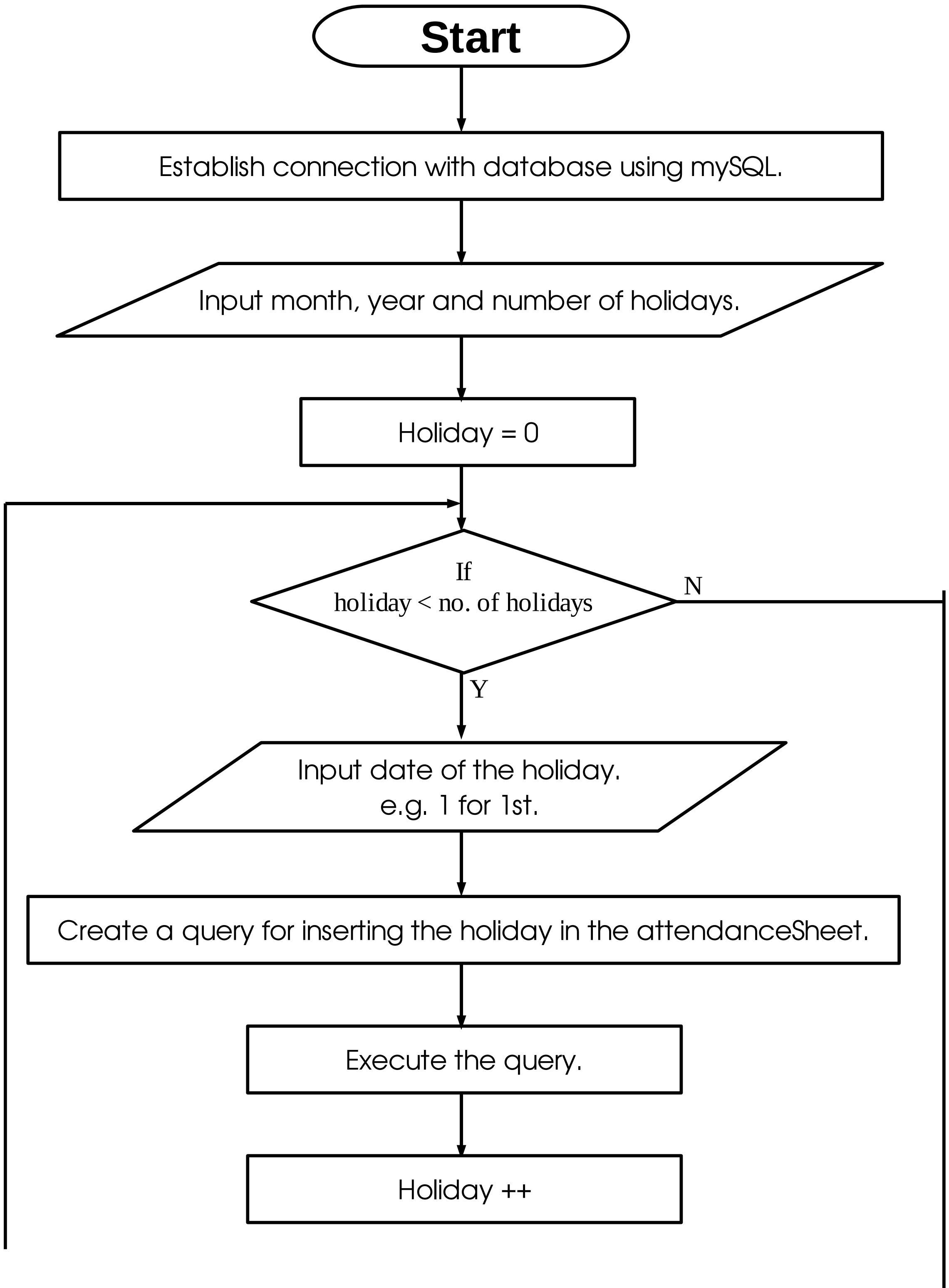
It should be accepted that every month there will be some holidays for whose accomodation the system omniPresence should have measures and means. Even if there are no holidays, there will be at least four Sundays for which all the employees are not expected to come to office. In order to mdify the attendance sheet database accordingly, the sheet prepare module has been incorporated into omniPresence. Basically, the function of this module will be to mark all holidays within each month in the attendance sheet database and ensure that the system doesn't mark all employees absent on a holiday/Sunday.

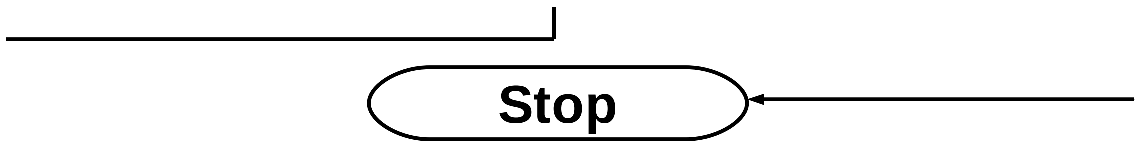
This is how the sheet prepare module operates. As the first step, a connection is established with MySQL. Then, the month, the year and the number of holidays are taken as input. Starting with the first holiday, the date of the holiday is entered into the system. For instance, 1 is entered if the first of any month is a holiday. A query is created to enter the respective date as a holiday. The query is then executed. The date for which the query was executed gets marked with a 'H' indicating a holiday. The process is repeated foe all the holidays that are there within that month.

Following depicts the flowchart, algorithm and complexity metrics of the sheet prepare module:

Flowchart for sheetPrepare

(This module is responsible for the preparation of the holidays in the attendanceSheet for every month.)


Start
Establish connection with database using mySQL.
Input month, year and number of holidays.
Holiday = 0
If holiday < no. of holidays
N
Y
Input date of the holiday. e.g. 1 for 1st.
Create a query for inserting the holiday in the attendanceSheet.
Execute the query.
Holiday ++



Phase 3:

Phase 3 is the part involving the actual marking of attendance of the employees. The attendance of the employees is marked in the table meant to be the attendance sheet. The naming convention of this table is somewhat special. We must note the fact that this attendance sheet table needs to be manipulated very carefully. This is because the table is liable to hold voluminous amounts of data. What we have done to reduce the complexity and workload on the table is that we have made attendance tables for each month in a year. For instance, the name of the table meant to be the attendance sheet for the month of March 2009 will be attendanceSheet0309. This naming convention would ensue a limited work load on the table as well the querying systems. It also gives a succinct idea abut the attributes that the table will contain. It s clear that the table will have at most 31 attributes, for each day of the month, along with a unique identifier, which we have chosen to be rNad. rNad is a random number generated for each employee. A question which arises here is that what is the significance of rNad and why use rNad when we have a much more poplar and widely used attribute in empID. The answer to this question is security. As already said, security is a major issue on which we have given tremendous importance, which is further enforced here. rNad, unlike empID, is system generated. rNad is generated by the procedure, rNadgenerator. rNad is an attribute on which none has any control as such. So, no one ca manipulate with it to make unauthorized amendments in his/her respective attendance. The case discussed here, which is most definitely avoided by the use of rNad, is a looming danger to the security of the system if we use empID as the unique identifier.

Another aspect of the functioning of Phase 3 is the creation of the tables absentList and presentList. As is obvious from the name, this table holds the list of employees who have been absent and present respectively on a particular day. Both the tables have the same set of attributes which are enlisted below:

- phone number: This field is the primary key and contains the phone numbers of the employees who are absent/present depending on the table in question. It also has to be not null which is obvious, it being the primary key.
- date: This field has the date of the day for which the tables are being made and populated. This is an important aspect because tables are made each day.
- name: This field will have the name of the employee who is absent/present on a particular day.

Another table that is used or rather created in Phase 3 is the guest database. Obviously, there may be some Bluetooth MAC addresses that the system finds not matching with all the employee records. This is a highly probable case because any organization is bound to have a number of gests, if not each day, then on some days. We maintain a separate database for warehousing records of the guests who have been detected by the system as well. The database GuestDB, will have the following attributes:

- GuestMAC: This will be the identifying factor which will separate one guest from the other .
- GuestEntry: This will hold the time at which the concerned guest enters the organization.
- GuestExit: This will hold records of the time at which the guest eaves the organization.

- DateOfEntry: This will hold details regarding the date on which the guest was in interaction with the organization.
- NameOfGuest: This will hold the name of the guest if is possible to retrieve that information.
- OtherDetails: This will hold other information regarding the guest which is relevant and needs to be recorded.

Now, let s take a look into what really happens in Phase 3. We know that in Phase 1, device detection and retrieval of the MAC addresses is done and the data is stored into the scanDump table. The devices detected in each scan are then saved session wise in the table lastDumpSession.
Now, what happens in Phase 3 is that for each session in the lastDumpSession, scanDump is accessed. The MAC addresses are retrieved from this table and are matched with the MAC addresses available from the workersDB database. Now we have the following cases in this regard:

- The MAC address is found to be one of the employees in the database, then the following is done:

    1. The attendance for the employee is marked in the attendance sheet for the particular day in correspondence with his/her rNad.
    2. The employee detected will be marked present in the table presentList. This is a kind of guarantee that the employee is present for that day.

- The MAC address is not matching with the records that are found in the database. Then the following is performed:

    1. The MAC address after being detected as being one of a guest, is stored into the GuestDB database and

the other records which the table demands are tried to be dereived.

- Those MAC addresses which are found in the WorkersDB and are not detected for any particular day are marked absent in the attendance sheet for the corresponding rNad. Also the absentList table is also updated with the details of employees who are absent. The above is done only if the employees for the undetected MAC addresses are not on sanctioned leave.

It is at this point that we have an option to clear the scanDump and lastDumpSession table if we want to. This is obviously a very helpful measure because the data in the scanDump and lastDumpSession have been processed and it will be beneficial to remove them from the system.
There is also a facility for notifying employees about the successful marking of their attendance. This is done by the help of an SMS sent by the system to the marked employees. This service is still pending and s in its finishing stages.
This is how Phase 3 works and the attendance of employees is marked. Thus the whole intention, the sole motive of automating the attendance making process is achieved.

Following is given an algorithm for the working of Phase 3 of the project:

```
Phase 3  main (void)
{
        LOCAL VARIABLES

        INITIALISING SYSTEM
                CONNECT TO DATABASE
                PROCEED IF SUCCESSFUL, ELSE END PHASE3
        END INITIALISATION

        PARSING QUERY TO ACCESS SCANDUMP, LAST DUMP SESSION, WORKERSDB
                FOR ROW(i) IN LAST DUMP SESSION
```

```
            {       FETCH ROW(j) IN SCAN DUMP
                    {       IF (SCANDUMP.MAC EXISTS IN WORKERS.MAC)
                            {
                                    PERFORM ATTENDANCE MAKING PROCESS
                                    (DONE BY FETCHMOD,SEARCHMOD AND MAKEATTEND)
                            }
                            ELSE
                            {
                                    PERFORM ATTENDANCE MAKING PROCESS FOR GUEST
                                    (DONE BY FETCHMOD,SEARCHMOD AND GUESTMOD)

                            }
                    }
            }
    ATTENDANCES MADE FOR ALL ENTRIES IN THE SCANDUMP

    NUMBER OF GUESTS DETECTED

    PROCEED FOR MARKING THE ABSENTEES
            FOR ROW(i)(PARTICULAR RNAD) IN ATTENDANCESHEET
            {
                    IF (CH =(1ST CHAR IN ATTEND <DATE> IS NOT `A') && _
                                                    ATTEND <DATE> NOT NULL)
                    {
                            IF (CH IS `P')
                                    HE IS PRESENT
                            IF (CH IS `L')
                                    HE IS ON LEAVE
                            IF (CH IS `H')
                                    HOLIDAY
                            ELSE
                                    INVALID ENTRY
                    }
                    ELSE
                    {
                            MARK ABSENTEES ATTEND <DATE>= `A'
                    }
    ABSENTIES MARKED

    PROCEED FOR SYSTEM SHUTDOWN
            CLEAR THE SCANDUMP AND LAST DUMP SESSION
            COLLECT AND SHUTDOWN SYSTEM
}
```

```
//END MAIN
```