# REMOTE SYSTEM MONITOR FOR LINUX

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

***COMPUTER SCIENCE AND ENGINEERING***

under the Supervision of

**Dr. Yashwant Singh**

By

**Saransh (091270)**

**Tanmmay Mahendru (091223)**

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

## Certificate

This is to certify that project report entitled "Remote System Monitor for Linux", submitted by Saransh (091270) and Tanmmay Mahendru (091223) in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

**Date:**

**Supervisor's Name**

**Designation**

## Acknowledgement

We express our sincere gratitude to our respected project supervisor Dr. Yashwant Singh, Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat under whose supervision and guidance this work has been carried out. His whole hearted involvement, advice, support and constant encouragement throughout, have been responsible for carrying out this project work with confidence. We are also grateful to him for providing us with required infrastructural facilities that have been highly beneficial to us in undertaking the above mentioned project.

We are sincerely grateful to Brig. S.P. Ghrera, Professor and Head of Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat for providing all necessary facilities for the successful completion of our project.

We would also like to thank the laboratory staff of Department of Computer Science and Engineering for their timely help and assistance.

Date:

Saransh (091270)
Tanmmay Mehandru (091223)

# Summary

With protocols such as VNC which provide an excellent graphic interface to the client because of the bit map transfer over the network a huge bandwidth is required which we find seldom in developing countries where 3G is still a new concept and therefore is quite costly. The same problem exists with other technologies too apart from the portability issues, no matter how well they perform on a high bandwidth network when it comes the using such technologies over a 2G mobile network they all tend to collapse.

The utility will allow the user to manage processes on his system as well as monitor the running processes, all from a handheld device or even a browser running on a remote computer. The core sentiment behind this approach is to minimize the data being transferred. This can be accomplished by using a lightweight data interchange format called JSON (Javascript Object Notation) for the transfer of only relevant data, unlike the X windows server, which transfers large amount of bitmap image data across the network. The data being transferred on network can further be reduced by using GZIP compression which is supported by most modern browsers, both mobile and computer based. The user interface of the application will be based on HTML5 and Cascading Style Sheets.

# TABLE OF CONTENT

# List of Figures

# Chapter 1. INTRODUCTION

## 1.1. Introduction

Wire is a web based remote monitoring tool for the linux/unix based operating system. It lets the user manage the running application on his machine and monitor the overall status of the system resources. The same happens with the help of a web based user interface which can be accessed on any browser with HTML5 support.

There has been a constant need to control one's personal computer or to reliably monitor the state of a program, from handheld devices like mobile phones or tablet computers. With the existing technologies, the bandwidth consumption is too much for a cellular network. The purpose of the project is to develop a server-client based application which provides the required functionalities without the need of transferring extraneous display information on the network so as to reduce the amount of data transfer.

The project aims at providing functionalities on a handheld device to start a program, kill a program, and monitor running programs and the machine as a whole. Some of the long term goals aim at implementing functionalities like locking or shutting down the machine remotely. After the completion of the project, it will provide on the move solution to control a personal computer, securely and efficiently. The resulting API will run on a neweraHPC server to allow for scaling up the utility to monitor a computer grid.

## 1.2. Genesis of Problem

There has been a constant need to control one's personal computer from handheld devices like mobile phones or tablet computers. I have seen people sitting right back with their machine, waiting for a program to exit so that they can start a new one. With high end smart phones coming up, be it Android, iOS or the Windows platform, developers have tried to successfully build very native remote desktop applications for the handheld devices. They work great if the device is connected on a WiFi Network or has a 3G connection, but the bandwidth consumption is still too much. In developing countries like India, where the network providers are still trying to bring the 3G technology to masses, the cost of running a VNC application on a mobile device is too high.

Besides, there was another reason. We all agreed on the point that it would be really great if one could sit back on a sofa and control the desktop on a far ahead table. And by controlling, we mean more than just media player.

## 1.3. Problem Statement

This project aims at designing user friendly, low bandwidth consuming multifunctional software which can access a remote system through internet/Wifi/Bluetooth when the user is mobile. User need not to carry around his system now with gadgets like cell phone or tablet or anything that supports internet connection and the capability to open a browser will do the work of accessing the HTML5 based user friendly web site through which user can not only monitor his system but also control it in a way that all the basic tasks are accomplished easily.

## 1.4. Objectives

The main motive behind this project is to provide users with software which can communicate to a remote system over a low bandwidth network. We have divided this objective into several smaller objectives which are as following:

### 1.4.1. User Friendly Front End

We intend to design a web based front-end with user interface optimized for mobiles, tablets and Personal Computers.

### 1.4.2. Server Development

We intend to develop a server which can communicate to a remote device and implement functionalities like start an application, kill, monitor running programs and the machine as a whole.

### 1.4.3. Low Bandwidth Communication

As discussed earlier the main aim behind the project is to reduce the size of data transfer on the mobile network.

### 1.4.4. Provide Extra Functionalities

In future functionalities like lock/shutdown, control the volume remotely, send special key signal to system functionality.

### 1.4.5. Adding Grid Capabilities

In the future after implementing basic functionalities in the software next step would be to make it work on a grid i.e. for larger number of systems combined.

## 1.5. Approach/Methodology Used.

Most inter-process communication uses the ***client server*** model. It refers to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server, typically to make a request for information. The client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

## 1.6. Organizational Thesis

### 1.6.1. Literature Survey

In this chapter the study and research work that has been carried out in the past has been summarized and conclusions has been derived for project work.

### 1.6.2. Remote Monitoring System for Linux

A general overview with constraints, capabilities and performance related issues has been discussed in this chapter

### 1.6.3. Design and Implementation

Designing of the software and the implementation of major modules that have been included in the project are listed under this chapter.

### 1.6.4. Conclusion and Future Scope

This chapter includes what a general summery of what has been done yet in the project and what more can be done in the times to come in this field and project.

# Chapter 2. LITERATURE REVIEW

## 2.1. Introduction

The X Window System (or X11) has long supported the basic need of graphical user interfaces (GUIs) and rich input device capabilities for networked computers. X11 was published in 1987, and there is no doubt that it is doing a great job when it comes to high speed network computers. The problem with X11 is that when using X across a network, bandwidth limitations can hinder the use of bitmap-intensive applications that require rapidly updating large portions of the screen with low latency. Even a relatively small uncompressed 640x480x24bit 30fps video stream can easily outstrip the bandwidth of a 100Mbit network for a single client. Virtual Network Computing (VNC) or Xpra solve some issues with the earlier approaches. But, to put it simply, the best performance is achieved only on thick clients with relatively high speed network connections. The project is built around the goal to achieve the kind of accessibility which VNC provides, and at the same time reduce the amount of data transfer on the network. In developing countries like India, where the network providers are still trying to bring the 3G technology to masses, the cost of running a VNC application on a cellular network is too high.

## 2.2. Related Technologies

There are plenty of existing ways to access a computer remotely. Most of them can be put under a few major categories. They are, X Window System, Virtual Network Computing, NX, Remote Desktop Protocol and Cross Protocol.

### 2.2.1. X Window System

The X Window System (commonly known as X11, based on its current major version being 11, or shortened to simply X, and sometimes informally X-Windows) is a computer software system and network protocol that provides a basis for graphical user interfaces (GUIs) and rich input device capability for networked computers. It creates a hardware abstraction layer where software is written to use a generalized set of commands, allowing for device independence and reuse of programs on any computer that implements X.

X is defined by a network protocol.[1] X uses a client-server model which accepts request for graphics display from client and displays it to the user, and receives user input (keyboard, mouse) and transmits it to the client programs.

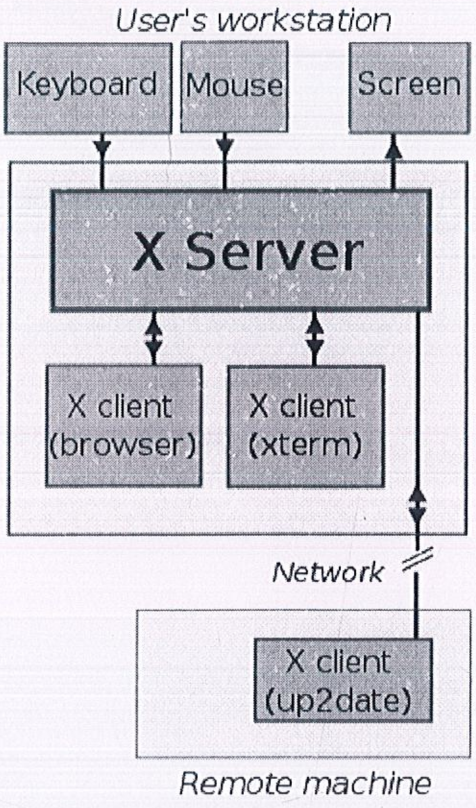


Fig. 2.2.1 X Windows System Architecture

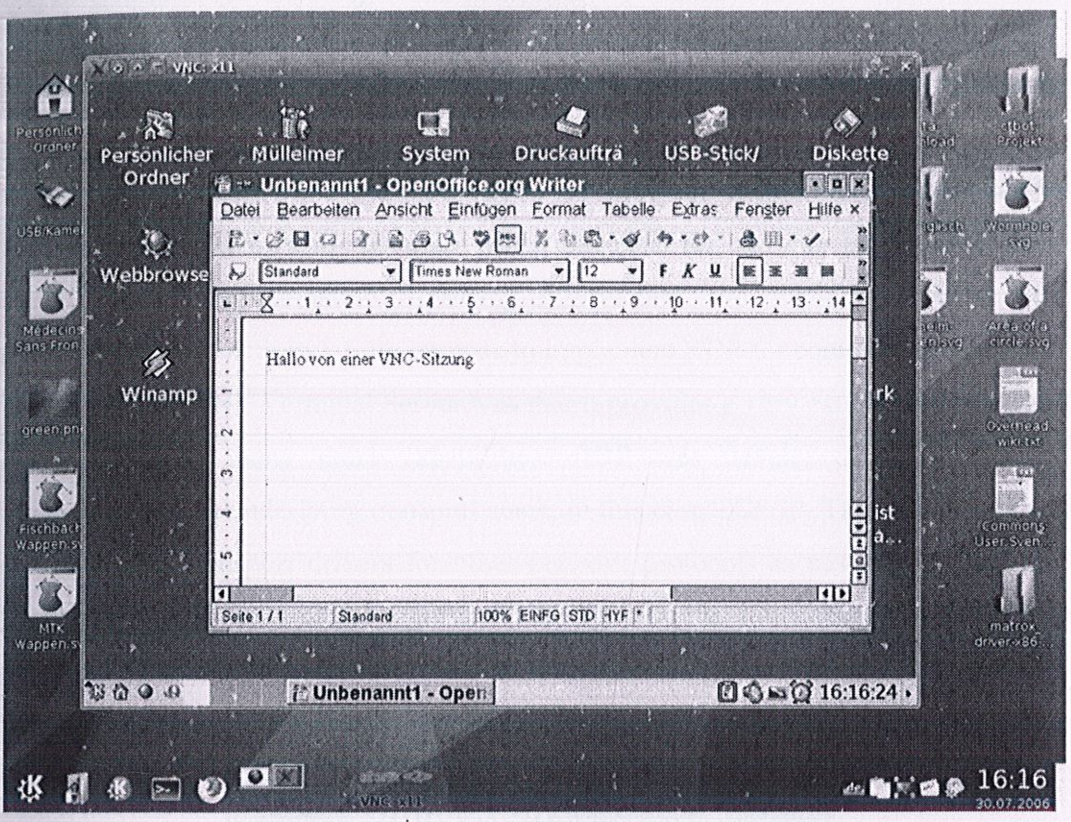
### 2.2.2. Virtual Network Computing

VNC is an ultra-thin client system based on a simple display protocol that is platform-independent. It achieves mobile computing without requiring the user to carry any hardware.[2] In the virtual network computing (VNC) system, server machines supply applications, data and an entire desktop environment that can be accessed from any Internet-connected machine using a simple software Network Computer. The Virtual Networking Computing (VNC) system is a thin-client system. Like all such systems, it reduces the amount of state maintained at the user's terminal. VNC viewers are exceedingly thin because they store no unrecoverable state at the endpoint. This contrasts with systems like X Windows, and allows arbitrary disconnection and

[1] R. W. Scheifler, J. Gettys, *The X Window System*, Digital Equipment Corporation and MIT Project Athena, April, 1986.

[2] T. Richardson, Q. Stafford-Fraser, K. R. WOOD, A. HOPPER, *Virtual Network Computing*, The Olivetti & Oracle Research Laboratory, February, 1998.

reconnection of the client with no effect on the session at the server. Since the client can reconnect at a different location--even on the other side of the planet—VNC achieves mobile computing without requiring the user to carry computing hardware.[3] That is the kind of accessibility we want to achieve as the final goal of the project.



Hallo von einer VNC-Sitzung

Fig. 2.2.2 VNC in Action

## 2.2.3. NX Technology

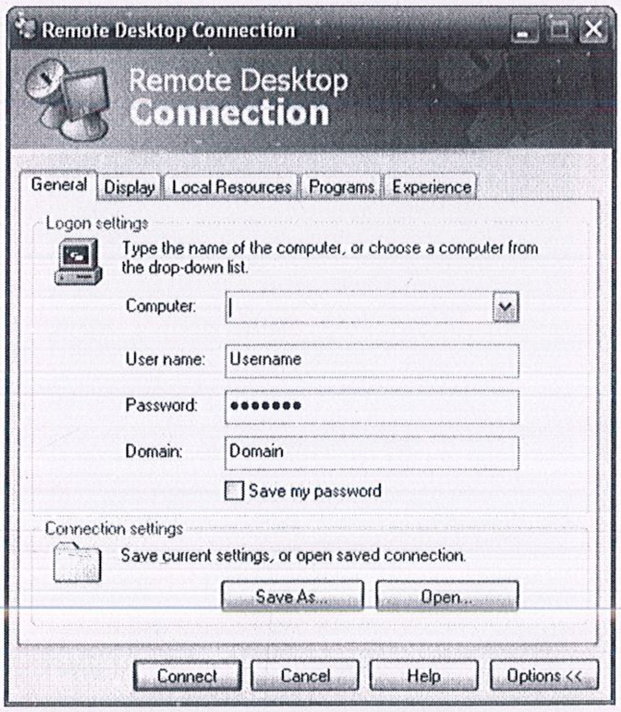
NX compresses the X11 data to minimize the amount of data transmitted. NX takes full advantage of modern hardware by caching all manner of data to make the session as responsive as possible. For example the first time a menu is opened it may take a few seconds, but on each subsequent opening the menu will appear almost instantly. NX is faster than its predecessors, as it eliminates most of the X round-trips, while dxpc and MLView only compress data. The two principal components of NX are nxproxy and nxagent. nxproxy is derived from dxpc and is started on both the remote

[3] "Microsoft Windows NT 'Hydra' and Windows-Based Terminals," white paper available at http://microsoft.com/ntserver/guide/hydrapapers.asp.

(client in X terminology) and the local (server in X terminology) machines simulating an X server on the client and forwarding remote X protocol requests to the local X server.

## 2.2.4. Remote Desktop Protocol

Remote Desktop Protocol (RDP) is a proprietary protocol developed by Microsoft, which provides a user with a graphical interface to another computer. It is based on, and is an extension of, the T-120 family of protocol standards. A multichannel capable protocol allows for separate virtual channels for carrying presentation data, serial device communication, licensing information and highly encrypted data (keyboard, mouse activity). Sole purpose to implement RDP for connectivity purposes within Windows NT Terminal Server was that it provides a very extensible base from which to build many more capabilities. RDP was developed to be entirely independent of its underlying transport stack, in this case TCP/IP. This means that we can add other transport drivers for other network protocols as customers needs for them grow, with little or no significant changes to the foundational parts of the protocol. These are key elements to the performance and extendibility of RDP. [4]



Type the name of the computer, or choose a computer from the drop-down list.

Computer:

User name: Username

Password: •••••••

Domain: Domain

☐ Save my password

Save current settings, or open saved connection.

Fig. 2.2.4. Remote Desktop Protocol Screenshot

[4] *Understanding the Remote Desktop Protocol, "Microsoft Windows NT Server 4.0, Terminal Server Edition"* available at http://support.microsoft.com/kb/186607.

### 2.2.5. Other Protocols

There are many other proprietary as well as open source protocols that performs that task of providing the graphical interface remotely. Macintosh has its own Apple Remote Desktop which gives complete access to the computers in a network running Remote Desktop Server.

## 2.3. Existing Products

There are products in the market which provide functionalities we are trying to put together, but each one of them has some drawbacks and some advantages.

### 2.3.1. Unified Remote

Unified Remote is an app that lets you control your entire Windows computer from your Android device. In short, it turns your device into a WiFi or Bluetooth remote control for all the programs on your computer. It is easily the most feature-filled PC remote available. With our app you can control a wide range of applications, including simple mouse and keyboard, media players, and other external hardware that can be connected to your computer.

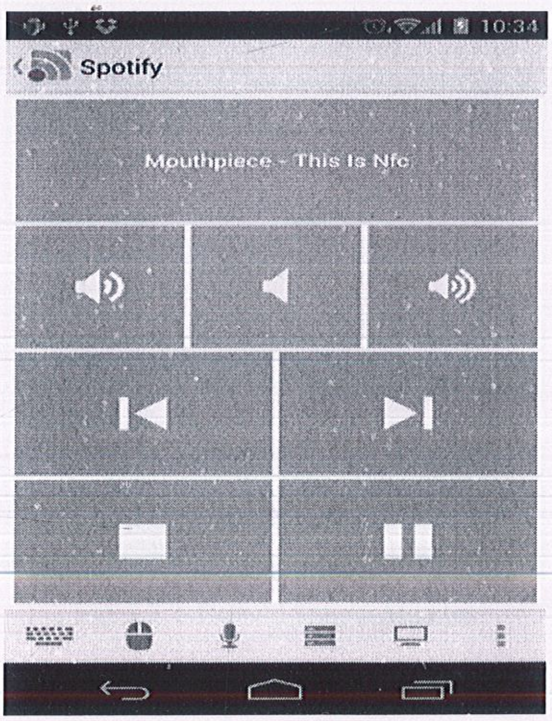
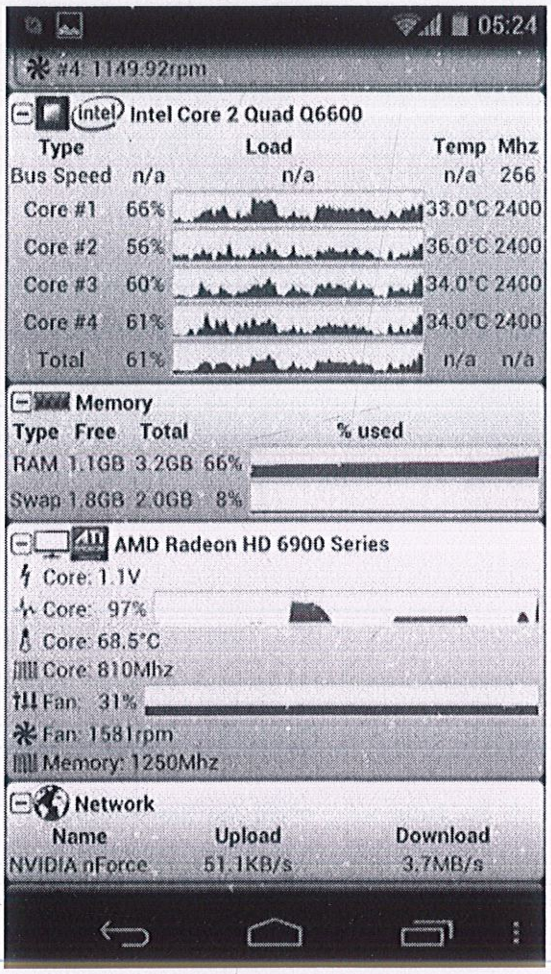


Fig. 2.3.1. Unified Remote.

### 2.3.2. Remote System Monitor

Remote System Monitor is an Android or BlackBerry software which allows to get advanced system and hardware information from your windows computers on your Android or BlackBerry devices over the network. Remote System Monitor allows to retrieve system information including graphic card information even if nobody is logged on the computer. You can use it to check your home cinema (HTPC), media centre, servers or desktop state. It is particularly useful to check your system state while playing games, allowing you to know if the computer is overheating, how your system deal with temperature and fan speed and how your games are using your computer resources (CPU, GPU, memory, etc..).



```
05:24
#4: 1149.92rpm

Intel Core 2 Quad Q6600
Type        Load                Temp    Mhz
Bus Speed   n/a      n/a        n/a     266
Core #1     66%                 33.0°C  2400
Core #2     56%                 36.0°C  2400
Core #3     60%                 34.0°C  2400
Core #4     61%                 34.0°C  2400
Total       61%                 n/a     n/a

Memory
Type  Free   Total        % used
RAM   1.1GB  3.2GB  66%
Swap  1.8GB  2.0GB   8%

AMD Radeon HD 6900 Series
Core: 1.1V
Core:  97%
Core: 68.5°C
Core: 810Mhz
Fan:  31%
Fan: 1581rpm
Memory: 1250Mhz

Network
Name            Upload       Download
NVIDIA nForce   51.1KB/s     3.7MB/s
```

Fig. 2.3.2. Remote System Monitor

### 2.3.3. TeamViewer

TeamViewer is a proprietary computer software package for remote control, desktop sharing, online meetings, web conferencing and file transfer between computers. The software operates with the Microsoft Windows, OS X, Linux, iOS, Android, and Windows RT operating systems. It is possible to access a machine running TeamViewer with a web browser.While the main focus of the application is remote control of computers, collaboration and presentation features are included.

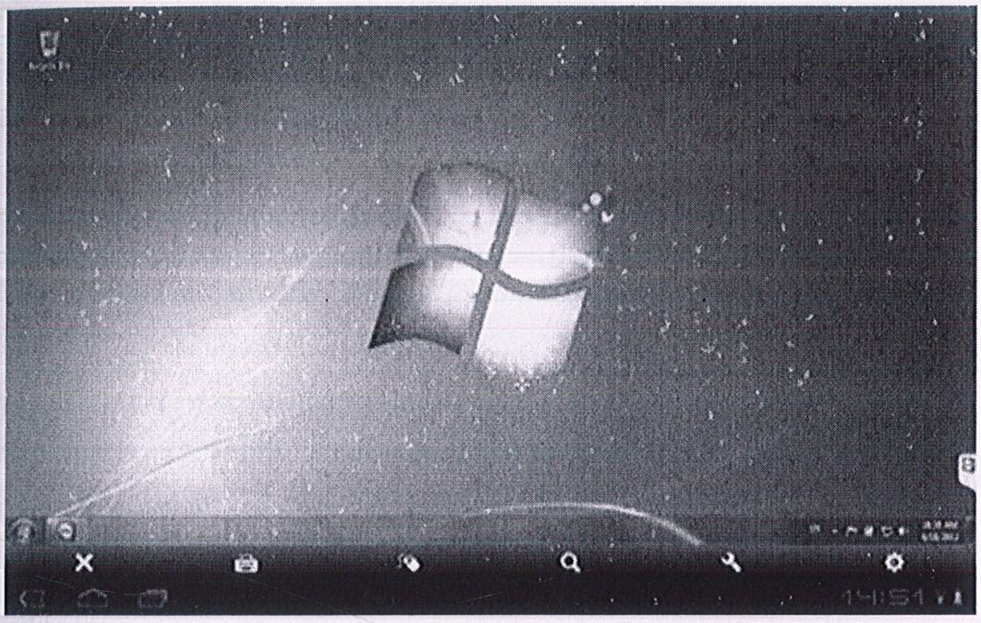


Fig. 2.3.3. TeamViewer running on an Android Device.

## 2.4. User Interfaces

The User Interface Design activity defines the front-end interface in which user will interact with the information system. There are two laws of user interface design.

**First Law:** A computer shall not harm your work or, through inactivity, allow your work to come to harm.

**Second Law:** A computer shall not waste your time or require you to do more work than is strictly necessary.

The user interface should mainly be user centered and as simple as possible so as to make the user feel comfortable when using it. Simple tasks should never require complex procedures, and complex tasks should get tailored to the human hand and mind. People of all ages and cultures should feel firmly in control, and never be overwhelmed by too many choices or irrelevant flash. When designing an interface it should be kept in mind to use buttons which convey the meaning behind the button easily and the user do not get confused between options. Also using soothing colors to design the interface attract more users. Transitions should be fast and clear; layout and typography should be crisp and meaningful. It is preferable to have App icons that are works of art in their own right. Just like a well-made tool, the app should strive to combine beauty, simplicity and purpose to create a magical experience that is effortless and powerful. When people use the app for the first time, they should intuitively grasp the most important features.

## 2.5. Software Interfaces

A piece of software provides access to computer resources (such as memory, CPU, storage, etc.) by its underlying computer system; the availability of these resources to other software can have major ramifications—sometimes disastrous ones—for its functionality and stability. A key principle of design is to prohibit access to all resources by default, allowing access only through well-defined entry points, i.e. interfaces.

The types of access that interfaces provide between software components can include: constants, data types, types of procedures, exception specifications and method signatures. In some instances, it may be useful to define public variables as part of the interface. It often also specifies the functionality of those procedures and methods, either by comments or (in some experimental languages) by formal logical assertions and preconditions.

## 2.6. Communications Protocols

A protocol should have a formal description. It may include signaling, authentication and error detection and correction capabilities.

A protocol definition defines the syntax, semantics, and synchronization of communication; the specified behavior is typically independent of how it is to be implemented. A protocol can therefore be implemented as hardware or software.

Communications protocols have to be agreed upon by the parties involved. To reach agreement a protocol may be developed into a technical standard.

Communicating systems use well-defined formats for exchanging messages. Each message has an exact meaning intended to provoke a defined response of the receiver. A protocol therefore describes the syntax, semantics, and synchronization of communication.

## 2.7. Grid Computing

Grid computing is the federation of computer resources from multiple locations to reach a common goal. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Although a single grid can be dedicated to a particular application, commonly a grid is used for a variety of purposes. Grids are often constructed with general-purpose grid middleware software libraries.
Grid size varies a considerable amount. Grids are a form of distributed computing whereby a "super virtual computer" is composed of many networked loosely coupled computers acting together to perform large tasks. For certain applications, "distributed" or "grid" computing, can be seen as a special type of parallel computing that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus.

## 2.8. Memory Constraints

In electronic digital computers, there are different limitations on the usable memory address space. Even if a microprocessor supports, for example, 32-bit addressing, the integrated circuit package may only allow external access to a lower number of address bits, restricting the memory that can be installed. In modern personal computers, some limits are due to the design of processor, others due to the design of chipsets, BIOS and other hardware and related electrical limitations. Operating system and application software on a hardware platform may not have the capacity to use the full address space physically available.

For performance reasons, all the parallel address lines of an address bus must be valid at the same time, otherwise access to memory would be delayed and performance would be seriously reduced. Integrated circuit packages may have a limit on the number of pins available to provide the memory bus. Different versions of a CPU architecture, in different-sized IC packages, can be designed, trading off reduced package size for reduced pin count and address space. A trade-off might be made between address pins and other functions, restricting the memory physically available to an architecture even if it inherently has a higher capacity. On the other hand, segmented or bank switching designs provide more memory address space than is available in an internal memory address register.

## 2.9. Reliability

Software reliability plays an important role in assuring the quality of a software. To ensure software reliability, the software is tested thoroughly during the testing phase. The time invested in the testing phase or the optimal software release time depends on the level of reliability to be achieved. There are two different concepts related to software reliability, viz., testing reliability and operational reliability.

## 2.10. Product Functionality

In information technology, functionality is the sum or any aspect of what a product, such as a software application or computing device, can do for a user. A product's functionality is used by marketers to identify product features and enables a user to have a set of capabilities. Functionality may or may not be easy to use but for a product to be successful the functionalities should meet the demand of the user.

## 2.11. Assumption and Dependency

Assumptions are circumstances that you are assuming to be true in order for the project to be successful. But to make the situation clear the product should be made in such a way that in later stages it does not create conflicts.

The best way to describe dependencies is to talk about work-breakdown structures (WBS). WBS is a way to breakdown larger tasks into smaller ones. When you are planning a project, you use this breakdown and add resources and cost. You link tasks together that shows that in order for one to happen, the first has to occur. It's the same with dependencies.

Dependencies are relationships between requirements. A linkage that shows that one requirement is dependent on another.

# Chapter 3. REMOTE MONITORING SYSTEM FOR LINUX

## 3.1. Introduction

Remote Monitoring System for Linux will allow the user to manage processes on his system as well as monitor the running processes, all from a handheld device or even a browser running on a remote computer. The core sentiment behind this approach is to minimize the data being transferred. This can be accomplished by using a lightweight data interchange format called JSON (Javascript Object Notation) for the transfer of only relevant data, unlike the X windows server, which transfers large amount of bitmap image data across the network. The data being transferred on network can further be reduced by using GZIP compression which is supported by most modern browsers, both mobile and computer based. The user interface of the application will be based on HTML5 and Cascading Style Sheets.

## 3.2. Agile Development Model

Agile software development is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle.

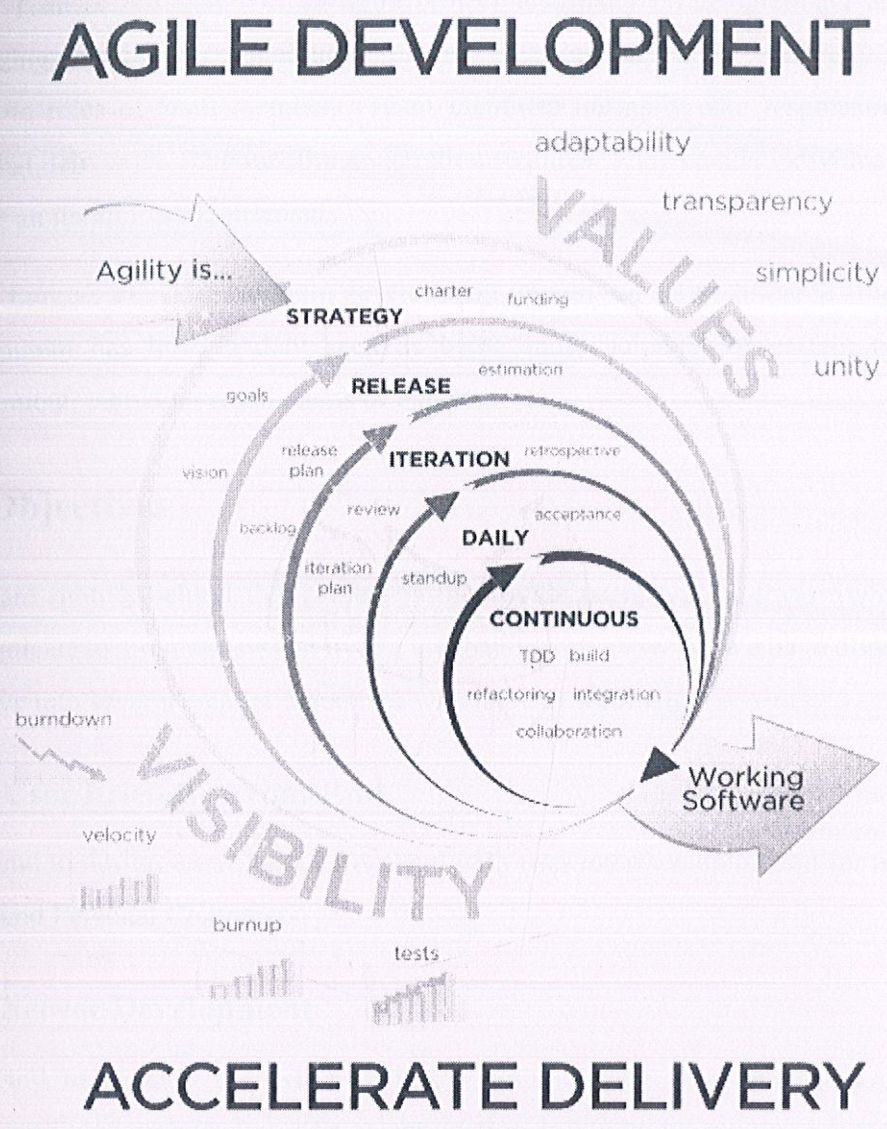


Fig. 3.2. Agile Development Model Process Cycle

Agile methods break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames (timeboxes) that typically last from one to four weeks. Each iteration involves a cross functional team working in all functions: planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration.[10] Multiple iterations might be required to release a product or new features.

Team composition in an agile project is usually cross-functional and self-organizing, without consideration for any existing corporate hierarchy or the corporate roles of team members. Team members normally take responsibility for tasks that deliver the functionality an iteration requires. They decide individually how to meet an iteration's requirements.

Large-scale agile software development remains an active research area. Agile development has been widely seen as being more suitable for certain types of environment, including small teams of experts.

## 3.3. Objectives

The main motive behind this project is to provide users with software which can communicate to a remote system over a low bandwidth network. We have divided this objective into several smaller objectives which are as following:

### 3.3.1. User Friendly Front End

We intend to design a web based front-end with user interface optimized for mobiles, tablets and Personal Computers.

### 3.3.2. Server Development

We intend to develop a server which can communicate to a remote device and implement functionalities like start an application, kill, monitor running programs and the machine as a whole.

### 3.3.3. Low Bandwidth Communication

As discussed earlier the main aim behind the project is to reduce the size of data transfer on the mobile network.

### 3.3.4. Provide Extra Functionalities

In future functionalities like lock/shutdown, control the volume remotely, send special key signal to system functionality.

### 3.3.5. Adding Grid Capabilities

In the future after implementing basic functionalities in the software next step would be to make it work on a grid i.e. for larger number of systems combined.

## 3.4. Architecture Overview

Most interprocess communication uses the ***client server*** model. It refers to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server, typically to make a request for information. The client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

The product being a typical client-server communication model, consists of two major components, the *host server* and the remote *http client*.

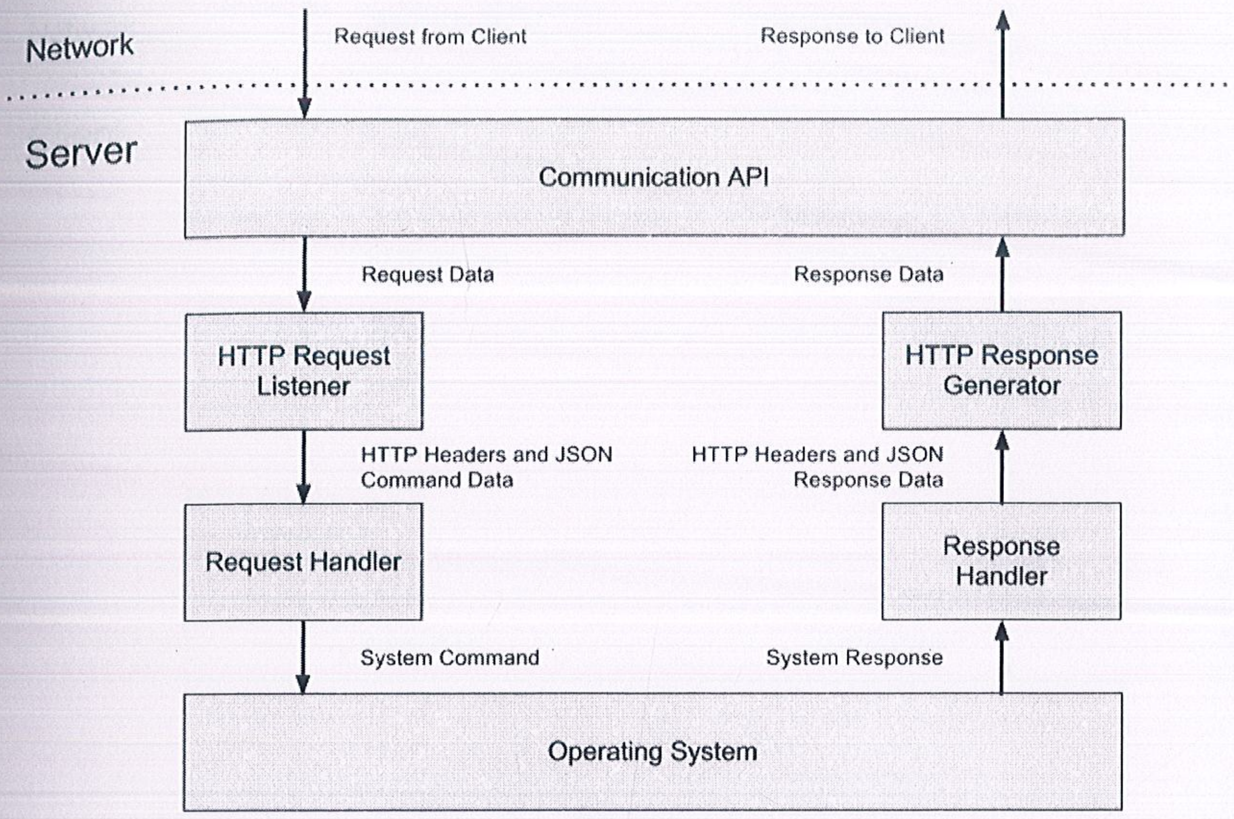


Fig 3.4.1. Overview to Server Architecture.

The server is responsible for accepting requests from the client, validating the requests, and executing the command on the operating system over which it is running. (*Fig. 4*) The server, also, accepts the response data from the operating system and forwards it to the client.

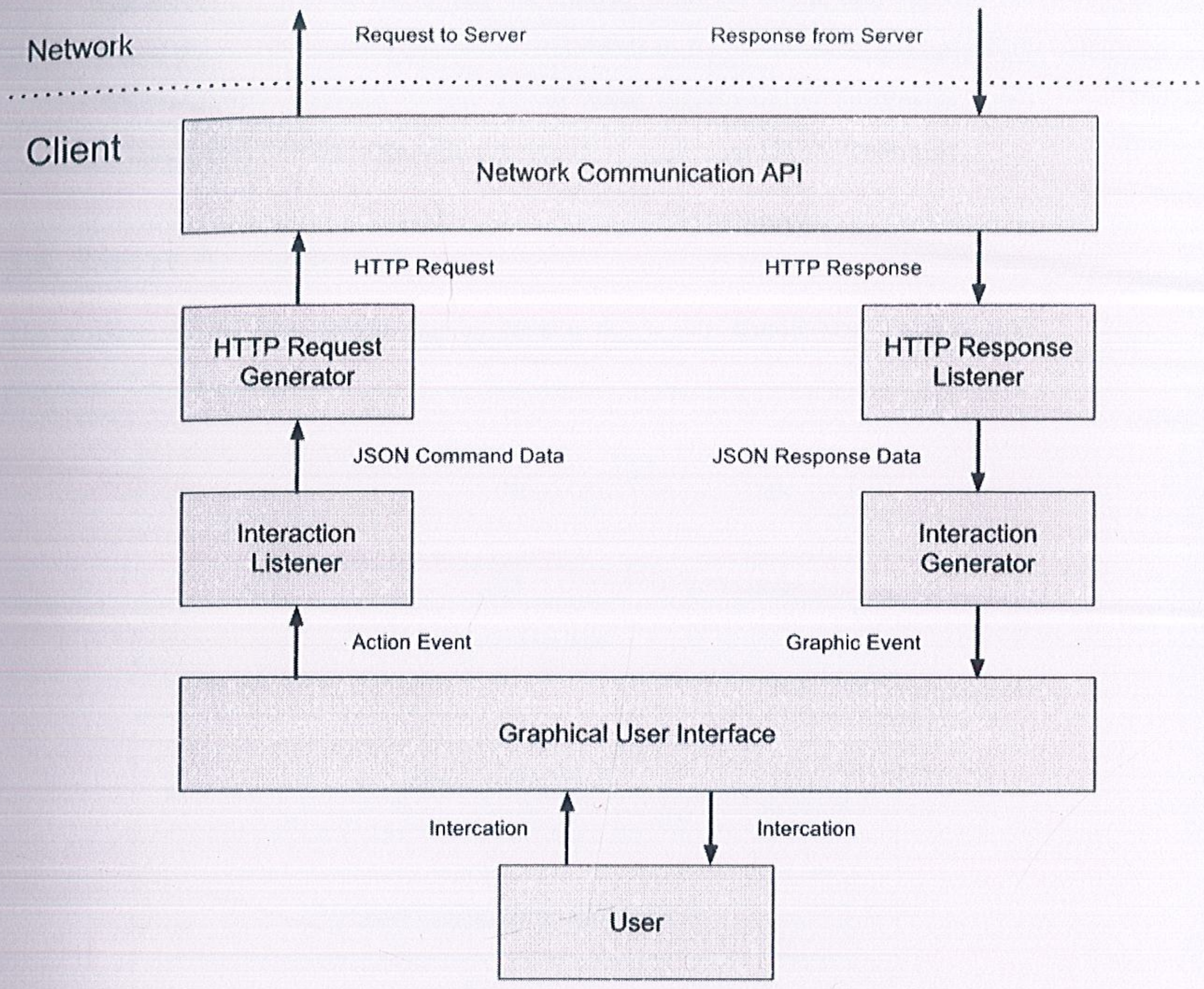


Fig 3.4.2. Overview to Client Architecture.

The HTML based client, presents a clean graphical user interface to the human. The GUI lets the user interact with the system to run the various commands he wants to execute on the remotely placed host machine. The interaction with the GUI are converted into the server understandable protocol by the underlying Javascript based Network Communication API (*Fig. 5*), and send over the network, to the host server.

The communication over the network takes place using the standard HTTP 1.1 protocol. The commands selected by the user are converted into JSON Data format, and appended to the HTTP Request Body. The request is then sent to the server.

The server accepts the HTTP request coming from the client, and decodes the JSON Data to extract the system command that needs to be executed on the operating system. The response received by the server from the operating system is again converted into JSON written to the client network stream as an HTTP Response.

## 3.5. Server Architecture

The working of the host server can be broken down into smaller components and processes that are going on.

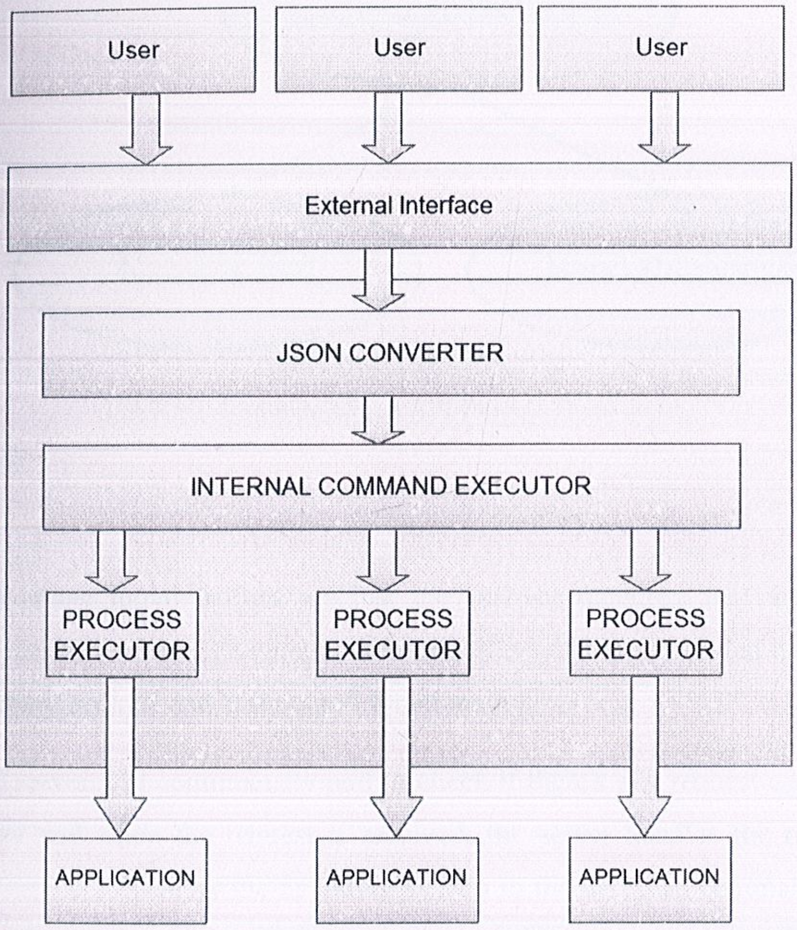


Fig 3.5.1. Activity Flow from User to Applications.

### 3.5.1. Communication API

Communication over the web takes place with the help of HTTP, a general request-response protocol in the client-server computing model. A communication API (or Application Programming Interface) would provide encapsulated tools and functions to communicate over a network.

The API would provide the following functionalities:

a Poll for request.

b Send Response.

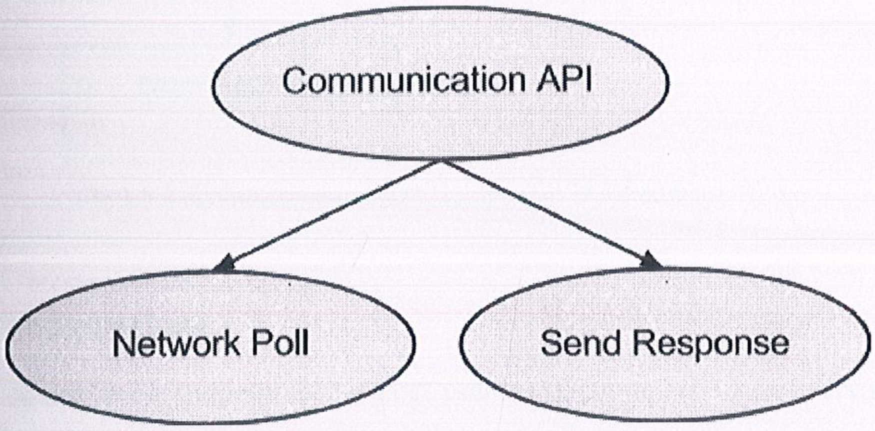


Fig 3.5.2. Communication API Process Breakdown.

The above functionalities are just the abstract functions that the API will provide for the ease of use. The functionalities will need to implemented with the help of non-blocking sockets and system poll functionality.

The server will continuously poll to check if there's any request coming from client or not and when the request is received, the server handles the request and writes the response given by the operating system to the executed command, back to the client. The execution over the server may take time which will make it unresponsive to other clients. Therefore, a non-blocking connection needs to be setup between the two. This takes the responsiveness of the system as a whole to another level.

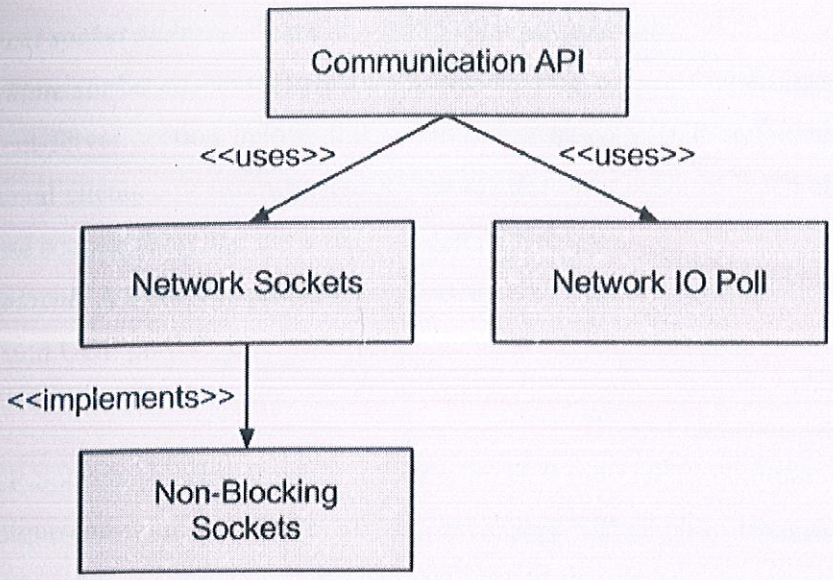


Fig 3.5.3. Communication API Component Breakdown.

### 3.5.2. Network Sockets

A network socket is an endpoint of an inter-process communication flow across a computer network. Today, most communication between computers is based on the Internet Protocol; therefore most network sockets are Internet sockets.

A socket API is an application programming interface (API), usually provided by the operating system, that allows application programs to control and use network sockets. Internet socket APIs are usually based on the Berkeley sockets standard.

A socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread.

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. The two processes establish their own socket. An Internet socket is characterized by a unique combination of the following:

1 ***Local socket address:*** Local IP address and port number.

2 ***Remote socket address:*** Only for established TCP sockets. As discussed in the client-server section below, this is necessary since a TCP server may serve several clients concurrently. The server creates one socket for each client, and these sockets share the same local socket address.

3 ***Protocol:*** A transport protocol (e.g., TCP, UDP, raw IP, or others). TCP port 53 and UDP port 53 are consequently different, distinct sockets.

TCP Sockets (or virtual ports) are used in TCP (and UDP) communication to identify unique end-to-end connections. They are called 'virtual ports' because a single physical connector can serve multiple connections. Each side of a socket connection uses its own port number, which does not change during the life of that connection. The port number and IP address together uniquely identify an endpoint. Together, two endpoints are considered a 'socket'. This makes TCP sockets the best option of our use.

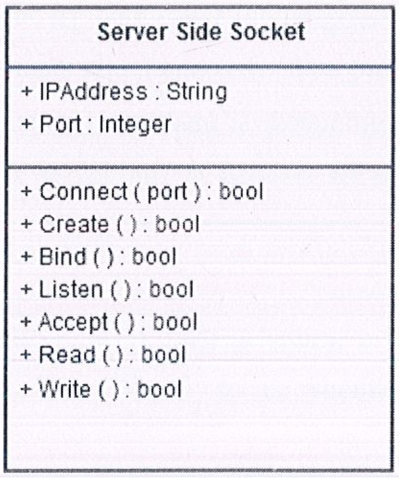


Fig 3.5.4. Class Diagram of Server's Network Socket.

The steps involved in establishing a **socket on the server side** are as follows:

1 Create a socket with the socket() system call.

2 Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.

3 Listen for connections with the listen() system call.

4 Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.

5 Send and receive data.

### 3.5.3. Non Blocking Sockets

By default, TCP sockets are in "blocking" mode. For example, when you call recv() to read from a stream, control isn't returned to your program until at least one byte of data is read from the remote site. This process of waiting for data to appear is referred to as "blocking". The same is true for the write() API, the connect() API, etc. When you run them, the connection "blocks" until the operation is complete.

When placed in non-blocking mode, the program never waits for an operation to complete. This is an invaluable tool to switch between many different connected sockets, and ensure that none of them cause the program to "lock up." So, the server can handle multiple http clients simultaneously. The problem with this approach is that the program cannot know which client is responding to the data it sent earlier. The solution was to make the server poll the network for response from the client.

### 3.5.4. Network IO Poll

'Polling' is the continuous checking of other programs or devices by one program or device to see what state they are in, usually to see whether they are still connected or want to communicate.

Network polling is controlled by poll policies. Poll policies consist of the following:

1 Poll definitions, which define the data to retrieve.

2 Poll scope, consisting of the devices to poll. The scope can also be modified at a poll definition level to filter based on device class and interface.

3 Polling interval and other poll properties.

### 3.5.5. HTTP Request

An HTTP request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.

```
        Request = Request-Line *(( general-header
                  |request-header)CRLF)
                  CRLF [ message-body ]
```

The request/response message consists of the following things.

1. Request line, such as GET /logo.gif HTTP/1.1 or status line, such as HTTP/1.1 200 OK.
2. Headers
3. An empty line.
4. Optional HTTP message body data.
5. The request/status line and headers must all end with <CR><LF> (that is, a carriage return followed by a line feed). The empty line must consist of only <CR><LF> and no other whitespace.
6. The optional HTTP message body data.

The first line of a typical HTTP request would look something like

**GET index.html HTTP/1.1**

This is called the *request line* of a HTTP Request. The request line is followed by HTTP header fields. They are components of the message header of requests and responses in the Hypertext Transfer Protocol (HTTP). They define the operating parameters of an HTTP transaction.

The header fields are transmitted after the request or response line, the first line of a message. Header fields are colon-separated name-value pairs in clear-text string format, terminated by a carriage return (CR) and line feed (LF) character sequence. The end of the header fields is indicated by an empty field, resulting in the transmission of two consecutive CR-LF pairs. Long lines can be folded into multiple lines; continuation lines are indicated by presence of space (SP) or horizontal tab (HT) as first character on next line.

A few standard HTTP headers are as follows:

| Accept

| Accept-Charset

| Accept-Encoding

| Accept-Language

| From

| Host

| User-Agent

HTTP Body Data is the data bytes transmitted in an HTTP transaction message immediately following the headers if there is any. It contains any data that needs to be required by the server to entertain the request.

### 3.5.6. HTTP Request Listener

HTTP Request Listener is a functionality running on the host machine, that accepts the data read by the communication socket. It breaks down the raw socket data into HTTP Request Headers and HTTP Body. The headers are decide the operating parameters of the request while the body contains the information required to handle the request. The data is then encapsulated into a structure and passed on to the request handler for further execution.

### 3.5.7. Request Handler

The request handler accepts the HTTP Data Structure from the Request Listener and strips out the JSON Data from the HTTP Body. The HTTP Body contains commands that need to be executed in the form of JSON (Javascript Object Notation), which helps in structuring the information as well as reduce the data size, and the time spent in handling the request.

A sample HTTP Body content would look like the following:

```
"command" : {
        "type": "start-app",
        "name": "firefox",
        "arguments": [
                               "www.google.com",
                               "www.facebook.com"
        ]
}
```

The handler will use the json structure to generate the system command and execute it on the operating system using the *system()* functionality. For example, the above json command would start a new firefox instance (if not already running), and open the urls *www.google.com* and *www.facebook.com*. The corresponding system command will look like

```
system ( "firefox www.google.com www.facebook.com"
);
```

The above statement will execute the command supplied in the quotes on the operating system and return a success status. If the command was successful, the response given by the application will be piped to the response handler for further processing and transmission back to the client.

## *A. Javascript Object Notation (JSON) Data*

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

1 A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

2 An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

1 An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).

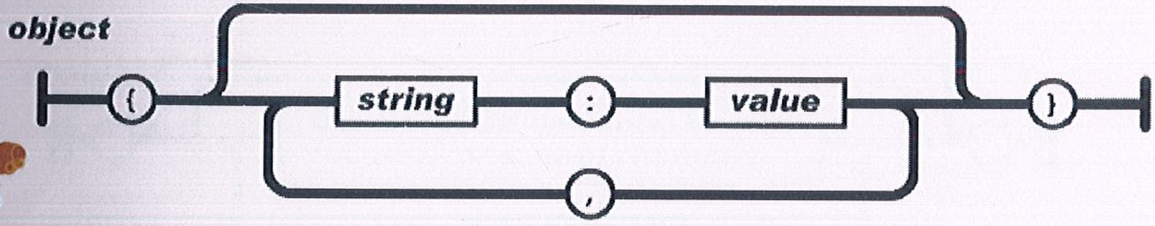


Fig 3.5.5.Object Notation in JSON [5]

2 An array is an ordered collection of values. An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by , (comma).

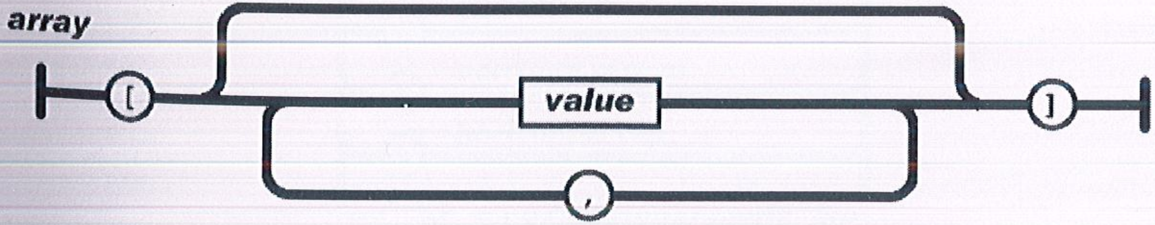


Fig 3.5.6. Array Notation in JSON [5]

3 A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.

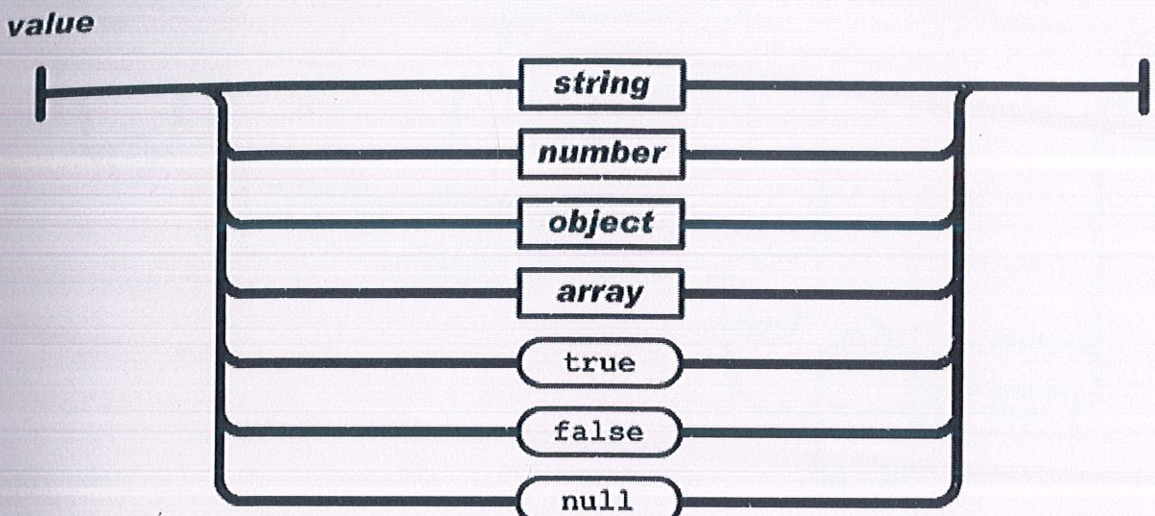


Fig 3.5.7. Value Notation in JSON [5]

4 A string is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.

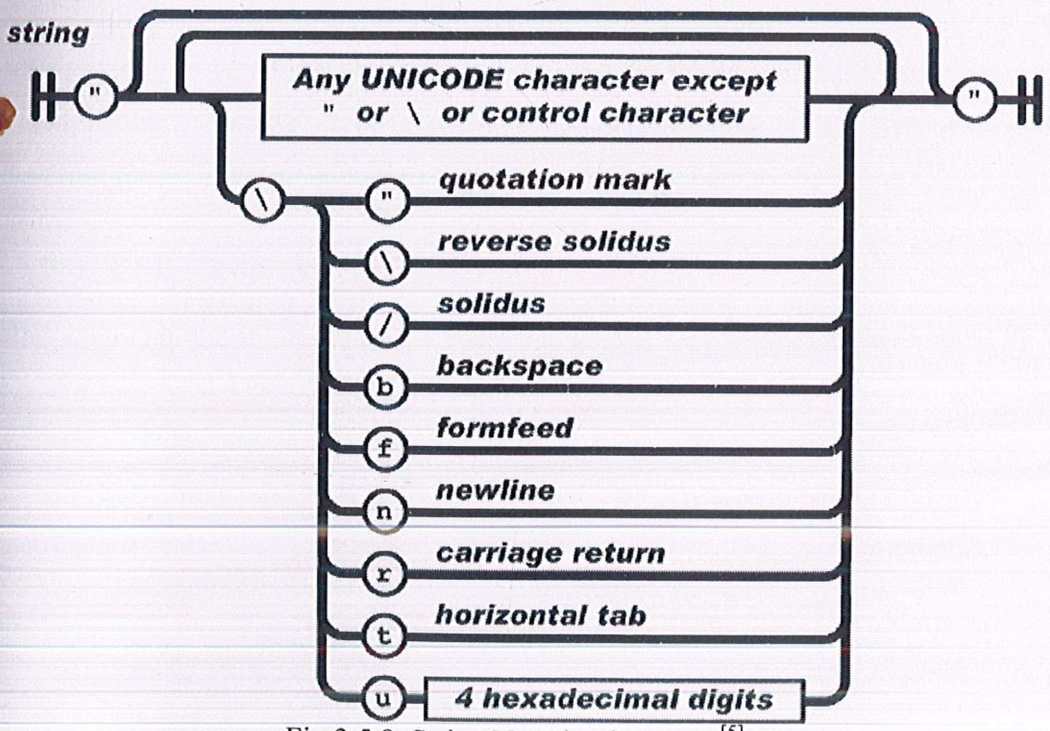


Fig 3.5.8. String Notation in JSON [5]

5 A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.

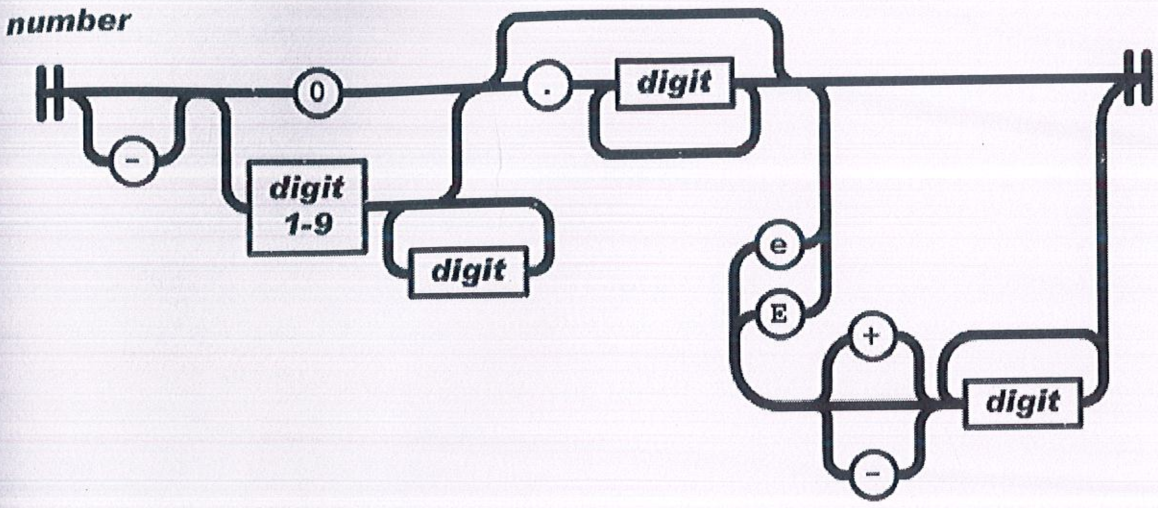


Fig 3.5.9. Number Notation in JSON [5]

### 3.5.8. Response Handler

The system response contains data given by any application that was executed by the user. Response handler accepts this response and encapsulates it into the JSON structure that can be understood by the client. The JSON Response is then forwarded to the HTTP Response Generator for generating a valid HTTP Response and forwarding it to the communication socket for sending it over to the client.

### 3.5.9. HTTP Response

After receiving and interpreting a request message, a server responds with an HTTP response message. A typical HTTP Response look like

```
Response        =    Status-Line
                     *(( general-header
                     | response-header
                     | entity-header ) CRLF)
                     CRLF
                     [ message-body ]
```

The first line of a Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

A http response would look something like this.

```
HTTP/1.1 200 OK
Date: Sun, 10 Oct 2010 23:26:07 GMT
Server: Apache/2.2.8 (Ubuntu) mod_ssl/2.2.8
OpenSSL/0.9.8g
Last-Modified: Sun, 26 Sep 2010 22:04:35 GMT
Accept-Ranges: bytes
Content-Length: 13
Connection: close
Content-Type: text/html

Hello world!
```

Content-Type specifies the Internet media type of the data conveyed by the HTTP message, while Content-Length indicates its length in bytes. The HTTP/1.1 webserver publishes its ability to respond to requests for certain byte ranges of the document by setting the header Accept-Ranges: bytes. This is useful, if the client needs to have only certain portions of a resource sent by the server, which is called byte serving. When Connection: close is sent in a header, it means that the web server will close the TCP connection immediately after the transfer of this response.

Most of the header lines are optional. When Content-Length is missing the length is determined in other ways. Chunked transfer encoding uses a chunk size of 0 to mark the end of the content. Identity encoding without Content-Length reads content until the socket is closed. A Content-Encoding like gzip can be used to compress the transmitted data.

The string "Hello world!" is part of the http response body. The response body can contain any information that can be used by the client and server to communicate the required data. The http client shall send instructions to the server as http request body, and the server shall respond to it with data in the response body.

### 3.5.10. HTTP Response Generator

The HTTP Response Generator, accepts the raw JSON Response Data from the response handler and encapsulates into a valid HTTP 1.1 type Response. It adds the valid HTTP Headers required by the HTTP Response, succeeded by the HTTP Body which contains the JSON Data. The finished HTTP Response data is piped to the communication socket for writing it back to the client network stream as a response to the request it made.

## 3.6. Client Architecture

The client is based on the HTML5 Application model. HTML5 and CSS3 provide beautiful and user friendly, yet a very simplistic user interface for interacting with the user. The underlying Javascript API is responsible for handling user interactions with the application and sending the commands over to the network.

### 3.6.1. Graphical User Interface

A GUI should strive to combine beauty, simplicity and purpose to create a magical experience that is effortless and powerful. When people use the app for the first time, they should intuitively grasp the most important features. The web technology today makes it possible to create rich and interactive user interfaces using HTML5, CSS3 and Javascript. HTML5 and CSS3, there are some interesting features born of the combination of these two languages, like more powerful and interactive applications, and a usability that is more focused on the user.

*A. Why HTML5 and CSS3?*

The reason to choose HTML5 is the versatility it offers and the at the same time keeps the resources very low. HTML5 is making the web platform more powerful in a number of different areas. HTML5's new features allows for developers to manage data, draw and reproduce video and audio, with more semantic elements. It also has a new canvas which makes it easier to integrate video elements. It introduces many cutting-edge features that allows to create apps and websites with the functionality, speed, performance, and experience of desktop applications. But unlike desktop applications, apps built on the web platform can reach a much broader audience using a wider array of devices. High-performance features like 3D CSS, turbocharge web

### 3.5.10. HTTP Response Generator

The HTTP Response Generator, accepts the raw JSON Response Data from the response handler and encapsulates into a valid HTTP 1.1 type Response. It adds the valid HTTP Headers required by the HTTP Response, succeeded by the HTTP Body which contains the JSON Data. The finished HTTP Response data is piped to the communication socket for writing it back to the client network stream as a response to the request it made.

## 3.6. Client Architecture

The client is based on the HTML5 Application model. HTML5 and CSS3 provide beautiful and user friendly, yet a very simplistic user interface for interacting with the user. The underlying Javascript API is responsible for handling user interactions with the application and sending the commands over to the network.

### 3.6.1. Graphical User Interface

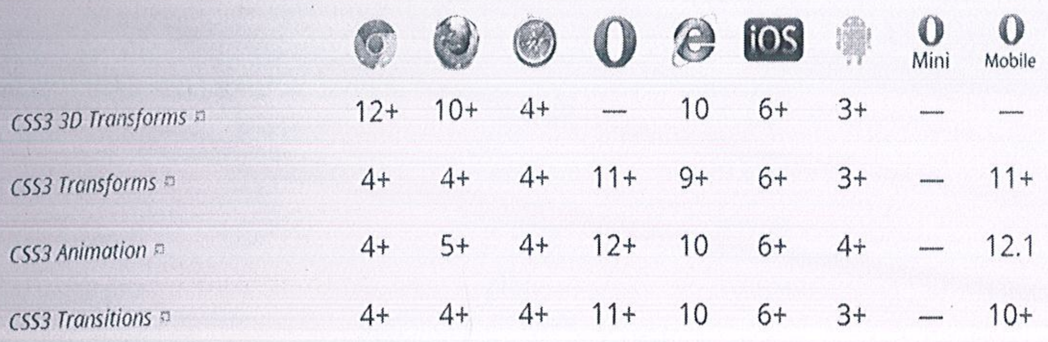
A GUI should strive to combine beauty, simplicity and purpose to create a magical experience that is effortless and powerful. When people use the app for the first time, they should intuitively grasp the most important features. The web technology today makes it possible to create rich and interactive user interfaces using HTML5, CSS3 and Javascript. HTML5 and CSS3, there are some interesting features born of the combination of these two languages, like more powerful and interactive applications, and a usability that is more focused on the user.

*A. Why HTML5 and CSS3?*

The reason to choose HTML5 is the versatility it offers and the at the same time keeps the resources very low. HTML5 is making the web platform more powerful in a number of different areas. HTML5's new features allows for developers to manage data, draw and reproduce video and audio, with more semantic elements. It also has a new canvas which makes it easier to integrate video elements. It introduces many cutting-edge features that allows to create apps and websites with the functionality, speed, performance, and experience of desktop applications. But unlike desktop applications, apps built on the web platform can reach a much broader audience using a wider array of devices. High-performance features like 3D CSS, turbocharge web apps with amazing 3D graphics and special effects. Graphical APIs and technologies let us create a compelling and immersive experience for your users and audience. To put it simply, HTML5 accelerates the pace of innovation.

Fig. 3.6.1. HTML5 Main Logo

CSS3, on the other hand, is all about design. It offers new possibilities of decoration with less markup language and less dependency on Javascript. A higher level of control allows you to change the colour of the text selection so that it matches the rest of the site's colour, make rounded box corners, drop-down menus, animated buttons, multiple backgrounds, web APIs and use font-face technology, among other features. It also allows the use of typographies that are not installed in the operating system. With the introduction of CSS3 it has never been easier to create rich and beautiful sites and applications in HTML. There are many new technologies and extensions to CSS3 including: 2D Transformations, Transitions, 3D Transforms and WebFonts to name just a few.



| | | | | | | iOS | | Mini | Mobile |
|---|---|---|---|---|---|---|---|---|---|
| *CSS3 3D Transforms* | 12+ | 10+ | 4+ | — | 10 | 6+ | 3+ | — | — |
| *CSS3 Transforms* | 4+ | 4+ | 4+ | 11+ | 9+ | 6+ | 3+ | — | 11+ |
| *CSS3 Animation* | 4+ | 5+ | 4+ | 12+ | 10 | 6+ | 4+ | — | 12.1 |
| *CSS3 Transitions* | 4+ | 4+ | 4+ | 11+ | 10 | 6+ | 3+ | — | 10+ |

Fig. 3.6.2. Different Browser Comparison

HTML5 and CSS3 are supported by Google's Chrome, Mozilla Firefox, Opera, Safari and Internet Explorer.

### 3.6.2. Interaction Listener

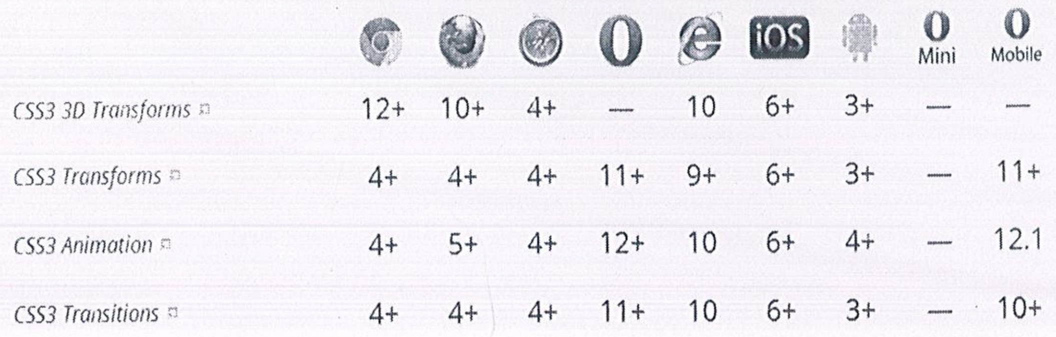
All the different visitors actions that a web page can respond to are called events or interactions. An event represents the precise moment when something happens.

An interaction listener, listens to the events fired by the user interface, and performs a task corresponding to the context of the graphic element. For example, when the button to start a new process is pressed, it brings up a new popup window with a input field to accept the name of the application that user wants to run.

jQuery is tailor-made to respond to events in an HTML page. Event methods trigger or attach a function to an event handler for the selected elements. jQuery Event Handling really allows the developer to separate content and behaviour, that is, the developer can define your content's structure first and only later take care of the specific behaviour.

The term "fires" is often used with events. Example: "The keypress event fires the moment you press a key".

Here are some common DOM events:



| | | | | | | iOS | | Mini | Mobile |
|---|---|---|---|---|---|---|---|---|---|
| *CSS3 3D Transforms* | 12+ | 10+ | 4+ | — | 10 | 6+ | 3+ | — | — |
| *CSS3 Transforms* | 4+ | 4+ | 4+ | 11+ | 9+ | 6+ | 3+ | — | 11+ |
| *CSS3 Animation* | 4+ | 5+ | 4+ | 12+ | 10 | 6+ | 4+ | — | 12.1 |
| *CSS3 Transitions* | 4+ | 4+ | 4+ | 11+ | 10 | 6+ | 3+ | — | 10+ |

Fig. 3.6.2. Different Browser Comparison

HTML5 and CSS3 are supported by Google's Chrome, Mozilla Firefox, Opera, Safari and Internet Explorer.

### 3.6.2. Interaction Listener

All the different visitors actions that a web page can respond to are called events or interactions. An event represents the precise moment when something happens.

An interaction listener, listens to the events fired by the user interface, and performs a task corresponding to the context of the graphic element. For example, when the button to start a new process is pressed, it brings up a new popup window with a input field to accept the name of the application that user wants to run.

jQuery is tailor-made to respond to events in an HTML page. Event methods trigger or attach a function to an event handler for the selected elements. jQuery Event Handling really allows the developer to separate content and behaviour, that is, the developer can define your content's structure first and only later take care of the specific behaviour.

The term "fires" is often used with events. Example: "The keypress event fires the moment you press a key".

Here are some common DOM events:

| Mouse Events | Keyboard Events | Form Events | Document/Window Events |
|---|---|---|---|
| Click | Keypress | Submit | Load |
| Dblclick | Keydown | Change | Resize |
| Mouseenter | Keyup | Focus | Scroll |
| Mouseleave |  | Blur | Unload |

Table. 3.6.1. DOM Events

### 3.6.3. HTTP Request Generator

The HTTP Request Generator accepts the data generated by the interaction listener, and uses the information provided by the user to generate JSON Commands. The JSON command is then sent over to the Communication API. The network communication api then sends the information to the server for the execution.

To put it simply, the HTTP Request Generator acts like a interface between the user interface and the corresponding server understandable command. The interaction is converted into a JSON Request Data structure which contains the command that needs to be executed by the operating system.

### 3.6.4. Network Communication API

There's an army of associated JavaScript APIs. Among the ranks are a few new technologies that open up how we communicate between client and server and across documents. XHR & XHR2 with CORS is the most common and simplistic method to communicate between server and an html based client.

All event handlers (with the exception of XHR) receive an event object containing a data property. This property includes the data sent as part of the message.

The event model (again with the exception of XHR) is mostly based around onmessage and postMessage or send. For example:

```
// in the recipient code
recipient.onmessage = function (event) {
  console.log('received message: ' + event.data);
};


// from the sender code
recipient.postMessage('hi      there');      //       or
recipient.send('hi there');
```

This is just a common model and isn't the exactly the same among all these technologies. The two key similarities are that they use:

1 A sending method (postMessage or send) on the recipient object, and

2 An event handler that listens for the message event and receives an event object containing a data property.

Very importantly, most browsers only support sending strings from sender to recipient, so we often need to JSON stringify and parse if we want to send anything other than a string.

### *A. XHR & XHR2 with CORS*

XHR can be both synchronous and asynchronous. XHR is the only API that (purposely) supports synchronous requests, meaning the execution of code will block until the callback fires.

There's nothing particularly new about XHR, but in XHR2 we can handle uploads, and there's a progress event to tell you how the upload or download is getting on.

The super shiny new toy in XHR2 is its support for Cross-Origin Resource Sharing (CORS). This means you can make an XHR request across domains, but only if the server you're connecting to allows it.

The request is as you'd expect from XHR:

```
var client = new XMLHttpRequest();

client.onreadystatechange = function () {

  if (this.readyState == 4 && this.status == 200) {
    alert('Application        Started:         '        +
this.responseText);
  }
};
client.open('GET', '/wire');
client.send();
```

If our server responds with a CORS header, however, we can put our XHR responder on another server.

When the following code is run in a browser that supports XHR2, the cross domain request succeeds!

```
var client = new XMLHttpRequest();
client.onreadystatechange = function () {
  if (this.readyState == 4 && this.status == 200) {
    alert('The most awesome-est person to follow: '
+ this.responseText);
  }
};
client.open('GET', '/wire');
client.send();
```

XHR usage is pretty common already, but XHR2 with CORS is a winner over JSON-P, particularly as you have finer control over the request, can handle timeouts, and can handle errors correctly.

### *B. Support for XHR & XHR2 with CORS*

- XHR support is pretty solid nowadays (even though IE6 uses ActiveXObject to get it going)

- XHR2 with CORS: Safari & Mobile Safari, Firefox 3.5, Chrome and IE8 (via XDomainRequest)

### *C. postMessage*

This API is older, but it's very useful if you want to get around the XHR same-origin rules. If you have an <iframe> document that can accept onmessage events from your origin (i.e., your site), then you can communicate across domains (and origins).

For example, a page that accepts an onmessage event might contain code such as this:

```
window.onmessage = function (event) {
  if (event.origin == 'mytrustedsite.com') {
    alert('my trusted site said: ' + event.data);
  }
};
```

This gives you the ability to send strings across two mutually trusted domains. (Remember that you can use JSON.stringify and JSON.parse to convert to an object to and from string format.)

### *D. Support for postMessage*

1 Chrome

2 Safari

3 Opera

4 Firefox

5 IE8

### 3.6.5. HTTP Response Listener

The HTTP Response Listener accepts the response data given by the server and analysis the HTTP data to strip out the JSON based response data. The JSON response is then passed on to the interaction generator to make changes to the graphical user interface.

### 3.6.6. Interaction Generator

The interaction generator displays the received information in the form of popups and display boxes. The response given by the executed application is displayed back to the user once the process has been completed. This lets the user know that the task he wanted to do has been successfully completed.

## 3.7. Product Functions

The product functionality is split between two components, the server and the external interface which works as a user front end. The server handles the requests coming from the client and responds to it by providing the appropriate data. It acts as an interface between the client and the operating system. The external interface works as a window to the operating system, and allows the user to manage and monitor the machine by collaborating with the server to fetch the required data.

## 3.8. User Characteristics

The end user of the project will be any user who would want to operate his/her operating system remotely from a handheld device. The handheld device will run a HTML based user interface to the user's personal computer.

## 3.9. Dependencies

The project is build on top of a fully functional High Performance Cluster, neweraHPC, designed to work exclusively on TCP/IP protocol layer with extensible plug-in support. Unlike other grid platforms it has a standard library including base functions and a small client program to distribute tasks to node machines. It achieves zero memory leaks even in the worst scenarios, and has a HTTP server integrated to support the seamless communication required by the HTML front end.

# Chapter 4. DESIGN AND IMPLEMENTATION

## 4.1. Introduction

In this chapter we discuss the design and how the functionalities are implemented in our project. For every project the designing and planning of a project plays a major role in shaping out the final overview of the project and if the implementation is done according to the design the project turns out to be in good shape else there is a possibility of bug detection at later stages.

## 4.2. Design Analysis

If the broader topic of product development "blends the perspective of marketing, design, and manufacturing into a single approach to product development," then design is the act of taking the marketing information and creating the design of the product to be manufactured. Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user.

### 4.2.1. Data Flow Analysis

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

#### Level 0

The Level 0 Data Flow Diagram, outlines the system and the individual actors that are going to interact with the system. It also outlines the major information that will be transferred between the actors and the system.

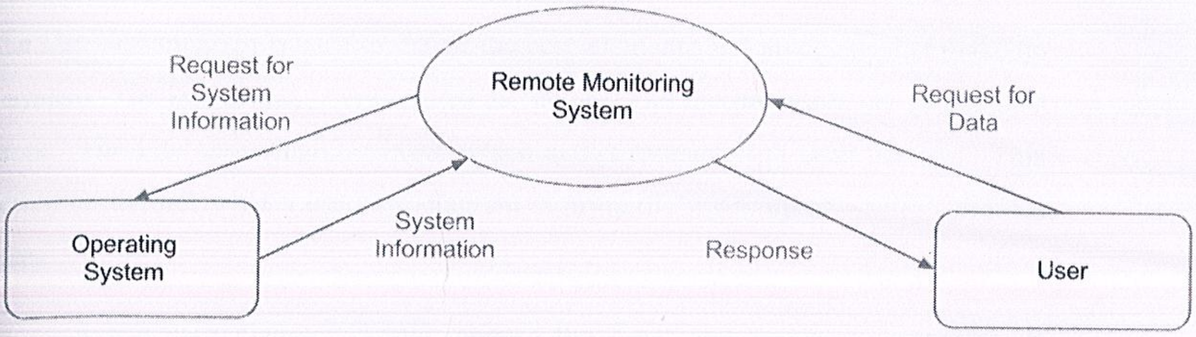


Fig 4.2.1. Level 0 DFD showing the system and the actors.

## Level 1

With Level 1, we get into the details of the system and list the processes that will be involved in the functionality of the system. It helps us to outline the functionalities that need to be implemented and the data that will be required to execute the corresponding functions.

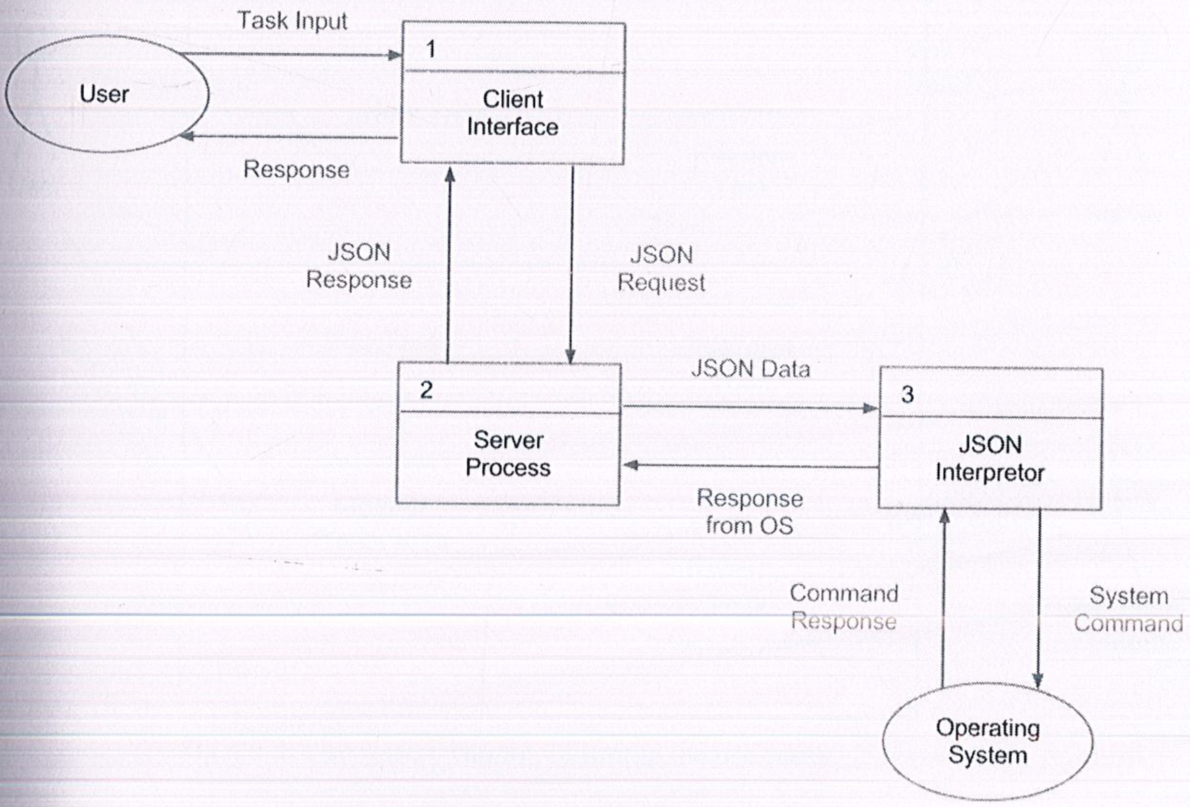


Fig 4.2.2. Level 1 DFD outlining the processes involved in the system.

### 4.2.2. Use Case Diagram

The Use case diagram is used to define the core elements and processes that make up a system. The key elements are termed as "actors" and the processes are called "use cases." The Use case diagram shows which actors interact with each use case. This definition defines what a use case diagram is primarily made up of—actors and use cases.

A use case diagram should capture the functional system components. It embosses the business processes within the system. While you traverse your system, you will learn significant system attributes that you model in the use case diagram. Because use case diagrams are simple in nature, they are free of technical jargon, use case diagrams are a great way to storyboard flows with users. Use case diagrams have another critical role. Use case diagrams define the system requirements being modeled and help write the scenarios later used in testing.

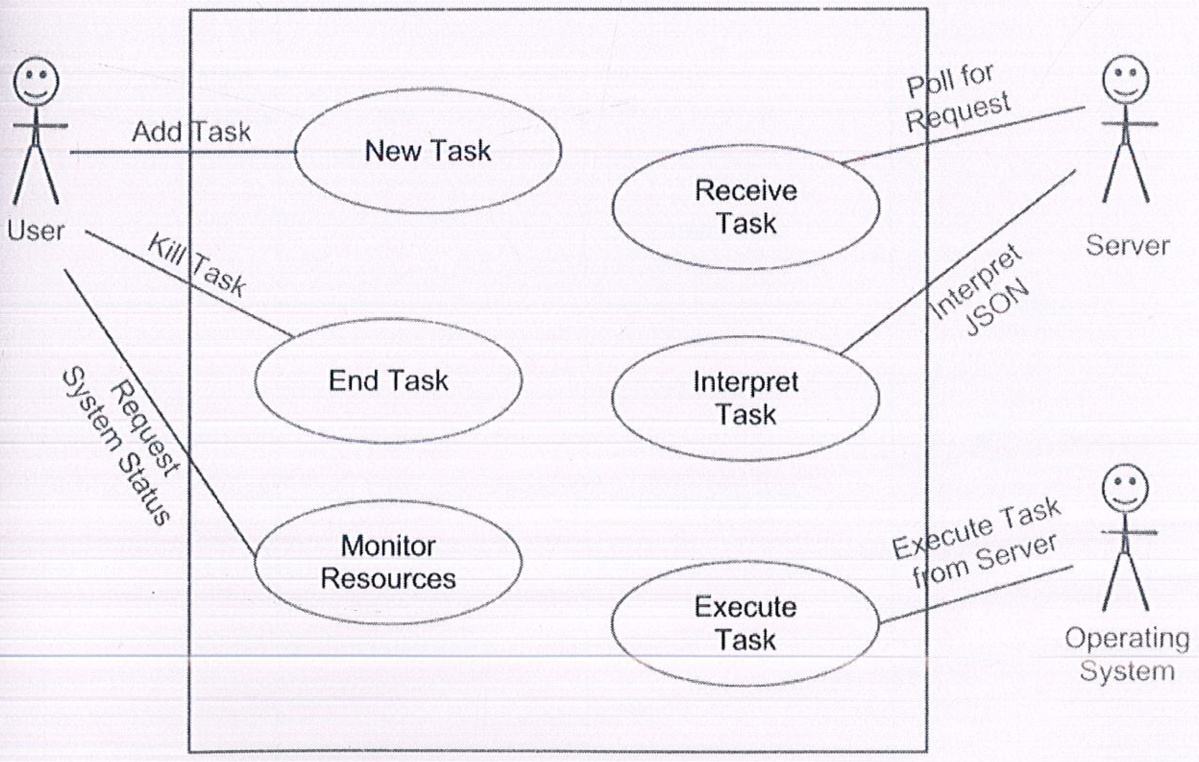


Fig 4.2.3. Use Case Diagram, outlining the use cases.

It is important to outline the actors that will interact with the system and the actions that the actors will perform on the system. This involves activities like starting a new task, ending a task, or monitoring the system resources.

The server is also an actor that will interact with the system. So is the operating system. The server will poll for the request coming from the clients, and execute the commands on the operating system.

## 4.2.3. User Interaction

The interaction overview diagram is similar to the activity diagram both visualizing a sequence of activities. The difference is that the individual activity in the interaction overview diagram is pictured as a frame, which can contain interaction - or sequence diagrams. These interaction/sequence diagrams are constructed with building blocks like: sequence, communication, interaction overview and timing diagram.

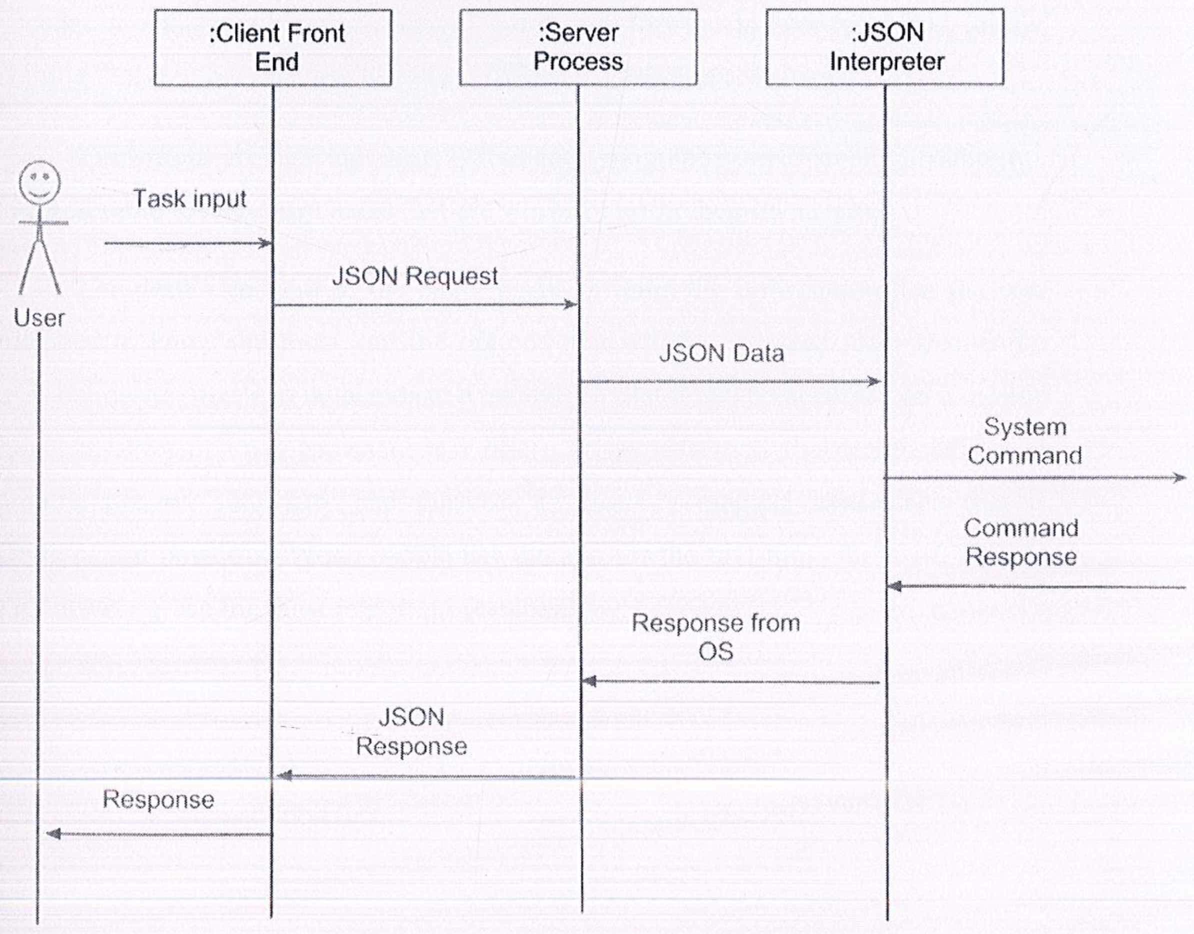


Fig 4.2.4. User Interaction Diagram, outlining the user activities.

The nodes in the diagram connect these sequence diagrams, which can be place in a specific order. With these elements the interaction overview diagram can be used to "deconstruct a complex scenario that would otherwise require multiple if-then-else paths to be illustrated as a single sequence diagram".

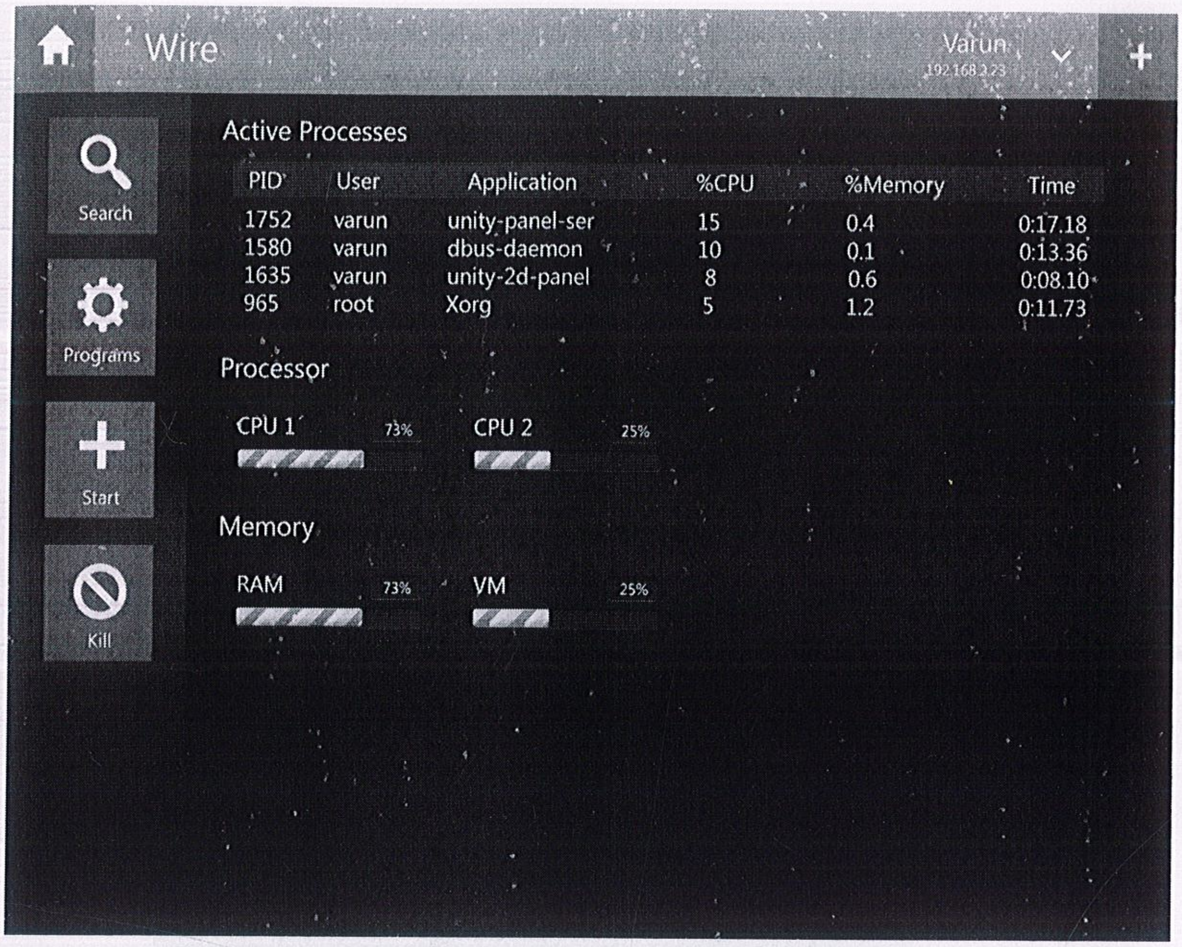
Except for the activity nodes the other notation elements for interaction overview diagrams are the same as for activity diagrams, such as initial, final, decision, merge, fork and join nodes. The two new elements in the interaction overview diagrams are the "interaction occurrences" and "interaction elements".

## 4.3. HTML Client Implementation

Client has been implemented in HTML5 with CSS3 over it to give it a graceful look and feel. The user interface is user centred and as simple as possible so as to make the user feel comfortable when using it. "Simple tasks should never require complex procedures, and complex tasks should get tailored to the human hand and mind." Keeping that in mind, the task has been divided into simple activities.

Transitions are fast and clear; layout and typography are crisp and meaningful. It is preferable to have App icons that are works of art in their own right.

The design concept of the client keeps in mind the information that the user will need to know the most and the options that will be accessed most frequently. Also, the design needs to be responsive enough so that it can be accessed on a mobile phone, a tablet or on a browser. Just like a well-made tool, the design strives to combine beauty, simplicity and purpose to create a magical experience that is effortless and powerful. When people use the app for the first time, they will be able to intuitively grasp the most important features.



Active Processes

| PID | User | Application | %CPU | %Memory | Time |
|---|---|---|---|---|---|
| 1752 | varun | unity-panel-ser | 15 | 0.4 | 0:17.18 |
| 1580 | varun | dbus-daemon | 10 | 0.1 | 0:13.36 |
| 1635 | varun | unity-2d-panel | 8 | 0.6 | 0:08.10 |
| 965 | root | Xorg | 5 | 1.2 | 0:11.73 |

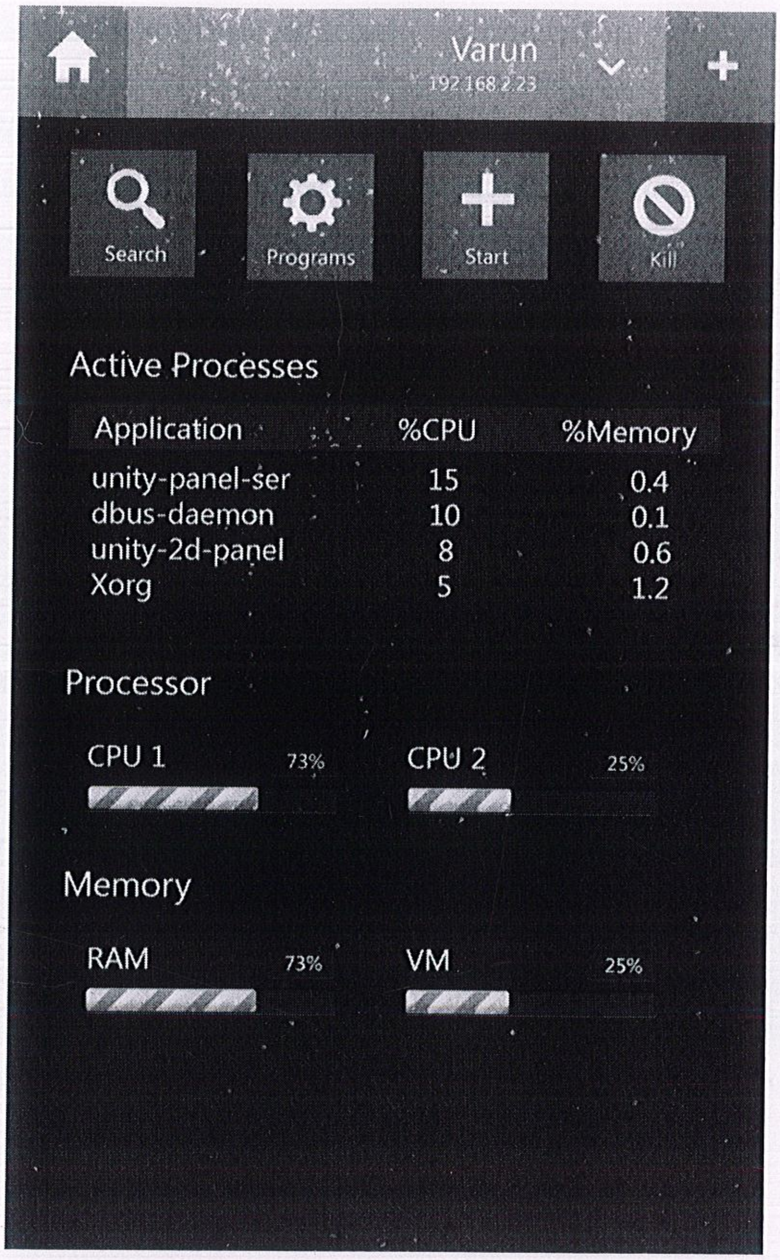
Processor

CPU 1 73%  CPU 2 25%

Memory

RAM 73%  VM 25%

Fig 4.3.1. Design concept of client on a Desktop Browser.



Active Processes

| Application | %CPU | %Memory |
|---|---|---|
| unity-panel-ser | 15 | 0.4 |
| dbus-daemon | 10 | 0.1 |
| unity-2d-panel | 8 | 0.6 |
| Xorg | 5 | 1.2 |

Processor

CPU 1 73% CPU 2 25%

Memory

RAM 73% VM 25%

Fig 4.3.2. Design concept of client on a Mobile Browser.

### 4.3.1. Functionalities Implemented

#### *A. Start Application*

The start application has been implemented in javascript. When the start application button is clicked on the screen, it brings down a small tray which has input field for entering the command and a Go button which fires the event to the javascript Interaction Listener. The command in then encapsulated into a JSON structure and handed over to the Communication API for sending it to the server.

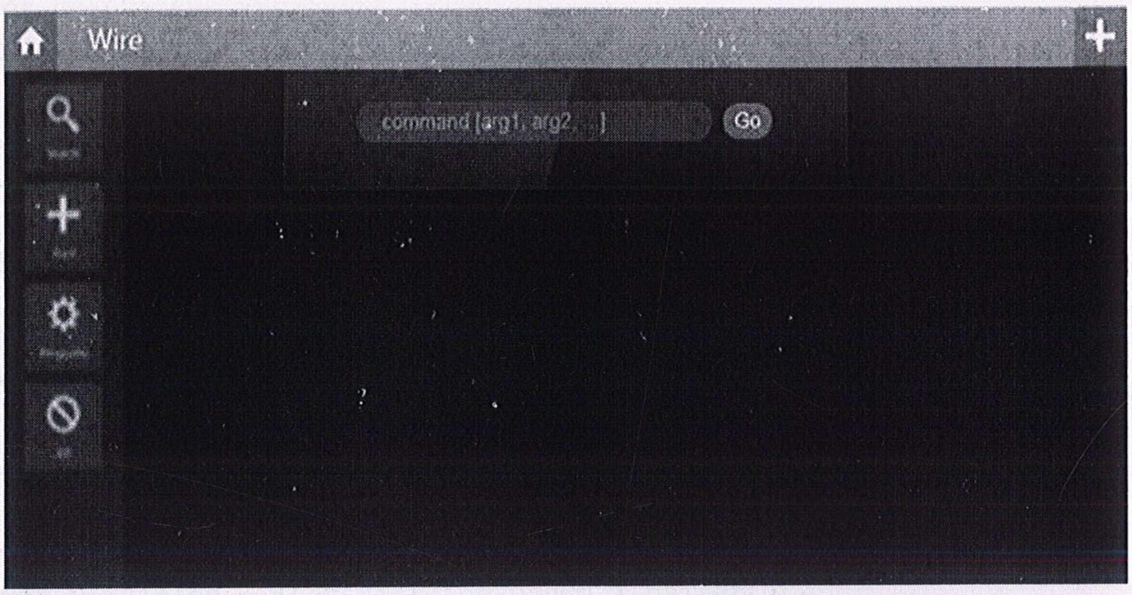


Fig 4.3.3. Snapshot of start functionality (the popup tray with the input field).

The jQuery or Javascript Code that is responsible for the UI interactions is as follows:

```
function registerAppStarter(e){

if($(e).is("#newapp") &&
($("#start-app-container").css("display") == "block"  ||
   $("#search-container").css("display") == "block") && flag){
if($('#start-app-container').css("display")=="block"){
flag = true;
$('#start-app-container').trigger("click");
return;
}
else{
$(".popup_container").trigger("click", function(){
flag = false;
$("#newapp_button").trigger("click");
});
return;
}
}
$("#start-app-container").fadeIn(0);
var marginTop = $("#start-app-popup").height() -
$("header").height();
$("#start-app-popup").css("margin-top", -marginTop);
$("nav").css("-webkit-filter", "grayscale(0.7) blur(2px)");
$("#start-app-popup").animate({"marginTop":"0",
"opacity":"show"},
"slow", "easeOutExpo", function(){
$("#command-name").focus();
});
$('#start-app-container').click(function() {
var marginTop = $("#start-app-popup").height() -
$("header").height();
$("#start-app-popup").animate({"marginTop":-marginTop,
"opacity":"hide"}, 600, "easeOutQuad", function(){
$('#start-app-container').fadeOut(0);
$("nav").css("-webkit-filter","grayscale(0) blur(0px)")
});
});

$('.popup').click(function(event){
event.stopPropagation();
});


}
```

The code that sends the data to the server is as follows:

```
            var json = 'start-app:' + command;
            var url = location.href;
            var xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function(){
            if(xmlhttp.readyState  ==  4  &&  xmlhttp.status  ==
200){
$("#command-name-submit").next("#response").html("<b>Execution
Successful!</b>");
                        $("#command-name").val("");
                        $("#command-name").focus();
                        $("#loader").fadeOut(200);
                  }
            }
            xmlhttp.open("POST", url, true);
            xmlhttp.send(json);
```

## *B. Search Programs*

The search program functionality allows the user to use the autocomplete option to select the command they want to execute. This way, the user doesn't actually need to memorize all the commands he uses. When the user starts typing, a drop down brings up all the related commands.

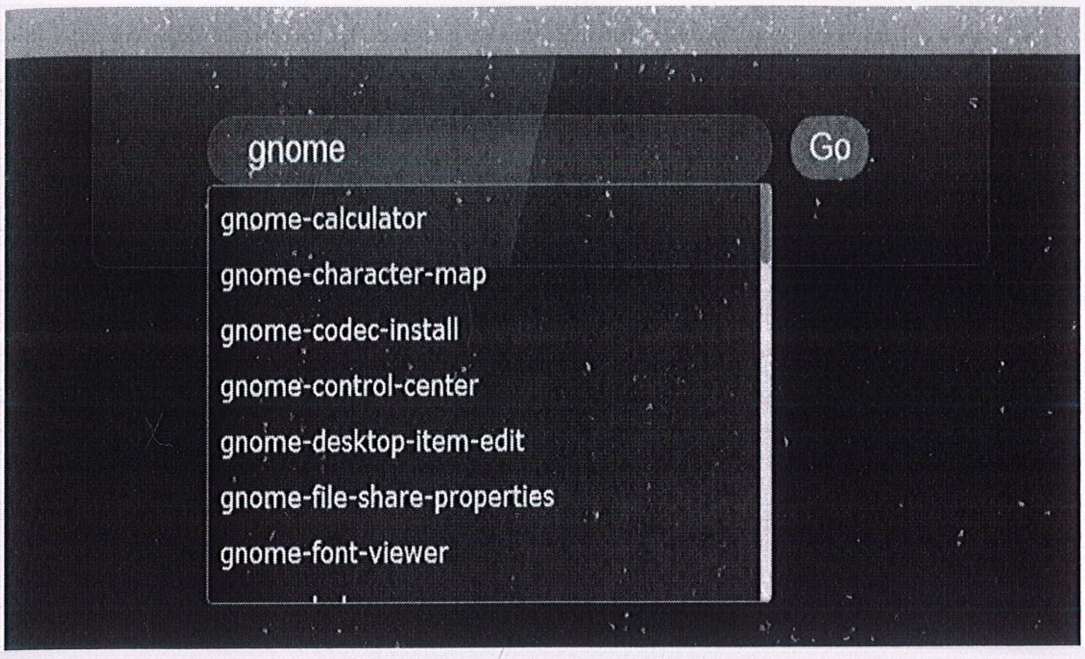


Fig 4.3.4. Snapshot of the search autocomplete functionality.

The javascript interaction listener responsible for bringing up the autocomplete functionality is as follows:

```
function loadAutocomplete(){

     if(autocomplete_flag == true) return;
     $("#loader").fadeIn(200);
     var json = 'list-apps,';
     var url = location.href;
     var xmlhttp = new XMLHttpRequest();
     xmlhttp.onreadystatechange = function(){
          if(xmlhttp.readyState == 4 && xmlhttp.status ==
200){
               var apps = xmlhttp.responseText.split(",");
               $("#command-name").autocomplete({
                    source: apps
               });
           $("#loader").fadeOut(200);
               autocomplete_flag = false;
          }
     }
     xmlhttp.open("POST", url, true);
     xmlhttp.send(json);
}
```

### 4.3.2. Client HTML5 Semantics

The top header is built upon the HTML <header> tag.

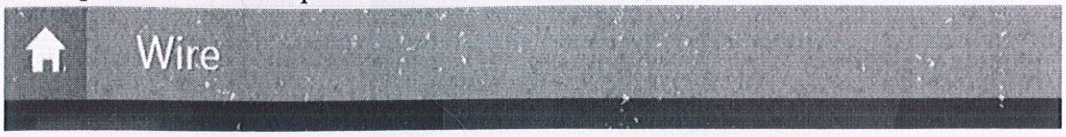


Fig 4.3.5. Snapshot of the Client Webpage Header.

```
<header>
     <div class="button-left"></div>
     <div class="button-right">
           <a href="#" id="newapp" class="button">
                 <img src="images/starttask-icon.png"
           border="0" />
           </a>
     </div>
     <div style="width:85%;margin:auto;margin-top:0.18em;font-
size:2em;color:#FFF;text-shadow:1px 1px 1px
rgba(0,0,0,0.3);">Wire</div>
</header>
```

The left navigation pane of the client has been built upon the nav tag of the HTML5 library.

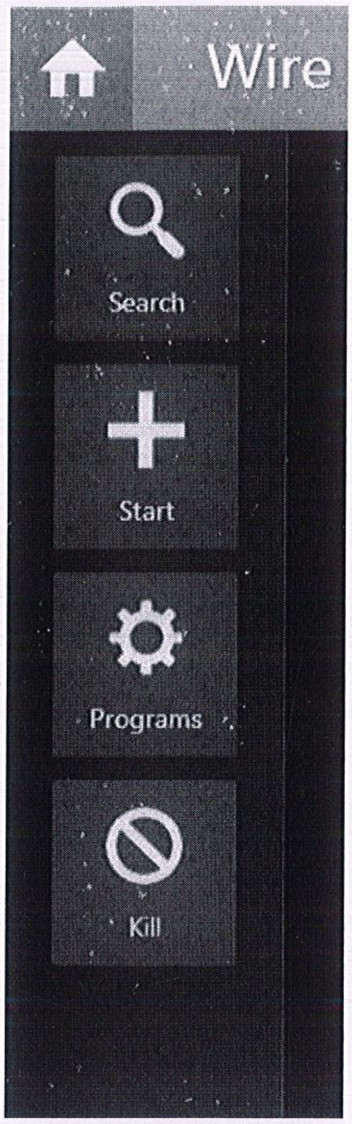


Fig 4.3.6. Snapshot of the Client Navigation Pane.

```
<nav>
<div class="app-button-container">
<div class="app-button" id="search_button">
<div class="app-icon">
<img src="images/search-icon.png">
</div>
<div class="app-name">Search</div>
</div>
<div class="app-button blue2">
<div class="app-icon">
```

```
<img src="images/starttask-icon.png" />
</div>
<div class="app-name">Start</div>
</div>
<div class="app-button blue2">
<div class="app-icon">
<img src="images/programs-icon.png" />
</div>
<div class="app-name">Programs</div>
</div>
<div class="app-button red">
<div class="app-icon"><img src="images/endtask-icon.png"
/></div>
<div class="app-name">Kill</div> </div></div>
</nav>


<div class="popup_container" id="start-app-container">
<div class="popup" id="start-app-popup">
<div class="container">
<input type="text" id="command-name" placeholder="command
[arg1, arg2, ...]" />
<input type="submit" value="Go" id="command-name-submit" />
<div class="response" id="response"></div>
</div>
</div>
</div>

<div class="popup_container" id="search-container">
<div class="popup" id="search-popup">
<div class="container">
<input type="text" id="search" placeholder="command name" />
<input type="submit" value="Search" id="search-submit" />
```

```
<div class="response" id="response"></div>
</div>
</div>
</div>
```

### 4.3.2. Search Autocomplete Functionality

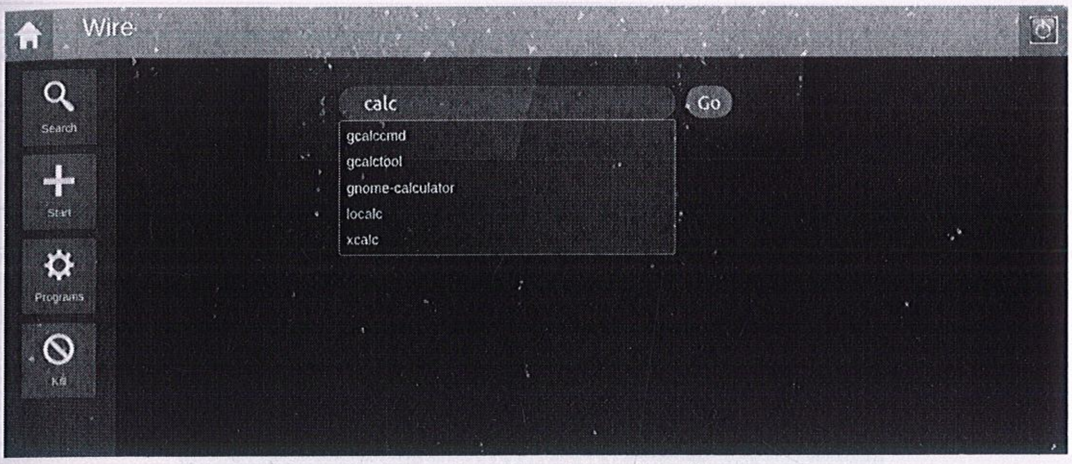


Fig. 4.3.7. Snapshot of Search AutoComplete Functionality.

```
function loadAutocomplete(){
          if(autocomplete_flag == true) return;
          $("#loader").fadeIn(200);
          var json = 'list-apps,';
          var url = location.href;
          var xmlhttp = new XMLHttpRequest();
          xmlhttp.onreadystatechange = function(){

          if(xmlhttp.readyState == 4 && xmlhttp.status ==
200){
                 var apps = xmlhttp.responseText.split(",");
                 $("#command-name").autocomplete({
                             source: apps
                 });
                 $("#loader").fadeOut(200);
```

```
                                        autocomplete_flag = false;
                            }
                    }
                    xmlhttp.open("POST", url, true);
                    xmlhttp.send(json);
}
```

### 4.3.4. Ajax Loader Animation

The Ajax Loader has been implemented using the CSS3 animation frames. HTML5 and CSS3 provides powerful combination to implement animations as nice and beautiful as were implemented using Flash in the earlier days.

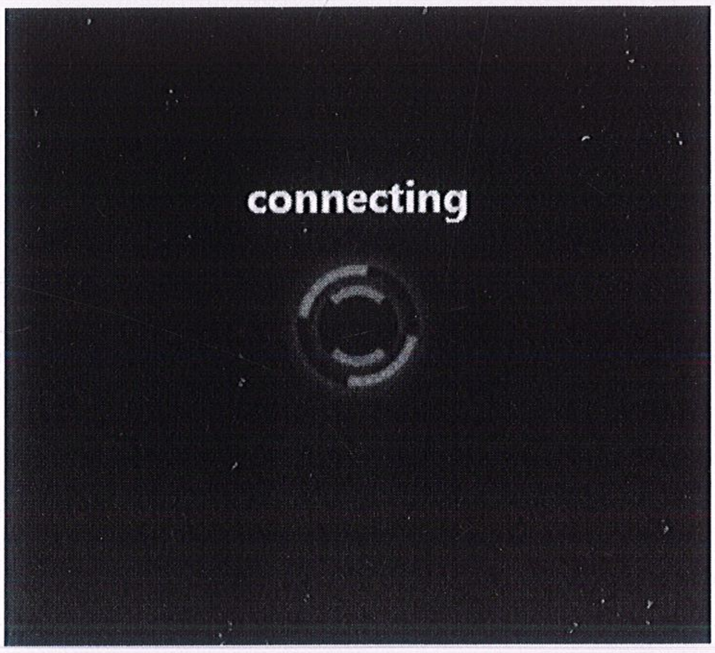


Fig 4.3.8. Loader that appears when communicating with the server.

```
<div id="loader">
      <div class="text">connecting</div>
      <div class="circle"></div>
      <div class="circle1"></div>
</div>

.circle {
```

```
            background-color: rgba(0,0,0,0);
            border:5px solid rgba(0,183,229,0.9);
            opacity:.9;
            border-right:5px solid rgba(0,0,0,0);
            border-left:5px solid rgba(0,0,0,0);
            border-radius:50px;
            box-shadow: 0 0 35px #2187e7;
            width:50px;
            height:50px;
            margin:0 auto;
            -moz-animation:spinPulse 1s infinite ease-in-out;
            -webkit-animation:spinPulse 1s infinite linear;
}
.circle1 {
            background-color: rgba(0,0,0,0);
            border:5px solid rgba(0,183,229,0.9);
            opacity:.9;
            border-left:5px solid rgba(0,0,0,0);
            border-right:5px solid rgba(0,0,0,0);
            border-radius:50px;
            box-shadow: 0 0 15px #2187e7;
            width:30px;
            height:30px;
            margin:0 auto;
            position:relative;
            top:-50px;
            -moz-animation:spinoffPulse 1s infinite linear;
            -webkit-animation:spinoffPulse 1s infinite linear;
}
@-webkit-keyframes spinPulse {
            0% {
                  -webkit-transform:rotate(160deg);
                  opacity:0; box-shadow:0 0 1px #2187e7;
```

```
           }
           50% { -webkit-transform:rotate(145deg); opacity:1;}
           100% { -webkit-transform:rotate(-320deg); opacity:0;
}
}
@-webkit-keyframes spinoffPulse {
           0% { -webkit-transform:rotate(0deg); }
           100% { -webkit-transform:rotate(360deg); }
}
@-webkit-keyframes glow {
           0% { text-shadow:0 0 30px #2187e7, -1px -1px black;
}
           50% { text-shadow:0 0 45px #2187e7, -1px -1px black;
}
           100%  {  text-shadow:0  0  30px  #2187e7,  -1px  -1px
black; }
}
```

## 4.4. Host Server

The server is build on top of a fully functional High Performance Cluster, neweraHPC, designed to work exclusively on TCP/IP protocol layer with extensible plug-in support. Unlike other grid platforms it has a standard library including base functions and a small client program to distribute tasks to node machines. It achieves zero memory leaks even in the worst scenarios, and has a HTTP server integrated to support the seamless communication required by the HTML front end.

The server accepts GET and POST requests from the client, analyzes the JSON data present in the request and uses it to generate system command that needs to be executed on the operating system.

```
#include <iostream>
#include <fstream>
```

```
#include <sys/stat.h>
#include <stdlib.h>

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <include/network.h>
#include <include/general.h>
#include <include/web_ui.h>

using namespace std;

namespace neweraHPC
{
   const char *request_type_strings[] =
   {
      "HTTP_INVALID",
      "HTTP_REQUEST_GET",
      "HTTP_REQUEST_POST",
      "HTTP_RESPONSE_GET",
      "HTTP_RESPONSE_POST"
   };

   rbtree_t *http_handlers;
   void sig_action(int);

   nhpc_status_t        http_handler_register(const        char
*handler_string, fnc_ptr_nhpc_t handler_function)
   {
      fnc_ptr_nhpc_t *func_trigger_local = new fnc_ptr_nhpc_t;
      (*func_trigger_local) = handler_function;
```

```
        nhpc_status_t          rv          =          http_handlers-
>insert(func_trigger_local, handler_string);

        return rv;
    }

    void http_init()
    {
        LOG_INFO("Initialize http handler");
        http_handlers = new rbtree_t(NHPC_RBTREE_STR);
    }

    void http_init(nhpc_socket_t *sock)
    {
        http_data_t *http_data;
        nhpc_status_t     nrv     =     read_headers(sock->headers,
&http_data);
        http_data->sock = sock;
        if(nrv == NHPC_SUCCESS)
        {
        LOG_INFO("HTTP                  Request                  type:
"<<request_type_strings[http_data->request_type]);
        http_request(http_data);

        delete_http_header(http_data);
        delete http_data;
        }
    }

    char* listApps()
    {

        FILE *fpipe;
    char line[50];
    char data[99999];
    int c = 0, i;
    if ( !(fpipe = (FILE*)popen("cat compgen.out","r")) )
    {
     LOG_INFO("Problems with pipe");
     return NHPC_FAIL;
    }

    while (fgets( line, sizeof line, fpipe))
    {
       i = 0;
       while(line[i] != '\n')
       {
           //cout<<line[i];
           data[c++] = line[i];
           i++;
       }
       data[c++] = ',';
    }
    data[c] = '\0';
    return data;
}

char* listProcess()
{
   FILE *fpipe1;
   char str1[50], data[99999];
   int c=-1, i;
   if(!(fpipe1=(FILE*)popen("ps -a","r")))
   {
           perror("Problem with pipe");
   }
```

```
    else
    {
            while(fgets(str1,sizeof str1 ,fpipe1)!= NULL)
            {
                    //printf("%s",str1);
                    if(c == -1){
                            c = 0;
                            continue;
                    }
                    i = 0;
                    while(str1[i] != '\n')
                    {
                            data[c++] = str1[i];
                            i++;
                    }
                    data[c++] = ',';
            }
        //data[c] = '\0';
    }
    pclose(fpipe1);
    return data;
}

int listAppsJSON(nhpc_json_t *json)
{
    FILE *fpipe1;
    char str1[50], data[99999];
    int c=-1, i;
    if(!(fpipe1=(FILE*)popen("cat compgen.out","r")))
    {
            perror("Problem with pipe");
        return NHPC_FAIL;
    }
    else
    {
          json->add_element(JSON_ARRAY, "apps", NULL);
            while(fgets(str1,sizeof str1 ,fpipe1)!= NULL)
            {

              if(strlen(str1) <= 1) continue;
              i = 0;
              while(str1[i] != '\n')
              {
                   str1[i] = str1[i];
                   i++;
              }
              str1[i] = '\0';
              json->add_element(JSON_OBJECT);
              json->add_element(JSON_STRING, "name", str1);
              json->close_element();
            }
         json->close_element();
         //data[c] = '\0';
    }
    pclose(fpipe1);
    json->close_element();
    return NHPC_SUCCESS;
}

int listProcessJSON(nhpc_json_t *json)
{
    FILE *fpipe1;
    char str1[500];
    int c=-1, i;
    if(!(fpipe1=(FILE*)popen("ps u | sort -n","r")))
    {
```

```
                perror("Problem with pipe");
            return NHPC_FAIL;
        }
        else
        {
            json->add_element(JSON_ARRAY, "process", NULL);
                while(fgets(str1,sizeof str1 ,fpipe1)!= NULL)
                {
                    if(c==-1){ c=0; continue;}
                          //printf("%s",str1);
                    i = 0;
                    while(str1[i] != '\n')
                    {
                          str1[i] = str1[i];
                          i++;
                    }
                    str1[i] = '\0';
                    cout<<str1<<endl;
                    json->add_element(JSON_OBJECT);
                    string_t *s;
                    s = nhpc_substr(str1, ' ');
                    json->add_element(JSON_STRING,    "user",     s-
>strings[0]);
                    json->add_element(JSON_STRING,     "pid",     s-
>strings[1]);
                    json->add_element(JSON_STRING,     "cpu",     s-
>strings[2]);
                    json->add_element(JSON_STRING,     "mem",     s-
>strings[3]);
                    json->add_element(JSON_STRING,     "tty",     s-
>strings[6]);
                    json->add_element(JSON_STRING,   "start",     s-
>strings[8]);
```

```
                json->add_element(JSON_STRING,    "time",    s-
>strings[9]);
                json->add_element(JSON_STRING,  "command",  s-
>strings[10]);

                //json->add_element(JSON_STRING, "ps", str1);
                json->close_element();
             }
           json->close_element();
           //data[c] = '\0';
       }
       pclose(fpipe1);
       json->close_element();
       return NHPC_SUCCESS;
   }

/* output test code*/

   int outputProcessJSON(nhpc_json_t *json,char *command)
   {
       FILE *fpipe1;
       char str1[5000];
       int c=-1, i;
       if(!(fpipe1=(FILE*)popen(command,"r")))
       {
             perror("Problem with pipe");
           return NHPC_FAIL;
       }
       else
       {
           json->add_element(JSON_ARRAY, "process", NULL);
             while(fgets(str1,sizeof str1 ,fpipe1)!= NULL)
             {
                    if(c==-1){ c=0; continue;}
                         //printf("%s",str1);
                    i = 0;
                    while(str1[i] != '\n')
                    {
                         str1[i] = str1[i];
                         i++;
                    }
                    str1[i] = '\0';
                    cout<<str1<<endl;
                    json->add_element(JSON_OBJECT);
                    string_t *s;
                    s = nhpc_substr(str1, '\n');
                    json->add_element(JSON_STRING,   "output",   s-
>strings[0]);
                    //json->add_element(JSON_STRING, "ps", str1);
                    json->close_element();
               }
          json->close_element();
          //data[c] = '\0';
     }
     pclose(fpipe1);
     json->close_element();
     return NHPC_SUCCESS;
  }


  void http_request(http_data_t *http_data)
  {
     nhpc_socket_t *sock = http_data->sock;

     if((http_data->request_type)    ==    HTTP_REQUEST_GET    ||
(http_data->request_type) == HTTP_REQUEST_POST)
```

```
      {
      if(((http_data->request_type)  ==  HTTP_REQUEST_POST)  ==
NHPC_SUCCESS)
         {
            if(sock->has_partial_content)
            {
               cout<<"Partial      content:"      <<      sock-
>partial_content << endl;

               char type[20], command[100];
            //memcpy(command, 0, 100);

               int   comma   =   strlen(sock->partial_content)   -
strlen(strstr(sock->partial_content, ","));
               int i, len = strlen(sock->partial_content);
               cout<<"Delimiter Position: "<<comma<<endl;

            for(i = 0; i < comma; i++){
                  type[i] = sock->partial_content[i];
               }
               type[i] = '\0';

               int c = 0;
               for(i = comma+1; i < len; i++){
                  command[c++] = sock->partial_content[i];
               }
               command[c] = '\0';


               cout<<"type: "<<type<<endl;
               cout<<"command:
"<<command<<","<<strlen(command)<<endl;
```

```
            char* response = "";
              if(strstr(type, "list-apps")){
                 nhpc_json_t *apps = new nhpc_json_t;
             listAppsJSON(apps);
             //response = listApps();
                 //cout<<"list:"<<response;
                 nhpc_strcpy(&http_data-
>custom_response_data, apps->get_stream());
              }
           /*if(strstr(type, "kill-app")){
           //char *comm="kill -9 ";
           //sprintf("%s %s",comm,command);
           int code = system(comm);
           nhpc_strcpy(&http_data->custom_response_data,
nhpc_itostr(code));
           }*/
              if(strstr(type, "start-app")){
                 int code = system(command);
                 nhpc_strcpy(&http_data-
>custom_response_data, nhpc_itostr(code));
              }
           if(strstr(type, "shutdown-ps")){
           int code = system("shutdown -h now");
           nhpc_strcpy(&http_data->custom_response_data,
nhpc_itostr(code));
           }
           if(strstr(type, "search-app")){
           char *comm="man ";
           sprintf("%s %s",comm,command);
           int code = system(comm);
           nhpc_strcpy(&http_data->custom_response_data,
nhpc_itostr(code));
           }
```

```
          if(strstr(type, "list-ps")){
           nhpc_json_t *process = new nhpc_json_t;
           listProcessJSON(process);
                //response = listProcess();
                //cout<<"process:"<<response;
                nhpc_strcpy(&http_data-
>custom_response_data, process->get_stream());
              }
         if(strstr(type,"programs-app")){
         nhpc_json_t *process = new nhpc_json_t;
         outputProcessJSON(process,command);
         nhpc_strcpy(&http_data->custom_response_data,
process->get_stream());
         }
             //cout<<sock->partial_contenlt<<endl;
          }


       }

      string_t *tmp_str = nhpc_substr(http_data->request_page,
'/');
      char *app_name = tmp_str->strings[0];

      LOG_INFO("Checking for: " << app_name);
      fnc_ptr_nhpc_t  *func_trigger_local  =  (fnc_ptr_nhpc_t
*)http_handlers->search(app_name);
      if(func_trigger_local != NULL)
      {
         LOG_INFO("Found http handler: " << app_name);
         nhpc_status_t nrv = (*func_trigger_local)(http_data);
      }

      nhpc_string_delete(tmp_str);
```

```
        char *file_path = NULL;
        file_path     =     nhpc_strconcat(HTTP_ROOT,    http_data-
>request_page);

        nhpc_size_t file_size;
        nhpc_status_t nrv;

           if(!(http_data->custom_response_data))
              nrv = nhpc_file_size(file_path, &file_size);
           else
              nrv = NHPC_FILE;

        if(nrv == NHPC_FILE_NOT_FOUND)
        {
           const   char   *mssg   =   "HTTP/1.1   404   Content   Not
Found\r\n\r\nContent Not Found\r\n";
           nhpc_size_t size = strlen(mssg);
           socket_send(sock, (char *)mssg, &size);
        }
        else if(nrv == NHPC_DIRECTORY)
        {
           const   char   *mssg   =   "HTTP/1.1   404   Content   Not
Found\r\n\r\nServer Doesn't Know How To Handle Directory\r\n";
           nhpc_size_t size = strlen(mssg);
           socket_send(sock, (char *)mssg, &size);
        }
        else
        {
           nhpc_headers_t *headers = new nhpc_headers_t;
           headers->insert("HTTP/1.1 200 OK");

              if(!(http_data->custom_response_data))
```

```
            {
               cout << "waiting for file" << endl;
            FILE *fp = fopen(file_path, "r");
            char *file_size_str = nhpc_itostr(file_size);

            headers->insert("Content-Length", file_size_str);

            /* Deciding mime types */
            if(nhpc_strcmp(file_path, "*.json"))
               headers->insert("Content-Type:
application/json");
            else if(nhpc_strcmp(file_path, "*.js"))
               headers->insert("Content-Type:
application/javascript");
            else if(nhpc_strcmp(file_path, "*.css"))
               headers->insert("Content-Type: text/css");



            headers->write(sock);

            delete headers;
            nhpc_string_delete(file_size_str);

            nhpc_status_t nrv;

            char buffer[10000];
            nhpc_size_t len;

            do
            {
               bzero(buffer, sizeof(buffer));
               len = fread(buffer, 1, sizeof(buffer), fp);
```

```
                    nrv = socket_sendmsg(sock, buffer, &len);
                }while(!feof(fp) && errno != EPIPE);

                fclose(fp);
                }
                else
                {
                    nhpc_size_t     len     =     strlen(http_data-
>custom_response_data);
                headers->insert("Content-Length",
nhpc_itostr(len));
                headers->insert("Connection: Keep-Alive");
                headers->insert("Content-Type: application/json");
                headers->insert("Keep-Alive: timeout=5, max=100");
                headers->write(sock);
                delete headers;
                    socket_sendmsg(sock,                    http_data-
>custom_response_data, &len);
                }
            }

            nhpc_string_delete(file_path);
            }
            else if((http_data->request_type) == HTTP_INVALID)
            {
            const    char    *mssg    =    "HTTP/1.1    403    Invalid
Request\r\n\r\nInvalid request\r\n";
            nhpc_size_t size = strlen(mssg);
            socket_send(sock, (char *)mssg, &size);
            }
    }
    nhpc_status_t    http_get_file(const    char    **file_path,
nhpc_socket_t  *sock,  const  char  *target_file,  const  char
*host_addr)
    {
        nhpc_create_tmp_file_or_dir(file_path,  "/tmp/neweraHPC",
NHPC_FILE);

        const    char    *command    =    nhpc_strconcat("GET    /",
target_file, " HTTP/1.1");
        nhpc_headers_t *headers = new nhpc_headers_t;
        headers->insert(command);
        headers->insert("User-Agent: neweraHPC");
        headers->insert("Host", host_addr);
        headers->write(sock);
        delete headers;
        nhpc_string_delete((char *)command);

        FILE *fp = fopen(*file_path, "w+");
        nhpc_status_t nrv;
        nhpc_size_t size;
        nhpc_size_t size_downloaded = 0;
        nhpc_size_t file_size;

        char buffer[10000];
        nhpc_size_t header_size = 0;

        do
        {
        bzero(buffer, sizeof(buffer));
        size = sizeof(buffer);
        header_size = 0;

        do
        {
            nrv = socket_recv(sock, buffer, &size);
        }while(nrv != NHPC_SUCCESS && nrv != NHPC_EOF);

        if(sock->have_headers == false)
        {
            nrv    =    nhpc_analyze_stream(sock,    buffer,    &size,
&header_size);
            if(nrv == NHPC_SUCCESS)
            {
                http_content_length(sock->headers, &file_size);
                nhpc_display_headers(sock);
            }
        }

        fwrite((buffer + header_size), 1, (size - header_size),
fp);

        size_downloaded += (size - header_size);

        }while(nrv != NHPC_EOF && size_downloaded != file_size);

        cout<<"Size Downloaded: "<<size_downloaded<<endl;

        fclose(fp);

        return nrv;
    }

    void http_response(nhpc_socket_t *sock)
    {

    }
```

```
};
```

### 4.4.1. Functionalities Implemented

*A. Command Interpreter*

The command interpreter strips out the information about the type of command it needs to perform from the HTTP Data that is receives from the client. It then uses that command type to perform the corresponding operation. sock->partial_content gives the HTTP Request Body from the client request that has been accepted.

```
char type[20], command[100];
int comma = strlen(sock->partial_content) -
            strlen(strstr(sock->partial_content, ","));

int i, len = strlen(sock->partial_content);

cout<<"Delimiter Position: "<<comma<<endl;

for(i = 0; i < comma; i++){
  type[i] = sock->partial_content[i];
}
type[i] = '\0';

int c = 0;
for(i = comma+1; i < len; i++){
  command[c++] = sock->partial_content[i];
}
command[c] = '\0';

cout<<"type: "<<type<<endl;
cout<<"command: "<<command<<endl;
```

## *B. Autocomplete Request Handler*

When the server receives the request for listing the applications that are installed on the user machine, it looks up the installed applications using the compgen package installed on the linux operating system. It then encapsulates the application list into a JSON Response Structure and sends it back to the client.

```
char* list_apps()
{
            FILE *fpipe;
            char line[50];
            char data[99999];
            int c = 0, i;
            if ( !(fpipe = (FILE*)popen("cat compgen.out","r")) 
)
            {
            LOG_INFO("Problems with pipe");
            return NHPC_FAIL;
            }

            while (fgets( line, sizeof line, fpipe))
            {
             i = 0;
             while(line[i] != '\n')
             {
                        //cout<<line[i];
                        data[c++] = line[i];
                        i++;
             }
             data[c++] = ',';
            }
            data[c] = '\0';
            return data;
}
```

*C. Application Start Handler*

When the server receives a request to start a new application, it strips out the information about the application that needs to be started and the arguments that will be passed to the application during execution time.

The command is provided by the command interpreter.

```
if(strstr(type, "start-app")){
        int code = system(command);
        nhpc_strcpy(&http_data->custom_response_data,
                         nhpc_itostr(code));
}
```

The application start handler then copies the response code returned by the application execution to the response data which sent back to the client by the communication api.

## 4.5. Running a Gnome Calculator using Wire

### Step 1: Open the Wire App

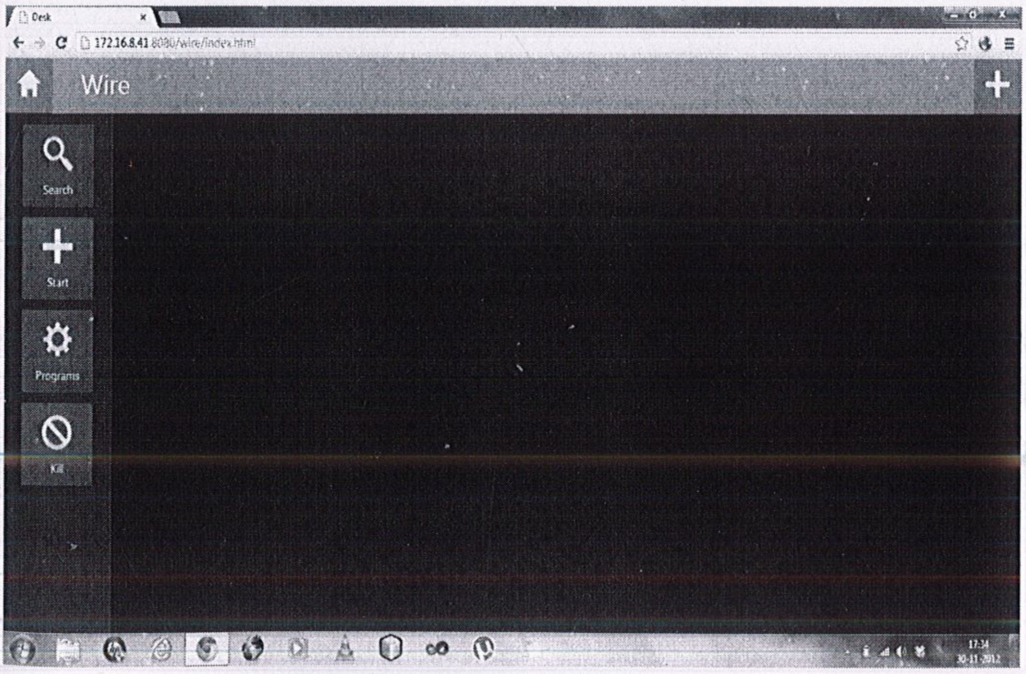


Fig. 4.5.1. Snapshot of Wire App Page.

## Step 2: Start a new Application

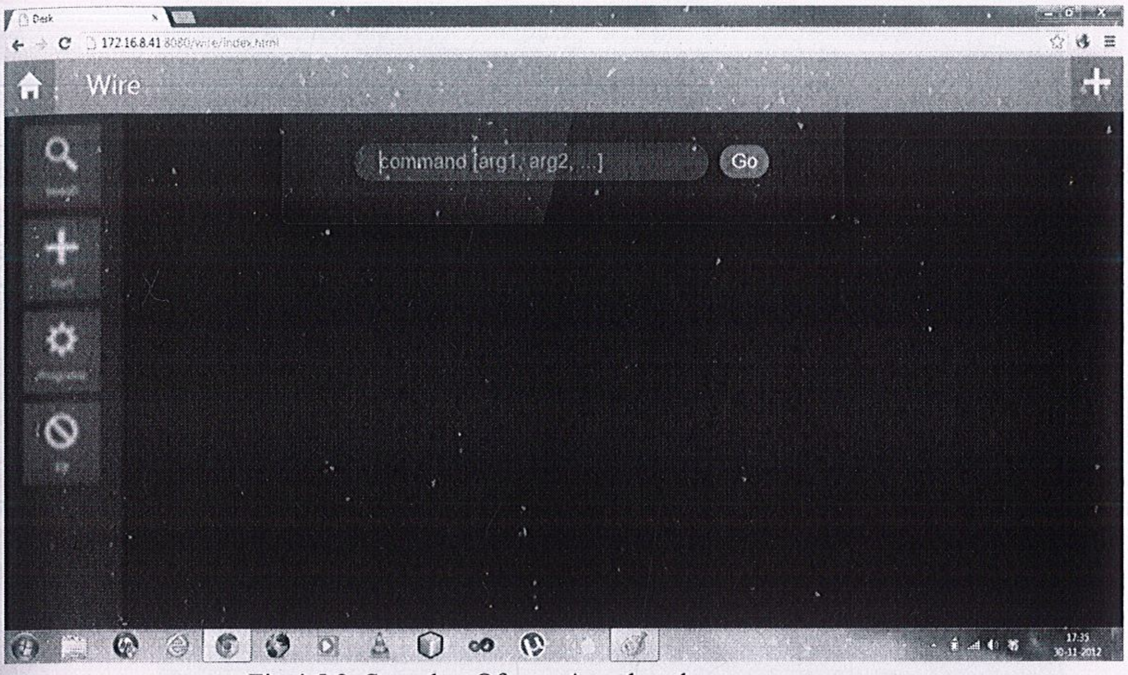


Fig 4.5.2. Snapshot Of new App dropdown menu.

## Step 3: Type in the command name

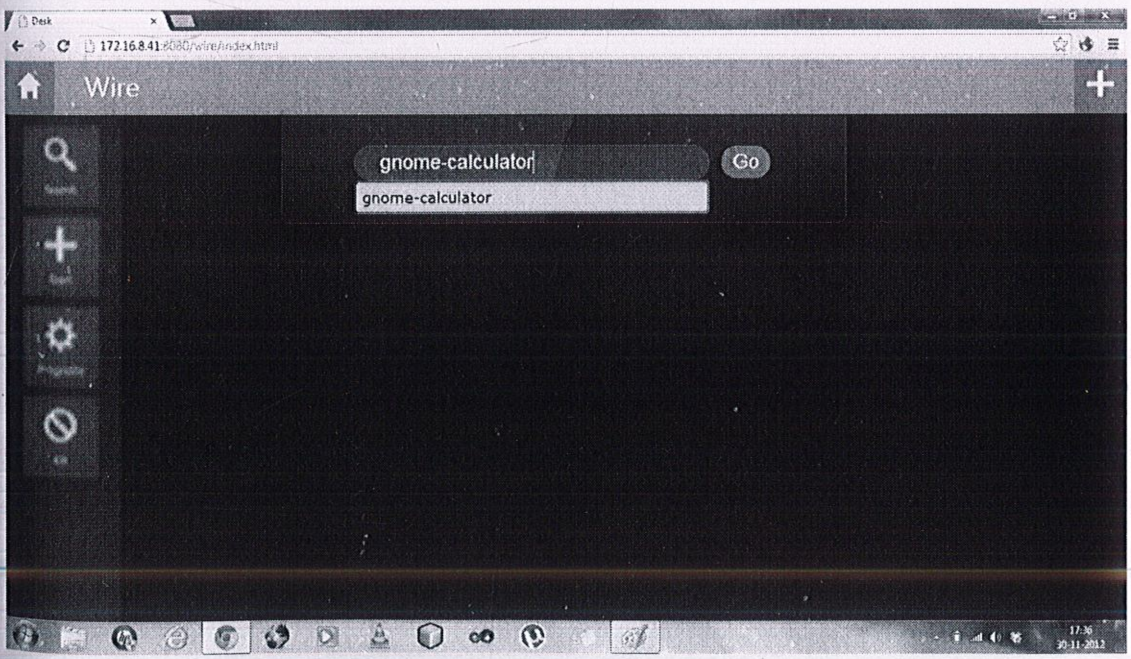


Fig 4.5.3. Snapshot Of AutoComplete in action

**Step 4: The Application is started**

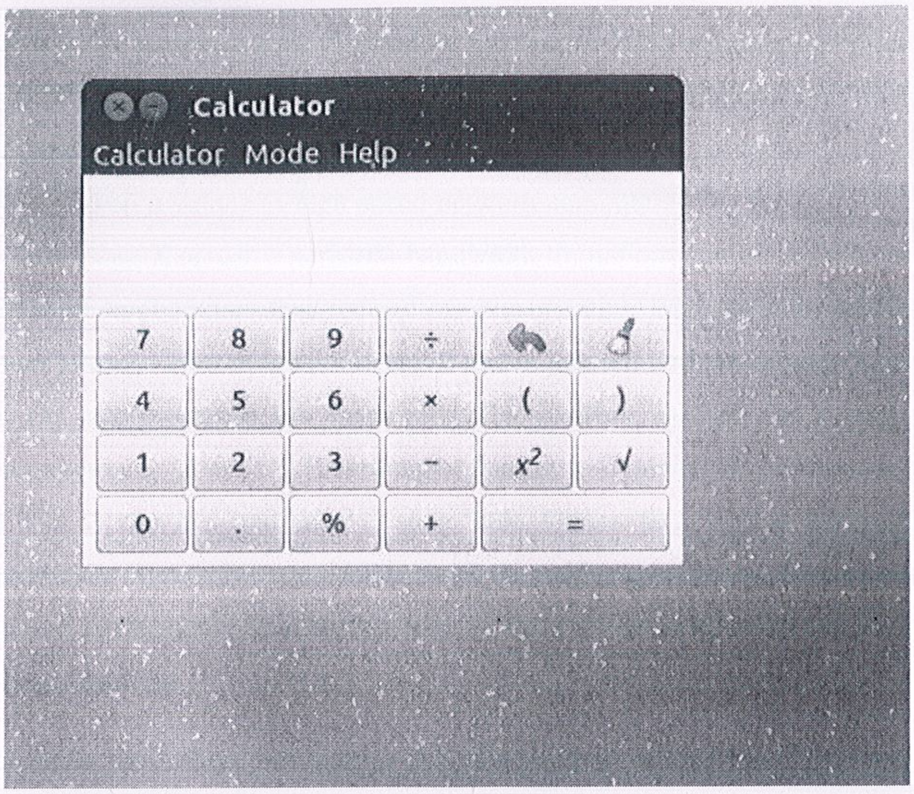
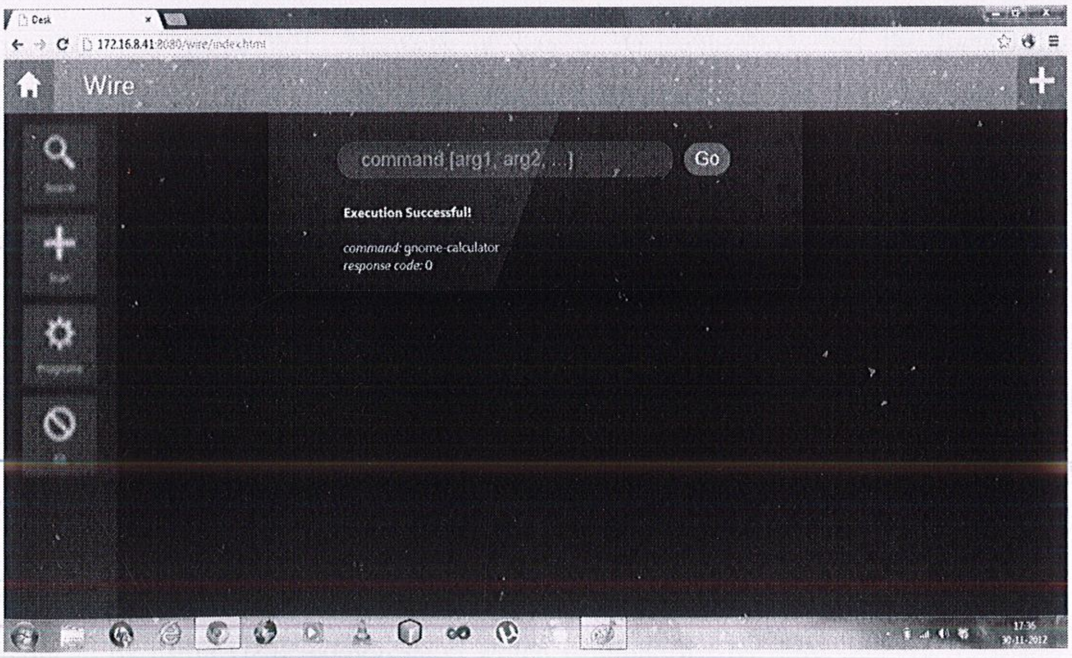


Fig 4.5.4. Snapshot Of Running application

**Step 5: System Response**



Execution Successful!

*command:* gnome-calculator
*response code:* 0

Fig 4.5.5. Snapshot Of execution response.

# Chapter 5. CONCLUSION AND FUTURE SCOPE

So as discussed in our section 2.2 the X Window System (or X11) has long supported the basic need of graphical user interfaces (GUIs) and rich input device capabilities for networked computers. X11 was published in 1987, and there is no doubt that it is doing a great job when it comes to high speed network computers. The problem with X11 is that when using X across a network, bandwidth limitations can hinder the use of bitmap-intensive applications that require rapidly updating large portions of the screen with low latency. Even a relatively small uncompressed 640x480x24bit 30fps video stream can easily outstrip the bandwidth of a 100Mbit network for a single client. Another approach called NX Technology attempts to improve the performance to the point that it can be used over a slow link. Others like Virtual Network Computing (VNC) or Xpra solve some issues with the earlier approaches. But, to put it simply, the best performance is achieved only on thick clients with relatively high speed network connections.

At the same time, there has been a constant need to control one's personal computer or to reliably monitor the state of a program, from handheld devices like mobile phones or tablet computers. With high end smartphones coming up, be it Android, iOS or the Windows platform, developers have tried to successfully build very native VNC applications for the handheld devices. They work great if the device is connected on a WiFi Network or has a 3G connection, but the bandwidth consumption is still too much. In developing countries like India, where the network providers are still trying to bring the 3G technology to masses, the cost of running a VNC application on a mobile device is too high.

The utility will allow the user to manage processes on his system as well as monitor the running processes, all from a handheld device or even a browser running on a remote computer. The core sentiment behind this approach is to minimize the data being transferred. This can be accomplished by using a lightweight data interchange format called JSON (Javascript Object Notation) for the transfer of only relevant data, unlike the X windows server, which transfers large amount of bitmap image data across the network. The data being transferred on network can further be reduced by using GZIP compression which is supported by most modern browsers, both mobile and computer based. The user interface of the application will be based on HTML5 and Cascading Style Sheets.

| Technology | Bandwidth | Graphics Display | Message passed using | Portable |
|---|---|---|---|---|
| X Windows System | High | Yes | Bit Map Form | No |
| Virtual Network Computing (VNC) | High | Yes | Bit Map Form | Yes |
| NX Technology | Medium | Yes | Encrypted SSH Sessions | Yes |
| Remote Desktop Protocol (RDP) | Medium | Yes | T-120 | Yes |
| RSML | Low | No | JSON | Yes |

TABLE 5.1. : Comparison between protocols

From the table shown above we can see the major differences between the protocols that exist and is being recommended. RSML although cannot display the graphic interface at the user end but still can be used for networks where high bandwidth is not an option. Also the portability clause is easily matched using RSML because of a lightweight front end.

With protocols such as VNC which provide an excellent graphic interface to the client because of the bit map transfer over the network a huge bandwidth is required which we find seldom in developing countries where 3G is still a new concept and therefore is quite costly. The same problem exists with other technologies too apart from the portability issues, no matter how well they perform on a high bandwidth network when it comes the using such technologies over a 2G mobile network they all tend to collapse.

# Chapter 6. References

[1] R. W. Scheifler, J. Gettys, The X Window System, Digital Equipment Corporation and MIT Project Athena, April, 1986.

[2] T. Richardson, Q. Stafford-Fraser, K. R. WOOD, A. HOPPER, Virtual Network Computing, The Olivetti & Oracle Research Laboratory, February, 1998.

[3] https://market.android.com/ as accessed in August, 2012.

[4] http://www.w3schools.com/ajax as accessed in August, 2012.

[5] http://www.json.org/ as accessed in August, 2012.

[6] http://json-p.org/ as accessed in August, 2012.

[7] http://stackoverflow.com/questions/145110/c-performance-vs-java-c as accessed in August, 2012.

[8] http://www.teamviewer.com/images/pdf/Teamviewer_SecurityStatement.pdf last updated in May, 2012.

[9] http://html5demos.com/ and http://www.w3.org/html/logo/ as accessed in September,2012.

[10] http://www.ehow.com/list_7408003_vnc-bandwidth-requirements.html as accessed in September, 2012.

[11] http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html

[12] http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html

[13] http://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html

[14] http://www.w3.org/TR/css3-animations/

[15] http://www.w3schools.com/html/html5_intro.asp

[16] http://www.cgisecurity.com/lib/XmlHTTPRequest.shtml

[17] http://html5doctor.com/methods-of-communication/

[18] http://www.w3.org/TR/XMLHttpRequest/

[19] http://www.w3schools.com/xml/xml_http.asp

[20] http://www.w3schools.com/ajax/ajax_xmlhttprequest_send.asp

[21] Agile Software Development, Software Engineering (9th Edition) by Ian Sommerville

[22] Design and Implementation, Software Engineering (9th Edition) by Ian Sommerville