Infrastructure Development for a Mind Attention Interface

Zhen Yang

September 2007

A thesis submitted for the degree of Master of Philosophy
of the Australian National University

This thesis is dedicated to my wife who is the inspiration in my life.

Declaration

The work in this thesis is my own except where otherwise stated.

Zhen Yang

01/09/2007

Acknowledgements

First, I would like to thank my parents. Without their support, this document would not have been written.

Secondly, I would like to thank my supervisor Henry Gardner, Andrew James and Peter Christen for their generosity, kindness, guidance and ideas they has contributed.

Thirdly, many thanks to my research colleagues and group mates: James Sheridan, Ben Swift and Torben Schou. As the senior research student in the group, James has provided much leadership and careful drive in the development of the Mind Attention Interface for the Wedge Theatre. I have had many long and useful conversations with both of them, and these interactions helped a lot when testing and improving my software.

I would also like to thank Samuel Inverso, we had worked together to contribute the BioSemi driver for BCI2000 project. Thanks to Hugh Fisher, who manages IT facilities in the group. Last, but not least, thanks to the department of computer science, who provide tutoring opportunities, scholarship and coffee for the last two years.

Abstract

The "Mind Attention Interface" (MAI) enables sensor measurements of a person's mind states, and attention states, to be used in a virtual environment. As well as serving as a platform for Human Computer Interface and Brain-Computer Interface research, the MAI aspires to build artistic installations, particularly in computer-generated music.

This thesis describes the development of the MAI System: the enabling software infrastructure of the Mind Attention Interface. It discusses the investigation and purchase of hardware and the design of the MAI System architecture. The requirements, design, and implementation of the MAI System, accompanied by a set of profiling tests, demonstrate the effectiveness of the design architecture and the overall usefulness of the system.

Contents

List of Terminology

Terminology

6DOF	Six Degree of Freedom
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BCI	Brain-Computer Interface
BDF	BioSemi Data Format
CPU	Central Processing Unit
DAQ	Data Acquisition
EDF	European Data Format
EEG	Electroencephalogram
fMRI	functional magnetic resonance imaging
GUI	Graphic User Interface
HCI	Human Computer Interface
MAI	Mind Attention Interface
MASE	Mind Attention Spectral Engine
MSS	Maximum Segment Size

NTP	Network Time Protocol
SDK	Software Development Kit
SECK	Spectral Engine Connection Kit
SMR	Sensorimotor Rhythm
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UTC	Universal Time Coordinated
VE	Virtual Environment
VR	Virtual Reality

Chapter 1

Introduction

For a long time, people have dreamt of using their mind as direct input to control machines. Researchers on Brain-Computer Interfaces (BCIs) have been working to build practical systems which can be used by disabled people and there have been significant results which show that computers can be slowly but effectively controlled by a human mind [1, 64]. "Mind" here refers to the combination of thought, emotion, perception, imagination, and other brain states. In the Mind Attention Interface (MAI), we intend to measure mind states by using Electroencephalograms (EEG).

For some time, psychology and neuroscience researchers have had a research focus on attention. Attention is the cognitive process of selectively concentrating on things that the mind is interested in [36]. It is one of the cognitive processes which is considered as having a most concrete association with the human mind, and it is closely linked with perception [41]. Audio and visual cues for attention are another source that we can use to help improve communication with computers. In the MAI, measurements of attention such as gaze, head or hand movements are obtained from dedicated recording systems.

The idea of the Mind Attention Interface is to create a platform for interacting with a virtual reality theatre using EEG and measures of attention. This thesis describes the construction of such extensive hardware and software platform,

which is called the MAI System. The rest of this chapter introduces the most important concepts behind the Mind Attention Interface.

1.1 Brain-Computer Interfaces

1.1.1 Background

A Brain-Computer Interface (BCI) is an interface between a person's brain and a computer. It is a computer interface that only uses signals from the brain, and which does not require any user motor activity [2] such as eye or other body potentials. Research on BCIs started in the 1970s and have developed rapidly in the 1990s [26]. BCIs can be classified in two major categories: invasive and non-invasive. In invasive BCIs, electrodes are implanted into the brain during neurosurgery, and these systems diagnose and repair damage brain functions. (Invasive BCIs will not be discussed further in this thesis) Non-invasive BCIs, on the other hand, detect brain signals from the surface of the brain, such as the scalp of a user, using an electrode cap. Electroencephalography (EEG) is the most commonly used non-invasive BCI due to its fine temporal resolution, ease of use and cost [64]. By converting the acquired analogue signals to digital, and sending them to a computer for processing, a BCI can be used to control certain electronic devices.

1.1.2 Electroencephalographic BCIs

The first electroencephalogram (EEG) recording was obtained by Hans Berger in 1929 [19]. An EEG is a graph of measurements of the electrical activity of the brain as recorded from electrodes placed on the scalp or on the cortex itself. Because electrical activity is closely involved in the way the brain works, EEG provides "direct measurements" of brain functions in comparison with technologies which rely on blood flow or metabolism which maybe decoupled from brain activity. Traditionally, this non-invasive technology has several limitations:

Firstly, scalp electrodes are not sensitive enough to pick out action potentials of individual neurons. Instead, the EEG picks up signals from the synchronised groups of neurons. Secondly, identifying the source locations of measured EEG potential is an invoice problem which means that these desired locations are very non-specific. In contrast with other brain-imaging techniques such as functional magnetic resonance imaging (fMRI) *, the spatial resolution in the region could be less than 3 millimetres. Thirdly, due to its susceptibility to noise, this method can be largely affected by the surrounding environments, such as the 50Hz AC noise (50Hz is the mains power frequency in Australia). On the other hand, EEG has several positive aspects as a tool of exploring brain activity. As mentioned above, it is an affordable system for most research groups. The setup and scalp preparation for modern EEG systems is relatively straightforward to use. In particular, its temporal resolution is very high. EEG has a time resolution down to sub-milliseconds compared with other methods for researching brain activities, which have time resolution in the order of seconds or even minutes.

The types of waves measured in the EEG signal are studied together with mental states. Major types of continuous, sinusoidal EEG waves are delta, theta, alpha, beta, and gamma. The only difference between these signal types is the frequency range † . They are listed in Table 1.1 with associated mental states.

Electroencephalographic BCI applications can provide disabled users with a new channel for sending messages to the outside world [67]. In [24], slow cortical potential (SCP) was used to control cursor movements on a computer screen. This rhythm shows up as large drifts in the cortical voltage which last from a few hundred milliseconds up to several minutes. In [67], similar cursor movement BCI is implemented by controlling the amplitudes of the sensorimotor activity signals such as the mu rhythm in the alpha frequency band (8 -12Hz). Other commonly

*Functional fMRI is the use of magnetic resonance imaging to measure the haemodynamic response related to neural activity in the brain. MRI is a non-invasive method used to render images of the inside of an object.

†There is no precise agreement on the frequency ranges for each type.

wave	frequency range	associated mental states
Delta	0-4 Hz	certain encephalopathy, deep sleep
Theta	4-8 Hz	trances, hypnosis, lucid dreaming, light sleep
Alpha	8-12 Hz	relaxation, calmness, abstract thinking
Beta	12+ Hz	anxious thinking, active concentration
Gamma	26 - 100 Hz	higher mental activity including perception, problem solving, fear, and consciousness

Table 1.1: EEG waves associated with mental states [13]

used signals in BCIs are Event-Related Potentials (ERPs), Visual Evoked Potentials (VEPs), and Steady State Visual Evoked Responses (SSVERs) [65, 66].

Among these signals, an ERP response to unpredictable stimuli called the P300 (or simply P3) is one of the most robust [11]. The P300 ERP appears as a positive deflection of the EEG voltage at approximately 300ms. Based on this phenomenon, P300 character recognition is designed in [14] to choose characters for disabled users: A 6×6 grid containing alphabet letters and numbers is displayed on screen and users are asked to count the number of times the row or column flash which contains the character they wanted to select. Every twelve flashes are called a set, which all rows and columns have flashed once. P300 amplitude is measured and the average response to each row and column is used to determine the selected character.

Although there are many existing BCI systems in use at present, the extensive training required before users obtain such technology and the slowness in com-

munication is the substantial barrier of using BCI systems. One implementation by Wolpaw in [66] using mu rhythm achieved 10.88 binary decisions per minute with 2 month training. In another word, a typical cursor move from the centre of a video screen to a target located at the top or bottom edge will take 3 seconds. Another example would be P300 character recognition system implemented by Donchin in [11], which achieved 9.23 binary decisions per minute online classification rate. The classification rate is the way BCI uses to compare the speed of the system. In P300 for example, more than five binary decisions (correct ones) are needed to choose a single symbol as there are totally 36 symbols. To summarise, users are normally trained through biofeedback for several weeks to months before they make use of the systems, and online classification rate are less than 15 binary decisions per minute [48].

1.2 Attention and the Mind as a Computer Interface

1.2.1 Attentive Interface

The so-called "Attentive Interface" is a relatively new category of user interface, which dynamically prioritises the information it presents to its users [59]. In [47], Selker gives a definition of Attentive Interface as: *"Context-aware human-computer interfaces that rely on a person's attention as the primary input"*. Here attention is epitomized by eye gaze which is one of the key features that have been widely used in Attentive Interfaces. Figure 1.1 shows a typical eye-gaze tracking system. In recent years, interactive applications using eye tracking have improved to the stage that they now allow people to type, draw, and control an environment[27]. In these interfaces, the synchronised rotation of the eyes is often used as a pointing device to point to and select a graphical object located at the intersection of the eye-gaze vector and the screen (see [53] for example).

Blinking, eye gestures and movements, as well as eye dwell-time can all be used in selection and control [57]. At present, training is essential to use attentive interfaces because none of the above methods is natural. Some experts believe that ongoing research may bring us generation interfaces with more natural eye movements for selection and control [39].

Figure 1.1: Gaze system by SeeingMachines^{TM}

1.2.2 Mind and Attention for Interaction with a Virtual Environment

In virtual environments, real-world human movements can be mapped to actions in a virtual world. The effect of these actions is often to do with manipulation of the entire objects in 3D spaces. Although most emphasis has been placed on physical controllers together with tracking systems, eye gaze has also been considered as an interface to a virtual environment [18]. Tanriverdi and Jacob, in [53], claim that eye movement-based interaction offers the potential of easy, natural, and fast ways of interacting in virtual environments. A comparison of the performance that eye movement and pointing based interactions in virtual environments was undertaken, and the results show that eye gaze based technique has speed advantage especially in distant virtual environment.

Bayliss in [3, 5] demonstrate examples of EEG based BCI systems in control of virtual environments. Her past research in [4] also gives us a clue of the effects that eye tracker and virtual environment impact on EEG recording, and the results indicate that neither eye tracker nor virtual environment will introduce significant noise than computer monitors to the EEG signal quality. Therefore, the Mind Attention Interface idea seems realistically, which all devices would work together without affecting others.

My infrastructure goal was to build a fundamental framework for Wedge virtual reality theatre that supported mind and attention data from different sources, in different formats, and which would go beyond a simple BCI interface. Firstly, the real-time analysis of brain states from EEG was planned to be quite sophisticated and to go beyond a simple classification of signals into frequency bands. Secondly, it was planned to provide natural feedback of a user's brain states to the evolution of an immersive virtual experience which included 3D graphics and surround sound. Thirdly, the measurements of eye-gaze-based attention from a user in virtual environment were intended to provide feedback on the brain state of a user. Eye gaze can also play a role in filtering EEG signals for interference from oculomotor events. Head position, orientation, and other control devices can also be used to interact with the virtual world. Figure 1.2 shows the feedback loop that we intend to create in the MAI.

1.3 This Thesis

This thesis describes infrastructure development for a "Mind Attention Interface". This Mind Attention Interface enables sensor measurements of a subject's mind states and attention states to be used as control and feedback data for a virtual environment. Sensor devices measure EEG brain activity, eye gaze, and head and hand movements. The MAI has been built for use in the Wedge, a two-walled immersive virtual environment which combines stereoscopic computer graphics with surround sound. Its goal is to contribute to Human Computer Interface

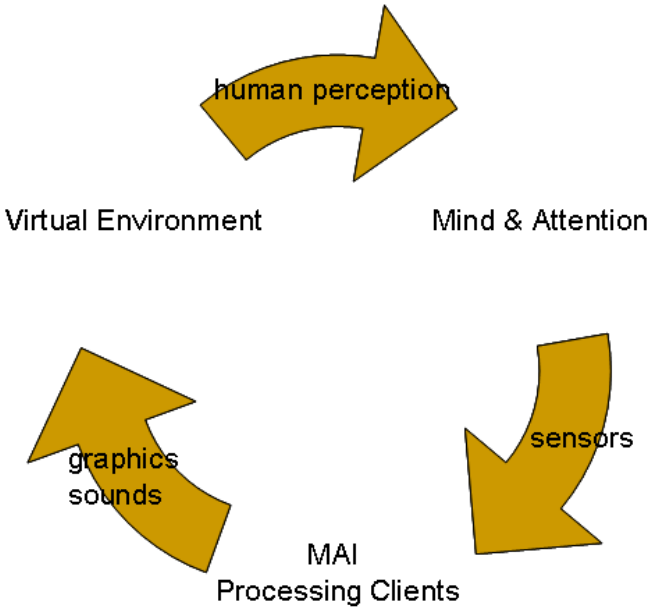

Figure 1.2: Feedback loop in the MAI

(HCI) and Brain-Computer Interface (BCI) research in virtual environments, and, by so doing, provide new insights into the human behaviour and the human mind.

The infrastructure of the MAI described here consists of four major layers: The first layer is the Data Acquisition (DAQ) layer which handles the device drivers, data transmission over the network, and timing. The second layer is a central server that handles peer connections, header information and provides the application layer clients with a gateway to access data over the network. The third layer is the signal-processing layer, which processes raw data and provides high-level information for further usage. The Spectral Engine Connection Kit (SECK) enables multiple processing modules to be used in this layer and thus distributes processing over the network. The last layer in the MAI is the application layer that connects to the MAI and display information in the virtual environment. My major contributions are the MAI System, which include first and second layers and software libraries created for the fourth layer. Although the signal-processing layer is largely the work of others, its interfacing components are also

a contribution of the present thesis.

When I commenced this project, the MAI had just been funded and no MAI infrastructure existed. In the last one and half years, my research has included the investigation and purchase of hardware, and the development of software for the MAI framework. This thesis describes my research process: System hardware and associated software are described in Chapter 2. Chapter 3 specifies the MAI System software platform requirements and states the studies over a set of comparable systems. Chapter 4 contains a discussion of the MAI software design considerations based on the requirements. Chapter 5 describes the implementation of the system enabling components. Utilities that create supportive environments for carrying out experiments are also described in Chapter 5. Subsequently, profiling results of the current system performance in comparison with another similar system are shown in Chapter 6. Overview of this project and future work are summarised in Chapter 7.

Chapter 2

System Hardware and Associated Software

The Infrastructure of the Mind Attention Interface relies heavily on the hardware devices used. The two major detection components are the EEG device and eye-gaze plus head tracking system. Much effort was put into identifying suitable candidates for these systems and they are described in some detail below. A survey was conducted in Section 2.1 to find the available hardware and a selection of devices that satisfied our requirements. Manufacturers software that is delivered with the hardware devices is described in Section 2.2. This software is used in certain parts of the MAI System.

2.1 Devices

2.1.1 Wedge

The Wedge is a two-walled, immersive virtual environment. It is an Australian invention designed and built at the Australian National University. It is able to display three-dimensional computer graphics in active stereo and has the capability to play three-dimensional spatialised sound using an 8-speaker array. Interaction in the Wedge, before the development of the MAI, has been driven

by a Logitech ultrasonic tracking system together with other controls. Figure 2.1 shows a user in the Wedge.

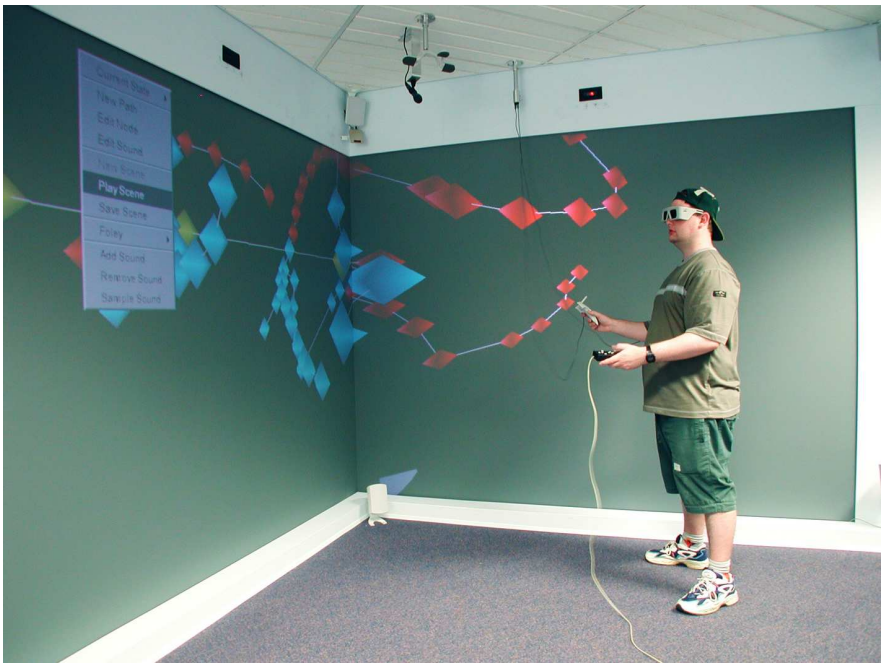

Figure 2.1: A participant in Wedge virtual reality theatre

In the Mind Attention Interface, user sits inside the Wedge space wearing an EEG cap, and, possibly, shutter glasses. A camera-based eye-gaze and head tracking device will be setup. Other controls will be used to control this virtual environment when necessary. Visual and audio stimulus are generated and displayed inside the Wedge and the attention and mind states of a human participant are recorded. Figure 2.2 shows the extension we made in the Wedge environment.

2.1.2 EEG System

EEG systems consist of electrodes and amplification hardware together with preprocessing software. Each electrode measures the electric potential at a location on a participant's scalp (or, more precisely, the differences in potentials between electrodes). Modern EEG systems are widespread and range from budget sys-

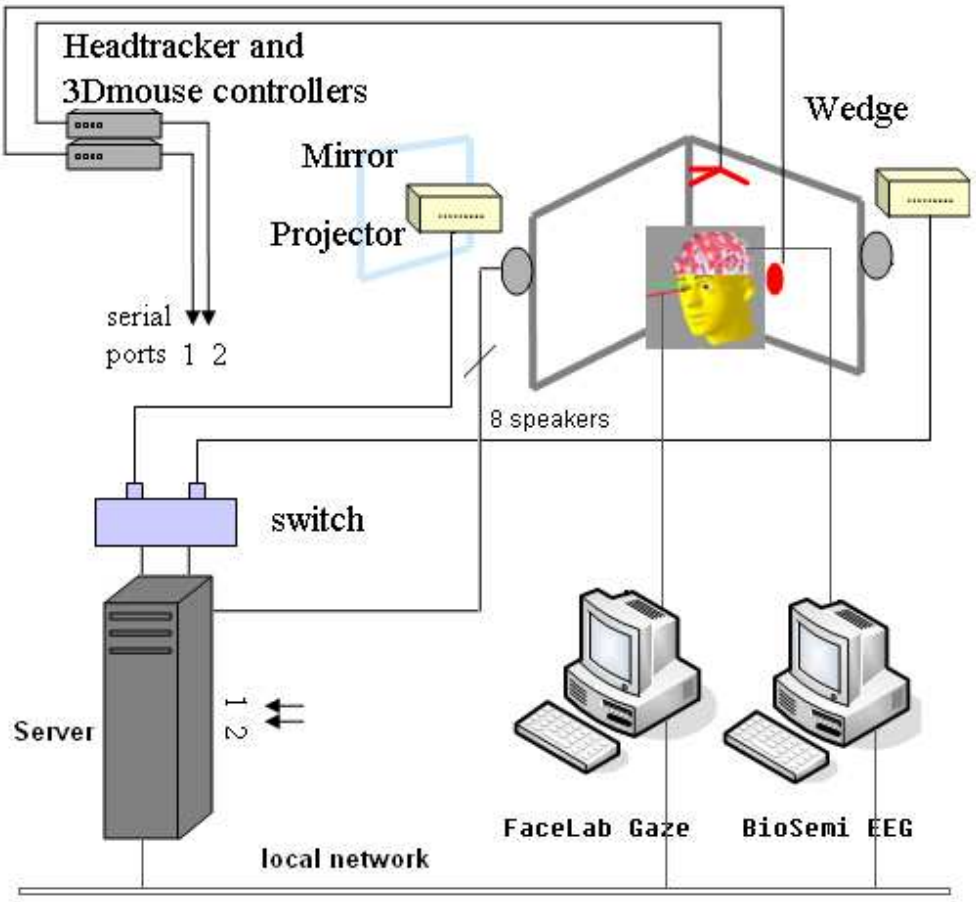

Figure 2.2: The MAI System hardware layout

tems, for hobbyists and game players, to diagnostic systems for hospitals and research clinics.

The Dutch company, BioSemi^{TM*} , is one supplier of EEG systems for research application. This company grew out the Medical Physics department of the University of Amsterdam and it claims to be the only commercial supplier of "active electrode" technology. The concept is that appropriate shielding, combined with driving part of the amplifier circuit as a feedback loop through the electrode, allows the use of high-impedance leads without picking up 50 Hz interference from the mains. Since high-impedance leads are used, the customary

*http://www.biosemi.com

skin preparations, of scrubbing or even shaving the scalp, are not necessary to lower impedance [30].

Because ease of preparation and low noise are important requirements for robustness and usability of an interface for a virtual environment, the BioSemi ActiveTwo EEG system was chosen for use in the MAI. The baseline ActiveTwo system includes 16 electrodes, which was thought to be sufficient as a starting point. Although up to 128 electrodes are possible, there is a trade off between the ease of use for real-time interaction and the accuracy of source localization as determined by the EEG. (A larger system is available for limited use, and for benchmarking, elsewhere at the Australian National University.)

Figure 2.3 shows the major parts of BioSemi system and how they connect together:

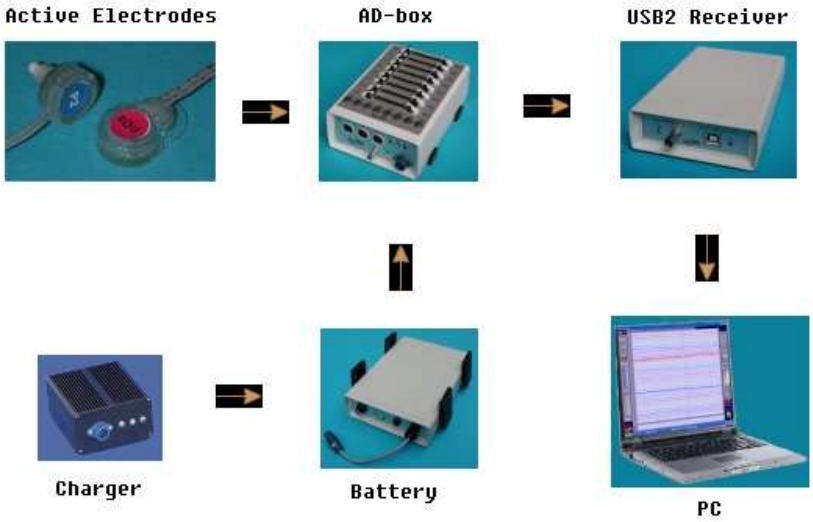

Figure 2.3: BioSemi ActiveTwo EEG system hardware components [7]

2.1.3 Gaze Tracker

Compared with EEG system, the choice of a gaze-tracking system for the MAI is much more open-ended as the market supports a range of trackers, which measure

head and eye positions, orientations and movements. Candidate trackers can be divided into head-mounted and remote systems. Our initial bias was towards remote systems because of the burden of wearing the EEG cap and, possibly, shutter glasses. Although it is possible to measure head position by other means, such as the Logitech ultrasound tracker in the Wedge, it was thought to be advantageous to combine head and eye tracking in the one system. We evaluated several commercial eye-gaze tracking systems, which are listed in Table 2.1:

Company	System	Approximate Price	Website
Applied Science Laboratories™(ASL)	Model R series	US$ 33000	www.a-s-l.com
SeeingMachines™	faceLAB®	USD$ 25000	www.seeingmachine.com
SensoMotoric Instruments™(SMI)	iView X Red	Euro$ 40000	www.smi.de
Arrington™	ViewPoint	US$ 10000	www.arringtonresearch.com

Table 2.1: Available remote gaze systems

A survey was conducted looking at the following features. This work was done by myself and others in the MAI group:

1. Cost

   The cost of the system needed to be considered as we had a limited budget and could only use part of our funding for the gaze tracker. We required a system under AUD$ 30,000. On this basis, we could not afford SMI system. ASL and Seeing Machines are both remote optical system and they are about the same price. On the other hand, Arrington has a competitive price and is much more affordable. If we purchased Arrington, we would be able to purchase a head-tracking device using the rest of the available funds.

2. Sample Rate

   Sample rate is one of the most important issues we need to consider. The sampling rate of the system is related to the sample interval and, therefore, reflects the response of the system. It also determines how much data will be generated, which reflects the bandwidth used by the device. The sample rate of head-mounted gaze-tracking system can be very high. For example, EyeLink®series from SR Research [†] provides tracking with sample rates up to 1KHz. Among the remote system we have evaluated, Arrington has a sample rate of 30Hz. Other systems vary from 60 Hz to 240 Hz, depending on the choice of camera. With higher sampling rates, we would be able to do offline studies with EEG data much more accurately.

3. Accuracy

   Accuracy is another important feature that cannot be ignored. Several optical systems we looked at had about the same accuracy. Seeing Machines had the advantage of using two cameras, which provides more head movement range and eye rotation range.

4. Latency

   System Latency was one of the features we paid most attention to. In this context, latency is the time between the moment data is sampled, and the moment it is output. Both hardware devices and software can cause latency due to the time it takes to process data. It depends on the hardware acquisition and the software algorithm of the eye-tracking technology. Seeing Machines, claims that it takes one and a half frames for their faceLAB software to calculate gaze data at 60Hz, which gives a latency of 25ms. Other systems latency is not listed in the specification documents and is generally unknown.

5. Integration

[†](http://www.eyelinkinfo.com/)

System integrations ability is a key to construction of the MAI, which combines the eye-gaze tracker, EEG cap, and stereo glasses for Wedge all together. As a consequence, devices that we select should not influence the use of other devices. For example, an eye-gaze system that requires the use of a headband or an optics monocle is not suitable.

	ASL	SeeingMachines	SMI	Arrington
Sampling and output rates	60Hz default, 120Hz, 240Hz optional	60Hz default	60Hz default, 100Hz optional	30Hz default
Visual range	50° horizontal, 40° vertical	90° horizontal	Unknown	70° horizontal
System Accuracy	0.5° visual angle	1.0° rotational angle	0.5° to 1.0°	0.25° to 1.0°
Resolution	0.25°	Unknown	0.1°	0.15°
Head movement	30×30cm at 80cm	50×50cm at 1.4m	40×40cm at 80cm	Unlimited
Operation together with shutter glasses	Unknown	Yes	High tolerance	No
Operation together with EEG cap	Yes	Yes	Yes	Signal Noise
Head tracking	No	Yes	No	No
Overview	Limited head movement and visual range	High sampling not available, stereovision system	Too expensive	Not suitable for MAI

Table 2.2: Gaze system comparison

Ultimately, the Seeing Machines faceLAB system is chosen for our gaze-tracking system after a serial of comparison in Table 2.2. Although this system does not offer the highest sampling rate and accuracy, we were still impressed by its visual range and tolerance to various complex environments. Besides, the remote technology gives user a wider head movement range and this is very important to virtual environment. Finally yet importantly, this native Australian company is on campus and we have test the system to verify our requirement, and it turned out that this robust and flexible stereovision solution is suitable for our human attention device.

2.1.4 Logitech 3D mouse

Logitech 3D mouse is a low cost ultrasonic device that operates with 6DOF (Six Degree of Freedom), and it has been used in the Wedge theatre since 1998. Receivers sample ultrasonic signals and reference them to transmitter coordinate array. With up to a 50Hz sampling rate, it covers a tracking space of 1.5m, with a cone range of 100°. Linear resolution is 0.1mm and angular resolution is 0.1°. Device latency is 72ms [63]. In the MAI, this 3D mouse might be used for some direct control. Figure 2.1 on page 12 shows a user holding the Logitech 3D mouse in the Wedge.

2.1.5 Wii^{TM}Remote Controller

The Wii Remote is the primary controller for Nintendo's Wii console. The main features of the Wii Remote are its motion sensing capability, which allows the user to interact with and manipulate items on screen via movement and pointing, and its expandability through the use of attachments. The Wii Remote has the ability to sense both rotational orientation and translational acceleration along three-dimensional axis. The Wii Remote was added into the MAI during 2007. This low cost[‡] controller might be used as a controller or pointing device in the

[‡]AU$69.95

Wedge.

2.1.6 A Note of my Individual Contribution

*EEG	BioSemi was the decision of others.
*Eye-tracking System	I contributed 20% of decision and purchase. The evaluation and comparison was entirely my work.
*Logitech 3D mouse	This device has been part of the Wedge system since 2001.
*Wii	I only contributed the driver for the MAI System for this device.

2.2 Manufacturer Software

After we purchased the hardware system, manufacturer software is an important component that needs to be studied. In this section, I have described three manufacturer software that has been used in our system, namely: ActiView, faceLAB V4, and Network Time Protocol FastTrack®.

2.2.1 ActiView

ActiView is open source acquisition software delivered with the BioSemi system. Figure 2.4 is a screen snapshot of ActiView. It is claimed [7] to be *"a complete acquisition program designed to display all ActiveTwo channels on screen and save all the data to (network) disk in BDF format".* This software is written in LabVIEW™graphical programming language[§] . It displays signals on screen, logs or replays data, and handles network functions. The layout is designed to give the user a quick and simple check of the quality of the data. Users can perform offline processing using a file that is saved by ActiView. It is also possible to develop

[§]Laboratory Virtual Instrumentation Engineering Workbench (LabVIEW) is a product of National Instruments™, which is a producer of automated test equipment and virtual instrumentation software

real-time LabVIEW programs and to integrate them with ActiView to do online processing. BioSemi system offers a dynamic link library (dll) for users to create their own acquisition driver when further developments are not LabVIEW based, so ActiView is only used as a validation tool in MAI System to display and check EEG data patterns.

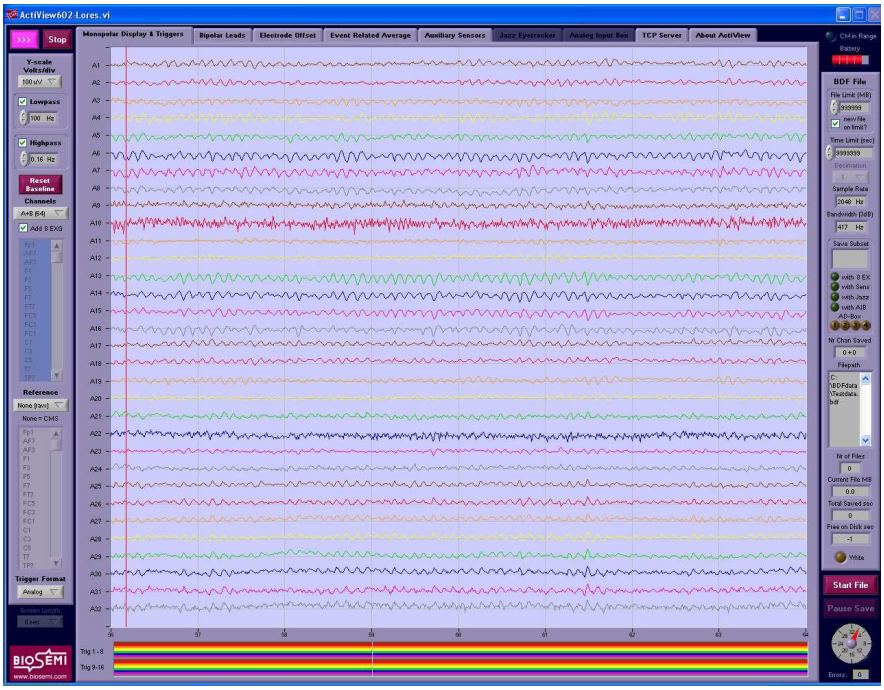

Figure 2.4: ActiView acquisition software

2.2.2 faceLAB V4

FaceLAB V4 is name of the software that comes with the hardware. It is an entire package including calibration, tracking, acquisition, analysis, displaying, and transmission. This handy tool builds in a fully integrated 3D world model, which allows us to explore human attention behaviours. We develop real-time applications using the freely available Software Development Kit (SDK) provided by faceLAB, and enabling our attention interface to localise salient facial features such as the eyes or head in real-time across network interfaces [46].The Graphic

User Interface (GUI) helped to setup and calibrate easily in both classic tracking mode and precision gaze mode. Figure 1.1 in page 6 is also a screen snapshot when the software is running.

2.2.3 Network Time Protocol FastTrack

Network Time Protocol (NTP) is an internet protocol designed to synchronise the clocks of hosts and routers over a network. Previous research shows that for systems which consider multiple peripheral devices will also have to consider network timing issues [44]. NTP is always the solution to synchronise computer clocks. The protocol was first developed by David L. Mills in 1988 [32]. Since then, four versions of the protocol have been released. NTP version 3, described in [33], is an internet draft standard, formalised in Request for Comments (RFC) 1305. NTP version 4 is a significant revision of the NTP standard, and is the current development version, but has not been formalised in a RFC. In the MAI System, faceLAB product NTP FastTrack is used to synchronise all of the hosts to the University Stratum[¶] 2 server. This product is based upon the NTP Version 4 distribution [46].

NTP does not synchronise computers directly to each other, but instead synchronises each computer independently to the official time standard of Universal Time Coordinated (UTC). UTC is a worldwide time reference independent of time zones, and is determined from a combination of time estimates from several institutions in different countries. It is expected to provide nominal accuracies of a few tens of milliseconds on wide area networks, sub milliseconds accuracies on local area networks. Figure 2.5 shows the NTP time reference hierarchy in the MAI System. Appendix C on page 85 gives a estimation of the system clock offset in one single day without NTP, and also shows the system clock in the MAI System compared to the standard time with NTP running.

[¶]The stratum levels define the distance from the reference clock and the associated accuracy.

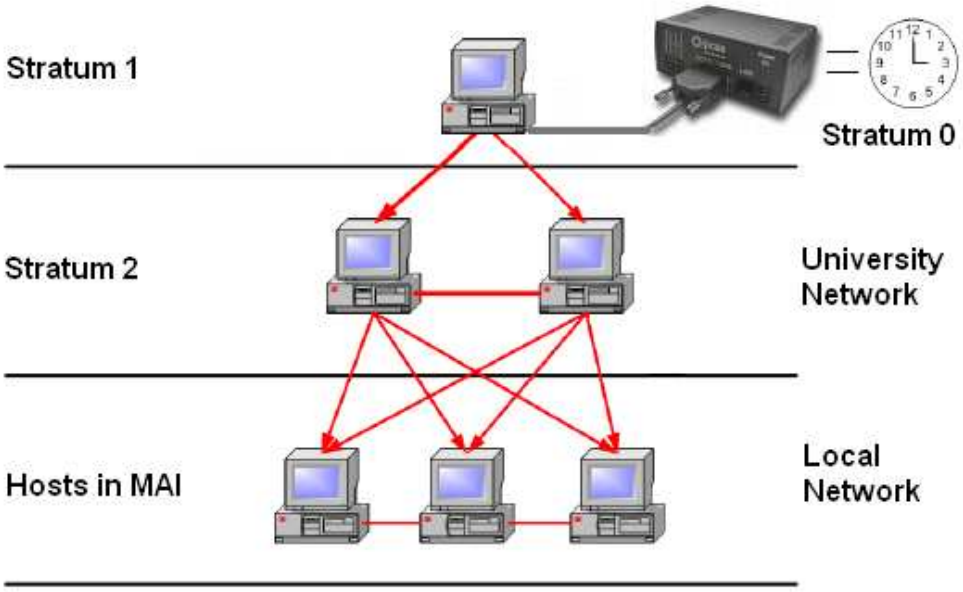

Figure 2.5: NTP time reference hierarchy in the MAI System. Stratum 0 are devices such as GPS clocks, and they locally connected to Stratum 1 computers. Australian National University has Stratum 2 computers which send NTP requests to Stratum 1 computers.

2.3 Summary

To summarise, the Mind Attention Interface hardware devices consist of the Wedge virtual reality theatre, the BioSemi EEG system, the SeeingMachines faceLAB eye and head tracking system, the Logitech 3D mouse, and the Wii remote controller. These devices with associated manufacturer software are the existing resources, and to use these devices as a whole environment, an enabling software platform for the hardware needs to be designed and implemented.

Chapter 3

MAI System Requirements

The Mind Attention Interface will facilitate an exploration of the usefulness of mind and attention as interfaces for a virtual environment. The MAI-Wedge platform also aspires to build artistic installations, particularly in computer-generated music, which will be a unique endeavour. Through the study of the way in which humans interact with these installations, it is hoped to throw some light on the inherent nature of mind and attention itself. Because of these demanding goals, the MAI System needs to be an environment that can be used for analysis as well as for production and performance. In Section 3.1, system requirements will be described. These requirements were raised in order to guide the design of a practical system. Section 3.2 holds a discussion over the pros and cons of the present comparable systems.

3.1 System Requirements

In brief, the MAI System needs to fulfil the following requirements:

1. A distributed system over the network

   This requirement will allow data acquisition, signal processing, and display applications to run over the network, which underlies the computing power of the whole system.

2. Support for a current hardware, and a plug-and-play interface for new hardware

   Current hardware should be included in the MAI System, and new hardware should be able to be integrated with the MAI system without additional modifications.

3. Compatible with the Wedge-based virtual environment

   The Wedge is the environment that will be used for production and performance.

4. Log and replay experiments

   Experiments running in the MAI need to be logged. The logging of experiments should not affect the performance of the running system. Log files should be completed and detailed so that further analysis and replay experiments can be achieved based on these files.

5. A development interface for the MAI applications

   The MAI System will be used both for research and for teaching purposes. Future users or developers should be able extend the current MAI System as well as the application clients.

6. Capability to support certain comparable systems

   The MAI System should have the compatibility with industry file format and be capable of using functionalities provided by comparable systems.

3.2 Comparable Systems

Before designing the MAI System, an evaluation was made to determine the adequacy and effectiveness of the comparable systems. This evaluation of competing software packages focuses on whether or not they met the system requirements. The result of this evaluation is outlined in the following sections.

3.2.1 Virtual-Reality Peripheral Network

This software was described in [44] as follows: *"the Virtual-Reality Peripheral Network (VRPN) is a set of classes within a library and a set of servers that are designed to implement a network-transparent interface between application programs and the set of physical devices (trackers, etc.) used in a virtual-reality (VR) system".* VRPN does not aim to provide an overall VR Application Programming Interface (API). It focuses on the sub-problems of providing a uniform interface to a wide range of devices and providing low-latency, robust, and network-transparent access to devices[62].

In VRPN, device types are pre-defined generically (such as "Tracker", "Button", etc.). Physical device classes are added in by inheriting VRPN base classes. VRPN library will need to be recompiled for new classes, thus the servers that use the VRPN library need to be rewritten in order to support new classes. The EEG and gaze-tracking systems are not pre-defined device types.

I experimented with the VRPN in the Wedge. The Logitech head tracker driver used in the wedge was included as an input to the VRPN server. The VRPN library was included in a client application in order to receive data from a VRPN server. It appears that adding new commercial devices into VRPN is not very convenient, especially when there is no access to the software development APIs for these devices. In most instances, software that comes with purchased devices only allows for slight modifications. For example, FaceLAB only allows data to be sent out to a network place via their software, and the Biosemi EEG device could only be accessed by ActiView or a dynamic link library. It is clear that not all these devices are ready to be added onto the VRPN framework.

To summarise, VRPN focus on virtual reality peripheral devices. We originally hoped that this software package could provide a device layer for the MAI System; unfortunately, it does not support all our current devices. Furthermore, it also does not have a plug and play interface, which might result in the recompilation of the entire system whenever a new device is added.

3.2.2 NeuroServer

As claimed in [37], NeuroServer is an open-source multi-platform EEG signals transceiver which uses the TCP/IP transmission protocol (see Section 4.2.2 on page 39 for details) and the EDF (European Data Format) file format. It is part of the OpenEEG * project. This software provides a standard EEG server that mediates between the raw EEG devices and all the various EEG applications that a user may wish to run to analyse the incoming EEG data. Data is transmitted and passed easily over a network (or the internet). Standard EDF conventions (see Section 4.2.4 on page 45) are used for header information and file storage. Besides NeuroServer, we were also interested in using other EEG applications that connect to NeuroServer using its protocol. Example such as BrainBay [9] rapid-development environment, which is part of OpenEEG project, has been used in the MAI application development by James Sheridan.

In brief, NeuroServer provides a basic software framework for network transceiver interface. This software met our first two and the last requirements; however, the other three requirements were not satisfied. NeuroServer was not specifically designed for the virtual environment because it focuses on EEG signals, whereas tracking devices were not considered.

3.2.3 VR Juggler

VR Juggler is an open source software which provides a platform for virtual reality application development. The claim is that programmers can build powerful VR applications using a solid framework that is independent of computer platform and VR system [61]. VR Juggler does not contain a graphic engine itself but builds on top of OpenGL, as shown in Figure 3.1.

VR Juggler is highly portable which can be supported on all commonly used desktop operating systems. VR Juggler 2.0 supports IRIX, Linux, Windows,

*The OpenEEG project is about creating a low cost EEG device and free software to go with it

FreeBSD, Solaris, and Mac OS X. In VR Juggler, all applications are written as objects that are handled by the kernel. VR Juggler uses the application objects to create the VR environment with which the users interact. Developers create their own application by implementing application object interfaces needed by the VR Juggler virtual platform.

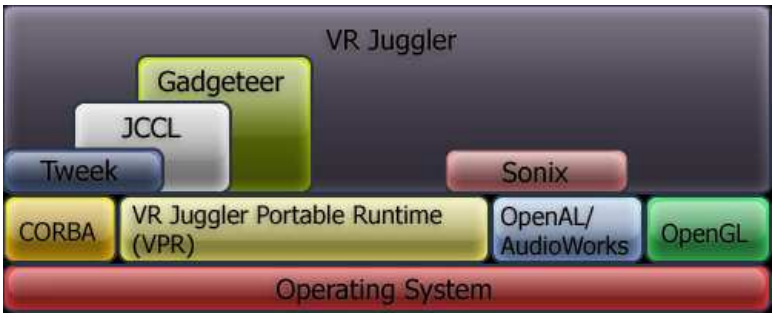

Figure 3.1: VR Juggler suite layered architecture (after [61]). The VR Juggler suite contains VR Juggler, "Gadgeteer", JCCL, TWeek, and Sonix.

Most significantly, the VR Juggler suite allows applications to work with VR peripheral devices like VR gloves and VR helmets using its device management system called "Gadgeteer". This device management system handles the configuration, control, acquisition, and representation of data from VR devices. Gadgeteer contains an input manager that treats devices in terms of abstract concepts such as "positional", "digital", "gesture", etc. It also contains a remote input manager that can share device samples between computers.

VR Juggler mainly focuses on creating a platform for VR systems. It can help a programmer to achieve the goal of "program once, use everywhere". However, the disadvantage is also obvious for users who are not so interested in using a different VR system. Firstly, a set of Wedge applications will have to be reconstructed using VR Juggler API. Secondly, VR device types are pre-defined and not extendable (just like VRPN), which stops us from including the EEG and gaze-tracking systems.

3.2.4 BCI2000

BCI2000, claims to be a general-purpose system for brain-computer interface (BCI) research [45]. It can be used for data acquisition, stimulus presentation, and brain-monitoring applications. A variety of brain signal-processing methods culminating in a number of applications have been created by a group of developers. BCI2000 is able to facilitate both the implementation of these BCI applications as well as the psycho-physiological experiments which rely on them.

The initial BCI2000 platform was written in Borland C++ under Microsoft Windows 2000/XP. The BCI2000 system model consists of four individual modules. These modules communicate through a documented network-capable protocol based on TCP/IP. Although each module can be developed in any programming language or platform, no developer has deviated from the original configuration [6].

The real-time capability in BCI2000, though not intended as a dedicated real-time system, is considered sufficient for BCI operations. As Windows is not a real-time system, BCI2000 was carefully designed so that it "depends as little as possible on potentially-lengthy operating system functions" [45]. With BCI2000, neuroscientist can manipulate a BCI system using different processing methods quickly. However, from our point of view, the MAI System is expected to go beyond a simple BCI interface. The goal of allowing attention feedback be used in virtual environment cannot be achieved in BCI2000.

3.3 Summary

By studying a set of related software packages, I discovered that there were no existing software that can fulfil all our requirements, as they more or less concentrates on a particular aspect. With the knowledge that I gained from my research, I realised that, it is possible that a combination of all the above competing system would go towards meeting most of our requirements. However, building the MAI

System in this way would sacrifice low-level control. In addition, extra works on interfacing between different software need to be done.

To conclude, the MAI System should be designed and implemented as a new project. In order to reuse existing resources, comparable system designs are referenced in the later design considerations, and a number of existing software (NeuroServer for example) will be used as a start point for our implementation.

Chapter 4

MAI System Iterative Design

As described in Chapter 3, a new MAI system needs to be designed to fulfil our requirements. The design decisions corresponding to the six requirements listed in Section 3.1 on page 23 is outlined below.

1. The whole system will be distributed over a network. The device drivers will be able to run on separate machines and transmit the acquired data across the network. These data can then be analysed through the processing engine or can be sent to application clients over the network. Application clients will be provided with both analysed data and raw data. A server-client architecture design, which uses a central server to manage all other components as clients, fulfils the first requirement.

2. Current hardware devices listed in Chapter 2 will be supported in the MAI System. A network interface for devices using a generic protocol to send data will be designed. This protocol will also be used in other parts of the system. This protocol will allow multi-platform devices to be easily plugged into the system. In another word, adding new devices into the system will only involve writing a driver for each device independently. Thus, overall, the system uses a unique interface for control and communication.

3. A system designed specifically for a Wedge-based virtual environment should

allow current Wedge applications to be plugged into it. This requirement can be fulfilled by supporting client applications operating in multiple operating systems. Each client application will communicate through the network interface. Thus, existing applications can be extended without modifying virtual environment components.

4. The logging requirement will be achieved by using a multicast datagram delivery mechanism, which will not greatly affect the performance of running system. Time stamps will be included in the log files for analysis and replay purposes. The logging server will generate separate log files for each device, which assists the playback server in replaying log files individually.

5. Software libraries will be created in order to allow the MAI applications to connect to the MAI System. Developing static and dynamic libraries on Windows platform will have priority, as most of the existing applications will be running on Windows platform.

6. The MAI System will provide conversion between factory format and industry format so that client applications have access to both formats. This conversion procedure will be placed at device level. In order to support applications that connect to NeuroServer, the generic protocol for the MAI System will be derived from the protocol used in NeuroServer.

As described above, the six requirements merely provides the framework the high-level design. The following chapter will focus on the details of the design culminating in the entire system design shown in Figure 4.1. The background to the design, the system architecture in terms of layers, low-level design considerations, and the design of the enabling components, will be included in the discussion.

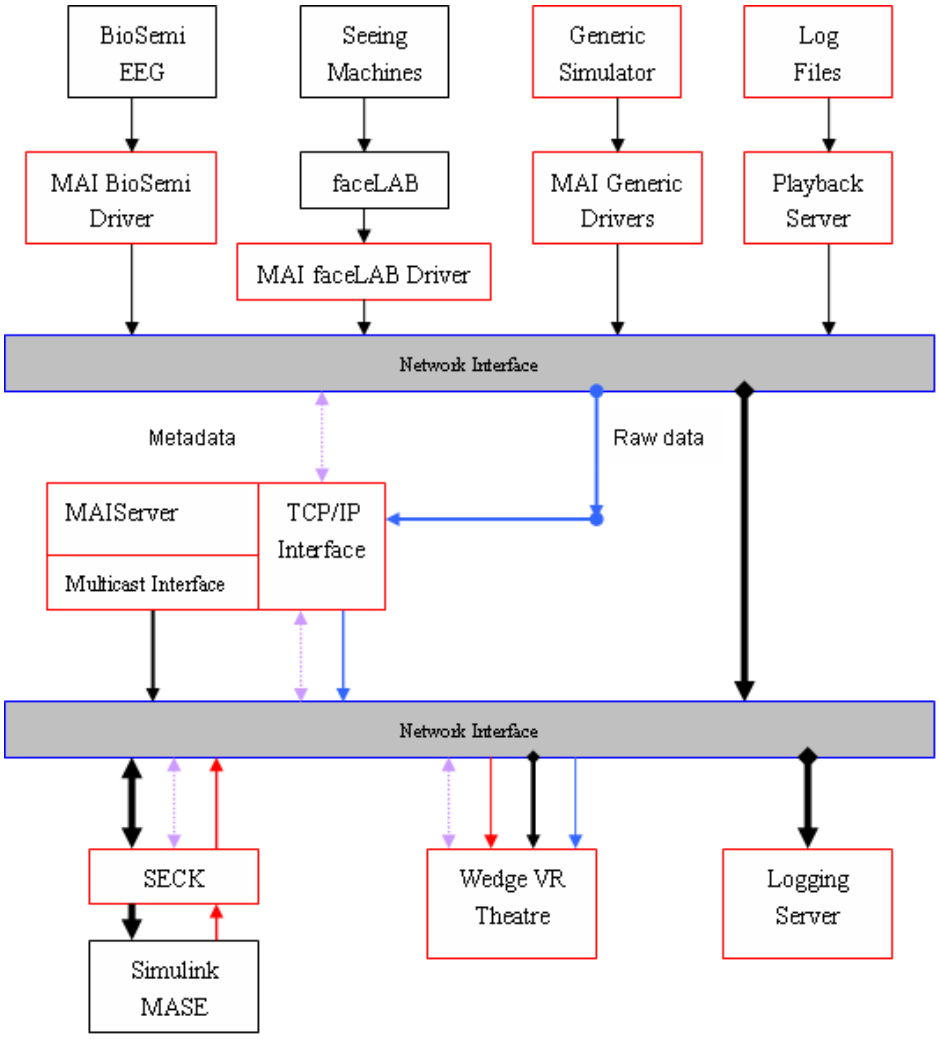

Figure 4.1: The MAI System design overview. Red blocks are modules that I need to implement. Different colour arrows represent different types of data. The line thickness represents the data throughput.

4.1 System Architecture and Layer Design

4.1.1 MAIServer Architecture Considerations

As mentioned in the design outline above, a server is required to serve both devices and applications. In this section, several common types of server architecture designs are described and compared. The MAIServer is the central server that maintains each module of the MAI System. TCP (Transmission Control Protocol)

is used to verify that all messages that has been sent to and from the MAIServer has been delivered. (Transmission protocols will be further discussed in section 4.2.2 .) Comer in his book [10] introduced three patterns in designing servers, which are summarised below:

- Concurrent, connection-oriented servers

  Servers of this type create a stream socket and produce a process to handle each new connection that they receive. A stream socket that uses the TCP protocol incurs high system overheads in establishing connection, in transmitting data, and in creating a new process. However, once the connection has been established, data transmission is reliable in both directions. Well-known examples are telnet and ftp servers, in which both will do a good deal of reading and writing over an extended period. In this model, processes do not share information with each other, which reduces the cost of communication. Theoretically, this design depends on the operating systems' time slicing mechanism to share the CPU among the processes, however, in reality, the arrival of data controls the rate at which processing takes place. This might cause time confusion if we use this pattern in the MAIServer. When two processes both have message in their queue, the time information are not shared thus sequence of packets are lost.

- Iterative, connectionless servers

  Servers of this type often use a datagram socket rather than a stream socket. No forking is involved in this model. Examples of these type of servers include time-of-day, and echo servers*. These servers handle each message in the same process with much less overhead compared to servers using stream sockets, however, communications are not reliable. Packets are not guaranteed to be transmitted successfully, since there is no mechanism to detect or handle packet loss.

*A server that merely echoes a message sent by the client.

- Single process concurrent servers

  This model has the capability of handling several clients simultaneously. The server spends most of its time blocked waiting for a message from the client, in other words, the connection I/O is dominated. As the name of this server indicates, only one process maintains a set of open connections, and listens to each for a message. The "select" system call is used to listen to new messages from multiple sources while blocked waiting. The process only wakes up and replies whenever it gets a message from clients. Unlike the connectionless model, this model provides reliable service.

The above server types are the most typical of server designs. The last design was chosen as our basic server framework. As described above, each design has its own advantage over others in some aspect, but none of them is suitable for all environments. Experience from well-known applications such as ftp server or time-of-day server demonstrate the best implementation of a design. In the MAI System, the role of the MAIServer is to maintain shared metadata and manage device status information. This requirement can be accomplished by a single process server. Without forking a new process, the MAIServer can handle and maintain a number of clients simultaneously, which saves the cost of interprocess communication. To conclude, the fundamental framework of the MAIServer is a single process concurrent server running over TCP, the detail design on top will be discussed later in Section 4.3.2 on page 47.

4.1.2 Signal-processing Layer Considerations

As has been noted in Figure 4.1 before, the layers are organised relatively straightforward except the position of signal-processing layer. Signal-processing layer, here called the Mind Attention Spectral Engine (MASE), is one of the important layers in the MAI System. This layer provides high-level features for application layer by processing raw data. Despite the pre-processing in the Data Acquisition (DAQ) layer (and possibly in the MAIServer), this layer need to be construct

in a high-level computing language so that it offers algorithm development, data visualisation, data analysis. However, as the processing layer is largely the work of others, my design consideration is how the MASE can be plugged into the MAI System.

My initial thought was to distribute the MAI System over the network with all layers following a sequential relationship, where the output of one layer becomes the input of another layer. The MAIServer will be sitting in between the DAQ layer and the API (Application Programming Interface). However, the idea of the MAI System itself seems more like a peer-to-peer network with information flow inside, and the MAIServer uses an uniform interface to serve all other peers. To position the MASE, I consider this layer as an individual client. On one hand, the MASE can be considered as an instance of application layer that handles signal processing; On the other hand, the MASE is also an outstanding device, which feeds high-level features into the rest of the system. Figure 4.2 provides three possible system layouts with the MASE.

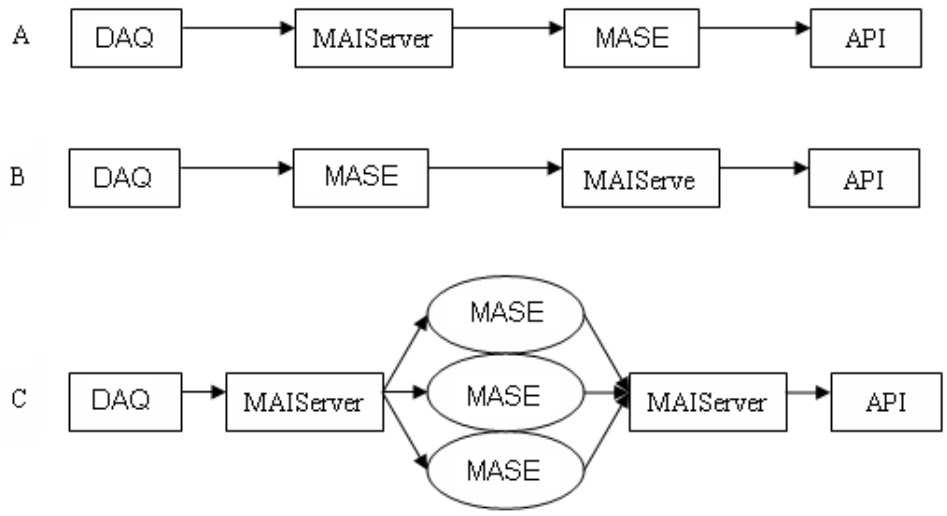

Figure 4.2: Three possible positions of the MASE in the MAI System

As can be seen, the MASE that is placed in the first two designs has functions overlap with the MAIServer. The direct connection between the MASE and any

other layers except the MAIServer will involve a connection interface for both incoming and outgoing messages, by which the MAIServer is design to provide. Besides, after raw data stream into the MASE, it will not be available again for the next layer unless an extra raw data tunnel is prepared. On the contrary, the last design interprets MASE as a plug-in to the system. (See Figure 4.1 for high-level overview) An "intermedia tool" called the Spectral Engine Connection Kit (SECK) will be design so that information can be interpreted for both sides without modification to original design. The problem in the last design then becomes tradeoffs between latency and bandwidth in the system. Table 4.1 states the pros and cons of the design. In comparison, the plug-in design allows multiple MASEs to be connected while design in series increases the complexity when extending the system.

	Design A	Design B	Design C
Pros	⋆ Low latency ⋆ Interface for input devices ⋆ Good for scientific research	⋆ Real time data feed into engine ⋆ Quick access to high-level processed data ⋆ Low latency	⋆ Engine plug-in as a device ⋆ Devices and clients all share the same incoming interface
Cons	⋆ More network traffic for engine to handle ⋆ Engine needs to consider algorithm as well as APIs	⋆ Interface for input devices ⋆ Raw data cannot reach clients	⋆ Add more latency
Conclusion	Complicated	Hard to implement	Extendable

Table 4.1: Comparison of the three engine designs

4.2 Low-Level Design Considerations

4.2.1 Communication Protocol

In the MAI System, communication between machines will be based on a set of pre-defined commands. These commands are preferred in text format and used in the network level communication. To ensure all parts of the system share the same interface when sending or receiving data, all the modules within the system must implement an interface following parts of the protocol. Therefore, new modules that follow this protocol can be connected to the existing modules like building blocks, where the interface just fit with each other. As a start point, line-oriented NeuroServer protocol is chosen to be the base commands. Our protocol is an extension to NeuroServer protocol that used in OpenEEG project for the purposes of communicating with relevant applications. A list of commands in the protocol with simple explanations is attached in Appendix A on page 81.

As can be seen, commands are used in either handshake or data transmission. Handshake commands establish connections, declare head information to the server, and check available resources in the system. In the same way, data transmission commands warp data in channels and these data will be decoded by receivers. This will solve the problem that periphery devices are usually hard to classify due to different data attribute. All types of data are sorted in the same way despite whether it is gaze data or EEG data. This is a replacement of the pre-defined abstract structure for each device type. In our system, handshake and data transmission will be delivered in separate paths. To be precise, header information is delivered through the MAIServer while data are delivered straight to destinations. A time stamp will also be attached at the end of the data command for further usage.

Finally, ASCII character encoding will be in use in the protocol. This allows server and client communicate in different platform regardless the order in which a

sequence of bytes are stored in different computing environment[†]. Unfortunately, this uses more bytes to store float or double values in comparison with how they store in memory. To optimize the network traffic, the protocol should also support sending out data as the format that are stored especially for signal-processing modules that requires high-resolution values. To summarise, this protocol is designed to enhance the communication performance as well as to support devices easily.

4.2.2 Transmission Protocols

Since the MAI System is a distributed system over the network, the data flow protocols become an important feature that can affect system behaviour. TCP (Transmission Control Protocol [43]) and UDP (User Datagram Protocol [42]) are two core protocols in the transport layer of the OSI (Open Systems Interconnection) seven-layer model of computer networks [69]. They will be both used in the MAI as described below.

Transmission Control Protocol

TCP provides a networked system with reliable data transmission between hosts. Packets are delivered in order, and checksums ensure the automatic detection of data-transmission errors. The point-to-point nature of TCP increases system costs by establishing a connection and bandwidth overhead to retransmit failed packets. To guarantee the ordering of packets, the TCP protocol handler may also arbitrarily delay the packets. Based on these characteristics, TCP has seen its greatest use in systems having a relatively small number of hosts and a small amount of data over network. TCP is usually referred to as TCP/IP, because IP (Internet Protocol) is the most important protocol in internet layer of the OSI

[†]Big-endian is an order in which the most significant value in the sequence is stored at the lowest storage address. Little-endian is an order in which the least significant value in the sequence is stored at the lowest storage address

model.

TCP is a common network protocol which has significant overheads due to the way in which it checks lost packets. It will be used in the MAI System for compatibility with other systems such as NeuroServer, as well as for safe delivery of handshake data, but in general it is problematic and the UDP is to be preferred for data transmission.

User Datagram Protocol

UDP removes most of the communication overhead introduced by TCP, but it also provides a poor service. It offers a simple delivery of data packets and has no reliability or ordering guarantees. As with TCP, packets are transmitted point-to-point between two hosts, however, because UDP transmission does not require the sender and receiver to establish an explicit connection, hosts can easily broadcast packets. Applications are more robust because they do not have to be dependent on a reliable connection being made. Due to the fact that UDP does not have the mechanism to ensure safe delivery all packets, the packet header is smaller than TCP header.

In the MAI System, UDP mode will be introduced when time-sensitive information is conveyed between clients and server and where data reliability is considered less critical. Among all system devices, EEG for example, a large amount of data is to be sent as quickly as possible. UDP mode is preferred as it sends data straight out like non-buffered TCP mode, but uses fewer resources and sends quicker.

In some situations where applications require TCP services such as packet ordering, mechanisms should be developed within the UDP packets to achieve a similar functionality without introducing significant overhead. For example, the device client can associate each data packet with a unique serial number generated by a packet counter. This allows the MAIServer reorder packets sequence from each source by inspecting the serial number contained in the data packet. Because

the UDP itself is unaware of these sequence numbers, the destination application receives the packets as soon as they can but uses serial number to manipulate lost data. Each source (device client) generates its own, independent, serial number sequence, and there is no relation between these numbers. To provide global ordering capabilities, time stamp mechanism can be introduced to indicate when the packet was generated and sent, and thus provide a reference for comparing packets and synchronizing different sources. Of course, a time stamp is based on the device client's system clock and the system needs to be synchronised using NTP.

4.2.3 Message Delivery

The rapid growth of the data in the MAI System has led to much interest in message delivery paradigms, which include unicast, broadcast, and multicast. Network performance, such as bandwidth, can be largely affected due to the different techniques used in these paradigms.

- Unicast

  Unicast is a one to one connection between the client and the server. It uses IP delivery methods such as TCP and UDP protocols. All Local Area Networks (LANs) and IP networks support unicast transfer mode, and famous unicast application such as ftp and telnet employ the TCP transport protocol.

  Figure 4.3 shows the message delivery mechanism in unicast. A unicast client has a direct relationship to the server and each client that connects to the server taking up additional bandwidth. Packets are sent from server to specified destination individually, and will be sent multiple times if there are multiple clients. For example, if 10 clients connect to a server receiving 100Kbytes per second, the server will send 100Kbytes per second to each client and requires 1Mbytes of the bandwidth. If a server has a bandwidth

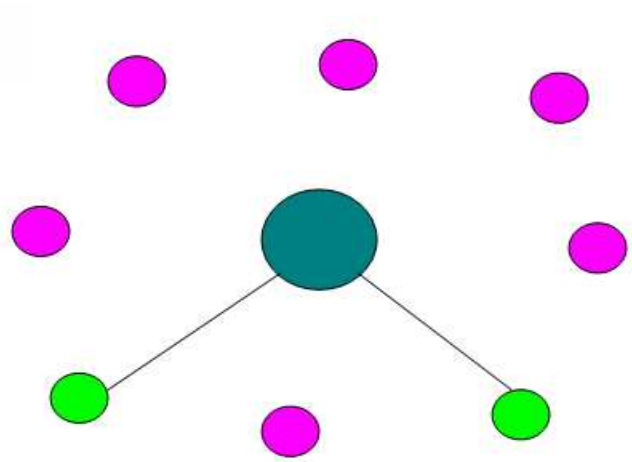

Figure 4.3: Unicast

limit of M, and we suppose there are N unicast clients connected to this server, each client will be restricted to a bandwidth of M/N. Not only are connections limited by the server bandwidth, but also multiple connections slow down the server performance because of the switching time between clients. Clients receive server packets with different delay times because the server sends out data individually.

- Broadcast

  Broadcast is a type of communication which delivers packets from one node to all other nodes. Any devices within the server broadcast domain will receive packets from the server. Broadcast employs UDP transport protocol. Broadcasting is largely confined to local area network technologies, most notably Ethernet. By design, most modern routers will block IP broadcast traffic and restrict it to the local subnet, as broadcast has a performance impact over wide area networks (WAN). Broadcast has advantage over Unicast on server performance, but it floods LANs and subnets. When a server tries to broadcast a huge amount of data, it heavily increases the network traffic. Media servers normally use multicast instead of broadcast. Figure

4.4 shows the message delivery mechanism in broadcast.

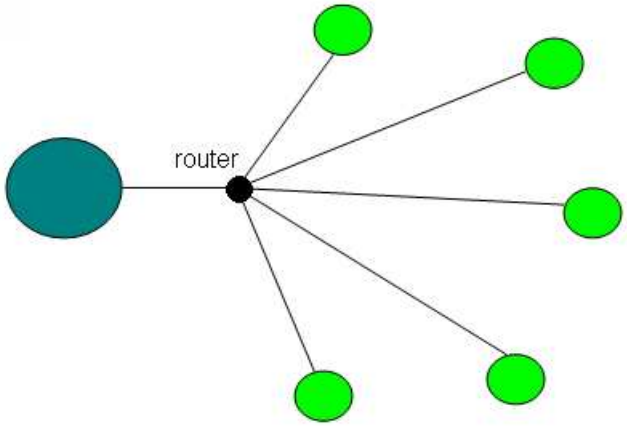

Figure 4.4: Broadcast

- Multicast

  Multicast is a networking technique for dynamic many-to-many communication, in which one or more servers deliver packets to a group of destinations. The multicast source relies on multicast-enabled routers to forward the same packet simultaneously to a set of clients. Multicast employs UDP transport protocol and the format of IP multicast packets is identical to unicast packets. Figure 4.5 shows the message delivery mechanism in multicast.

  Unlike broadcast transmission, multicast clients receive a stream of packets only if they have previously joined the specific multicast group address. Sending data to a multicast group address is similar to transmitting radio signals on a particular frequency. Just as you must tune a radio receiver to the particular frequency to receive the radio signal, you must join a multicast group to receive the multicast packets. Membership of a group is dynamic and controlled by the receivers. Senders have no idea about group

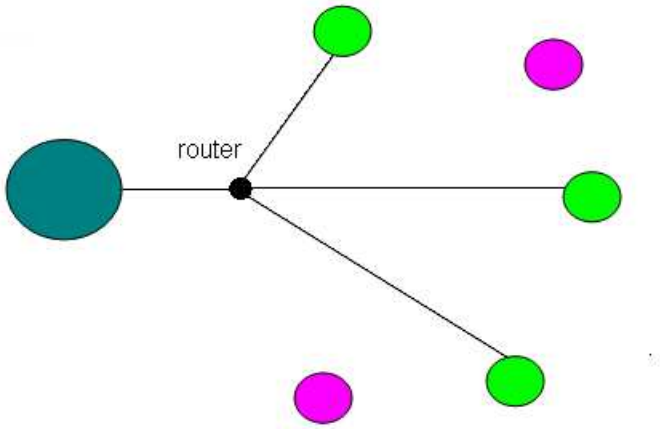

Figure 4.5: Multicast

membership. The routers in a multicast network attempt to minimize the transmission of packets across parts of the network.

The multicast mode is useful when a group of clients require a common set of data at the same time. Where there is a common need for the same data required by a group of clients, multicast transmission may provide significant bandwidth savings (up to 1/N of the bandwidth compared to N separate unicast clients). Multicast transmission also guarantees data delivery at the same time. Multicast on the Internet is generally not practical because only small sections of the Internet are multicast-enabled.

With the investigation over these three mechanisms, a decision over when and where to use each delivery mechanism in the MAI System has been made. Although the communications within the system are peer-to-peer based, it is highly recommended that unicast over TCP in use only for handling handshake command operations. Due to the fact that large amount of data was generated in the MAI System, multicast channel is preferred to enhance network performance by reducing bandwidth and latency.

4.2.4 Data Format

In terms of what was stated in the design requirements, the system requires the compatibility of standard data format. Since the MAI System already deals with few types of devices at present, the generic interface that I designed should support the existing industry standard file format (or manufactory specification if no industry standard exists for such type of device).

The format of header information that we adapt is the European Data Format (EDF). It is a simple and flexible format for exchange and storage of multi-channel biological signals. In our system, EEG is one of the main input signal sources. Therefore, EDF header has been chosen, which is the de-facto standard for EEG records since published. Its flexibility of labelling channels extent our device range to a new level. Users will be able to parse the header file as most of the BCI systems use EDF file format. Appendix B on page 83 is an table of EDF file format header. The reserved fields in the header are used in the MAI System as a channel description by Data Acquisition (DAQ) layer. Thus, this file header can also be used by other device drivers.

Except for headers, raw data also need to be formatted into a normalised standard. This standard is part of the interface communication protocol which has been described in section 4.2.1 on page 38. In brief, data in the protocol is normalised into channels. As can be seen, the key of this normalisation procedure is to understand the factory format. There is less concern on gaze data in comparison to EEG data due to its exclusive compatibility, by which universal standard has not yet published. EEG data however, uses EDF format for general with the exception that BioSemi EEG system uses an extended version called BDF format. The major change is the 24-bit data resolution compared with 16-bit depth EDF format. All in all, original factory format will be studied and converted into new format before streaming into the MAI System. We will take advantage of various devices and their high precision data, as well as provide compatibility options such as BDF to EDF conversion.

4.3 MAI System Enabling Components Design

In this section, I will discuss the detail design of the major MAI System enabling components. These components are layers or modules which support the MAI System.

4.3.1 Data Acquisition Layer

Data Acquisition Layer is a collection of device drivers. It focuses on acquiring data from the devices and formatting these data into a generic protocol. In addition, it also preserves the ability of pre-processing between acquiring and formatting when necessary. Figure 4.6 shows the data flow design in this layer and my contribution.

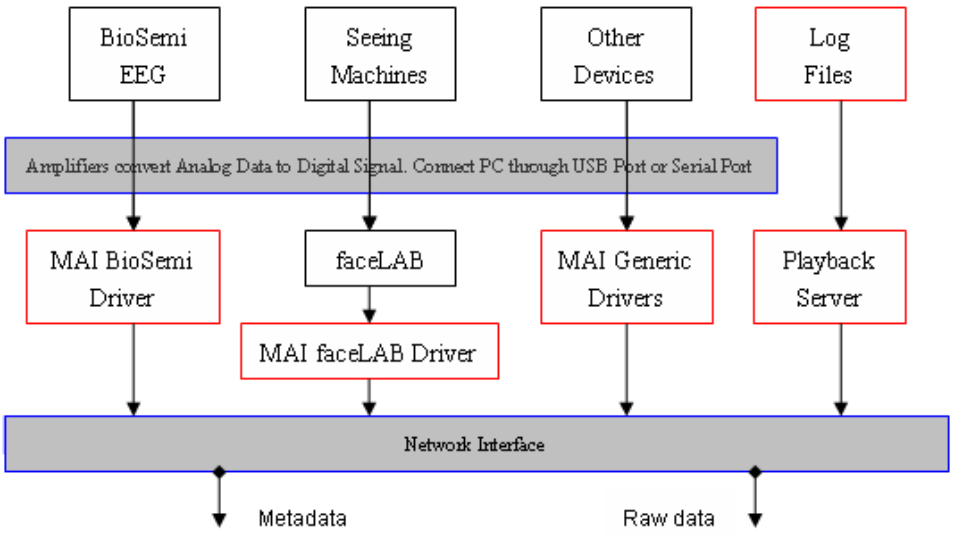

Figure 4.6: Data Acquisition layer design overview. Red blocks are modules that I need to implement.

As you might notice, gaze data is a bit different from other data. This is because faceLAB takes control over the raw data and offers processed data in its own format. Actually, commercial devices were more or less protected in a

way that we are not able to access it from the lowest level. Usually, libraries or SDK (Software Development Kit) are provides as an extension toolkit for further development. Therefore, the MAI driver for each device should be created separately. This layer design is similar to an Operating System, which accept new device but requires the driver to be "installed". No standard driver can be used by several devices even though they are of the same type. The generic interface is used for device to communicate with the system. Compared with other systems, the way in which the interface is designed to support various devices is the main differences. The interface only rules how data is formatted, while other systems are sensitive to the data itself.

Furthermore, a generic simulator and a playback server are necessary for experimental purpose. The design ideas are the writing of virtual device driver which manipulate the interface communication with the MAI System. By doing so, from the MAIServer perspective, the DAQ layer is a network interface that provides data in a proposed format. In short, this layer will be powerful and flexible to accomplish a set of tasks in the future.

4.3.2 MAIServer

The MAIServer can be divided into two parts: the outer network interface, and the inner functions that maintains the client sessions.

In short, the MAIServer is the central server of the system, which manages device and client connections. After considering several server architecture designs, I chose to extend the single process concurrent server described on page 4.1.1. In comparison with NeuroServer, which also uses signal process concurrent design, my design separates raw data and metadata into two different channels. In the Figure 4.7 of the MAIServer structure that I designed, the TCP/IP interface is the only interface available to clients. Using reliable transmission, metadata will be delivered safely for the MAIServer to create client sessions. In order to be compatible with the OpenEEG project, raw data is allowed to go through a

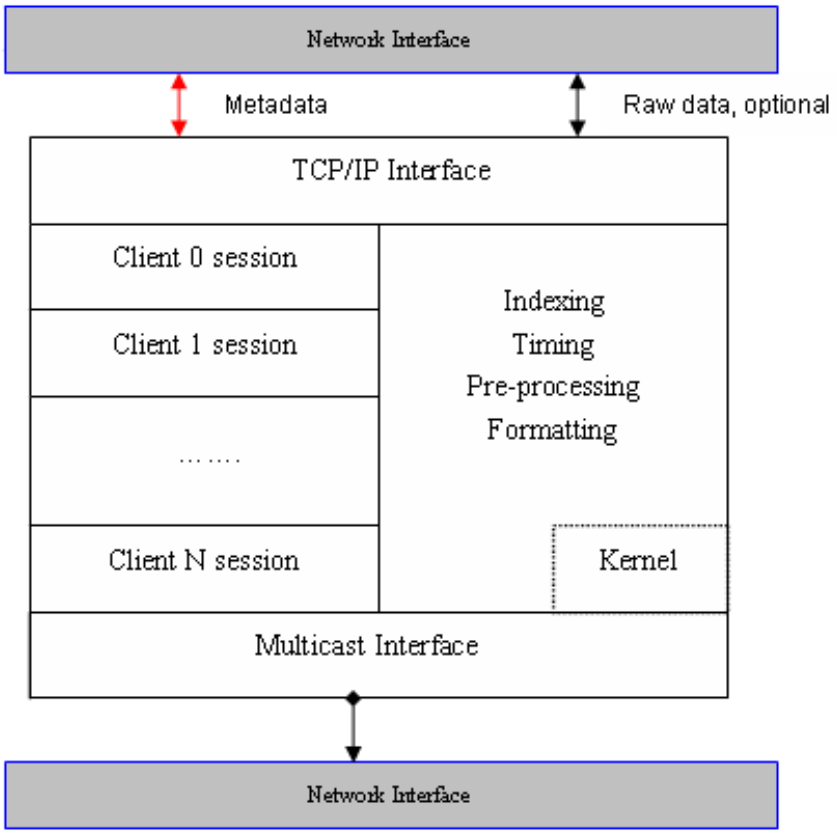

Figure 4.7: The MAIServer design overview. It offers indexing, timing, pre-processing,formating

TCP/IP interface but will be internally forwarded by a multicast interface. In comparison with the NeuroServer design, this solution will dramatically improve the performance of the MAIServer, reduce the whole system latency, distribute the system overhead and cut down the bottlenecks. Figure 4.8 compares the data flow in the compatible mode and multicast mode.

Generally, I/O is recommended as bottleneck of the whole system. Figure 4.8 shows the network I/O reduced in the MAIServer by switching a driver mode to multicast. As the total traffic goes through the MAIServer in compatible mode on the left, this design is more likely to become the bottleneck of the system because of the data throughput capacity of the MAIServer. On the contrary, the MAIServer in multicast mode only handles metadata and helps clients to

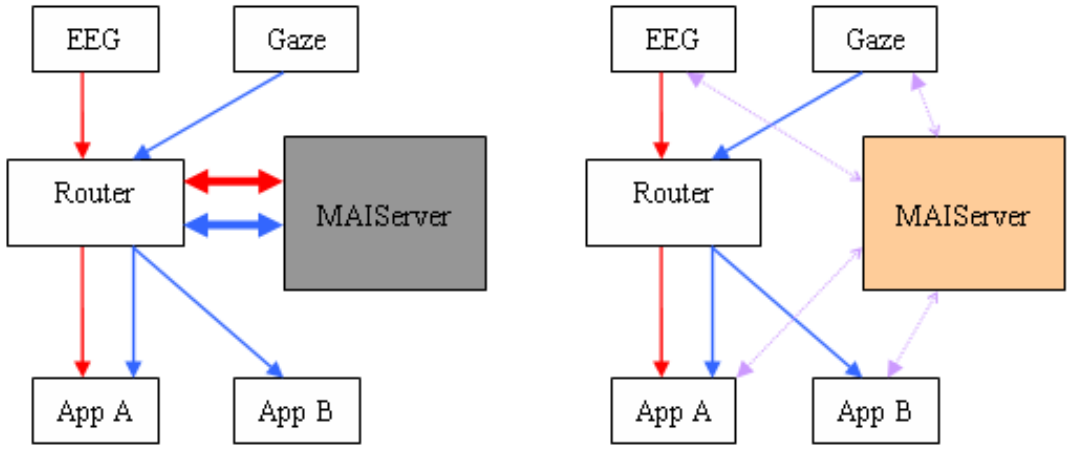

Figure 4.8: Two different MAIServer runtime modes. Different colour arrows represent different types of data. The line thickness represents data throughput.

establish peer-to-peer (p2p) connections.

4.3.3 Software Libraries

The software libraries that I intended to create are a set of open source, reusable C/C++ libraries to support the creation of future MAI applications. These libraries will provide a toolset for connecting to the MAI Interfaces. With the MAI libraries, users gain flexibility of fast programming with simple socket interface and data parser objects. Communication protocols in the libraries will be function-based and protected from unnecessary modification.

Chapter 5

Implementation

In this chapter, the implementation of the Mind Attention Interface System will be based on the design described in Chapter 4. To start with, a high-level overview of the hardware layout is shown in Figure 5.1 to demonstrate the possible devices and computers in the system.

As we can see from the Figure 5.1, a number of devices and computers are involved in the MAI System. The hardware is classified in four categories based on the MAI System layers: Measuring devices, together with their controlling computers, are located in the Data Acquisition layer. Devices or computers that display data are located in the application layer. The computer that runs the MAIServer is in a separate layer. Finally, yet importantly, machines that run the Spectral Engine are located in the Signal Processing layer. To make the hardware in these layers function together, the MAI System enabling components (including device drivers, MAIServer, software libraries) need to be constructed. In addition, the implementation of a generic communication protocol is equally important. Overall, the implementation will follow the outline that I described in previous chapters, and can be divided into three parts: software environment considerations, MAI System enabling components implementation, and construction of useful utilities.

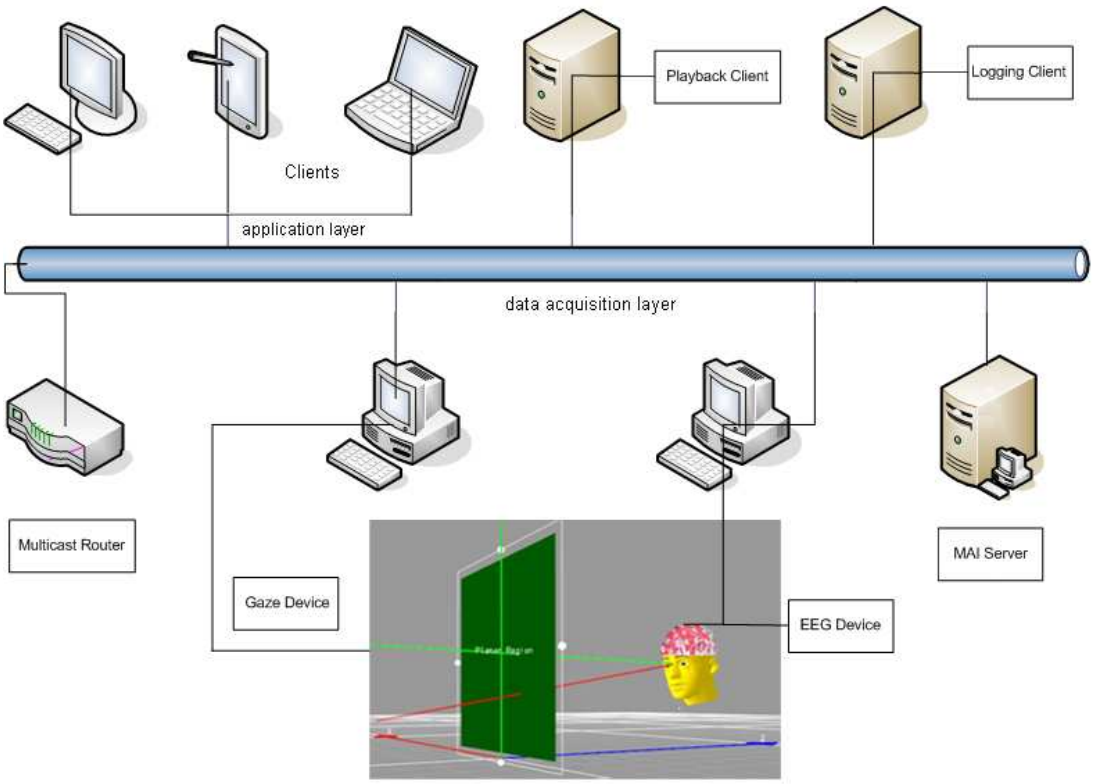

Figure 5.1: The MAI System hardware layout

5.1 Software Environment considerations

- Operating System

  Microsoft Windows 2000/XP were chosen as our initial operating system. Windows is the world's most popular operating system and hardware devices companies target Windows users for marketing purposes. Thus, to ensure we put our most effort on system design and usability, we developed our Mind Attention Interface modules on Windows.

- Programming Language

  C/C++ is a high-level programming language with low-level functions. It has become one of the most popular commercial programming languages since 90s [54]. Most of the comparable systems we have evaluated use

C/C++ as their initial programming language [6, 61, 62, 37]. To construct an extendable system and archive more flexibility, infrastructure of the MAI System is developed under C/C++. The reason that we use both C and C++ is that part of our drivers are build on top of the commercial C/C++ API that comes with hardware. Concerning the variety devices and applications that might be added into the system, the MAI System is not limited to one programming language. The development of supplementary modules can be done in other forms.

Mind Attention Interface enabling software including the device drivers, the MAIServer, the Application Programming Interface libraries, and some MAI clients have been developed under the above system configurations. However, due to the network protocol used in the MAI System, new devices and new clients can be developed in other languages or platforms, and added as "plugged and play" hosts. One example would be Wii remote controller purchased in early 2007, the driver is developed under C# programming language. Another example was given by Sheridan in [49], he developed a sound demonstration in Wedge virtual reality theatre using Java. (Modification has been done to the original implementation to be able to communicate with the MAI System.)

5.2 MAI System Enabling Components Implementation

The first implementation was to provide a collection of the MAI drivers for the Data Acquisition layer. This involves the fetching, normalising, and sending data. The sending part that uses the generic protocol in communication with other layers is the most important step. It was implemented as an interface that can also be used in other layers.

Another key issue was the implementation of transmission protocol used in the communication procedure. As described earlier in Section 4.2.2 on page 39,

data transmission over TCP is used for compatibility with the OpenEEG system. However, the NeuroServer implementation of TCP transmission introduced large latency due to the buffering of data. Modifications have been done to reduce the latency by sending data in non-buffered TCP mode. Appendix D gives a detail discussion over the optimisations to the TCP transmission in the MAI System. In summary, the MAI System supports three transmission mode: buffered TCP mode, non-buffered TCP mode, multicast mode. Generally, multicast mode, which employs the UDP for transmission, is used as the default data-delivery mode.

With regards to the MAIServer, the NeuroServer source code was modified and used as a fundamental framework of a single process concurrent server. Later on implementation included the generic interface modification, the timing and pre-processing, and finally the support of two extra transmission modes (non-buffered TCP and multicast).

The libraries that I created are instances of the MAI Application Programming Interface. The libraries fetch and buffer data in a ring-buffer data structure, and these data can be accessed locally without network programming. Although windows libraries are platform dependent, the advantage of creating these libraries is to reuse them across multiple projects in Windows platform. The methodology that I use to summarise the interface is also an innovation. Further library development on other platforms following the same function declarations can be used for application migration.

As has been mentioned, Spectral Engines are largely the work of others, except the interfacing components. The Mathworks product, MATLAB®and SIMULINK®was chosen as their development language and environment as they claimed to perform computationally intensive tasks faster than traditional programming languages such as C/C++. The collaboration between Spectral Engine researchers allows me to evaluate the interface connection usability when a third party software trying to plug into the MAI System. Therefore, the connection

part was redeveloped several times as each time we come across problems. On one hand, this is because the variation of software packages that the connection interface dealt with. On the other hand, the efficiency of implementation needs to be improved.

To accomplish the task, my first attempt is to create MEX (MATLAB Executable) applications. MEX is a built-in utility that enables us to call C code in MATLAB by compiling your code into a MATLAB Executable. This work reuses the libraries source code, but only serves MATLAB Engine. SIMULINK is the graphical extension for MATLAB and implementation is similar. When xPc Target * was introduced, I eventually gave up the old model and designed the Spectral Engine Connection Kit (SECK). The SECK gives the opportunity of using the standard network utilities comes with the engine. For example, xPc Target which supports neither TCP nor IGMP† can now use UDP receiver. In terms of the time spent on cooperation, both sides now concentrate more on parsing incoming data instead of developing modules that cannot be reused. To conclude, SECK interpreted information to match the destination interface for both sides without modification to original design.

5.3 Utilities implementation

In the MAI System, utilities have been designed to provide better user interaction and offline analysis ability. The Logging Server and the Playback Server are instances of these utilities in application layer.

- Logging Server

  It is important to record and retain experimental data for research purposes.

*A MathWorks product which claims to be a high-performance environment that enables you to connect your SIMULINK models to physical systems and execute them in real time on PC-compatible hardware.

†Internet Group Management Protocol, a communications protocol used to manage the membership of Internet Protocol multicast groups.

Data can be recorded in the driver layer. Device drivers not only multicast messages, but also log them in a standard file format. (EEG in EDF/BDF format and Gaze data in faceLAB format) However, this type of logging might add extra output latency in some cases such as writing large amount of data to harddisk. In this case, the Logging Server can be used. This server registers to the multicast group, receives data from network, and logs everything happening within the system in ASCII text format. It is running on a separate machine and does not add extra overhead to the existing system. Data from multiple devices will be logged through a single interface and stored separately into files. The log file format is specially designed and can be used to replay the whole experiment. It records the time when each incoming message arrives, which will be used for synchronisation and playback.

The advantage of the driver-layer log files are that we can distribute or share them with other research groups. It can also be used in other systems such as LabVIEW as an input. Thus, we can compare experimental results in different systems using the same input file. In contrast, the Logging Server format can be read and analysed easily. In addition, the time stamps show the latency and jitter introduced by network and application.

- Playback Server

  Playback is a function that I introduced to validate results. For example, we might just replay one set of data many times, and apply different signal processing methods to it. We might also use some well-recorded log files for testing purposes, and thus, we do not need human subjects all the time.

  Messages that arrive at destination machines will jitter due to software issues and network transmission. I have tried to build an environment that has the same behaviour as when the experiments were carried out originally. Thus, I prefer to use time stamps instead of the device-cycle clock while

playback. The replay function is triggered by time stamps in the log file. Time stamps would show the exact sequence and interval of any action in the system. This is extremely convenient for devices that do not have a fixed sampling rate, for example, a keyboard that I used to send control or label messages.

Besides the above utilities, I have also developed the Generic Simulator that can be used to simulate an input device in the DAQ layer. This utility can read from a data set or generate random data, which gives me the ability of quick testing.

5.4 System Verification

To conclude, the implementation of the MAI System has been accomplished by constructing modules marked in red block shown in Figure 4.1 on page 33. In order to verify the implementation, all modules have been tested individually to make sure the correctness of data throughput. The whole MAI System has also been verified by running the configurations outlined:

- The MAI input layer has been tested with the following devices: BioSemi, faceLAB, generic simulator, and Playback Server.

- The application layer has been tested with the following clients: a graphical demonstration in the Wedge virtual reality theatre including sound, and the Logging Server.

- The Spectral Engine Connection Kit (SECK) has been tested to ensure reliable operation of the Mind Attention Spectral Engine (MASE).

- Devices and clients have been configured to use either TCP mode or multicast mode.

- The multicast interface in the MAIServer has been enabled for multicast clients.

  At the time of writing, the MAI System has also been verified via practical means by other group members. Ben Swift has used the system to build a computer music instrument, which will eventually be driven by a human mind [52]. This instrument uses the BioSemi EEG driver, the MAIServer, the Spectral Engine Connection Kit (SECK), Swift's Mind Attention Spectral Engine (MASE), the music generation engine, and the sound system of the Wedge theatre. Figure 5.2 shows this so-called Mind-Modulated Music interface:

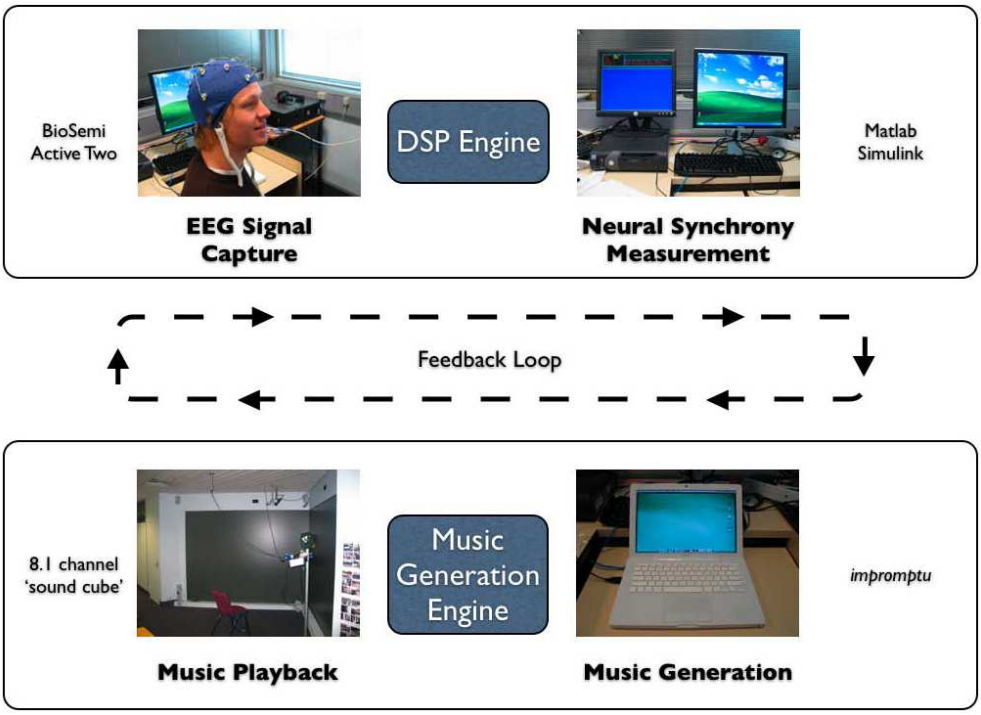

Figure 5.2: Mind-Modulated Music in the Mind Attention Interface, from [52].

The Mind-Modulated Music interface has been used to demonstrate the use of the MAI System to coordinate the activities of five computers over a local area network. Signals from the BioSemi EEG device were sent via multicast to an instance of the MAIServer, which monitors the status of

all clients. The MASE was deployed on a dedicated machine to provided "neural synchrony" measurements to the whole system as a Digital Signal Processing (DSP) engine. The SECK was used as a bridge between the EEG raw data and the MASE, and finally sent music information to impromptu [20] to generate live music into the Wedge theater.

Chapter 6

Evaluation and Testing

The software validation tests, mentioned in the previous chapter, give confidence that data flows correctly though all parts of the Mind Attention Interface System. Performance profiling of the MAI System is the topic of this chapter and this has involved a set of experiments on key parts of the system. Figure 6.1 shows a view of the test system setup with the profiled modules shaded.

The MAI System is not trying to perform operations that require conventional, real-time predictability. However, studying timing and latency issues provides feedback on the system design. By running a set of tasks described below, I have measured the latency caused by the data acquisition layer (in Section 6.1) and comments are made about how these results compare with other software. I have also examined the performance of the network interface in different system configurations (in Section 6.2). In the last Section 6.3, profiling tests have been done to measure the data throughput capacity of the MAIServer.

6.1 Data Acquisition Latency

Data acquisition latency is caused by two factors: device-level latency and system latency. The first is the average time it takes to output data after it has been captured by hardware. This is determined by the hardware (by the sampling

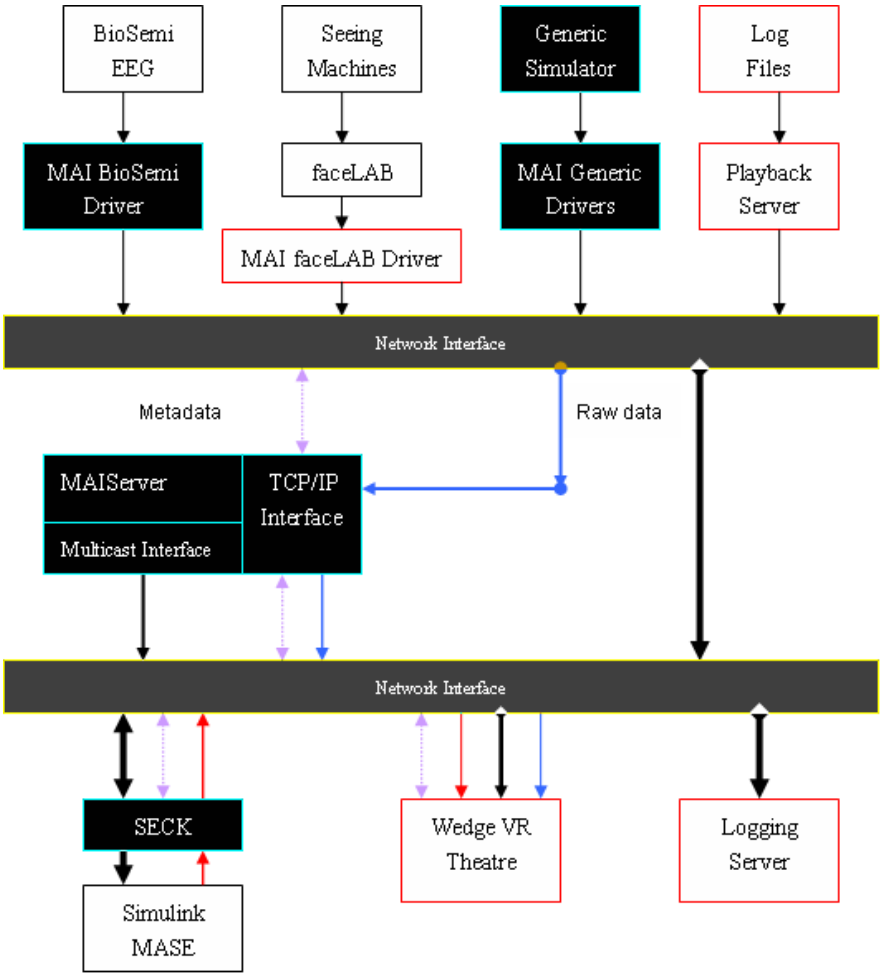

Figure 6.1: MAI test system setup after Figure 4.1 on page 33. Shaded modules have been profiled.

rate for instance) and the algorithms used in the manufacturers' software. For example, faceLAB introduces 25ms latency to the SeeingMachines gaze-tracking system as described in Section 2.1.3. Any additional latency introduced by the MAI drivers can be called "system latency". This is measured from the time when the data has been output until the MAI driver is ready to send the data on to the rest of the MAI System.

The role of each of the MAI drivers is to fetch, format and send the data. One example is the algorithm that I used in the BioSemi MAI driver, which consists of four steps:

1. The driver checks whether the required amount of data is ready.

2. If the data is ready, go to step 4.

3. If the data is not ready, then sleep for 10 ms and go to step 1.

4. The driver formats and processes the data. Loop back to step 1.

As we can see, the last step will always be similar in different device drivers. The differences between the MAI drivers are how they check for the data in Step 1, and how long they sleep in Step 3. For instance, the BioSemi EEG device places digitised data in a memory buffer, whereas SeeingMachines sends data out to network through UDP. For Step 1, the checking of EEG data is done by calling a manufacturer-supplied library, whereas the checking of gaze data, which is obtained from another computer, is done by listening to a socket. With regards to the sleep time in Step 3, the time stamps that we can obtain from the Windows operating system are generally limited to a maximum resolution of 10 or 15 milliseconds, depending on the underlying hardware. The sleeping time needs to be a factor of this resolution. Because all MAI drivers conform to the four steps outlined above, it is sufficient to consider only one particular MAI driver to understand the nature of the system latency in the DAQ layer. Since the BioSemi EEG device has the highest demand for system resources, network bandwidth and has the highest sampling rate among all existing devices (which means that it is more sensitive to latency in comparison to other devices), it was used in the following experiments to study the system latency.

As has been noted, the time resolution in the Windows operating system is not sufficient to give reliable information about latency. In order to measure latency with a resolution greater than 10 milliseconds, I developed a timing module which uses the Central Processing Unit (CPU) of the driver computer as a high-resolution hardware counter. This module returns CPU time that has elapsed since the time that execution of the program commenced. The number of clock cycles measured by the hardware counter can be converted to time in millisec-

onds by dividing by the CPU frequency. This enables a measurement of system latency with high resolution and low overhead.

The BioSemi EEG device was set up to run on 16 channels at 2048 Hz, sending data to the MAIServer in TCP non-buffered mode. The driver was tuned to fetch a different number of data blocks from the buffer each time. (One EEG data block contains all samples generated within one cycle, which is $3*N$ bytes, with N being the number of channels.) Table 6.1 shows the system latency against the number of blocks fetched each time.

No. of Blocks	data generation time (ms)	latency in step 4 (ms)	system latency (ms)	latency to select 64 blocks (ms)
1	0.49	0.33	0.39	$0.39 * 64 = 24.96$
8	3.91	2.28	3.35	$3.35 * 8 = 26.80$
32	15.63	9.68	14.21	$14.21 * 2 = 28.42$
64	31.25	19.74	30.84	$30.84 * 1 = 30.84$

Table 6.1: system latency in milliseconds against the number of blocks fetched each time. The data generation time is calculated from the sampling rate as the time required for the hardware to generate a certain number of blocks. PC configuration was Intel®Core 2 CPU@2.13GHz, 2GB RAM

As can be seen in Table 6.1, one noticeable trend is that the system latency is always smaller than the data generation time. This shows that the driver is consistently processing data faster than the device can output data, which will prevent the memory buffer overflowing or being overwritten. The latency introduced by algorithm Step 4 increases linearly against blocking factor (1, 8, 32, 64) because the blocking factor represents the amount of data that the driver needs to process in each algorithm cycle. With an increase in the blocking factor, the system latency also increases. The last column shows the system latency normalised to a blocking factor of 64 blocks. We can see that the system latency increases with fewer fetch times. This is because for larger blocking factors, the

data generation time is longer; it is then more likely that the driver will end up sleeping in Step 3. In theory, system latency can be reduced by checking less data in Step 1, and by decreasing the sleeping time in Step 3. While sleeping time in the MAI System cannot be reduced to less than 10 ms, the minimal latency (0.38 ms) can be achieved by fetching one block at a time. Because smaller blocking factors increase the system overhead, in the MAI System, I normally fetch 8 EEG blocks at a time, and, in average, 3.35 ms latency is introduced into the BioSemi MAI driver.

My last approach to demonstrate the effectiveness of the data acquisition layer is by comparing it with another, general-purpose system. Among the systems that we have examined, BCI2000 is the only comparable system that provides measurements of the system latency in data acquisition. The measurement results listed in [45] are based on different EEG systems and the data acquisition algorithm is not mentioned. Nevertheless, the shortest latency listed is 3.22ms, which still gives us some idea of the average acceptable latency that another system can achieve, and this is comparable to that obtained using a blocking factor of 8 as described above.

The BioSemi driver that I wrote has been contributed to the BCI2000 archive as part of an ongoing collaboration with that scientific community.

6.2 Network Performance

6.2.1 Performance measurement

Network performance is one of the most important aspects of the MAI System because the whole system is highly distributed. Because of this, network transmission has been monitored and data throughput has been measured for representative system configurations. The comparison of the network performance of the three different transmission modes (multicast, buffered TCP, non-buffered TCP) described below was undertaken to verify the network design of the MAI System.

The results also indicate the advantage of each transmission mode, which means that the system can be optimally configured for best performance.

In the same way that I tested the driver latency, the BioSemi EEG device was used in the network experiments for data generation. The device was run with 16 channels at 2048 Hz, and the data was transmitted to the MAI System using each of the three transmission modes. The System Performance Monitor (SPM), which is part of the Microsoft Administrative Tools, was used to log the real-time data throughput of the MAI System. This utility was running on a separate machine and acquired performance data over the network. Table 6.2 shows partial results of three individual tests:

	Multicast	Buffered TCP	Non-buffered TCP
Bytes Sent/sec	333470	267714	352589
Packets Sent/sec	2041	200	2044
Bytes per Packet Sent	163	1338	172
%Processor Time	7.9	0.8	7.7

Table 6.2: The network performance and data throughput when the BioSemi driver runs in different modes. The term packet is used to refer to segments in TCP and datagrams in UDP. Bytes Sent/sec is the rate at which bytes are sent over the network, including headers. Packets Sent/sec is the rate at which packets are sent. % Processor Time is the percentage of elapsed time that the processor spends executing a non-idle thread. (Explanations from Microsoft Developer Network [35]) Buffered and non-buffered TCP modes were run on the PC configuration: Intel Core 2 CPU@2.13GHz, 2GB RAM. Multicast mode was run on the PC configuration: Intel Pentium^{TM}M @1.86GHz, 1GB RAM

Note that the above tests were carried out in a networked time-sharing environment with the presence of other processes affect the data slightly. (For example, 11 TCP segments were introduced for the Multicast mode.) Since the BioSemi generates 2048 samples per second in the experiment, we can see from Table 6.2, the packets-sent-per-second shows the number of EEG data blocks in each packet. As we can see, a packet is the combination of 10 EEG samples in

buffered TCP mode, whereas it is one single EEG sample in other two modes. Thus, in order to send the same number of EEG samples, buffered TCP mode sends fewer packets and, therefore, uses less processor time.

As mentioned previously in Section 6.1, one EEG sample contains all data generated within one cycle, which is $3 * N$ bytes, with N being the number of channels. When the EEG sample was wrapped by generic protocol in a text format, the size of a data message is larger than the original size. (see Section 4.2.1 on page 38 for details) In the above experiments, 16 channels EEG was used and each data message was 130 bytes in average. We can see that the Bytes-per-Packet-Sent in Table 6.1 is larger than the data message size. (Bytes-per-Packet-Sent is calculated by dividing the number of Bytes Sent/sec by the number of Packets Sent/sec.) This is due to the TCP/IP (or UDP/IP) header included in each packet. In our case, TCP transmission over IPv4 adds 40 bytes header to each packet, whereas UDP transmission over IPv4 adds 28 bytes header to each packet.

Since fewer headers were added to send same number of EEG samples, buffered TCP mode saves bandwidth in average compared with two other modes. However, the disadvantage is also obvious. Each packet in the buffered TCP mode contains 10 EEG samples in average, and latency caused by buffering is 5 EEG samples generation time. This latency seems small for the BioSemi EEG device. For example, when BioSemi EEG device runs at 2048Hz, this latency is 2.44 ms. However, considering devices run at 60Hz (SeeingMachines gaze tracking system for instance), there is 83.33 ms delay.

In summary, the experiments show the data throughput of three transmission modes. The results are consistent with the theory, which validate our initial design considerations:

- Buffered TCP mode was designed to support EEG applications that connect to NeuroServer. It provides reliable data transmission, uses least network bandwidth, but has the most latency.

- Non-buffered TCP mode was designed to reduce the latency of buffered TCP mode. It provides reliable data transmission, but uses more network bandwidth. Both buffered and non-buffered TCP modes rely on the data throughput capacity of the MAIServer, which will be discussed in the next section.

- Multicast mode employs UDP transmission protocol, which provides unreliable data transmission. However, this design has the least system latency and best performance, and uses less network bandwidth than non-buffered TCP mode. This design also removes the bottleneck of the system by sending raw data directly to clients.

6.2.2 Integrated Performance with Mind-Modulated Music Interface

Although the emphasis of this thesis has been primarily on measurement of the system performance in a test environment, the actual performance of the system with a user has also been studied. A particular integrated system, the Mind-Modulated Music application that has been described in Chapter 5.4 on page 58, was used to measure the actual system performance compared to the previous test systems. The main aspects examined were the bandwidth usage and the latency introduced. The EEG device was run with 16 channels at 2048 Hz, and data was transmitted to the MAI System using multicast mode, which has been validated as the most efficient transmission mode. Data was received by the Spectral Engine Connection Kit (SECK) and retransmitted to the Mind Attention Spectral Engine (MASE). After the MASE processed the brain signals, a set of music data was sent back to the SECK. This music data was then send from the SECK to the music generation engine.

Due to the fact that the output from the MASE was generated at a frequency of 4Hz and an average size of 10 bytes [52], we can see that the bandwidth impact

from MASE to SECK was significantly smaller than that due to the raw EEG data. Using the figures given on page 67,(each data message of 16 channels EEG was 130 bytes in average), approximately 0.015% more data is expected to be received by SECK compared to what the MAIServer received in TCP mode. A average of 369840 Bytes Received/sec was recorded under these tests, and the processor time on the SECK machine is slightly smaller, at 7.6%, compared to the last column in Table 6.2 on page 66. This small difference was because the SECK shares the same architecture as the MAIServer, but is less complicated in the way it handles data.

With regards to the latency introduced, the primary concern is the extra hops introduced by the MASE. In a system that does not require the use of the MASE, data was transmitted directly from the device driver machine to the destination (in multicast mode). However, when the MASE was introduced, an extra 2 hops occur as shown in design option C in Figure 4.2 on page 36. Since the MASE was unable to communicate with the MAI System without the use of SECK, 4 hops will be finally introduced in the real world system. Therefore, the main differences between this integrated system and previous test systems are the extra network latency introduced shown in the left bottom corner of Figure 6.1. In particular, the MASE in this system provides the measurements of neural synchrony to the music generation engine: it down sampling the EEG data to 256Hz, uses a complicated algorithm to measure neural synchrony, and then outputs results at 4Hz. (A frequency considered sufficient for the music engine to generate meaningful data). Because it is out of the scope of this thesis to study the latency in the MASE, the focus was to estimate the time spent in SECK and network latency of each hop. Using the timing module mentioned in the previous section, an average of less than 0.1ms was introduced by the SECK from the time a data packet arrived to the time a data packet was forwarded. The network latency introduced in each hop can be estimated to the value of round-trip delay time, which is the time elapsed for a message to travel from a computer, across

a network to another computer, and back. In a LAN, this time is always small enough to be insignificant. A system network tool "ping" was used in our system to estimate this round-trip time, and the result was less than 0.5ms. (Windows Operating System shows $< 1ms$ and Linux system show $0ms$) Because of the differences between the input and the output of the MASE, as well as the overlap window algorithm it used to process data, it is almost impossible to correlate the output with a certain number of input EEG samples and, therefore, the latency was not measured in an accurate way in this particular system. Nevertheless, the maximum network latency due to the extra hops was under 3ms.

The human performance of the system performance has been estimated using an informal "think and listen" method at various data acquisition rates. In order for users to indentify the response from a music engine, music will be played in the Wedge after it has been generated. Feedback from one music lover indicated that the concentration on the musical stimulus did seem to be rewarded with an increase in musical complexity; however, changes in the musical stimulus occur slowly in response to his thoughts and attention. In addition, some comments suggested that the visual feedback seems to be a distraction of human attention. The creator of the music engine said that the slowness response was due to the design of the music generation engine, and recommended that users closed their eyes in order for a clear signal to be collected. Another noticeable trend is that the music did not seem to be affected by EEG sampling rate. This can be explained by the fact that this particular MASE had an input threshold of 256Hz. To summarise, the music generation application generated listenable music, and provided a chance to study the performance of the MAI System with a human user. The latency of the MAI System with the MASE plug-in largely depended on the real world network, whereas the bandwidth was considered sufficient for existing software.

6.3 MAIServer Data Throughput Capacity

In the MAI infrastructure, the data throughput capacities of all the software components are limited by the bandwidth of the network. The MAI system is carefully designed to avoid hitting this limitation, by distributing each component over a local area network (LAN). Thus, software such as the MAIServer have a full bandwidth access to the Ethernet, and the challenge has become to optimise the data throughput capacity each component can handle. As discussed in Section 4.1.1 on page 33, a single process concurrent server architecture has been chosen for the MAIServer, and, in this section, experiments are described which profile the data throughput capacity of the MAIServer in TCP mode. Since the Spectral Engine Connection Kit (SECK) uses similar architecture to the MAIServer, this profiling also indicates the data throughput capacity of the SECK.

In the MAIServer, each incoming packet triggers an execution, and the length of the execution time determines the data throughput capacity of the MAIServer. There should be enough time for the MAIServer to process each message before system overload. When the MAIServer finishes processing the current message, it blocks and listens for the next message. With an increase in data throughput, the MAIServer spends more time in processing and less time in blocking. The first part of the performance measurement is the profiling of the MAIServer with different amount of data throughput, which gives us the precise time costs of individual functions. The second part is to find out the MAIServer data throughput capacity by increasing the serving overhead of the MAIServer. As has been noted, while the MAIServer runs with Multicast mode, execution will only involve the processing of metadata. The amount of data is relatively tiny in comparison with the raw data. As a result, experiments were set up in non-buffered TCP mode.

Two machines were used in the experiments: one machine ran the MAIServer and the other one ran the generic simulator. The MAIServer was configured to forward raw data through the multicast interface to possible multicast clients. This setting simulated the heaviest conceivable overhead. A number of generic

simulators between 5 to 30 were connected to the MAIServer and sent data that was randomly generated for 3 minutes. Each generic simulator was generating 16 channel data at 50Hz, and the average packet size was 90 bytes. Table 6.3 shows the time costs in MAIServer against the number of clients connected. The results were collected using the Visual Studio Performance Tool. *

No. of Client	Kbytes Received/sec	%Blocking-time
5 clients	22	97.822
15 clients	66	93.492
30 clients	132	91.992

Table 6.3: MAIServer profile against number of clients connected. Kbytes Received/sec is the amount of data received by the MAIServer. %Blocking-time is percentage of time the MAIServer blocks.

As we can see from Table 6.3, with the increase of clients connecting to the MAIServer, system blocking-time is decreasing gradually. Instead, the MAIServer spends more time on handling packets from clients. When blocking-time approaches zero, the MAIServer approaches its data throughput capacity.

The second test is designed to simulate high sampling rate devices such as the BioSemi EEG device to obtain the value of data throughput capacity. Similar to the previous setup, one generic simulator was used and this client sent packets that were chosen from a set of real EEG data. With an increase of sampling rate, the amount of data the MAIServer can receive hit a peak of 2026 Kbytes/sec.

In addition to the measurement of data throughput capacity, the profiling experiments also give us an overview of the runtime performance of the MAIServer. The most time-consuming functions are the memory copy and string format, which cost 38.5% of the processing time. In summary, although BioSemi system total throughput remains constant at 1.5 Mbytes/sec, which is way below the

*Microsoft Visual Studio 2005 is Microsoft's flagship product for computer programmers. It offers a rapid development platform for C/C++ applications.

bandwidth limit of the Ethernet, it is obvious that the MAIServer running in TCP mode could only handle 2Mbytes/sec. Once more devices or clients are connected to the system, both data throughput and bandwidth will affect the overall system performance. The above tests indicate that TCP mode is not suitable for heavy data transmission tasks. Since multicast will not be affected by this limitation, it is considered more robust and efficient. In contrast with TCP mode, it is able to handle much more devices and clients, without hitting the peak of data throughput.

Chapter 7

Discussion

7.1 Contribution

The Mind Attention Interface System is a flexible platform which is intended to support research into the interactions between the human mind, and human attention, and virtual environments. The goal of my project has been to design the MAI System architecture, develop infrastructure parts, and to support development of future applications.

Figure 7.1 shows the four major layers of the MAI framework. The infrastructure constructed implements indispensable modules of this system as well as a set of usable software applications. It is self contained enough to be able to set up and run experiments with human participants. It should be easily extended to incorporate new input devices, new signal processing algorithms, and any number of client applications. My original contributions are summarised below:

- Data Acquisition Layer

  I have worked with several different types of devices while developing this layer. Expensive commercial devices like BioSemi or SeeingMachines provide their own software for data acquisition while other devices, like trackers, offer straight acquisition through the serial or USB ports. Due to the above reason, there is no standard data input channel. Data are collected

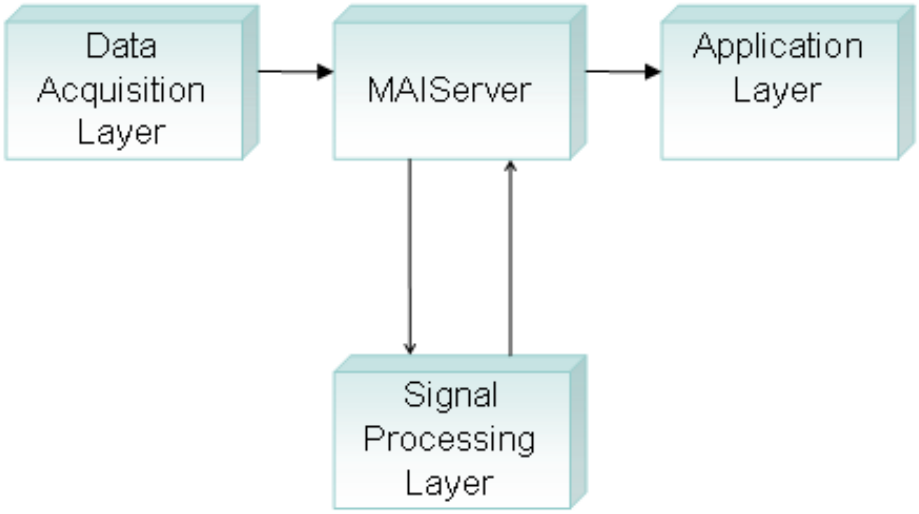

Figure 7.1: Four major layers of the Mind Attention Interface

from the network, acquisition library, serial port or USB port. Developing drivers for these devices required different techniques, and we expected the implementation of new device drivers could be done easily. Other systems would limit the driver to be in a framework such as inheriting base device. These structures limit the use of devices that are outside the current framework. Examples system such as VRPN [62] or VRJuggler [61] do not have a Gaze or Brain base class, let-alone devices that are designed for other platforms or operating systems. I have tried to provide an overall format for outgoing data so that developers will gain more control over the driver development. Finally, I have extended the command-based protocol that is used in the OpenEEG (see Section 3.2 on page 24) project. The protocol allows a server and clients to communicate in a predefined language.

- Mind Attention Interface Server

  The MAIServer is the central server that manages connections and metadata in the Mind Attention Interface. The MAIServer employs TCP for

its transport layer to guarantee safe delivery of metadata. It treats devices and clients as peers without making a distinction. Most of the peer-to-peer information flow does not go through the MAIServer unless the devices are specified to do so. Multicast is introduced in data transmission. Thus, the system does not rely on limited computing power or the bandwidth of the central server. Header information and client connection status are maintained in the MAIServer sessions. Clients use enquiry commands that are pre-defined in the protocol to acquire metadata. In order to support applications that use NeuroServer protocols, the MAIServer has a special mode to support data coming through TCP. Although this particular mode limits our device-sampling rate, we benefit by being able to use various existing applications.

- Application Programming Interface Library

  The creation of an API helps potential users to develop MAI applications. This software library aims at easy access to the MAI devices and log files, as well as network data. I have implemented a library, which will be linked into code while compiling. The library contains the source code, and header file declarations. While library files might be updated in a future revision, the interface will not change for API users. Users do not need to worry about how things are implemented in the library, as algorithms inside the library are completely hidden from users. The interface shows the incoming and outgoing data format while details are hidden behind the library.

- Spectral Engine Connection Kit

  The Mind Attention Spectral Engine (MASE) is the signal-processing module, which connects to the MAIServer using our Spectral Engine Connection Kit (SECK). SECK enables the connection between the MAIServer and the MASE by fetching and converting data. It provides a two-way connection, serving both sides as an input channel and output channel. By request-

ing device information from the MAIServer, SECK receives data from the multicast address and sends them to the MASE through UDP. SECK also registers a device that represents the MASE in the MAIServer, and multicasts processed data that are received from the MASE. Using SECK, the MASE is treated equivalently as peers by the MAIServer. In addition, there are no restrictions to have multiple signal processing peers: the available sources are all registered in the MAIServer, and outputs from one spectral engine could even be used as an input to another spectral engine.

7.2 Future Development of the Mind Attention Interface

The existing system provides a good example of how to build a platform that fulfils the specified requirements. (See requirement details in Section 3.1 on page 23.) The present MAI System can readily be expanded in the following aspects:

1. Data Acquisition Layer

   Hardware supported in this layer can be expanded when new devices are purchased.

2. Application Layer

   At present, the Mind Attention Interface only offers a limited number of applications. However, the present research agenda of the MAI will result in a number of new client applications to study the human mind and attention in a virtual environment.

3. Application Programming Interface

   Libraries that I have currently developed can be redeveloped in other programming environments to support more operating systems. Although user applications for the MAI System are not restricted to Windows, the libraries I offer do greatly have this limitation.

7.3 Conclusion

The Mind Attention Interface (MAI) aspires to pioneering research in extending Brain-Computer Interfaces by tracking attention in a virtual environment. The inflexibility of most other BCI software systems, and the special requirements of this MAI System, necessitated the construction of the platform described in this thesis. In my design, the performance limitations of the MAI System were eased by moving to a distributed system using a client-server model. Thus, data throughput capacity, rather than processing power, became the critical aspect of this distributed system. The central MAIServer extends the single-process-concurrent-server architecture by using separate message delivery mechanisms for raw data and metadata. In this way, system data throughput capacity can be greatly increased by using peer-to-peer data communication between devices and clients. These design techniques can be used to extend similar systems with performance issues.

At the time of writing, the author also contributed to the writing of two papers, [50] and [52], which describe the Mind Attention Interface and its music generation application. In addition, a considerable amount of imaginative programming tasks has been accomplished, particularly with the construction of the four components mentioned above. The engineering style project started with a thorough investigation of the research requirements and possible hardware options, and then followed by commendable efforts in constructing enabling software and conducting performance evaluations.

The platform that I have built offers a practical experimental and development environment which facilitates the research, evaluation, and expansion of the MAI studies. It is a fundamental part of the Mind Attention Interface and it will prove useful in the future development of the Mind Attention Interface.

Appendix A

Mind Attention Interface Communication Protocol

- 200 OK \r \n
- 400 BAD REQUEST \r\n
- DISPLAY //to enter Display role
- EEG //to enter EEG role
- GAZE //to enter GAZE role
- SECK //to enter SECK role
- CONTROL //to enter Controller role
- ROLE //to print out the current role for this client
- STATUS //to print a table showing the client status, including client role and index
- WATCH <clientIndex> //to receive data from server
- UNWATCH <clientIndex> //to stop receive data from server

- CLOSE //to close an connection
- GETHEADER <clientIndex> //to get client header
- SETHEADER <EDFHeaderPacket> \r\n //to get client header
- ! <packetCounter> <channelCount> <sample0> <sample1> < Time stamp in Driver> \r\n //driver in TCP mode
- ! <clientIndex> <packetCounter> <channelCount> <sample0> <sample1> <Time stamp in Driver> <Time stamp in MAIServer> \r\n //MAIServer in TCP mode
- # <clientIndex> <packetCounter> <channelCount> <sample0> <sample1> <Time stamp in Driver> \r\n //driver in Multicast mode
- 00<clientIndex> <packetCounter> <channelCount> <sample0> <sample1> <Time stamp in Driver> \r\n // 00 is the starting symbol, each number is a float type which uses 8 bytes

\r : a carriage return
\n : a newline character and
// : comments

Appendix B

European Data Format Header

Length in bytes	EDF Header	Description
8 bytes	Byte 1: “0” Bytes 2-8 : “ ”	Identification code
80 bytes	User text input	Local subject identification
80 bytes	User text input	Local recording identification
8 bytes	dd.mm.yy	Startdate of recording
8 bytes	hh.mm.ss	Starttime of recording
8 bytes	(ASCII)	Number of bytes in header record
44 bytes	e.g.: “BIOSEMI”	Version of data format
8 bytes	(ASCII)	Number of data records “-1” if unknown
8 bytes	e.g.: “1”	Duration of a data record, in seconds
4 bytes	e.g.: “257” or “128”	Number of channels (N) in data record
N x 16 bytes	e.g.: “Fp1”, “Fpz”, “Fp2”, etc	Labels of the channels
N x 80 bytes	e.g.: “active electrode”,	Transducer type
N x 8 bytes	e.g.: “uV”, “Ohm”	Physical dimension of channels
N x 8 bytes	e.g.: “-32768”	Physical minimum in units of physical dimension
N x 8 bytes	e.g.: “32767”	Physical maximum in units of physical dimension
N x 8 bytes	e.g.: “-32768”	Digital minimum
N x 8 bytes	e.g.: “32767”	Digital maximum
N x 80 bytes	e.g.: “HP:0,16; LP:500”	Prefiltering
N x 8 bytes	e.g.: “2048”	Number of samples in each data record
N x 32 bytes	(ASCII)	Reserved

Table B.1: EDF File Format Header

Appendix C

System clock offset and Network Time Protocol solution

In the MAI System, NTP is used to synchronise computers in the system. To prove why this is necessary, a calculation of system clock offset without NTP in one single day is followed:

$24 hours * 60 minutes * 60 seconds = 86400 seconds$

The precision of system clock is 16 bit to record 1 second:

$2^{-16} second = 15 microsecond$

System clock ticks: $86400 * 2^{16} = 5.74 * 10^9$ times per day

PPM (parts per million) is one part in 10^6, a precision of 10^{-6}

As a result, in one day, the maximum error ticks are:

$7 PPM * ticks = 40180\ times$ (7 PPM for a normal PC)

Total time offset is $40180 * 15\ microsecond = 609\ ms$

Figure C.1 shows the system clock offset compare to standard time in two months with NTP running. Figure C.2 shows the system clock offset in a typical day. As can be seen, the offset is limited mainly within 10 ms. Considering that the actual time offset within a few hours would normally limited to few millisecond, the results consist with theory. Better results are expected with the purchase of GPS clock installed in local area network.

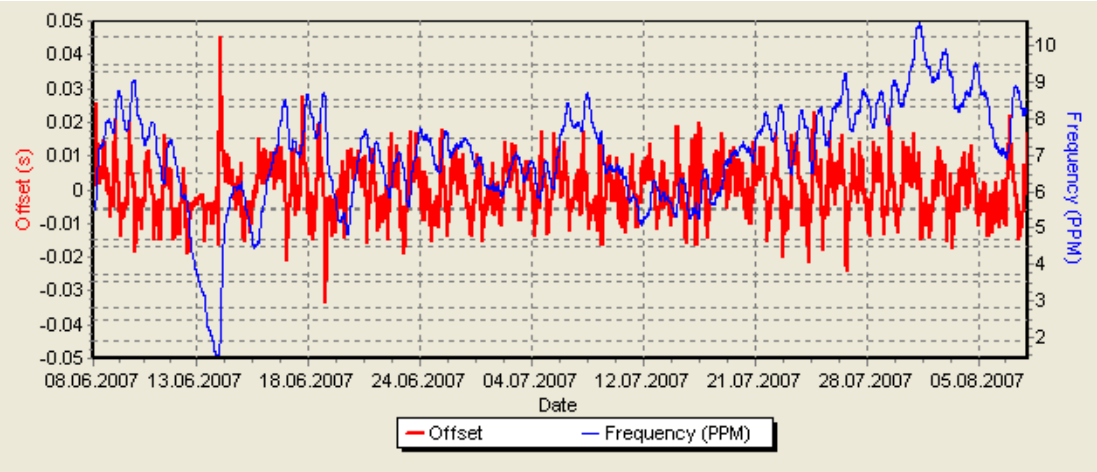

Figure C.1: System clock offset compare to standard time in two months with NTP running. NTP time reference hierarchy in the MAI System after Figure 2.5

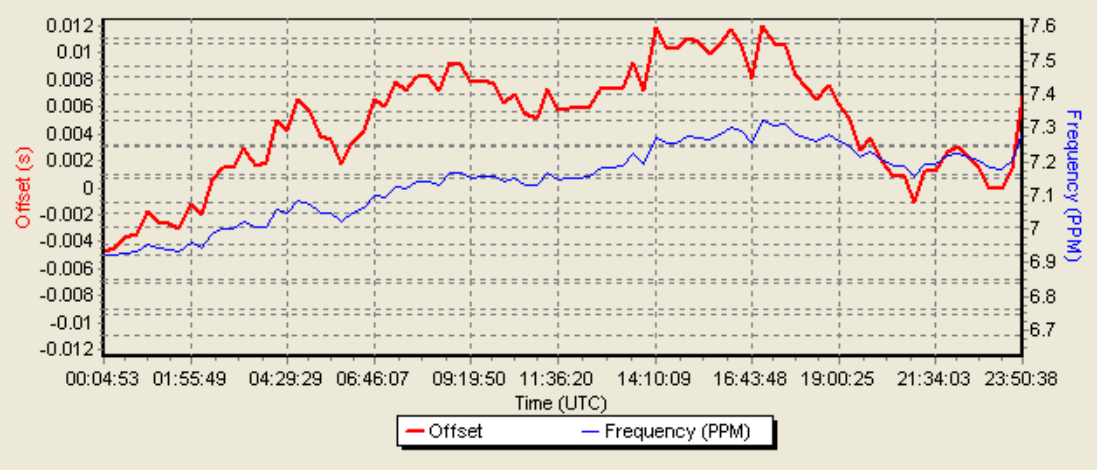

Figure C.2: System clock offset compare to standard time in a typical day with NTP running. NTP time reference hierarchy in the MAI System after Figure 2.5

Appendix D

TCP mode implementation

The following discussion explained mechanisms of transmitting packets using TCP, as well as the optimizations that I applied to original TCP configurations.

The design in Figure D.1 shows a common client-server model for network applications with acknowledgement (ACK). As discussed in Section 4.2.2 on page 4.2.2, when using TCP protocol for data transmission, there are mechanisms to make sure each message been delivered successfully.

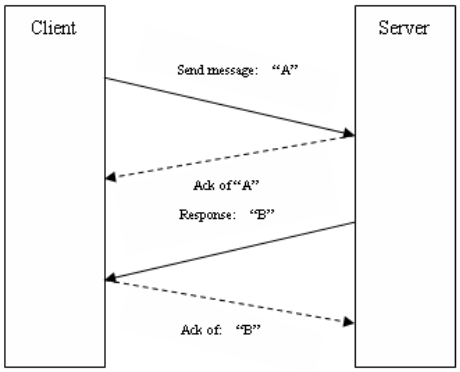

Figure D.1: Simplified TCP mechanism with individual acknowledgement

To avoid sending too many messages, TCP allows us to combine an acknowledgement with a response to an operation. Thus, it almost halves the number of

packets sent by increasing the amount of data in each packet as in Figure D.2.

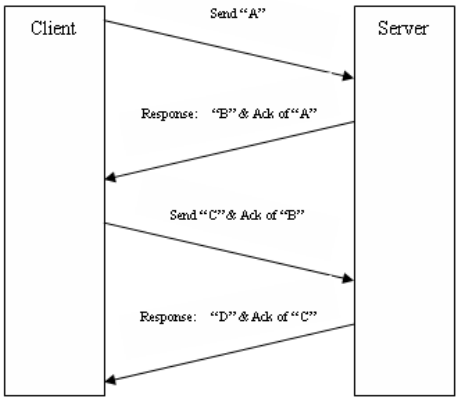

Figure D.2: Simplified TCP mechanism with attached acknowledgement

However, there is a problem with the behaviour of TCP as shown in Figure D.2. As can be seen, this only happens when the server responds to the client. In case where there is no server response, a 200ms timer is set. When 200ms has expired, the server will then send a bare acknowledgement to the client. Because such a 200ms delay is intolerable for the MAI, I have forced the MAIServer to send a special "200 OK" response message when any packet arrives in TCP mode.

Unfortunately, this solution is not as straightforward as it might be. In TCP/IP, to optimize performance at the application layer, a socket copies data buffers from application send calls to a kernel buffer. Then, the stack uses its own algorithms (such as Nagle algorithm [10]) to determine when to actually put the packet on the wire. In most cases, the send completion in the application only indicates the data buffer been copied to the kernel buffer and does not indicate that the data has hit the network medium. This is happening to both client and server side. Therefore, I developed two modes for TCP transmission: the buffered TCP mode that uses the original configurations to optimize performance; the non-buffered TCP mode that send data straight out to reduce latency.

The implementation for non-buffered TCP mode is not just simply turn the buffer to zero. The Nagle algorithm only allows one small segment that has not yet been acknowledged in a TCP/IP connection, which is supposed to avoid having small data packets congest the network. As each of our data packet as well as response message is smaller than "segment size" which on Ethernet is 1460 bytes, it will be blocked due to the previous acknowledgement has not yet been received. Therefore, I disable the Nagle algorithm in non-buffered mode.

In conclusion, depending on the devices resolution, different strategies may apply to different devices in MAI System. Device such as EEG has a resolution down to sub-milliseconds. Other devices like gaze or tracker only have a resolution about 20ms. Non-buffered mode is preferred when data are time critical, however it uses much more resources in comparison with the buffered mode where performance is also important when sending large amount of data. The pros and cons of two modes also related maximum tolerant latency. Results will be shown in Chapter 6 on page 61.

Bibliography

[1] A. Barreto, S. Scargle, and M. Adjouadi. A real-time assistive computer interface for users with motor disabilities. *ACM SIGCAPH Computers and the Physically Handicapped*, pages 6–16, 1999.

[2] J. Bayliss. *A Flexible Brain-Computer Interface*. PhD thesis, University of Rochester, 2001.

[3] J. Bayliss. Use of the evoked potential p3 component for control in a virtual apartment. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Rehabilitation Engineering]*, 11(2):113–116, 2003.

[4] J. Bayliss and D. Ballard. The Effects of Eye Tracking in a VR Helmet on EEG Recording, 1998.

[5] J. Bayliss and D. Ballard. A virtual reality testbed for brain-computer interface research. *Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Neural Systems and Rehabilitation]*, 8(2):188–190, 2000.

[6] BCI2000. http://www.bci2000.org/, June 2007.

[7] BioSemi. http://www.biosemi.com/, June 2007.

[8] N. Birbaumer, N. Ghanayim, T. Hinterberger, I. Iversen, B. Kotchoubey, A. Kubler, J. Perelmouter, E. Taub, and H. Flor. A spelling device for the paralysed. *Nature*, 398(6725):297–8, 1999.

[9] BrainBay. http://www.shifz.org/brainbay/, June 2007.

[10] D. Comer and D. Stevens. *Internetworking with TCP/IP: volume III: client-server programming and applications (Windows sockets version)*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1997.

[11] E. Donchin, K. Spencer, and R. Wijesinghe. The mental prosthesis: assessing the speed of a p300-based brain-computer interface. *Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Neural Systems and Rehabilitation]*, 8(2):174–179, 2000.

[12] T. Egner, T. F. Zech, and J. H. Gruzelier. The effects of neurofeedback training on the spectral topography of the electroencephalogram. *Clinical Neurophysiology*, 115(11):2452–2460, Nov. 2004.

[13] J. Evans and A. Abarbanel. *Introduction to quantitative EEG and neurofeedback*. Academic Press San Diego, Calif, 1999.

[14] L. Farwell and E. Donchin. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalogr Clin Neurophysiol*, 70(6):510–23, 1988.

[15] D. Friedman, R. Leeb, A. Antley, M. Garau, C. Guger, C. Keinrath, A. Steed, G. Pfurtscheller, and M. Slater. Navigating virtual reality by thought: First steps. *Proceedings of the 7th Annual International Workshop on Presence*, pages 160–167, 2004.

[16] D. Friedman, M. Slater, A. Steed, R. Leeb, G. Pfurtscheller, and C. Guger. Using a brain-computer interface in highly-immersive virtual reality. *IEEE VR Workshop, Chicago*, 2004.

[17] X. Gao, D. Xu, M. Cheng, and S. Gao. A bci-based environmental controller for the motion-disabled. *Neural Systems and Rehabilitation Engineering,*

IEEE Transactions on [see also IEEE Trans. on Rehabilitation Engineering], 11(2):137–140, 2003.

[18] M. Garau, M. Slater, V. Vinayagamoorthy, A. Brogni, A. Steed, and M. A. Sasse. The impact of avatar realism and eye gaze control on perceived quality of communication in a shared immersive virtual environment. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 529–536, New York, NY, USA, 2003. ACM Press.

[19] P. Gloor. Hans Berger on the Electroencephalogram of Man. *Electroencephalography and Clinical NeurophysiologySuppl*, 28:1–36, 1969.

[20] Impromptu. http://impromptu.moso.com.au/, Sep 2007.

[21] G. D. Jacobs and R. Friedman. Eeg spectral analysis of relaxation techniques. *Applied Psychophysiology and Biofeedback*, 29(4):245–254, Dec. 2004.

[22] Y. Kitamura, Y. Yamaguchi, I. Hiroshi, F. Kishino, and M. Kawato. Things happening in the brain while humans learn to use new tools. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 417–424, 2003.

[23] D. R. Koller, M. R. Mine, and S. E. Hudson. Head-tracked orbital viewing: an interaction technique for immersive virtual environments. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 81–82, New York, NY, USA, 1996. ACM Press.

[24] A. Kübler, B. Kotchoubey, T. Hinterberger, N. Ghanayim, J. Perelmouter, M. Schauer, C. Fritsch, E. Taub, and N. Birbaumer. The thought translation device: a neurophysiological approach to communication in total motor paralysis. *Experimental Brain Research*, 124(2):223–232, 1999.

[25] E. C. Lalor, S. P. Kelly, C. Finucane, et al. Steady-state vep-based brain-computer interface control in an immersive 3d gaming environment.

EURASIP Journal on Applied Signal Processing, 2005(19):3156–3164, 2005. doi:10.1155/ASP.2005.3156.

[26] M. LEBEDEV and M. NICOLELIS. Brain-machine interfaces: past, present and future. *Trends in neurosciences(Regular edition)*, 29(9):536–546, 2006.

[27] C. Lee, C. Jang, T. Chen, J. Wetzel, Y. Shen, and T. Selker. Attention meter: a vision-based input toolkit for interaction designers. *Conference on Human Factors in Computing Systems*, pages 1007–1012, 2006.

[28] J. Lee and D. Tan. Using a low-cost electroencephalograph for task classification in HCI research. *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 81–90, 2006.

[29] S. Mason and G. Birch. A general framework for brain-computer interface design. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Rehabilitation Engineering]*, 11(1):70–85, 2003.

[30] A. Metting van Rijn, A. Peper, and C. Grimbergen. High-quality recording of bioelectric events. *Medical and Biological Engineering and Computing*, 28(5):389–397, 1990.

[31] J. Millán. Adaptive brain interfaces. *Communications of the ACM*, 46(3):74–80, 2003.

[32] D. Mills. Network Time Protocol (version 1)-specification and implementation. *DARPA Network Working Group Report RFC-1119, University of Delaware, September*, 1989.

[33] D. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. *Network*, 1992.

[34] M. Moore, P. Kennedy, E. Mynatt, and J. Mankoff. Nudge and shove: frequency thresholding for navigation in direct brain-computer interfaces. *Conference on Human Factors in Computing Systems*, pages 361–362, 2001.

[35] MSDN. http://msdn2.microsoft.com/en-us/library/ms803998.aspx, June 2007.

[36] R. Näätänen. *Attention and Brain Function*. Lawrence Erlbaum Associates, 1992.

[37] NeuroServer. http://openeeg.sourceforge.net/doc/sw/NeuroServer/, June 2007.

[38] J. V. Odom, M. Bach, C. Barber, M. Brigell, M. F. Marmor, A. P. Tormene, G. E. Holder, Vaegan, and Vaegan. Visual evoked potentials standard (2004). *Documenta Ophthalmologica*, 108(2):115–123, Mar. 2004.

[39] R. J. Peters and L. Itti. Computational mechanisms for gaze direction in interactive visual environments. In *ETRA '06: Proceedings of the 2006 symposium on Eye tracking research and applications*, pages 27–32, New York, NY, USA, 2006. ACM Press.

[40] G. Pfurtscheller, D. Flotzinger, and J. Kalcher. Brain-computer interface: a new communication device for handicapped persons. *Journal of Microcomputer Applications*, 16(3):293–299, 1993.

[41] M. Posner and S. Petersen. The Attention System of the Human Brain. *Annual Review of Neuroscience*, 13(1):25–42, 1990.

[42] J. Postel et al. User Datagram Protocol, 1980.

[43] J. Postel et al. Transmission Control Protocol, 1981.

[44] I. Russell M. Taylor, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. Vrpn: a device-independent, network-transparent vr peripheral system. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61, New York, NY, USA, 2001. ACM Press.

[45] G. Schalk, D. McFarland, T. Hinterberger, N. Birbaumer, and J. Wolpaw. Bci2000: a general-purpose brain-computer interface (bci) system. *Biomedical Engineering, IEEE Transactions on*, 51(6):1034–1043, 2004.

[46] SeeingMachines. http://www.seeingmachines.com, June 2007.

[47] T. Selker. Visual attentive interfaces. *BT Technology Journal*, 22(4):146–150, Oct. 2004.

[48] H. Serby, E. Yom-Tov, and G. Inbar. An improved p300-based brain-computer interface. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Rehabilitation Engineering]*, 13(1):89–98, 2005.

[49] J. Sheridan, G. Sood, T. Jacob, H. Gardner, and S. Barrass. SoundStudio4D-a VR Interface for Gestural Composition of Spatial Soundscapes. *Proceedings of The International Conference on Auditory Display 2004*, pages 1–7, 2004.

[50] J. Sheridan, Z. Yang, B. Swift, H. Gardner, A. Riddell, and A. James. Construction of a mind attention interface. submitted to IEEE Virtual Reality Conference, 2008.

[51] V. Surakka, M. Illi, and P. Isokoski. Gazing and frowning as a new human–computer interaction technique. *ACM Trans. Appl. Percept.*, 1(1):40–56, 2004.

[52] B. Swift, J. Sheridan, Y. Zhen, and H. J. Gardner. Mind-modulated music in the mind attention interface. In *OZCHI '07: Proceedings of the 2007 conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: design: activities, artifacts and environments*, pages 83–86, New York, NY, USA, 2007. ACM.

[53] V. Tanriverdi and R. Jacob. Interacting with Eye Movements in Virtual Environments. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 265–272, 2000.

[54] TIOBE. http://www.tiobe.com/tpci.htm, June 2007.

[55] H. Tramberend. Avocado: A Distributed Virtual Reality Framework. *Proceedings of the IEEE Virtual Reality*, page 14, 1999.

[56] T. Vaughan. Guest editorial brain-computer interface technology: a review of the second international meeting. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Rehabilitation Engineering]*, 11(2):94–109, 2003.

[57] B. M. Velichkovsky and J. P. Hansen. New technological windows into mind: there is more in eyes and brains for human-computer interaction, 1996.

[58] D. Vernon, T. Egner, N. Cooper, T. Compton, C. Neilands, A. Sheri, and J. Gruzelier. The effect of training distinct neurofeedback protocols on aspects of cognitive performance. *International Journal of Psychophysiology*, 47(1):75–85, Jan. 2003.

[59] R. Vertegaal. Designing attentive interfaces. *Proceedings of the symposium on Eye tracking research and applications*, pages 23–30, 2002.

[60] R. Vertegaal. Attentive user interfaces. *Communications of the ACM*, 46(3):31–33, 2003.

[61] VRJuggler. http://www.vrjuggler.org/, June 2007.

[62] VRPN. http://www.cs.unc.edu/Research/vrpn/, June 2007.

[63] C. Ware and R. Balakrishnan. Reaching for objects in VR displays: lag and frame rate. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(4):331–356, 1994.

[64] J. Wolpaw, N. Birbaumer, D. McFarland, G. Pfurtscheller, and T. Vaughan. Brain-computer interfaces for communication and control. *Clinical Neurophysiology*, 113(6):767–791, 2002.

[65] J. Wolpaw and D. McFarland. Multichannel EEG-based brain-computer communication. *Electroencephalogr Clin Neurophysiol*, 90(6):444–9, 1994.

[66] J. Wolpaw, D. McFarland, G. Neat, and C. Forneris. An EEG-based brain-computer interface for cursor control. *Electroencephalogr Clin Neurophysiol*, 78(3):252–9, 1991.

[67] J. Wolpaw, D. McFarland, and T. Vaughan. Brain-computer interface research at the wadsworth center. *Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Neural Systems and Rehabilitation]*, 8(2):222–226, 2000.

[68] J. R. Wolpaw and D. J. McFarland. Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans. *Proceedings of the National Academy of Sciences*, 101(51):17849–17854, 2004.

[69] H. Zimmermann. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. *Communications, IEEE Transactions on [legacy, pre-1988]*, 28(4):425–432, 1980.