Application of Distributed System in Neuroscience: A Case Study of BCI Framework

Indar Sugiarto[1], Iwan Handoyo Putro[2]
[1,2]Department of Electrical Engineering - Petra Christian University
Jl. Siwalankerto 121-131, Surabaya – 60236, Indonesia
email: [1]indi@petra.ac.id, [2]iwanhp@petra.ac.id

Abstract

A distributed system approach has been applied for development of electroencephalography based BCI (brain-computer interface) framework. In this method, the computing processes of BCI's sub systems are distributed over networked computers. The three most important tasks of a BCI system are signals acquisition, feature extraction and classification, and command translation or mapping. Distributing those three highly CPU's resources consuming processes in a distributed system will reduce computing complexity of BCI framework thus increasing reliability of overall system performance. In this paper, the BCI framework is developed using C++ programming language which incorporate CORBA (Common Object Request Broker Architecture) module as the communication protocol manager among BCI's sub systems. After writing the BCI framework programs, we measure the programs' performance through standard object oriented analysis metrics. The result shows that the average complexity of the distributed system architecture sub module is only 2.8 where the maximum complexity of overall architecture is measured as 6.0. The proposed method can then be extended to incorporate the more human-readable visualization strategy such as 3D contour representation of brain activities.

Keywords: Neuroscience, distributed system, brain-computer interface, object oriented design

1. Introduction

Distributed computing deals with all forms of computing, information access, and information exchange across multiple processing platforms connected by computer networks. One obvious application of this method is in the area of complex and concurrent computing, where a certain task must be accomplished using limited performance computer system. In the field of neuroscience, there is a quite new research interest which requires several computing processes to be performed simultaneously and is called brain-computer interface (or BCI in short). BCI system, as its name implies, tries to create a direct bridge between human CNS (central nervous system) with a computer or machine through neurophysiologic signals generated by the brain. It then creates a new pathway for the brain to carry its message [1]. At some extent, it is better to describe a BCI system as a hybrid system in which all of its components collaborate together to form an integrated system, as written in IEEE Signal Processing Magazine Volume 25 Number 1, January 2008: "A brain computer interface is a system that includes a means for measuring neural signals from the brain, a method/algorithm for decoding these signals and a methodology for mapping this decoding to a behavior or action" [2]. This description of BCI system sounds more technical than the one described in [3] since it emphasizes the importance of measurement-decoding-mapping interaction within the system. Those three core components of any BCI system require specific amount of processing resource in order to work as ideal as possible as in "dream" BCI [4]. The following

diagram shows how those three components work to construct a BCI system [3].

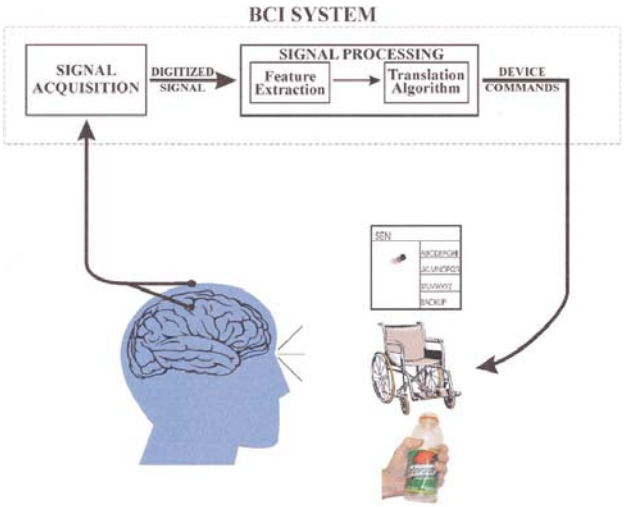

Figure 1. BCI system in general. Here the measurement task is performed by the Signal Acquisition sub-system, decoding and mapping tasks are performed by the Signal Processing sub-system.

Accomplishing those three tasks in only one of today's general computer is unlikely to be realizable. Hence, distributing those three tasks on a well-defined distributed system sounds more logical. The following diagram shows a common system setup for brain-computer interface experiment in medical application [5].

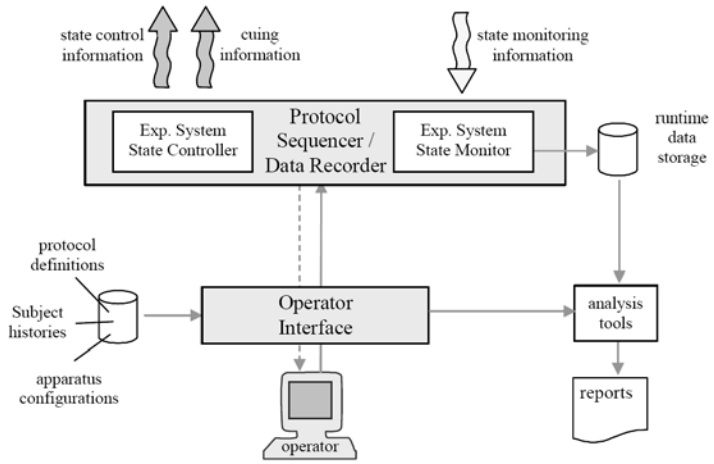

Figure 2. System setup for common brain monitoring experiment in medical application.

This paper aims to show how this mechanism could be well performed using general purpose computers arranged in a network. This paper is presented as follows. After giving an introduction, which explains the motivation of the research, the theoretical background about distributed system is explained. The detail of the proposed distributed design will be covered in section 3. Discussion about the implementation issue and its experiment result will be given in section 4. This paper will be closed with discussion and conclusion in section 5.

2. Distributed System Using CORBA

A distributed system is a collection of independent computers that appears to its users as a single coherent system [6], [7]. Furthermore, distributed system deals with collection of difference hardware and software systems which consist of more than one process or programs that running under a tight controlled rule. In distributed system, an application is split up into parts and distributed to clients who will run it locally. Unlike parallel processing, distributed application have to face the heterogeneous environments, network links of varying latencies, and unpredictable failures in the network or the computers [8].

There are many different types of distributed computing systems, such as clusters, grids, peer 2 peer [6], [9]. Each of them has their own advantage and disadvantage; however it all has the same purpose to connect users and resources in a transparent, open, and scalable way. In reality, various types of distributed systems and applications have been developed and are being used extensively in the real world. Those systems and applications are used in the area of cracking security, particle physics, weather, climate and geography, economics and finance, graphic and visualization, and bioinformatics [7].

The Common Object Request Broker Architecture (CORBA) is an emerging open distributed object computing framework that is standardized by the Object Management Group [10]. CORBA help the developer to automate some of the common network programming tasks in distributed computing. Those include object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; as well as operation dispatching [10].

The Common Object Request Broker Architecture (CORBA) is structured to allow integration of a wide variety of object systems. Figure 3 shows all of the components in the CORBA ORB architecture.

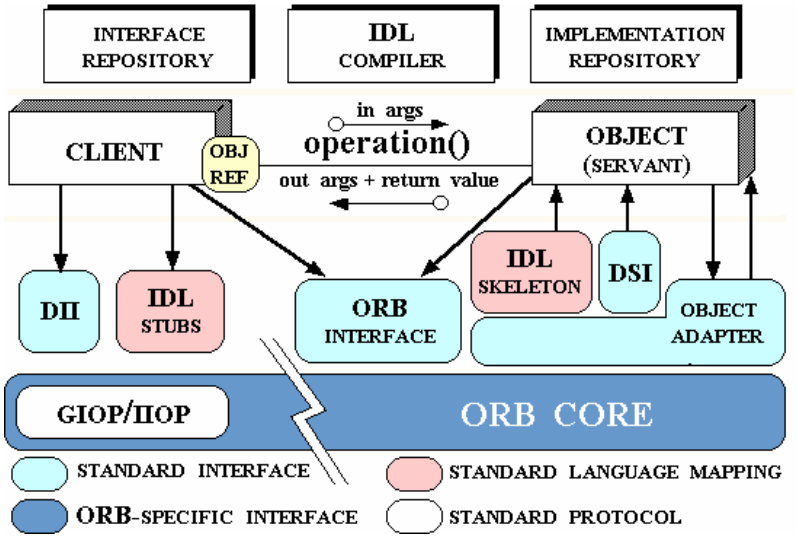

Figure 3. CORBA ORB Architecture [10]

Object is a CORBA programming entity that consists of an identity, an interface, and an implementation [10]. That programming entity, Servant, defines the operations which support a CORBA IDL interface and can be written in a variety of languages, including C, C++ and Java. Other program entity, Client, has a responsibility to invoke an operation on an object implementation. In this scenario, accessing the services of a remote object should be transparent to the caller [10], [11].

Functionally, The ORB provides a mechanism for transparent communications request between client and targeted object implementations. By decoupling the client from the details of the method invocations, distributed programming is simplified as that client requests appear to be local procedure calls [10]. In detail, the ORB would find the related object implementation when clients invoke a certain task. Furthermore, the ORB should activate that object, deliver the request to it and pass any response to the correct destination.

In the CORBA architecture, CORBA IDL stub and skeletons has an important role. It binds the client, server and ORB itself and also has a responsibility to automate the transformation between CORBA IDL definitions and the target programming language. The inconsistency problem that would be arisen between client stubs and server skeletons would be reduced by the implementation of compiler.

3. Distributed Design of BCI Framework

In a standard EEG-based medical experiment, there are at least three components of data abstraction which work simultaneously: data acquisition, signals database, and signals visualization. But in an applicable EEG-based BCI system, there are at least seven major components which are required to be synchronized: data acquisition, signals database/storage, feature processing (extraction and classification), visualization (temporal or spatial), command generation for actuator(s), command database, and feedback acquisition. The following diagram shows high level data abstraction used in our BCI Framework.

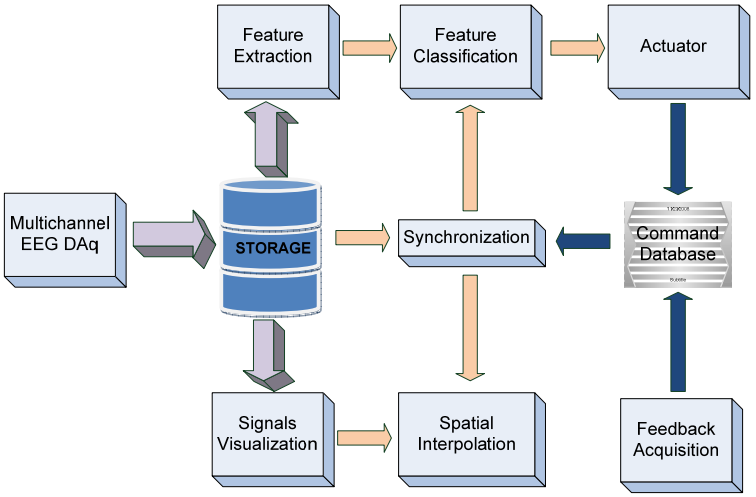

Figure 4. High level data abstraction of the BCI framework.

The entire system works as follows. After EEG signals are sampled and acquired, these raw data are stored in the hard drive, either in standard ASCII text file or binary file. The same data are fed to two other sub-systems: signal processing (for feature extraction and classification) and visualization. In the signal processing part, the features being extracted are highly depended on the mental task given by the experiment protocol to the subject. In the visualization part, the EEG signals can be displayed spontaneously (in time domain) or being interpolated for spatial visualization. After the features are extracted, certain translation algorithm is required to decode the feature into executable commands for actuator. In order to record the activity of the actuator, a command database program is required to keep track and store the commands. After a certain mental task has been given, the subject brain will react in a certain manner which will produce specific rhythmic signals. These feedback signals are

then being recorded and/or processed for further analysis. The following sequence diagram shows the early part of the work of BCI framework.

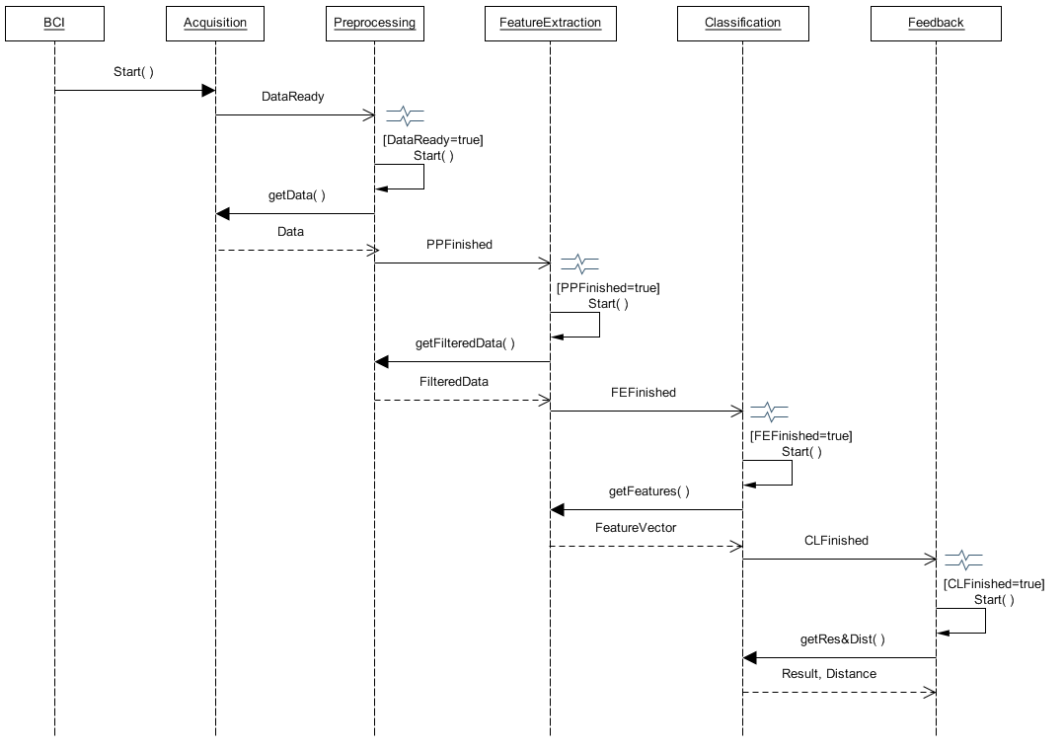

Figure 5. A part of sequence diagram in a BCI framework.

In order to synchronize all works in the framework, we develop a synchronization protocol which generate timing signals and request/acknowledge signals for data interchange between sub-systems. This synchronization process is handled by object management system using CORBA. The program was developed using ACE/TAO CORBA library.

4. Result and Discussion

The BCI framework is developed mainly using C++ language programming which incorporate several libraries from open source references. The following diagram shows the UML (Unified Modeling Language) representation of a networking part of the distributed system implementation of the BCI framework (note that not all part of the BCI framework is displayed here).

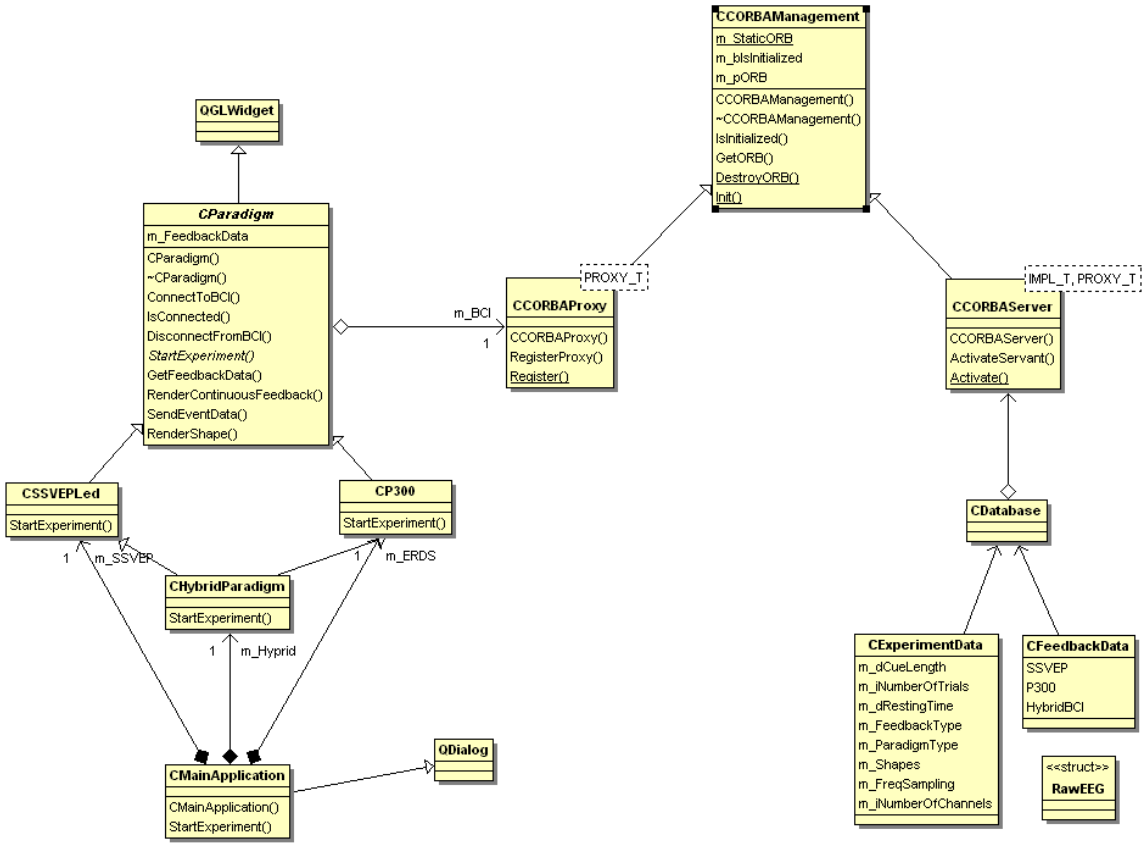

Figure 6. Representation of distributed design implementation in UML.

There are several methods that can be used to test software in order to maintain such level of software quality. At some points, software testing can be considered as part of Software Quality Control. After finishing the design process (also completing verification procedure), the program should undergo validation process through several testing procedures, statically or dynamically. Static testing is a form of software testing where the software's code is inspected without actually running the software. Dynamic testing, on the other hand, will execute the software and inspect the behavior of the software during run-time. In this paper, static code analysis is performed by measuring several metrics in order to collect information about software complexity and vulnerability. Not all of metrics for C++ will be used in this paper; only the following metrics are used: Lines of Code (LOC), Methods per Class, Complexity, and Block Depth. We use Source Monitor program developed by Campwood Software to asses our code. Figure 7 shows the metric measurement result for the distribution system part of the BCI framework program.

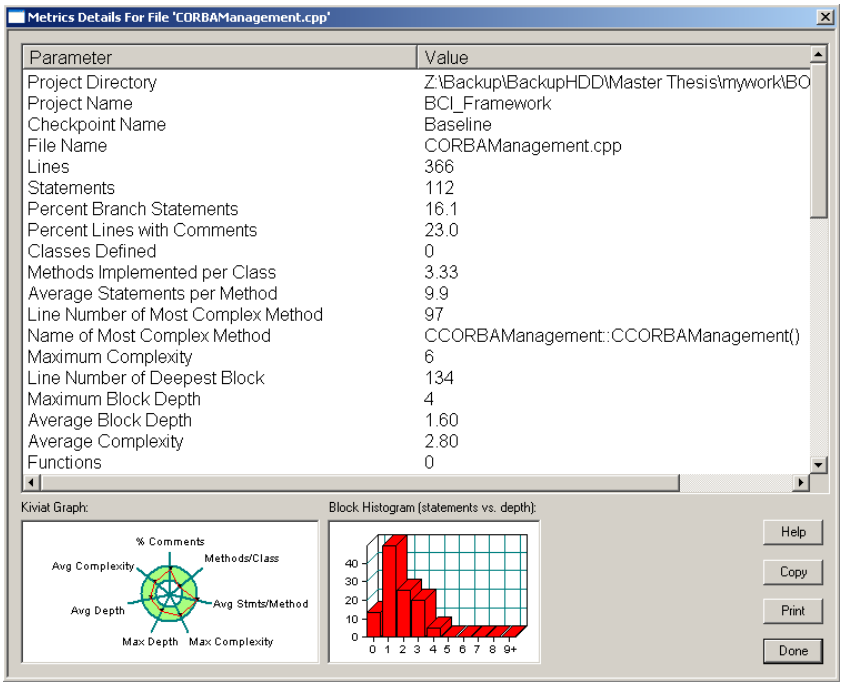

Figure 7. Metric measurement result for the distributed system part of the program.

We optimized the code so that all important aspects are confined within the acceptable and reasonable range. For example, we define the maximum of complexity as 8.0 according to cyclomatic complexity measurement based on McCabe algorithm. This complexity metric is a kind of directed acyclic graph which represents the flow of control within each function as a measure of the minimum number of test cases to ensure all parts of each function are exercised. We can use this metric as a measure for the detection of code which is likely to be error-prone and/or difficult to maintain. When measuring software complexity, a significant complexity measure increase during testing may be the sign of a brittle or high-risk module and it is a very strong component of the Maintainability Index measurement of maintainability [12]. Another important metric is the block depth, which measures how many nested blocks are exists within one upper level block. Maximum Block Depth is calculated from the top level of the source code, but namespaces are not included in this block depth metrics. Average Block Depth is the average nested block depth weighted by depth. Figure 8 shows Kiviat Metrics Graph for the corresponding metrics shown in figure 7.

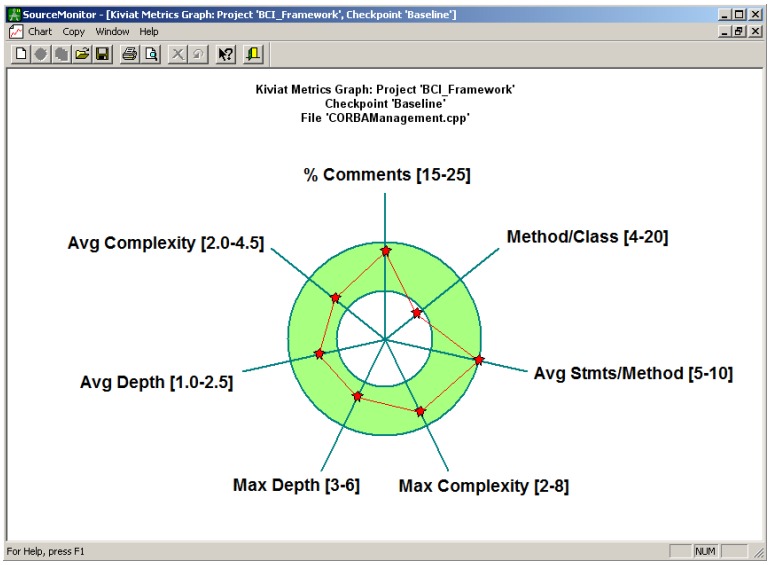

Figure 8. Kiviat Metrics Graph for distributed system part of the program.

We also measure how much network bandwidth is utilized by the system. We implement the system in a Local Area Network (LAN) using standard Ethernet adapter for each computer in the network. About 25% of available bandwidth is utilized when all programs in the framework run. Hence, the system doesn't need network management enhancement to increase the throughput of the overall network. Figure 9 shows the network utilization graph captured when the BCI framework is in active state.

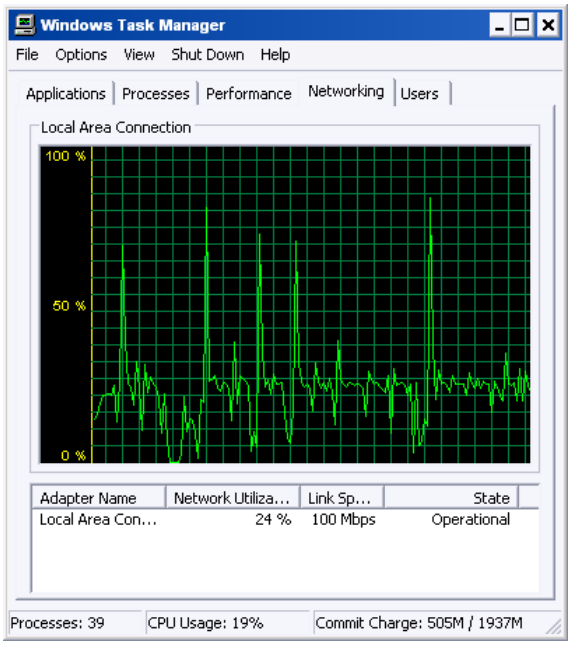

Figure 9. Network utilization graph showing BCI framework activity in a distributed system.

5. Conclusion

A distributed system approach has been applied for development of electroencephalography based BCI (brain-computer interface) framework. Using a distributed

system mechanism, we can allocate more computing power to reduce system complexity, thus improving system's reliability. The BCI framework is developed using C++ programming language which incorporate CORBA (Common Object Request Broker Architecture) module as the communication protocol manager among BCI's sub systems. Using static code analysis, the experiment shows that the average complexity of the distributed system architecture sub module is only 2.8 where the maximum complexity of overall architecture is measured as 6.0. Network utilization is only 25% and it is not a critical value. The proposed method can then be extended to incorporate the more human-readable visualization strategy such as 3D contour representation of brain activities.

References

[1] Jonathan R. Wolpaw, "Brain-Computer Interfaces as New Brain Output Pathways", *Journal of Physiology*, January 25, 2007.

[2] Indar Sugiarto, "Display and Feedback Approaches for BCI Systems", *Master Thesis*, University of Bremen, 2008.

[3] Jonathan R. Wolpaw, Niels Birbaumer, Dennis J. McFarland, Gert Pfurtscheller, Theresa M. Vaughan. "Brain-Computer Interface for Communication and Control," *Clinical Neurophysiology* 113, 2002: 770.

[4] Julies Kronegg, "Goals, Problems and Solutions in Brain-Computer Interfaces: a Tutorial," BCI-Info, 2005, www.bci-info.tugraz.at/Research_Info/documents/presentations/files/kronegg-2005__tutorial_bci.ppt (accessed April 2008).

[5] S. G. Mason, M. M. Moore Jackson, and G. E. Birch, "A General Framework for Characterizing Studies of Brain Interface Technology", *Annals of Biomedical Engineering*, Vol.33, No.11, pp.1653-1670, November 2005.

[6] Nadiminti, K., Assuncao, M., Buyya, R., "Distributed Systems and Recent Innovations: Challenges and Benefits", http://www.gridbus.org/~raj/papers/InfoNet-Article06.pdf (accessed Nopember 28, 2008).

[7] A. Tanenbaum and M. Van Steen, "Distributed Systems: Principles and Paradigms, Prentice Hall", Pearson Education, USA, 2002.

[8] Yuan, X., "Parallel and Distributed Systems", www.cs.fsu.edu/~xyuan/cis5930/lect1_intro.ppt (accessed Nopember 28, 2008).

[9] Martin Crane, Karl Podesta, "Distributed Systems (A very basic introduction of real world examples)", http://www.computing.dcu.ie/~kpodesta/distributed/ (accessed Nopember 1, 2008).

[10] Douglas C. Schmidt, "Overview of CORBA", http://www.cs.wustl.edu/~schmidt/corba-overview.html (accessed December 12, 2008).

[11] OMG, "Common Object Request Broker Architecture: Core Specification", http://www.omg.org/docs/formal/04-03-12.pdf (accessed December 8, 2008).

[12] VanDoren, Edmond. Halstead Complexity Measure. January 10, 1997. http://www.sei.cmu.edu/str/descriptions/halstead.html (accessed May 8, 2008).