Model Development, Testing and Experimentation in a CyberWorkstation for Brain-Machine Interface Research

Prapaporn Rattanatamrong[1], Andréa Matsunaga[1], Pooja Raiturkar[1], Diego Mesa[1], Ming Zhao[2], Babak Mahmoudi[1], Jack DiGiovanna[3], Jose Principe[1], Renato Figueiredo[1], Justin Sanchez[1], and José Fortes[1]

Abstract— **The CyberWorkstation (CW) is an advanced cyber-infrastructure for Brain-Machine Interface (BMI) research. It allows the development, configuration and execution of BMI computational models using high-performance computing resources. The CW's concept is implemented using a software structure in which an "experiment engine" is used to coordinate all software modules needed to capture, communicate and process brain signals and motor-control commands. A generic BMI-model template, which specifies a common interface to the CW's experiment engine, and a common communication protocol enable easy addition, removal or replacement of models without disrupting system operation. This paper reviews the essential components of the CW and shows how templates can facilitate the processes of BMI model development, testing and incorporation into the CW. It also discusses the ongoing work towards making this process infrastructure independent.**

I. Introduction

Brain-machine Interfaces (BMIs) are a promising technology for restoring communication and control to those with diseases or dysfunction of the nervous system. The objective of BMI research is to understand the mapping from a brain's neural activity to behavior in order to produce commands to control artificial limbs. To discover unknown aspects of systems-based neural encoding and decoding for complex tasks, computationally challenging real-time modeling is needed to understand the interactions between multiple brain subsystems, learning and behaviors [1].

Toward this goal, we created an experimental test bed, called the BMI CyberWorkstation (CW), distributed across two research laboratories in the University of Florida (UF) campus to provide the necessary resources, create parallel execution environments and guarantee the real-time response needed for low latency sensorimotor control. In our previous work [1,2], we have demonstrated how BMI control schemes (Recursive Least Square and Reinforcement Learning based BMI) can be implemented and tested in online and offline closed-loop experiments on the CW. Significant speed improvement in experiment execution has been observed when compared to the performance that can be achieved in a typical neurophysiology lab setting.

This work is supported in part by the National Science Foundation under Grant No. CNS-0540304, CNS-0821622 and the DARPA project N66001-10-C-2008. The authors also acknowledge the support of the BellSouth Foundation.

[1] University of Florida, Gainesville, Florida, USA, 32610.
[2] Florida International University, USA.
[3] ETHZ, Zurich, Switzerland.

In this paper, we present the CW's software techniques that enable its versatility in supporting BMIs using single or combinations of models (e.g. mixture of experts). Improvements of the model development framework are also discussed. This framework allows rapid creation, evaluation and integration of new BMI models, and in addition facilitates the maintenance of the CW.

This paper is organized as follows. Challenges in building infrastructure to support closed-loop BMI experiments are discussed in Section II. Section III overviews the CW architecture. The BMI-model template and the CW's experiment engine are described in Section IV. Section V presents improvements of the BMI model development framework. Section VI concludes on the importance of a flexible software development framework for BMI research.

II. Challenges of Remote Closed-Loop BMI Experiments

The working of a typical closed-loop BMI can be divided into three phases that occur periodically: data acquisition, data processing and prosthetic control.

The data acquisition senses *in-vivo* brain signals; a feature commonly used to quantify these signals is single-unit action potentials or "spikes" [3]. Analog-to-digital conversion and online digital signal processing (DSP) are required to detect spikes, and a sorting algorithm categorizes spikes according to the individual cells that produce them. The management of this phase is non-trivial because of the drastically varying complexity of real-time DSP computation, which depends on the type of neural data and neuro-scientific research purpose [4].

Next, the sorted spike train must be decoded using a model that translates patterns of neural data into the appropriate motor-control commands in the data processing phase. For complex neurological tasks, such as arm and hand movements in 3-dimensional space, an ensemble of concurrent movement models may be used instead of a single complex model [5]. Efficient parallel model execution in the cyber-infrastructure then becomes mandatory.

Neural decoding output may be used directly as control commands for a prosthetic device or serve as high-level (abstract) instructions for a set of low-level controllers. Signals that capture the behavior of the prosthetic device (e.g. its trajectory end position), and resulting environment changes (e.g. object displacement) are provided as feedback to the live subject and brain models. This enables the subject to decide on future actions and the models to learn or adapt online, completing the closed-loop BMI operation.

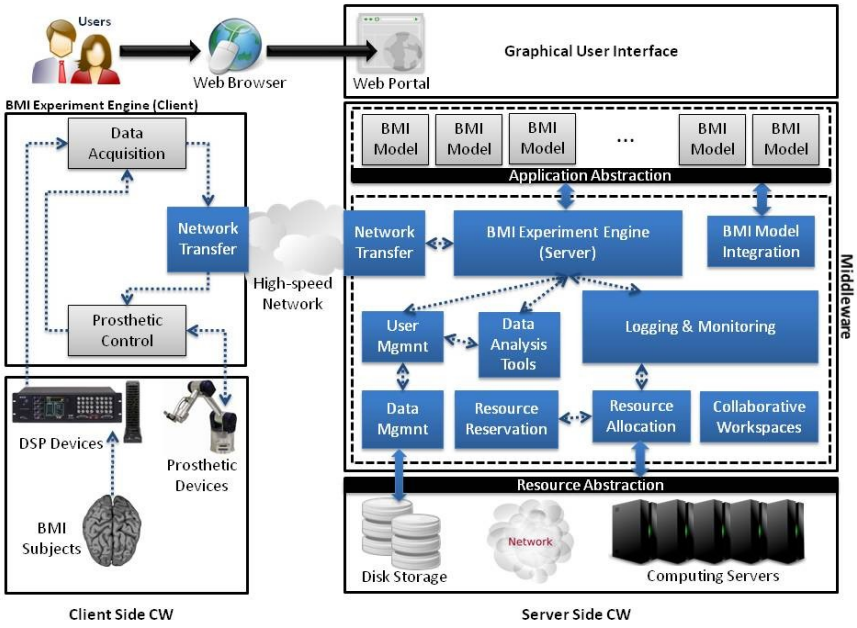

Fig. 1. The conceptual architecture of the BMI CyberWorkstation

Overall, the time taken by each BMI cycle includes the time taken by the above-described phases plus the time needed for communication among the tasks. In the general case, the locations of the data acquisition, data processing and prosthetic device can be distinct; potentially introducing significant communication delays that can violate real-time requirements of ongoing experiments when efficient and reliable network communication mechanisms are absent.

The CW has been successfully used to conduct online and offline closed-loop BMI experiments that include *in vivo* data acquisition (in online experiments), reliable network communication with error checking and buffering mechanisms, parallel computation of models, and real-time robot control. Details of the CW mechanisms and case studies can be found in [1] and [2].

III. Architecture of the CW

Fig. 1 summarizes the CW's architecture. The client-side CW, usually situated in a Neurophysiology laboratory, is responsible for data acquisition and prosthetic control. Through a high-speed network, the brain-activity data and necessary sensory feedback collected in the client side are transferred to the server side. The server-side CW, hosted in a Computing laboratory, processes the received data and returns the results back to the client-side CW for prosthetic control.

The client-side CW includes BMI subjects, instrumental resources (e.g. implant electrodes, sensors, DSP devices, etc.) and a client program of the CW's experiment engine. The client program provides coordination between the CW and BMI models during data acquisition and prosthetic control. The server-side CW consists of computational resources (e.g. computing units and data storage), middleware and graphical user interfaces. A portlet-based portal provides CW functionality through easy-to-use interfaces while hiding the complexity of the middleware.

The middleware has an application layer and a service layer. The application layer contains all supported BMI models—abstracted from the middleware's service layer by a model template described in Section IV.A. The service layer provides the following modules.

- *User Management* enables user authentication and authorization.
- *Logging and Monitoring* provides statistics and information about jobs and resources for other modules.
- *Resource Reservation and Allocation* assign necessary resources to efficiently run experiments.
- *Server-side Experiment Engine* launches and manages the experiments' data processing phase.
- *Model Integration* facilitates addition of newly developed models into the CW.
- *Network Transfer* enables reliable real-time data communication between the client and server sides.
- *Data Management* organizes safe and easy-to-retrieve data storage of experiments for post-processing.
- Tools for data visualization, analysis and group collaboration enable follow-up reviews and studies of experiments and knowledge sharing among researchers.

In the following section, we focus on the components of the architecture that 1) allow various developers to conveniently contribute their models and 2) enable users to re-use BMI models in online and offline studies.

IV. The CW's Software Techniques

The CW is designed so that it can be re-used for different types of BMI experiments, eliminating the overhead of re-building the software infrastructure needed for every BMI research experiment. The key is to allow flexible and efficient reconfiguration of the CW so that different models

can be easily "plugged in". The CW offers a "plug-and-play" experiment engine and enables the generalization of models by using a BMI-model template.

A. Generic BMI Model Template

The BMI model template, as shown in Fig. 2, consists of the model structure and the parameter interface. The model structure defines necessary subroutines corresponding to the previously described phases in BMI experiments (e.g. data acquisition, data processing and robotic control) and other management routines (e.g. initialization, loading input and cleanup, etc.) with deferred implementation. The BMI Code Library makes available useful subroutines that can help developers to rapidly implement their models by reusing code. These subroutines have clearly defined input and output arguments and can be included as inline code in any BMI model implementation. For a given subroutine, BMI developers can provide their own code or reuse code from the library. The parameter interface is a set of parameters that the experiment engine needs in order to communicate with the model. The model-specific static and dynamic parameters are separated from the model code in parameter files, providing an easy way to test and modify constants/parameters and compare BMI models with different parameter settings.

The BMI template, defined in C++, allows different modeling approaches, which may require different inputs and outputs, to be implemented in BMI models but still be accessible by the experiment engine through the same subroutine calls and parameter names. The template offers easy model integration and maintenance since the model code is decomposed into a set of small, highly independent, closed subroutines, which can be called from another subroutine and can be separately compiled. Hence, this simplifies readability and ease of testing.

The CW provides a code package that includes the model template, the code library and a blank model for new model development. Developers provide the implementation in subroutines according to the template; the CW can guarantee that the models will communicate and interact correctly with other components in the system.

B. Plug-and-Play Experiment Engine

The experiment engine is a client/server program. It coordinates BMI models and middleware service modules, needed to capture, communicate and process brain signals and motor-control commands. As shown in Fig. 3, multiple BMI models, such as RLS and RLBMI [1,2], and other service modules can be simultaneously connected to the experiment engine. This approach decouples all functionalities of the CW from each other and from the experiment engine, and allows these modules to be added or removed from the system with minimal changes to existing code of the experiment engine.

The experiment engine is implemented as a set of C++ programs with message-passing library calls. When a user creates a new experiment in the CW, the middleware automatically generates an experiment-run file containing all necessary information (e.g. the experiment directory

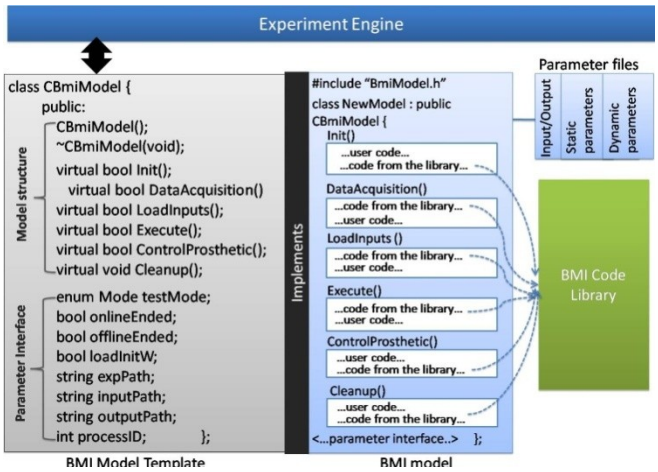

Fig. 2. Generic BMI Model Template engine

location, names of selected models, etc.) for other middleware modules and a job-submission request file. The server-side experiment engine reads the experiment-run file and submits the job request to instantiate parallel processes to execute user-selected models in the CW's computing cluster. Then both the client-side and server-side experiment engine set up necessary socket connections through the *Network Transfer* module. During the experiment, the engine interacts with BMI models using the model interface and structure (as described in the previous section), uses the *Data Management* module to record outputs, and provides a graphical view of output data via the *Data Visualization* module. Details of this interaction are shown in Fig. 3. An asterisk next to the step numbers denotes that the steps reiterate during a BMI experiment.

V. The BMI Model Development Framework

In early versions of the CW, several parts of the model code were coupled to the experiment engine, so it was not easy to investigate alternative communication protocols or approaches to improve the performance of the CW in supporting real-time experiments. In addition to the redesigned interface of BMI models to our CW code, as presented in the previous section, we discuss in this section further improvements of the BMI model development framework.

A. Model Implementation

While BMI models developed in C++ can deliver efficient codes necessary for peak performance in real-time experiments, it is important to include an alternative language for other developers who cannot conveniently port their code to the language that they are not familiar with. To address this need, the template and the experiment engine have been extended to be compatible with BMI models implemented in MATLAB, which is the most commonly used language by the computational BMI research community. Having direct access to this environment within the CW will enable users to rapidly move from concept to real-time experimentation.

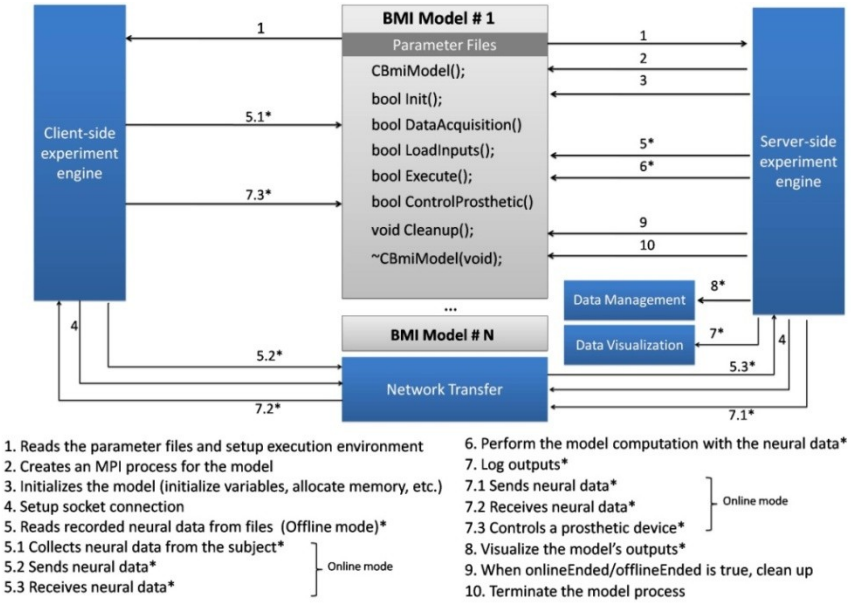

Fig. 3. The simplified structure of the CW's experiment engine

We are integrating a simple source code editor that provides a dual presentation of the BMI model code structure and the source code for a user-selected subroutine. This user interface facilitates automatic formatting of BMI-model codes (such that it follows model template specifications), and incorporates tools for developers to easily include code from the code library into their code.

B. Model Testing

The CW's administrator needs to facilitate the validation and calibration of the new BMI model before its integration into the production system. With increasing numbers of developers, this process can become a bottleneck since it generally needs iterations of code development, model configuration, execution and analysis of model output.

To reduce development time, the new CW design allows users to examine how well their model code works, using the portal interfaces without intervention from the CW administrator. The model integration module will generate an automated testing unit which clones the experiment engine with the new model plugged in. Test results become available in the user's workspace to validate correctness. Direct model implementation and testing on the CW increase model compatibility and, as a result, model integration requires less effort.

C. Model Integration and Maintenance

With the use of the experiment engine and the model template, the integration of new models and their maintenance can be seamlessly done. Successfully tested models can be submitted to the production site, along with input and configuration files needed for integration. In the future, the CW will provide a user interface that links with the model testing procedure, which automatically generates necessary files for a model submission request for users. While the integration will still require the administrator's intervention, it will be minimal.

VI. Conclusions

This paper presents three innovations in the CW system design, namely its software structure, the BMI model template and the experiment engine used in the CW's model development framework. They enable the decoupling of the logic of BMI models from the underlying infrastructure and thus expedite the deployment of new models and experiments by the system users, while minimizing CW maintenance and management by system administrators. The template allows users to easily create their model code by re-parameterization and to instantiate appropriate subroutines specified in the template. The experiment engine offers a plug-and-play capability that allows different models to be flexibly plugged in the CW. The portal interface improvements assist and partially automate model implementation, testing and integration. We believe that our software techniques are key to the efficiency of the CW since they enable models from different backgrounds and approaches to be easily combined and linked to create new and interesting possibilities in BMI research.

References

[1] J. DiGiovanna, P. Rattanatamrong, M. Zhao, et al. "CyberWorkstation Architecture for Computational Neuroscience," Frontiers in Neuroengineering, 2009.

[2] M. Zhao, P. Rattanatamrong, J. Digiovanna, et al. "BMI cyberworkstation: Enabling dynamic data-driven brain-machine interface research through cyberinfrastructure," in Engineering in Medicine and Biology Society, 2008. EMBC 2008. 30th Annual International Conference of the IEEE, 2008, pp. 646–649.

[3] N. G. Hatsopoulous and J. P. Donoghue, The Science of Neural Interface Systems, Annu Rev. Neurosci, 32, pp.249-266.

[4] M. S. Lewicki, "A review of methods for spike sorting: the detection and classification of neural action potentials." Network (Bristol, England),vol. 9, no. 4, November 1998.

[5] B. M. Yu, C. Kemere, G. Santhanam, et al. "Mixture of trajectory models for neural decoding of goal-directed movements," J Neurophysiol, vol. 97, no. 5, pp. 3763–3780, May 2007.

[6] Liferay portal framework. [Online]. http://www.liferay.com