Unlock: A Framework for Programming Brain-Computer Interfaces

James Percent
james@shift5.net

January 7, 2014

Contents

DRAFT

1 Overview

Our goal is to develop BCI technology to help individuals suffering from locked-in syndrome (LIS) [1] communicate. LIS is characterized by an almost complete inability to communicate. With current technology, unfortunately, we cannot achieve this goal.

We think, if more smart people are able to painlessly collaborate, then we have a better shot at accomplishing this goal. Therefore, our [simple] strategy is to focus on providing an awesome programming framework that makes it easier for researchers and engineers to build BCI-based systems. To this end, we employ a few simple ideas.

DRAFT

We encapsulate as much as possible; this is kind of technical mumbo jumbo for trying to say that we separate concerns. We'll talk more about this later, but the basic idea is that you should not need to know every corner of Unlock to create a new decoding algorithm or add a language model or add a new acquisition device or create a new application.

These could be different people with different skill sets, and we want them to be able to dive into their area and be productive quickly. One of the problems with BCI technology is that it is inherently cross-disciplinary; therefore, we want to lessen this burden as much as possible.

To support this goal, we need great documentation. The documentation needs to be encapsulated too. You should be able to get a high-level idea of how the whole system works and also locate details about your specific area of interest without arduous effort.

Finally, automate whenever possible; extra batteries included. We want researchers and engineers to be able to get going quickly. Learning about all the dependencies and taking a bunch of manual steps can be overwhelming. Especially if someone just wants to hack out a quick idea.

In summary, the goal is to give you a great platform to hack out an idea.

The rest of this document is structured as follows. First we do a quick technical introduction. After that, we discuss how to install and use Unlock, from a user perspective. Next, we cover both some of the internals from a high-level. Finally we close with a example.

Development of the Unlock framework is funded by the Adaptive Brain-Computer Interactions (ABCI) Capstone Project [4].

2 Technical Introduction

The infrastructure Unlock provides can be logically separated into two distinct layers: the application layer and the BCI layer. These layers can in turn be further decomposed.

The BCI layer is defined by the signal-processing chain [2]. The signal-processing chain includes signal acquisition and reconstruction, feature extraction and decoding, and command generation [3]. Many BCI systems also include a stimulation component, which is actually processed at the application layer; however, logically, the stimulation component is part of the BCI.

DRAFT

For example, a paradigm based on Visually Evoked Potentials (VEP), flashes a graphical sequence at the user, which is subsequently interpreted in the signal-processing chain [6]. But the actual flashing is handled at the application layer.

We'll talk in more detail about the BCI layer in Section 5.1, but for now there is one important take away. At each point in the pipeline we just discussed, the system is designed so that you can easily decorate, replace or otherwise manipulate the data flow. Each step in the pipeline is encapsulated under an interface contract.

The application layer is model-view-controller (MVC) **??** inspired. Don't worry too much if you're not familar with MVC. It's just a logical framework for structuring graphical user interfaces (GUIs).

The basic idea is that each application consists of three types of objects: models, views and controllers. The controllers accept user input (in this case BCI commands) and pass the inputs to the correct models. Models handle

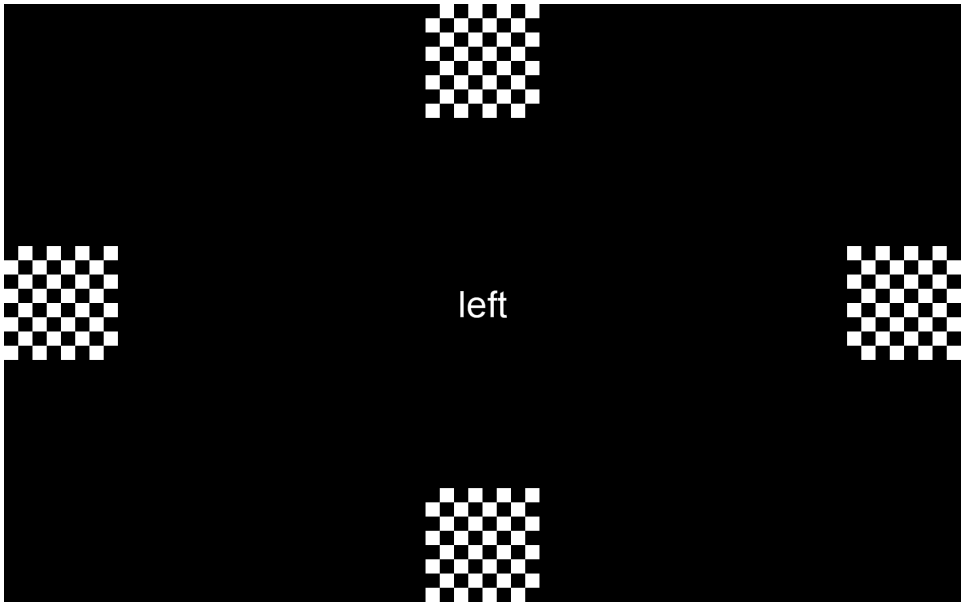

Figure 1: Data flow through Unlock

DRAFT

state transformation (the business logic if you will). Finally, views observe the state objects (models) and render themselves based on the current state of the world.

If you think about it, this also creates a pipeline in the application layer. The take away here is that these components are also encapsulated under interfaces.

Figure 1 depicts the data flow through Unlock.

3 Installation

3.1 Windoze

3.2 Linux

3.3 Mac OSX

4 Usage

5 Software

5.1 BCI Layer

DRAFT

In Section 1 we defined the signal-processing chain to be signal acquisition and reconstruction, feature extraction and decoding, and command generation.

Most of the signal acquistion code is written in C++. The C++ layer is dynamically linked into the Python runtime using Boost Python [7]. The C++ signal acquisition module exports a single polymorphic interface inspired by UNIX device files, and several factory methods for creating the signal acquisition infrastructure.

Unlock supports two modes of acquisition: blocking acquisition and nonblocking acquisition. Blocking acquisition is is simply a thin wrapper around the device interface. It will block on a I/O descriptor until a write is available.

We will discuss Unlock's application layer in detail in Section 5.2. But at this time it is important to understand at least one aspect: application layer is essentially a graphics event loop that gets called by the OS for each screen refresh. Blocking on each screen refresh can cause all kinds of problems. In fact, blocking based acquisition is only supported for device testing purposes.

To acquire the signal without blocking, Unlock supports a nonblocking ac-

quisition mode. To accomplish this, Unlock creates a platform independent C++ thread that transfers data between the two threads using a lock-free algorithm.

5.2 Application Layer

The Unlock application programming layer is a model-view-controller (MVC) [5, **?**] inspired framework for graphical user interface (GUI) development. Commands flow through the BCI layer to a control layer that interprets the state changes and sends the changes to visualization module.

5.3 Building Unlock for Development

5.4 Example

DRAFT

References

[1] Bauer, G. and Gerstenbrand, F. and Rumpl, E. Varieties of the locked-in syndrome. Journal of Neurology. 1979.

[2] Sippi, C., Sippi, P., Computer Dictionary and Handbook. 1972.

[3] He B., Gao S., Yuan H., and Wolpaw, J. Neural Engineering, Chapter 2, BrainComputer Interfaces. 2013.

[4] CELEST Capstone Project 1: Adaptive Brain-Computer Interactions. http:celest.bu.edu/about-us/capstone-projects/adaptive-brain-computer-interactions.

[5] Reenskaug, T. A note on DynaBook requirements. Xerox PARC. 1979.

[6] OShea, R. P., Roeber, U., Bach, M. Evoked potentials: Vision. Encyclopedia of Perception. 2010.

[7] Boost Python. http:www.boost.org/doc/libs/1_54_0/libs/python/doc/.

[8] Unlock. http:github.com/NeuralProsthesisLab/unlock.

[9] Gtec MOBILab. http:www.gtec.at/Products/Hardware-and-Accessories/g.MOBIlab-Specs-Features