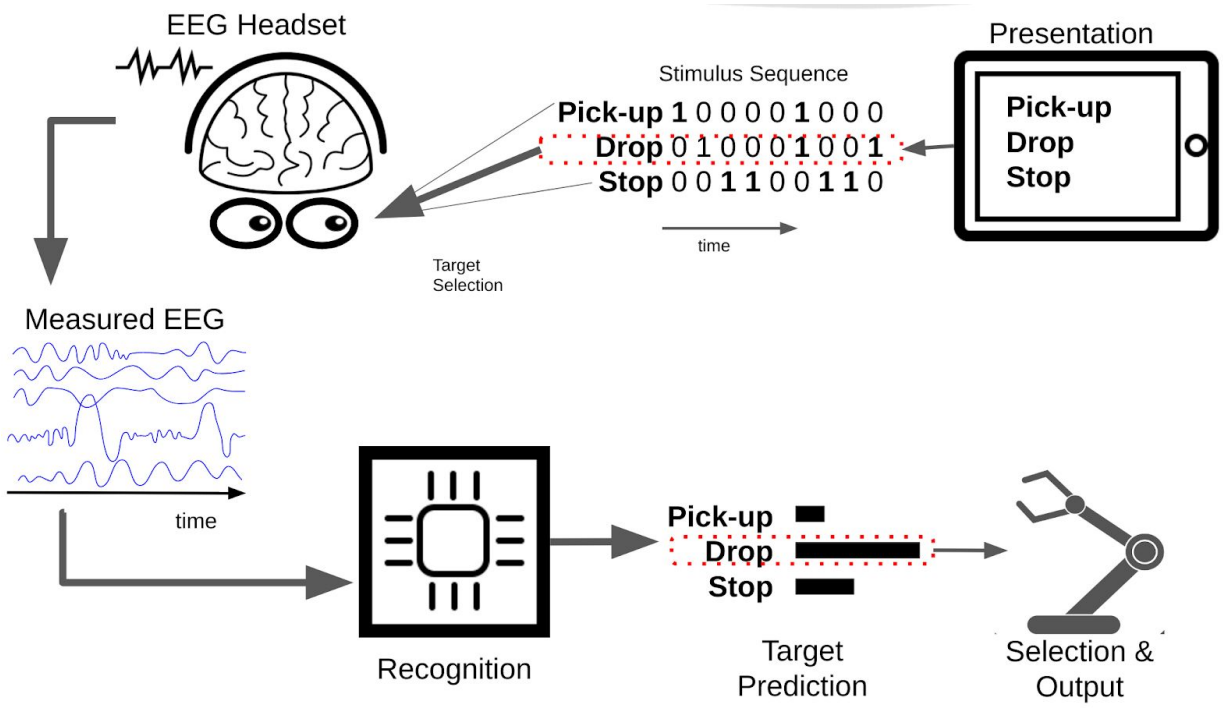
Utopia : Guide for Implementation of new Presentation and Output Components

Utopia : System Overview

Utopia is a Brain Computer Interface (BCI) system which allows users to control computers and other electronic devices by looking at flickering objects. As each object has a unique flicker sequence, by measuring the brain's response to this unique sequence Utopia can identify the target object the user is looking at and hence cause that object to be selected.

In the BCI literature this type of BCI is called a visual evoked-response BCI, as the brain response (which we use) is generated (or evoked) by the visual flickering of the target object. Below is an illustration of how a classic evoked response BCI works schematically:



To briefly describe this schematic:

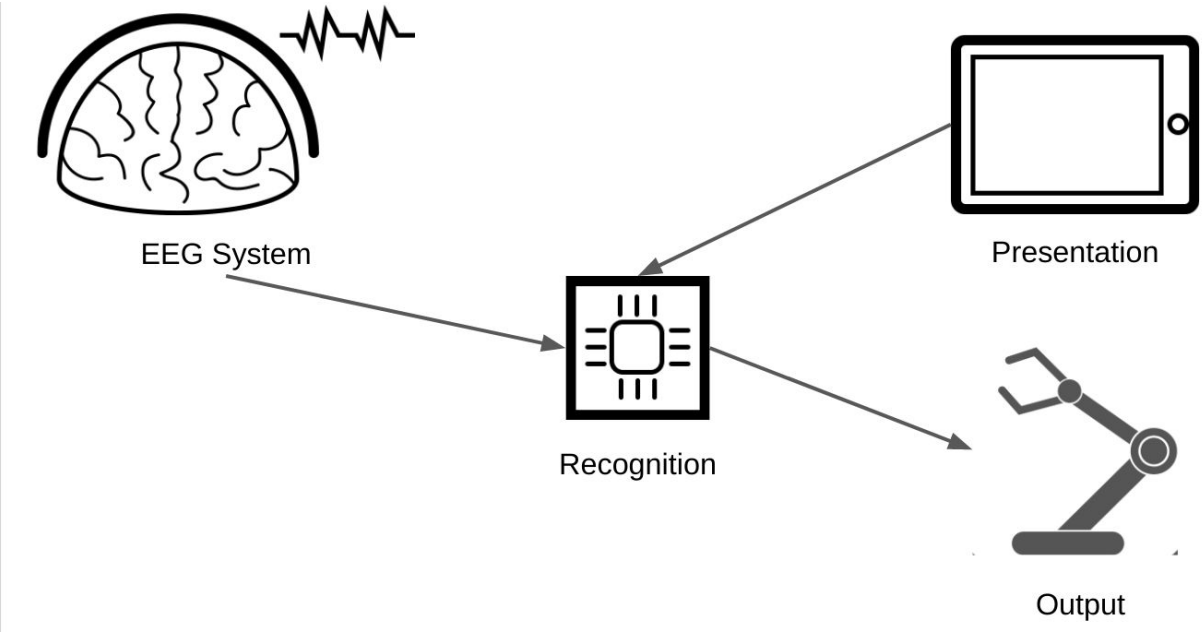
1. Presentation: displays a set of options to the user, such as pickup, drop, and stop.
2. Each of these options then flickers with a given unique flicker sequence (or stimulus-sequence) with different objects getting bright and dark at given times.
3. The user, looks at the option they want to select to select that option.
4. The users brain response to the presented stimulus is measured by EEG - due to the users focus on their target object, this EEG signal contains information on the flicker-sequence of the target object.
5. The recognition system uses the measured EEG and the known stimulus sequence to generate predictions for the probability of the different options being the users target option.
6. Finally, the selection and output system decides when the prediction is confident enough to select an option and generate the desired output -- dropping something in this case.

There are many types of (visual) evoked response BCI, all of which work in basically the same fashion. What may differ is the modality in which the stimulus is presented (such as visually, orally, or tactically) and/or the particular properties of the ‘flicker’ pattern and associated brain response.

The Utopia system uses a particular ‘flicker’ pattern based on pseudo-random-noise codes, specifically gold-codes, which cause the brain to generate a Visual Evoked Potential. In the BCI literature this type of BCI is called a code-based Visual Evoked Potential (c-VEP).

Physical components of the Utopia BCI

The main physical components of the Utopia System illustrated below:



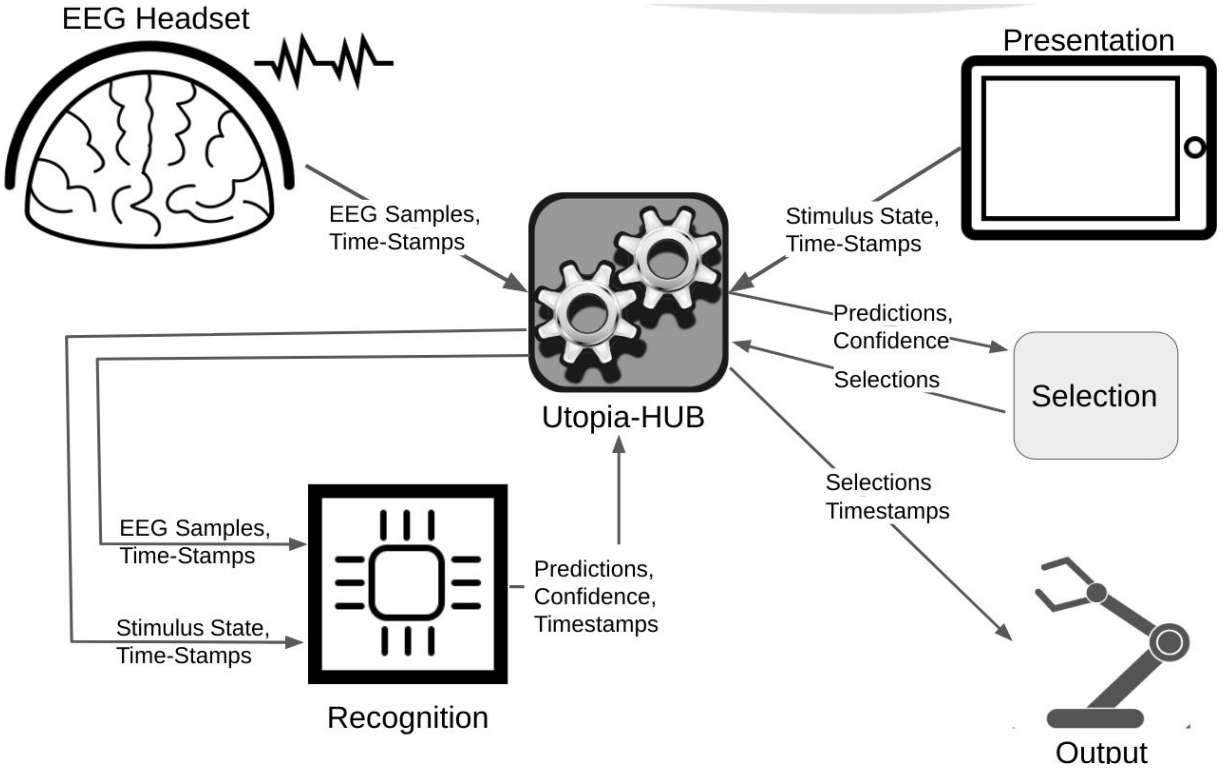
The 4 main components and their purpose are:

- **Presentation**: This component is usually a screen of some sort, e.g. LCD, or tablet-computer, but may be any system which is able to generate a rapid controlled visual change, such as an LED array or a laser-pointer. The purpose of this component is to present the selectable objects and 'flicker' them in the required sequence such that when the user looks at their *target object* (that is the object they want to select) a detectable brain response is evoked. It must also send information on the actual flicker sequence of each object to the Reogniser so it can identify the users target object.
- **EEG system:** This component usually consists of 3 sub-components: a) a headset that sits on the users head and ensures a physical connection between the measurement electrodes and the users scalp. b) an amplifier which measures the relevant brain response and converts it into digital signals, c) a transmission system (such as USB, WiFI or Bluetooth) which can send these measurements to the Recogniser so it can identify the users target object.
- **Recogniser** : This component is the physical host of the software that runs the machine learning algorithms and combines the information on the presented stimulus (from the Presentation component) and the measured brain response (from the EEG component) to **recognise** the users intended target and send these **object selections** to the output component. Usually, this component is physically a single-board computer, such as a raspberry pi, but may be a purely software component running on the CPU of another component (such as CPU of a tablet or desktop computer used for presentation).

- **Output**: In many cases this component is integrated as part of the presentation component, such as a tablet-computer, but may be a completely independent device, such as a robot or TV, or home-automation device (e.g. door opener/closer.) The purpose of this device is to take the **object selections** generated by the Recogniser and generate the desired output. For example, in a virtual-keyboard application this may be to add the selected letter to the current sentence, or in a home-automation setting this may be to change to the selected TV channel or open the indicated door.

Functional roles of the Utopia BCI

The physical components of the Utopia BCI map fairly straightforwardly onto the functional roles within the utopia system. The below diagram illustrates the main functional roles of the Utopia system and the communication messages used to communicate between them. The arrows indicate the direction of information flow between components.



The main functional roles in the utopia system are:

- **Presentation:** The presentation component is responsible for presenting the stimuli to the user, and sending information about this presentation to the **Recogniser** so it can identify the users selected target. How the presentation is actually performed is up to the presentation component. In particular the stimulus modality (visual, audio, tactile) and flicker sequence used are the responsibility of the presentation system. We advise

you to go through our examples and frameworks (see XXXX) for best practices of presenting c-VEP stimuli.
The stimulus information is sent to the recogniser using **Utopia Message-spec** messages (see the XXXX Utopia Message-spec for details) using **STIMULUSEVENT** messages. The particular transport used for the messages will depend on the hardware implementation, but currently we use TCP-IP sockets over WIFI. For ease of implementation we provide example presentation implementations with both cVEP stimulus display and TCP-IP based message sending in the following development languages/frameworks (see XXXX for details):

- SWIFT/iOS
- Unity
- Python, with pygame
- Java, with libGDX
- MATLAB/OCTAVE

- **EEG Acquisition:** The EEG acquisition system is responsible for getting digital EEG samples from the EEG hardware and transmitting them to the **Recogniser**.
  This information is transmitted to the recogniser in a number of different formats -- though mainly using the Fieldtrip Realtime Buffer protocol (see Overview of the realtime buffer - FieldTrip toolbox for more information) in the current Utopia system.
  As this transmission and protocol is internal to the Recogniser and irrelevant to developing new presentation and output components it is not discussed further here
- **Utopia-Hub**: This component is responsible for connecting together all the other components of the system. Basically, this is a broker for Utopia Message Spec messages, which accepts incoming messages and forwards them between different clients, for example forwarding all STIMULUSEVENT messages from Presentation clients to the Recogniser component and in turn forwarding all PREDICTEDTARGETPROB messages from the Recogniser to all output or selection clients.
  The Utopia-Hub is a purely software component running on the same platform as the Recogniser. The particular transport used for the messages depends on hardware, but the current system uses TCP-IP sockets over WIFI or Ethernet. In particular the Utopia-Hub allows incoming and outgoing client connections on **TCP port 8400** and **incoming only** connections on **UDP port 8400**.
  The utopia-hub also includes additional functionality making it easier to use and debug the utopia-system including;
  - Sending and monitoring **HEARTBEAT** messages from and to clients. These messages are sent a regular intervals to check for liveness of the clients within the utopia-hub and to detect hub crashes within clients.
  - Providing auto-discovery services, via Simple Service Discovery (SSDP) and mDNS to allow automatic detection of the utopia-hub from clients.

  As the Utopia-Hub is an internal component to the Recogniser component, it's actual operation is not discussed further here. (Though FYI: it is a single-threaded network server implemented in JAVA).

For ease of implementation in clients we provide a full implementation of utopia-hub **CLIENTS** including auto-discovery, heartbeat monitoring, and message sending and receiving for all message types in the following platforms/frameworks (see XXX for more information):

- SWIFT / iOS
- C# / Unity
- Python
- Java
- MATLAB/OCTAVE

- **Recogniser:** This component is responsible for using information on which flicker sequences have been presented to the user (received as **STIMULUSEVENT** messages from the presentation component(s)) and what EEG measurements have been made (received from the EEG Acquisition component) to generate **target predictions** sent as **PREDICTEDTARGETPROB** messages to the selection component. As the predictions of the recogniser are uncertain, the target predictions consist of a:
  - Predicted target object - which is the object the recogniser thinks is most likely to be the users intended target object.
  - Predicted target error - which is the recognisers estimate of the chance that it's identification of the target object is **incorrect**.

  Internally, it does this by using **machine learning** techniques to learn a mapping from EEG to a predicted flicker sequence. This predicted flicker sequence is then compared with information on the actual flicker sequences shown to the user by the presentation component to identify the most likely target object. As the actual techniques used to learn this mapping and compare the predicted and actual flicker sequences are internal to the Recogniser component these will not be discussed further here.
- **Selection :** The selection component is responsible for taking **target predictions** as generated by the **Recogniser** and turning these into **Selections** for which the output system should generate the desired output. How the target prediction information from the recogniser and any contextual information available to the selector is used to make these selections is up to the selection system, for example in a virtual keyboard application the context within a word may be used to make selection of the most likely next letter easier. Note: in many cases, the selection system integrated with that of the output system as this has the contextual information available to improve selections -- such as knowing the word typed so far in a virtual keyboard.

  In most cases the selection system is simply one of thresholding on the Predicted target error of the target predictions given by the **Recogniser**.

  Information on target predictions is transmitted to the output component using **Utopia Message-spec PREDICTEDTARGETPROB** messages (see XXXX the Utopia Message-spec for details). The particular transport used for the messages will depend on the hardware implementation, but currently we use TCP-IP sockets over WIFI. For ease of implementation for we provide example Predicted target error thresholding selectors using TCP-IP based message receiving in the following development languages/frameworks (see XXX for details):

    - SWIFT / iOS
    - C# / Unity
    - Python
    - Java
    - MATLAB/OCTAVE
- **Output**: The output system is responsible for transforming selections made by the user (received as **SELECTION** messages) into their desired output. What this output is and how it is delivered is the responsibility of the output system.
  Note: in many cases, such as with a virtual-keyboard, the output role is integrated with the presentation role.
  Information on user selections is transmitted to the output component using **Utopia Message-spec SELECTION** messages (see the Utopia Message-spec for details XXX). The particular transport used for the messages will depend on the hardware implementation, but currently we use TCP-IP sockets over WIFI. For ease of implementation we provide example output using TCP-IP based message receiving the following development languages/frameworks (see XXX for details):
    - SWIFT / iOS
    - C# / Unity
    - Python
    - Java
    - MATLAB/OCTAVE

System Mode Switching: Calibration and Prediction

The core of the Utopia system is the machine learning system in the recogniser which transforms EEG + StimulusEvents into target predictions. For this transformation to work the recogniser needs so-called **calibration** data from which the ML algorithms are **trained**. To get this calibration data the system works in one of two main modes, Calibration and Prediction.

- **Calibration Mode:** In this mode the system is gathering data with which to train the machine learning module. This means that in this mode the Selection and Output roles are not used. Further in this mode the Presentation module must somehow **instruct the user** which target they should attend to.
- **Prediction Mode:** In this mode the machine learning module has been trained (model fitting has been completed) and the Utopia system can be used to make output selections based on brain responses. This means that in this mode all the components, Recoginser, Presentation, Selection and Output work as described above.

Switching between these two modes is achieved by sending MODECHANGE messages to the utopia-hub (which in turn forwards these messages to all other components.)

Overview of steps required for implementing Utopia Roles in PREDICTION mode

Implementing the OUTPUT role in PREDICTION mode

As mentioned above, the output role consists of listening for Selection messages and performing the desired output when one is detected. Thus it is perhaps the simplest role to implement in the utopia system.
There may be many output components responsible for generating specific outputs when particular selections happen. For example, one output component responsible for controlling the TV when TV control objects are selected, while another is responsible for controlling the room temperature, with another for controlling a robot arm. To allow these multiple output modules to operate at the same time, the **objectID** of different objects are used to detect when a particular output should trigger. Thus it is important that *the objectIDs for different outputs be unique and output modules only generate output for the objectIDs they are responsible for.*
The basic steps for implementing Output are:

1. **Implement** the Utopia Message Spec for **Receiving** ALL messages[1], and as a minimum **PARSING and SENDING** SELECTION and PREDICTEDTARGETPROB messages. Alternatively, if available for your platform use the provided framework to get the full utopia-client functionality.
2. **Connect** to the Utopia-HUB server. If auto-discovery is implemented in your platform, you may use auto-discovery to get the location of the Utopia-Hub. If not then this may have to be manually configured / requested from the user.
3. (Optionally) Send a SUBSCRIBE message to only receive the SELECTION messages necessary for an OUTPUT module (this can be beneficial to reduce message bandwidth).
4. **Listen** for SELECTION messages, and parse them.
5. Given the content of the SELECTION message **check** if the selected **objectID** is one which this output component should trigger output for. If this is not the case then ignore this selection message.
6. **Generate** the desired output.

Implementing the SELECTION role in PREDICTION mode

As mentioned above, the SELECTION role consists of listening for PredictedTargetProb messages and identifying when the confidence in the target is sufficiently large to generate output.

---

[1] one can use the message length to skip messages you dont understand

Note: one possible ‘complexity’ of implementing this role is that due to network delays both the Presentation and Recogniser component may continue to generate Prediction messages for the previous ‘trial’ after a target has been selected. Thus, it is common to use a ‘debounce’ technique where one makes a selection once the target confidence is sufficiently high, but then inhibit further selection events until the confidence drops sufficiently low to indicate a new trial has begun.

The basic steps for implementing Selection are:

7. **Implement** the Utopia Message Spec for **Receiving** ALL messages[2], and as a minimum **PARSING and SENDING** SELECTION and PREDICTEDTARGETPROB messages. Alternatively, if available for your platform use the provided framework to get the full utopia-client functionality.
8. **Connect** to the Utopia-HUB server. If auto-discovery is implemented in your platform, you may use auto-discovery to get the location of the Utopia-Hub. If not then this may have to be manually configured / requested from the user.
9. (Optionally) Send a SUBSCRIBE message to only receive the PREDICTEDTARGETPROB messages necessary for a SELECTION module (this can be beneficial to reduce message bandwidth).
10. **Listen** for PREDICTEDTARGETPROB messages, and parse them.
11. Given the content of the PREDICTEDTARGETPROB message **decide** if the target confidence is sufficiently high that a prediction should be generated.
12. **Send** a SELECTION message indicating that the given object has been selected for output.

Implementing the PRESENTATION role in PREDICTION mode

As mentioned above, the Presentation role consists of presenting visual flicker stimuli to the user and sending STIMULUEVENT messages to the utopia-hub to describe what stimulus has happened.

Important Note: Stimulus Timing Accuracy

As the connection between **when** a flicker happens and **when** the users brain responds to this flicker is critical for the performance of the BCI. It is **very important** that accurate timing information is provided for when the visual stimulus changes. In general, this means that the Presentation system needs the ability to know when the visual stimulus has changed to **at least** 4 milliseconds accuracy, ideally higher accuracy may lead to higher performance. It is also important to use a (relatively) high performance clock for recording time-stamps -- at least 1ms resolution.

In our experience for most operating systems, when using a screen for output, this level of timing accuracy is only achievable when using relatively low-level high-performance graphics

[2] one can use the message length to skip messages you dont understand

systems -- usually those used for developing games. This means directX on Windows, openGL or Vulcan on unix or Metal on iOS. In addition some form of VSYNC trigger is required for maximum timing performance.

Implementation

The basic steps for implementing Presentation are:

1. **Implement** the Utopia Message Spec for **receiving** ALL messages[3], and as a minimum **PARSING and SENDING** STIMULUSEVENT messages. Alternatively, if available for your platform use the provided framework to get the full utopia-client functionality.
2. **Connect** to the Utopia-HUB server. If auto-discovery is implemented in your platform, you may use auto-discovery to get the location of the Utopia-Hub. If not then this may have to be manually configured / requested from the user.
3. (Optionall) Send a SUBSCRIBE message to register to only receive the SELECTION messages needed for the presentation module when also giving selection feedback to the user.
4. **Load** the codebook which describes the type of flicker sequence the presentation system will use. Again, if available for your platform/language you can use our provided frameworks to load cVEP codebooks. Alternatively, we provide codebooks a simple textual format which can be transformed into your language of choice.
5. **Present** the flicker codes for the current set of selectable objects. As mentioned above, keeping track of the **time** when the visual stimulus is seen by the user is very important, so this should be recorded during stimulus presentation. Further, every selectable object requires a unique **objectID** (which is a number between 1 and 255) so the recogniser can track how these objects state changes over time and so the output module knows which object was selected. The Presentation module is responsible for choosing these objectIDs and ensuring they **do not clash** with objectIDs used by other (possibly simultaneously running) presentation modules.
6. **Send** the STIMULUSEVENT messages, consisting of the timestamp of when the visual stimulus changed and the updated visual state of the changed objects. (Note: this is allows for delta encoded stimulus information if message bandwidth is limited.)
7. (Optionally) If also giving user selection feedback. Receive SELECTION messages from the output module, parse the message and give a visual indication to the user of the selection -- e.g. by pausing the flicker stimulus and highlighting the selected object for 1 second in a different color.

[3] one can use the message length to skip messages you dont understand

Overview of steps required for implementing Utopia Roles in CALIBRATION mode

Implementing the PRESENTATION role during CALIBRATION

As mentioned above, the Presentation role consists of presenting visual flicker stimuli to the user and sending STIMULUEVENT messages to the utopia-hub to describe what stimulus has happened. In Calibration mode the additional changes are:

1. Either:
    a. Send a MODECHANGE event to tell the system to enter CALIBRATION mode.
    b. (Or) Listen for MODECHANGE event telling the presentation system to enter CALIBRATION mode.
2. The user must be instructed with which object they should attend to. It is the Presentation module's responsibility to decide how this is done. Two approaches used in the MindAffect internal Utopia system are:
    a. Highlight the current instructed target object by making it a different color for 1 second before the actual trial flicker begins.
    b. Only present a single object on the screen for the user to select (and instruct them to concentrate on it.)
3. The Recogniser must be informed which of the presented objects is the one to which the user has been attending. This is simply achieved in the normal STIMULUSEVENT messages by adding a 'virtual' objectID with value 0 which contains the stimulus information for the current target object. That is, if for example the object with objectID==4 is the current target out of 10 objects (with id's 1-10), then instead of sending STIMULUSEVENT with the state of objectIDs 1-10 we send an event with objectIDs 0-10 (i.e. 11 objects) where the value for objectID=0 is identical to that of objectID==4.
4. As the Selection system is not running, send an additional event to indicate the end of the calibration trial. In the utopia system this is the NEWTARGET event.
5. Either
    a. Send a MODECHANGE event at the end of calibration to tell the system to exit calibration mode. (For example by switching to PREDICTION or IDLE mode.)
    b. (Or) Listen for a MODECHANGE event telling the presentation that it is time to leave calibration mode.
6. (Optionally). Give the user feedback on the expected performance of the machine learning model trained during calibration. During calibration the Recogniser sends PREDICTEDTARGETPROB events which contain the confidence of the recogniser that it could have correctly predict the instructed target object by the end of each trial if it had not been told. Thus, this is an indication of the expected performance of the trained model. The Presentation system may use this indication to give feedback to the user on

the model performance, so they know what performance they can expect and/or can choose to try re-calibrating to get a better machine learning model.
In or MindAffect utopia system, this indication is provided by reporting the final PREDICTEDTARGETPROB received after the MODECHANGE event which indicates the end of the calibration phase.

Implementing the SELECTION role during CALIBRATION

During calibration SELECTION should be disabled. Thus it should:

1. Listen for MODECHANGE event telling the SELECTION system to enter CALIBRATION mode, and then disable the SELECTION system until.
2. Listen for MODECHANGE event telling the SELECTION system to enter PREDICTION mode, and then re-enable selection operation.

Implementing the OUTPUT role during CALIBRATION

During calibration OUTPUT should be disabled. Thus it should:

1. Listen for MODECHANGE event telling the OUTPUT system to enter CALIBRATION mode, and then disable the OUTPUT system until.
2. Listen for MODECHANGE event telling the OUTPUT system to enter PREDICTION mode, and then re-enable output operation.