Defining Standard Gaze Tracking API

Oleg Špakov
University of Tampere
Kanslerinrinne 1
Tampere, FIN-33014, Finland
oleg.spakov@uta.fi

Abstract

In this paper we argue of the need for a definition of API that would support the system-independent development of applications with gaze input. Our proposal for such API is based on minimal yet extendable sets of functions and data layers for data transmission from data producers to data consumers. We also hope that some kind of standard API based on our proposal will be supported by eye-tracking community defining data layers and creating system-supporting modules.

Author Keywords
Gaze tracking; API standardization; gaze data protocol.

ACM Classification Keywords
H.5.2. User interfaces: Input devices and strategies.

General Terms
Human Factors; Design.

CHI 2013 Workshop on "Gaze Interaction in the Post-WIMP World", April 27, 2013, Paris, France.

Introduction

Recent improvements in eye-tracking technology supported by increasing use of eye-tracking systems for commercial purposes (e.g., for usability services) has prepared the ground for the upcoming mass marketing. High competition between manufacturers of high-end eye-tracking systems, numerous low-cost solutions based on ordinal web-cameras and free software, and increasing interest to the domain from IT industry giants like Google, Fujitsu, Sony, Microsoft and others[1] are already starting to push the prices of commercial systems down[2] and should eventually be found build-in into mobile devices and affordable for desktop PC users

Despite this clear progress and anticipation of a wide spread of the technology there are still some issues to be solved before it becomes truly mature. One of these issues in this domain comes from the fact that each eye-tracking system requires a separate piece of code to communicate with end-user applications and handle data it provides. It means that, if nothing changes on this side, even when the technology becomes accessible and affordable for regular users, most of gaze-based applications will support a limited number of systems and require a very exact match between what is supported by a gaze-based application and what is installed/embedded on the user device. One can

[1] For example, see http://www.geekchunk.com/reviews/3764/
[2] For example, see http://phandroid.com/2013/01/07/tobii-eyesight-controlled-computer-to-be-google-glass-partner/

imagine how complex would be the software development and compatibility maintaining if end-user applications require a separate code for a mouse of each vendor simply to respond to mouse events. This issue becomes even greater when eye tracking is taken away from desktop computers into pervasive and mobile devices that have even a wider diversity in functionality than conventional desktop PCs.

As a consequence, most of the existing gaze-based applications can interact with a certain eye-tracking system only. Some applications like OGAMA[3] (open-source gaze path analysis software) consequentially increase the number of supporting systems. However, there was at least one attempt to develop some sort of middleware which serves as a unifier of the way to interact with eye-tracking systems and receive data using same protocol regardless to the system in use. ETU-Driver[4], a COM library, provides transparent access to several commercial and web camera based systems. All gaze-based applications that use this middleware may start using a new system without upgrade once a supporting ETU-Driver module for this system is developed and installed. While being used for developing many gaze-based applications (mostly, by the author and his colleagues), this solution has few restrictions. In particular, the data protocol is fixed as a COM-structure; it make impossible to access system-specific features: for example, it misses fields to accommodate eye position in camera view that is reporting by many remote eye trackers.

Some of other possible issues in the exiting solution were discussed by Daunys and Vyšniauskas in [1], where authors proposed an alternative approach by using one of the MS Windows messages (WM_COPYDATA) to communicate with eye-tracking system systems and receive data via a dedicated application named "COGAIN Brocker". Authors claim that this approach allows faster data transmission from a system to a client application. Probably, it also could allow using flexible data protocols, although the authors do not discuss this aspect of their solution.

[3] http://www.ogama.net/

[4] http://www.eyeinteract.com/show.php?app=27

Eye-Tracking Uniform Development Tool

A discussion on a search of new ways for the convenient system-independent development of gaze-based application has been initiated few years ago on the COGAIN forum[5]. Although no clear agreement was archived on how the new middleware or SDK should look like, it gave a clear hint that manufacturers of eye-tracking systems would not like to have a functionality or output from their products somehow restricted: developers should have a full access to the services a certain system supports and provides.

We suggest that definition of some basic yet extendable functionality and a layered data protocol may work best in this case and lead to support of this initiative by market players in this domain. "Layered" protocol is the one that may contain various sets of data; one layer is "predefined" and contains a minimal set of very basic data that each system must be capable to provide, and more layers can be defined by the community as a "de-facto" standard. The basic sets of functions (interfaces) could be extended to provide full native system API.

In this solution, each eye-tracking system is supported by a dedicate module that implements predefined

[5] http://www.cogain.org/forum

interfaces (like in ETU-Driver) and properly formats data according to the definitions of layers. An end-user application that interacts with the system specifies what data layers it requires. Since data flow via same channel (event), XML is supposed to be used to format data (other protocols, like JSON may be appropriate as well). Definitions of data layers should be publically maintained (e.g, placed on some public resource, such as COGAIN.org, where a discussion for new definitions can take place). We suggest naming the predefined protocol as "Basic" with the following variables: a timestamp (s), X and Y gaze coordinates relative to the screen (pixels or normalized values).

Systems available on the machine in use can be detected using so-called system browsers. Browsers also report when a system becomes (un)available. There can be several browsers installed on the same machine, and each browser can support more than one system. It is expected, that each manufacturer provides a browser that supports all its trackers. Technically, the browser interface can be implemented in the module where the system interface is implemented.

For MS Windows operating system we suggest to implement the system, browser and supplementary interfaces in COM libraries. To unify the way the browsers could be loaded from end-user applications, we suggest placing IDs of classes (CLSID) that implement the browser interface into a dedicated registry key (say, as a key name in HKCU/Software/ETUDE/Browsers, and the key value stores its name). The list of the interfaces and their IDs (IID) could also be published on a public resource, like GitHub, to allow the development of system and browser modules available to all.

To make the solution extendable, we propose also a definition of an interface for plugins. Plugins may request a certain data layer(s) and produce another data layer(s). Plugins and system modules must implement same interface designed for data producers, so that the data delivery to end-users applications could be unified. The list of plugin CLSIDs could be placed into HKCU/Software/ETUDE/Plugins.

Developers of gaze-based applications have to follow these steps when using this solution for obtaining input from eye trackers: 1) load all browsers and plugins listed in HKCU/Software/ETUDE, 2) connect to one of available eye trackers which 3) acknowledges the request for required data layer(s). Due to the restriction in space, we list in Figure 1 only the summary of the interfaces used in our solution. The source code in C# with interface definitions, implementation of two systems modules (gaze tracker emulation by mouse, and Tobii T/X series by Tobii Technologies), a browser, utilities for modules and gaze-based application developers (including a "manager" that accommodates browsers and plugins loading routines and some other helping functionality), and a proof-of-concept is available on SourceForge[6].

Conclusion
We drew one of the main technological issues for the fast introduction of eye-tracking technology into the everyday use and discussed the next steps on the way to the standardization of eye-tracking API. The suggested milestones of this API consist of 1) common minimal interfaces, 2) flexible and extendable data protocol, and 3) openness in setting up this standard.

[6] http://sourceforge.net/projects/eyetrackingsdk/

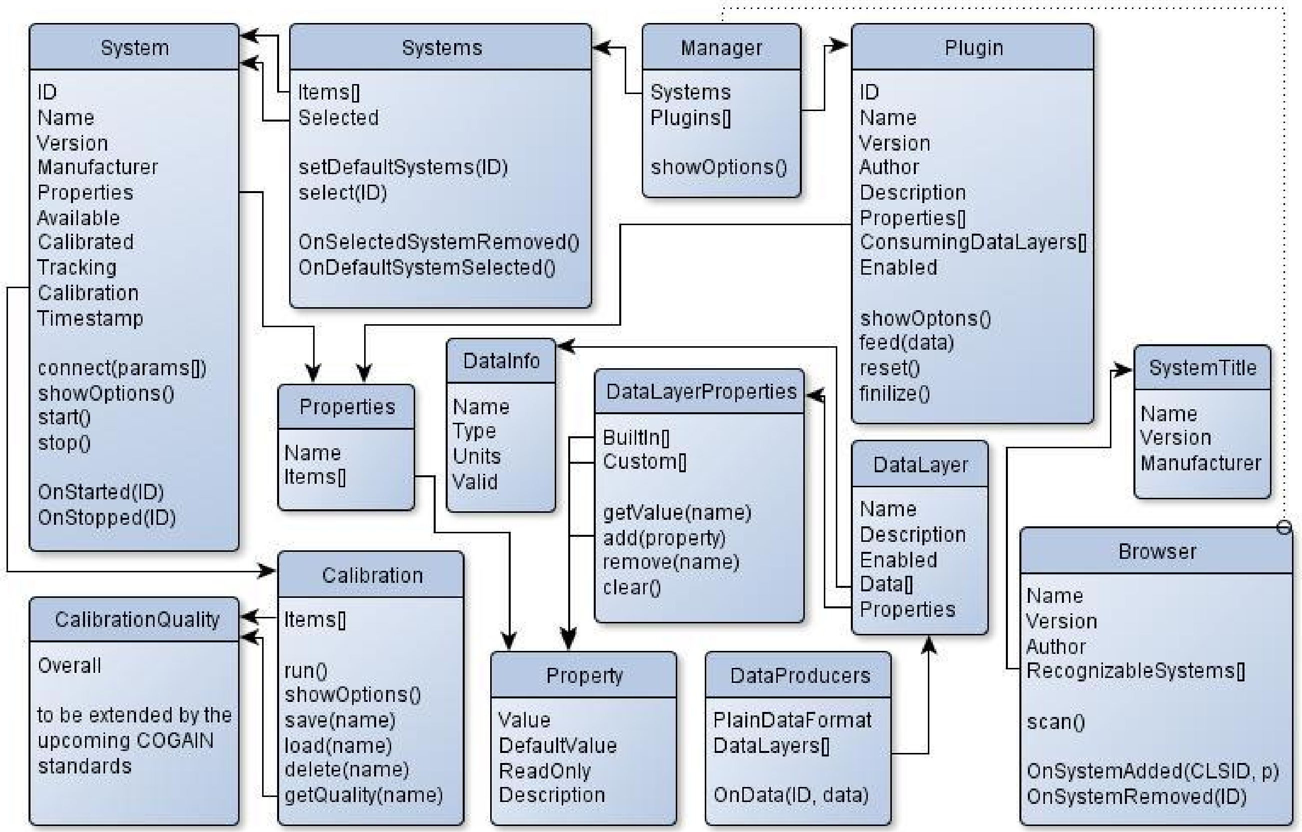

Figure 1. The list of interfaces in the proposed standard eye-tracking API

Acknowledgements

We thank all the participant of the discussion on COGAIN forum for their contribution into the vision expressed here and Päivi Majaranta for the useful comments on this manuscript.

References

[1] Daunys, G., Vyšniauskas, V. Eye Tracker Connectivity: Alternatives to ETU-Driver. *In Proceedings of the 5th Conference on Communication by Gaze Interaction (COGAIN 2009): Gaze Interaction For Those Who Want It Most*, Lyngby:DTU, 2009. ISBN 9788764304756. 77-80, 2009.