Enhancing Usability of the Human-Computer Interfaces with Software Architecture

R. KASPRZYK, J. SROCYŃSKI, K. STELMACH, P. SZADKOWSKI, K. WAWRUCH
rkasprzyk@wat.edu.pl

Institute of Computer and Information Systems
Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw

The article tells about two projects, which pay a great attention to the architectural side of a non-standard human-computer interface. The first project is THEIA, which is a touchless human-computer interaction system. The main target is to give the end-user an ability to interact with a computer only by his/her sight. By looking at areas of the screen the user triggers actions in the operating system he/she is using. The second project is OLIMPUS designed for creating and integrating modern user interfaces supporting any non-standard interaction sources. The project aims at simplifying a development process of interfaces collaborating simultaneously with many different sources of interaction and their dynamical exchanging as well.

Keywords: interaction, interface, architecture.

1. Introduction

Nowadays there are a lot of possibilities of the interaction with a computer – good quality webcams make it possible to create an image analysis based on interaction, the growing power of CPUs allows to perform speech recognition in real-time and much more. But, in fact, any attempt to create such an advanced solution is by now uneconomical. The problem is that most of the revolutionary interface solutions, to show their potential, need a dedicated implementation in the end-user software. Doing it in only a few applications will change nothing. Doing it with most of popular software is too expensive way.

In this article two projects are described, which are related to the human computer interaction area. The difference is, that they both are trying to solve the problem of practical usage of them with their architecture. The first one – THEIA – by delivering a relatively easy way to connect any application to its interaction engine or extend its functionality. The second one – OLIMPUS – by creating a platform for integrating custom interfaces with a prepared consumer application, without any need of any changes in the used components and without any additional costs of such integration.

2. The THEIA project

The THEIA (stands for: THEIA – The Handy Eye Interaction Adapter) system was planned as a universal interaction software tool for touchless interaction with the personal computer. The communication with a standard PC is based on an eye position recognition and further a user's gaze point by analyzing the image obtained from a regular web camera.

The main reason for developing such a solution is to help a lot of people with many different disabilities (for instance "locked-in syndrome", paresis or paralysis) use a computer in order to acquire electronic sources of knowledge, or simply contacting the world. Moreover, the THEIA system can monitor such user's activity and alarm a carer in an emergency. In addition to facilities delivery to disabled people, THEIA has also aimed at business, industry and institution sectors as well as end user facilities providing increases of interactivity with computers and bringing benefits from it. General functional requirements are presented in the Fig. 1.

Fig. 1. Use Case Diagram for THEIA
However, making THEIA the competitive product on the market means eliminating disadvantages of existing solutions with similar capabilities, the project has also some assumptions connected with non-functional requirements, which are as follows:

- system supported by a regular web camera or a better one
- computing a user's gaze point only based on image data, without any face markers, infrared radiation etc.
- optimizing performance's issues as much as possible, at least over a dozen images (about 15 FPS) obtained from a camera should be analyzed
- minimizing CPU and memory usage for the rest of the running programs.

3. THEIA's prototype outline

The idea was to design a new system steering a dedicated user interface able to handle the whole functionality of a normal operating system nowadays and meet the requirements brought by the future.

As far as the architecture is concerned, it is built around three main components:

- Core component
- Overlay Manager
- Overlays' system.

Adding a communication system mainly based on pipe technology and design a communication protocol makes the architecture very modular. The whole idea is depicted in the diagram (Fig. 2).

The Core component is responsible for obtaining image captured by the web camera and analyzing it (Fig. 3). The analysis, preceded by smoothing obtained image data, converting it to a gray scale image and equalizing its color's histogram, includes firstly user's face position recognition and secondly eyes' areas detection, using appropriate models of the face and eyes. At the next stage the eyes' centers are being calculated in the following steps:

- finding eyes' pupils (mainly black color) inside eyes' areas boundary (mainly white to gray color)
- enlarging found eyes' pupils' images using an extra algorithm, developed at the ITA institute, Military University of Technology, sharpening the edges of objects visible in the image (more information is included in [15])
- detecting edges with Canny algorithm (an idea of edges detecting can be found in [5]) and further Hough Transform (more in publication [6]) in the enlarged eyes' pupils' images
- computing pupils' centers using obtained circles from Hough Transform.

The pupils' centers data are the basis for calculating a user's gaze point directed at the monitor screen. In addition, once face is found, it is constantly being tracked by THEIA Core and the whole analysis is being repeated. Whenever the user's face is lost, THEIA Core will automatically search for it once again. All these steps are performed in such a way that covers most non-functional requirements.
All computed data is passed through to the Overlay Manager, which handles the received data and controls the behavior of all components by triggering suitable actions. Moreover, it retrieves all requests sent from overlays as well as sends its own messages to both Overlays and the Core component. The whole process is reduced to decode the obtained messages, identify the requested action, encode a new message and send it to its receiver. This method is strictly connected with the communication protocol implying sending only formatted text messages according to the following pattern:

FUNCTION_NAME, param1, ..., paramN

The parameters' list can vary depending on the specific function name. Some examples have been given below:

COORD, X, Y
ACTIVEAREA, AREA_ID
CALIBRATION

Furthermore, the design protocol does not assume elements connected with identifying sender and receiver of the message as far as the stack mechanism supports management of the overlays. It assures that only one overlay can be active at the same time and as a result has exclusive right to sending and receiving messages. This mechanism provides a sort of automatic feedback messaging. On the other hand, the communication between the Core component and Overlay Manager is uniquely identified by the separate pipe connection. There can be only one such connection simultaneously. Such a solution allowed to fairly well collaborate with the used pipe technology and make it easily extensible with new components, like overlays.

Each Overlay is independently of one another working module providing special set of functionalities as are available in the operating systems. However, they can collaborate with one another only by communicating with Overlay Manager, giving a possibility to combine different functions to cover almost all, both basic and more sophisticated, areas of using a computer. Nevertheless, Overlays' system possesses some restrictions, connected with a computing precision of the gaze point, because of poor camera quality. All this has forced to design a special grid in which every single grid's field corresponds to a specific activity triggering implemented functionality. The grid could be any size but still it never consists of number of rows and columns grater than several ones.

The entire project was developed using Microsoft technologies. Every component was implemented in C# under .NET Framework 3.5. The core component was mainly based on EmguCV library (please visit home page to learn more [4]), which is a set of a different kind of image operations based on OpenCV project adapted for C# (more in [2], [3]). As it has been mentioned before, the models of the face and eyes for their detection are stored in several separate XML files as a part of EmguCV library. As far as graphical user interfaces (GUI) for each overlays are concerned it was designed as WPF application, including some 3D interactive components, if necessary, using 3DTools DLL library.

Having regard to some system's limitations and used solutions, that should be improved, THEIA will allow controlling computer only by the gaze. Such an opportunity has been noticed and positively evaluated by many organizations who have become acquainted with the concept of THEIA. As a result, it has won several awards both at home and abroad.

4. The revision of the objectives

The prototype of the THEIA system was a good basis for further analysis. It allowed to check, how much a created solution can be useful. The evaluation of the practical usefulness of the system, based on the market review and real world tests, have given some interesting observations, and changed some targets of the project.

The fact, which seems to be certain after some research, is that there appeared other systems in development, which offer the same functionality as the THEIA's eye-tracking core. Basic targets for those projects are nearly similar to THEIA's, and they are progressing very well. For example there is the Cyber-Eye system[7], created in the Multimedia Systems' Department at Gdańsk University of Technology. This system is based on infra-red lighting, and offers great accuracy of sight detection. The another example is the OpenGazer system [8], which performs exactly the same activities as THEIA's core module. It has a support of the Samsung company, the Gatsby Foundation, and the European Commission (as it is a part of AEGIS program). Both systems are in an advanced stage, and practically are more accurate and universal than THEIA's mechanisms. In that case any further development of the eye-tracking module of THEIA should be considered as pointless.
On the other hand, THEIA’s extensible architecture performed very well in practice. It was its triumph card, because most of the innovative interfaces have not developed any suitable way to implement them in the real world, like THEIA’s overlays’ system or communication protocol. The chance to bind any application with THEIA’s core with a small effort, or freedom with extending its functionality became a crucial advantage of this project.

Because of this an idea has emerged to try not only to create another way of communication with the computer, but mainly focus on what can be built from already existing human-computer interface devices and software solutions, and how to bind them to maximize the benefits for the end-user, but do it in a cheap and easy way. That is how the OLIMPUS project was initiated, the successor of THEIA (OLIMPUS stands for Overall, Light, Interactive, Multi-Purpose User-interface System).

5. The OLIMPUS overview

The OLIMPUS project aims to create a software platform which delivers to the end-user an easy to use but robust tool to bind the available ways of interaction with application software. The platform should give an ability to define what types of interaction the user wants to use with the applications, and for which actions in software applications each way of interaction is responsible. Those responsibilities should be able to dynamically change or expand on users’ demands.

Usually, in most consumer applications, the functionalities are able to be triggered by certain actions. These actions are strictly connected to certain interaction events (for example a key shortcut is related to an interaction with the keyboard only). The general scheme of such a relation is illustrated in Fig. 4. That makes all applications bind to their dedicated interaction events, and makes them immune to the unusual interaction methods, as far as these interaction methods are no trying to pretend that events.

The basic idea for the OLIMPUS project is to detach the interaction method from an action in software, and create an ability to freely match different interaction methods with the needed functionality. To achieve this there is a need to isolate an event listener from the application (Fig. 5), and through this make it switchable with another. That moves the interaction dependence from the application to the dedicated event listener. Then, with changing the listener, a change of the interaction method with the application takes place.

Fig. 5. Proposed action triggering

The OLIMPUS structure is based on this idea. It consists of three main components:

- interaction plugins, which are the mentioned above event listeners, isolated as a standalone applications capable of inter-process communication
- interface plugins, which are the mentioned above software applications, capable of receiving action signals for triggering actions
- the core component, which is a software tool responsible for connecting the interaction plugins with the interface plugins in a desired manner.

The interaction plugins are responsible for receiving data from an interaction input device connected to the computer, generate some action signals based on them, and send these signals to the core component. The core component is able to find out which interface plugin should use these signals, and deliver them to it. This manner of performing action triggering in applications makes an opportunity to freely select the interaction method to suite the user's demands, or power in the interface plugin with many interaction methods simultaneously (as in the Fig. 6).
Of course there is a need to prepare the applications to act as the interface plugin, or to create the interaction plugin, which is utilizing some interaction method. Because of that a large-scale implementation of this solution needs a solid support from applications' developers and interaction methods creators. But preparing software to work with OLIMPUS should be much easier and cheaper than altering applications every time, when new suitable interaction method will show up, and gives great possibilities in a customization of the interaction without any further costs.

6. The OLIMPUS core

The OLIMPUS core is the main part of the solution. Its design is still at the development stage, but already most of its fundamental functionality is designed and implemented. The whole platform is created with Java technology [10], because of its great set of libraries and extensions, suitable for such applications. The specific technologies will be described with related functions of the application. The core is responsible for critical activities of the whole platform, such as:

- executing and initializing all of available plugins
- performing the configuration of plugins, including a universal calibration GUI for interaction plugins, if needed
- creating a scheme of connections between plugins
- routing interaction signals between plugins accordingly with the created scheme.

Those are the main areas of the OLIMPUS' core functionality, and they are going to be explained a bit in this part.

As mentioned earlier, it was decided, that plugins should be delivered to the platform, as stand-alone applications, capable of inter-process communication with the core. It makes adding new plugins to the system much easier, because there is no need to change anything in the core itself. There is only a need to give it information which plugin has to be connected to the core and what it is capable of. Secondly, it makes all plugins technology-independent – the only requirement is that the plugin must support the OLIMPUS' way of communication. Plugins themselves are responsible for:

- interaction plugins – delivering interaction signals from user-interaction devices,
- interface plugins – utilizing action signals (e.g. for in-plugin actions).

As a communication technology was chosen XML-RPC [11]. It is a well defined, easy to use and widely implemented method of remote procedure calling. It uses HTTP as transport and XML as encoding. In practice it needs a dedicated, lightweight HTTP server running on each application connected to the core for receiving requests, and a technology-dependent client library to send requests.
During this process, the core executes the plugin with parameters – ports for configuration communication. The first parameter of the plugin execution is the port on which the core's XML-RPC server is running. The second one is the port on which the plugin should start its own XML-RPC server. When the plugin is ready, it sends a confirmation signal to the core. Finally, the core sends to the running plugin information about "forward port", the port to which the plugin should send action signals – it will be explained later.

When all these ports are set, it allows the plugins to receive configuration requests from the core. The plugin configuration does not include the acquisition of data about plugin capabilities – these data are stored within the extra file delivered with the plugin, which is read before the plugin execution. The configuration is performed by a specific API, and covers such areas as defining active interaction methods or controlling the state of the plugin (Fig. 8). By now, there are specified three characteristic states for plugins:

- running state – the plugin is running, but it is not sending or receiving any interaction signals to the core,
- active state – the plugin is running, and performing its interaction activities,
- calibration state – an optional state, in which the plugin, if needs to, can perform its own calibration or initialization processes.

The next thing is a scheme of connections between plugins, which is needed, because the core must know, where to direct signals from interaction plugins. It is created before plugins' activation. The structure, which is capable of storing the whole scheme is presented in Fig. 9 as a class diagram. This structure is relatively simple, but sufficient. The plugin class stores all the data about the plugin, including a group of interaction sets, of which it is capable. An interaction set is meant as a pack of action signals, which can be generated by the plugin, grouped by a similar purpose or some other criterion (but meant to be used together). The scheme contains information to which plugin a signal from a specific interaction set should be sent, and delivers operations, which support use of these data. It is achieved by joining instruction sets within the scheme in pairs – source and target instruction set of the same interaction type. But what does it mean – source and target?
There is a need to describe instruction sets. Because an instruction set is related to some section of a specific type of human-computer interaction, there can be a huge variety of such sets. To bring some order, there has been created a classification of instruction sets. First of all, there are two types of them – sources and targets. Sources are associated with interaction plugins, and are meant to send interaction signals. Targets are meant to receive signals, and they are associated with interface plugins (like consumer applications). In addition instruction sets are grouped by their control capabilities. The classification (Fig. 10) consists of 4 groups:

- configuration sets – only used for plugin configuration, and access to core controls
- complete sets – capable of handling the interaction with computer individually
- dependent sets – unable of handling the interaction with computer individually, but it is possible with some other dependent sets
- supplementary sets – prepared only to add more functionality to interaction, insufficient to be treated as a significant way of interaction.

There is an attempt to categorize dependent sets, to get an ability to evaluate when a group of such sets could be treated as one complete set. The most promising subdivision of dependent sets is based on their interaction purposes and is presented in Fig. 11. With such categorization there is a chance, that the core will be able to automatically ensure, if the interface plugin, according to the active connection scheme, have enough interaction plugins connected to it to be fully operational for the user.

The connection schemes are crucial for the routing, or the so-called forwarding process. When the plugins are active, some of them (interaction plugins) are sending signals to the core, other (interface plugins) are waiting for those signals from it. The core must find out, using the connection scheme, where specific signals should be delivered, and direct them to the right interface plugin. This process is almost similar to the work performed by a standard LAN switch [12]. That is why the core sends to plugins, during the initialization process, the additional port number – a forwarding port. On this port the core is running a “raw” HTTP server (the server which does not decode requests into RPC calls, but takes them, as they are). All interaction plugins are sending interaction signals to this server. It is meant to read a request as a XML message (which are XML-RPC calls), to read from it to which plugin and instruction set this call was sent, check in the scheme where this call is supposed to be delivered, and send it there. For delivering such functionality the core is using a HTTP Components library [13] created by Apache Software Foundation. This library delivers a complete toolset of low-level components focused on HTTP and associated protocols.

The whole process of forwarding, illustrated in Fig. 12, creates for all plugins an illusion of dynamically changeable set of signals/interaction methods. Additionally, it allows to dynamically change the scheme of connections, without disturbing work of plugins.

The final result is that the end-user can bind any application (which was prepared to work as an interface plugin) to any interaction plugins he
wants, without worrying about compliance – if the plugin supports the same interaction set as the application – it will work smoothly.

7. OLIMPUS' possible applications

The OLIMPUS is not an application but a platform for building applications. Therefore OLIMPUS not providing any direct functionality for the end-users. However, it could be an invaluable, user friendly solution supporting using a computer just by adding the proper set of interaction and interface plugins.

The first area, which comes to mind, where OLIMPUS could be a useful and functional solution is an area of personal computing. Home users are greedy for custom human-computer interfaces, which makes using a computer more enjoyable. OLIMPUS would deliver to them an ability to adjust the way of interacting with their favourite applications without additional costs, or specialistic knowledge, just by adding new interaction plugins. The interaction plugins would be created by both individual creators, or companies.

Secondly, this solution could be a really helpful in creating dedicated sets of software. OLIMPUS would be a good basis for integrating software applications with created for them interaction methods. It would make such solutions easily customizable, with a perspective of future change of the way of communication with such application at low cost, or the future reuse of the interaction method in another OLIMPUS-based projects. Finally, as a well tested tool it is minimizing the risk of failures at system integration point.

The existence of such technology is widely needed, especially where the diversity of the interaction methods with the computer is the key factor. The perfect example is the market of software for disabled people [14]. There is often a need to adapt special software to specific needs of some special group of people. In most cases the level of their disability is quite diverse, so they need different ways of interaction with the computer. With OLIMPUS, one application can by used by many different patients, and the only things which are changing are interaction plugins, adapted to the level of disability of the user. OLIMPUS perfectly meets such dynamically changing requirements on demand and gives a possibility to reduce cost as well as time for adjusting a new configuration.

Once OLIMPUS is installed all that has to be done is to install and run new or lacking plugins which automatically will be attached to the whole working system. It means OLIMPUS could be used independently of needs of practically anyone. What is more, OLIMPUS could support any combination of plugins and interfaces including also those yet to be developed in the future. On the one hand, OLIMPUS offers extensible API for developers giving innumerable opportunities to create and design completely new plugins and interfaces limited only by their imagination. On the other hand, strictly defined API provides backward compatibility and ensures that all new plugins and interfaces will be working correctly even if it is hard to predict their functionality right now.

8. Summary

The development of non-standard ways of user-computer communicating is becoming more and more rapid. By now mainstream computers have enough power to bear interaction based on voice recognition or image analysis. Additionally, more and more projects are popping out, trying to deliver such interaction experience. But because of the lack of support and inability to utilize them with existing software, they remain unused. Maybe this situation can be changed by projects like OLIMPUS, which are trying to find an easy and cheap way of adapting such interaction mechanisms to the real world usage.

The key aspect is the ability to change the controller binding with the destination application without a need to make expensive changes. That ability is the very base of the OLIMPUS system. In the same way, the system can be adapted to the habits and different user preferences in the ways of controlling a computer system. Above all, OLIMPUS allows using different interaction methods to the same application, or using one interaction method to power in many applications. This kind of solution, focusing on the architectural side of the human-computer interaction problem, could be a proper approach to the problem of adapting custom interfaces into everyday use.

9. Bibliography


Koncepcja architektury oprogramowania do komunikacji człowiek-maszyna

R. KASPRZYK, J. SROCZYŃSKI, K. STELMACH, P. SZADKOWSKI, K. WAWRUCH

W artykule przedstawiono koncepcje dwóch autorskich rozwiązań problemu komunikacji człowiek-maszyna. Obie koncepcje zostały zrealizowane w postaci prototypów o kodowych nazwach THEIA i OLIMPUS, potwierdzających słuszność założeń i ich realizowalność. Projekt THEIA (The Handy Eye-Interaction Adapter) miał na celu zbudowanie platformy pozwalającej na swobodną interakcję z komputerem za pomocą jedynie wzroku. Poprzez analizę obrazu ze standardowej kamery internetowej, program ustala strefę ekranu, na którą patrzy aktualnie użytkownik. Wraz z nakładkami na system operacyjny, będącymi integralną częścią projektu THEIA, umożliwia to pełną, a przy tym prostą obsługę komputera.

OLIMPUS (Overall, Light, Interactive Multi Purpose User-interface System) jest platformą do tworzenia i integracji nowoczesnych interfejsów użytkownika z dowolnymi, niestandardowymi źródłami interakcji. Tym samym OLIMPUS upraszcza proces wytwarzania interfejsów symultanicznie współdziałających z wieloma źródłami interakcji i ich dynamiczną podmienną.

Słowa kluczowe: komunikacja człowiek-maszyna, interfejs człowiek-maszyna, architektura.