Toward real-time multimodal processing: EyesWeb 4.0

Antonio Camurri[1], Paolo Coletta[1,2], Alberto Massari[1], Barbara Mazzarino[1],
Massimiliano Peri[1,2], Matteo Ricchetti[1,3], Andrea Ricci[1], Gualtiero Volpe[1]

[(1)] Infomus Lab – Laboratorio di Informatica Musicale, DIST – University of Genoa, V.le Causa 13, 16145 Genova, Italy
[(2)] NumenSoft s.n.c. di M. Peri & C., V.le Brigate Partigiane 10/4, 16129 Genova, Italy
[(3)] Eidomedia s.a.s., Via Assab 4/2, 16131 Genova, Italy

Abstract

The EyesWeb open platform (www.eyesweb.org) has been originally conceived at the DIST-InfoMus Lab for supporting research on multimodal expressive interfaces and multimedia interactive systems. EyesWeb has also been widely employed for designing and developing real-time dance, music, and multimedia applications. It supports the user in experimenting computational models of non-verbal expressive communication and in mapping gestures from different modalities (e.g., human full-body movement, music) onto multimedia output (e.g., sound, music, visual media). It allows fast development and experiment cycles of interactive performance set-ups by including a visual programming language enabling mapping, at different levels, of movement and audio into integrated music, visual, and mobile scenery.

EyesWeb has been designed with a special focus on the analysis and processing of expressive gesture in movement, midi, audio, and music signals. It was the basic platform of the EU-IST Project MEGA (www.megaproject.org) and it has been employed in many artistic performances and interactive installations. However, the use of EyesWeb is not limited to performing arts. Museum installations, entertainment, edutainment, therapy and rehabilitation are just some of a wide number of different application domains where the system has been successfully applied. For example, EyesWeb has been adopted as standard in other EU IST projects such as MEDIATE and CARE HERE in the therapy and rehabilitation field, and EU TMR MOSART. Currently, it is employed in the framework of the EU-IST project TAI-CHI and in the 6FP Networks of Excellence ENACTIVE and HUMAINE. EyesWeb users include universities, public and private research centers, companies, and private users.

1 Introduction

Our paper presents EyesWeb 4, a contribute to the area on music and interaction (Rowe 1993, 2001; Chadabe 1996), on expressive content communication in active spaces where integrated human movement (e.g., of a music performer, or a dancer), visual, and music languages concur as a whole perceived entity. EyesWeb contributes to research, experiment, and build applications in multimodal scenarios where different communication channels are used in human-computer interaction.

In the area of multimodal applications, a number of systems are available that let users to work on audio or video streams, such as PureData (Puckette, 1996), Max/MSP (www.cycling74.com), Isadora (www.troikatronix.com), vvvv toolkit (vvvv.meso.net) and others. However, such systems are often limited in that they are particularly oriented toward a modality of interaction, i.e., they might perform well only when working with video or audio. In other cases, the limitations are in the number of physical devices that can be managed by the system. Furthermore, designing multimodal interface is not only a matter of working with streams of different types, but mainly concerns the ability to work at different abstraction levels: in the framework of the EU-IST project MEGA (Multisensory Expressive Gesture Applications, www.megaproject,org) a conceptual framework for expressive gesture processing has been defined, structured on four layers (Camurri, Mazzarino, Ricchetti, Timmers, and Volpe, 2004). EyesWeb is a temptative to design an open platform that can be used at the different levels.

In recent years, EyesWeb has been satisfactorily used by our lab both for research purposes and for several types of applications, in museum exhibits or in the field of performing arts. Moreover, the platform has been made freely available on the Internet and the number of users has rapidly grown. This has enlarged the field of applications of the software platform, which has brought us to redesign the software in order to support the new requirements.

This paper will explore the new requirements in Section 2, and will explain the new characteristics of the software in Section 3, with an in-depth analysis of the added features and concepts. The new graphical user interface is briefly introduces in Section 4. Finally, Section 5 will give some concluding remarks.

2 From EyesWeb 3 to EyesWeb 4

Intensive use of the EyesWeb platform up to version 3.x has brought to evidence a number of new requirements. Many of these requirements have been faced by releasing updates to the existing EyesWeb version. However, some new requirements implied a deep revision of the software at different levels. This led us to the decision of redesigning the system from scratch, keeping into account original and new requisites, and trying to forecast possible future requirements.

Users of the EyesWeb platform are spread over a number of different fields, from education to performing arts, from industry to research, and more. This widespread use of the system has the consequence that user requirements have been collected and summarized through different means. One main channel has been the availability of public newsgroups (see www.eyesweb.org) where users can post support requests, comments, and suggestions. Other channels have been a direct contact with final users (e.g., research centers), and, of course, our direct use of the software both for research purposes or for several types of applications, from artistic and museum installations to therapy and rehabilitation.

Such different user requirements have been collected and discussed in depth, before becoming the software requirements for the new EyesWeb version described in this paper. In the following, we'll focus on such new requirements, trying to put in evidence the difference with previous releases.

In brief, main requests concern conceptual issues on multimodality, and issues concerning usability, performance, robustness, interoperability with other systems, optimizations for some common operations.

U*sability*, concerns the availability of a *subpatch* mechanism and some deep modifications of the scheduling algorithm. Providing subpatches means to provide a mechanism to hierarchically group a subset of a patch (up to a complete patch) to form a single component that can be managed as a single block. This has been one of the first requisites emerged from the use of EyesWeb 3.x, as the complexity of patches grew up as EyesWeb was used in scenarios of increasing complexity. More importantly, subpatches have been implemented with the perspective of supporting future meta-levels in which a supervisor software can activate and control different (sub)patches dynamically. This is particularly useful for implementing the indirect interaction strategies (see Camurri et al, 2004 in these proceedings).

Concerning modifications of the scheduling mechanism, they are oriented to hide some inner details to the user (e.g., the difference between active and passive blocks in version 3), and to manage inside the kernel some synchronizations issues which, in the previous version, had to be faced by the final user. Thus, synchronization of audio and video is now supported and managed in the EyesWeb language. From the implementation point, EyesWeb tries to use a single clock to schedule the patch execution.

Performance concerns the optimization of the kernel when managing audio and video streams, as well as the exploitation of the characteristics of the actual processors. In particular, a main focus is on multiprocessors systems. Nowadays, motherboards with dual processors are available at reasonable prices; the new version of the EyesWeb kernel is built in such a way that dual (or multi) processors computers can be exploited at best.

Robustness is obtained mainly by completely separating the graphical user interface from the kernel. As a consequence, kernel execution will not suffer of possible bugs of the user interface. Standalone applications which do not need user interface and have to operate for days in unsupervised environments (e.g. a museum) will take advantage of the versions with a minimal (or without) GUI. Moreover, the interaction paradigm between the interface and the kernel has been greatly simplified if compared with previous versions. Two main methods are used to communicate from the interface to the kernel or to notify events from the kernel to the interface. Such methods will be explained in detail in the next section.

Interoperability with other systems concerns the capability of the platform to embed plugins from existing systems. The previous version of EyesWeb already supported standard plugins, such as the VST plugins. However, the implementation of the adapter between EyesWeb and the external plugins was not simple, as EyesWeb blocks and EyesWeb GUI were strictly coupled. The simplification of the communication paradigm between the EyesWeb kernel and the EyesWeb interface implies that their coupling has been reduced. Thus, the implementation of the adapter for existing plugins will be greatly simplified.

Finally, *optimizations* for some common operations implies that some modules and datatypes are implemented natively inside the kernel. This implies that optimizations can be performed by the execution engine for such objects, as it can make more assumptions on their implementations. Among such modules we may enumerate flow control blocks (i.e., the blocks that were previously included in the Generic library), operations on basic datatypes (strings, integer, booleans, etc.), and more.

3 EyesWeb 4.0

The new version of EyesWeb introduces a number of features and concepts which were not available in the previous version. This section is devoted to the explanation of such novel characteristics and to give the motivation for their existence. In particular, the focus of this Section will be more on the new kernel features than on the user interface features (however, the interface has

been renewed and improved with a comparable number of novel characteristics).

The first feature is the distinction between the kernel engine and the patch editor. The kernel engine is contained in a separate dynamic-link library (dll) on which the user interface relies for some services. On the opposite, the kernel does not rely on the user interface for its proper working. Consequently, different interfaces might be provided to edit as well as to execute patches. As a matter of fact, besides the main editor which we commonly refer to as the user interface, two more execution interfaces will be provided. The first is a simple command line interface which runs in Windows console mode; the second is a Win32 application which runs hidden and just displays an icon in the tray area (the small area were the Windows clock is usually displayed). Both these interfaces do not provide editing capabilities; they only let users execute patches with no further overhead.

In the kernel, new concepts have been introduced: clocks, devices, catalogs, kernel objects, subpatches, pins, and collections.

3.1 Clocks

Clocks are the objects which are responsible of providing the current reference time, and to generate proper triggers and alarms when needed. Although more than one clock might be used to schedule the patch execution, the default behaviour of EyesWeb will be to use a single clock. Multiclock behaviour must be forced by the user choices, or are adopted in rare cases. EyesWeb can provide an internal clock, if no one of the objects in the patch can provide its own clock; such *default* clock is based on the Windows multimedia timer. Objects in the patch can provide their clock and ask the system to use that one. However, during the initialization of the patch for its execution, the kernel elects a preferred clock and, if some rare conditions are not verified, that specific clock is used to schedule the patch. The criteria to choose which clock to use when more that one is available is to give priority to renderer blocks (e.g., the sound playout blocks), than to source blocks, and finally to the other types of blocks. If this criterion is not sufficient to establish the winner clock (this may happen if more than one clock is available in the class with the highest priority), a weighted priority is computed based on the values assigned to the outputs of the blocks by the blocks developers. If this further criterion is not sufficient, the patch is executed using more that one clock, and the possibility of jitters is signalled to the user.

The new concept of clocks, besides reducing synchronization issues for the final users, is mainly useful to support multimodal interaction. As a matter of fact, interaction of different streams is simplified by handling the various streams with a common clock. Moreover, when a common clock cannot be used as the streams come from intrinsically unsynchronized sources, a synchronization mechanism can be provided natively by the kernel.

3.2 Devices

Devices are internal objects which manage the interface with hardware. They cannot be used directly by the user, but they are used by blocks developers to interact with hardware. This new layer of abstraction, which was not available in the previous EyesWeb version, adds the possibility to map the available physical hardware resource to the virtual devices used by blocks. This makes patches more portable among different computers, even if the hardware configurations of the systems are not equivalent

3.3 Catalogs

Catalogs are responsible to enumerate the available blocks, subpatches, clocks, devices, and datatypes, and to instantiate and deallocate such objects. They also have the responsibility to provide authorship information, i.e., name and description of the authors and company that developed the module, as well as information about the licence of that block. In brief, catalogs carry meta-information about the blocks, and provide a factory for their instantiation.

The delegation of the enumeration and factory responsibilities to an object which is not the kernel itself, provides a powerful mechanism to simplify the support of plugins from other platforms. In facts, the methods to enumerate or to instantiate plugins from other platform can vary considerably from one system to another; e.g., the methods to enumerate and instantiate plugins from DirectShow architecture is completely different from the method to enumerate VST plugins. Delegating these responsibilities to an object which is not the kernel itself loosen the coupling between the kernel and the objects (blocks, datatatypes, etc.). A further advantage is given by the fact that catalogs can be implemented both inside or outside the kernel, thus, it will be possible to provide the support for new types of plugins without the need to modify the EyesWeb core.

3.4 Kernel objects

The concept of kernel objects in another main difference with the previous EyesWeb version. With kernel objects we refer to some EyesWeb objects (i.e., blocks, datatypes, clocks, subpatches, or devices) which are embedded in the kernel and which have a priviledged access to the kernel. This is a main difference with the approach of the previous architecture: in EyesWeb versions up to 3.x all blocks and datatypes were treated as external plugins. EyesWeb did not rely on the exhistence of any of these objects for its proper working. At the first glance this new approach might strengthen the dependencies between EyesWeb and some objects, hence limiting expansibility of the system. However, you must consider that the use of kernel objects is limited to the components which really need access to the kernel

internals. Among such components we may enumerate control flow blocks (switch, for or while loops, conditional operations, etc.), basic datatypes (integers, doubles, booleans, strings, etc.), basic clocks (multimedia timer) or devices (keyboard, serial, or mouse). Moreover, this approach has some more advantages: one of them is the homogenization between datatypes and parameters. Previously, data which flowed from outputs to inputs of blocks was completely different from data which flowed to parameters. The former was implemented through external plugins, which we called datatypes, whereas the latter was implemented through a limited set of standard types (int, double, bool, char *). With the new approach, the basic standard types are implemented as kernel objects. Thus they have both the advantages that the kernel can safely rely on their existence, and that they have the same standard interface of all other datatypes.

3.5 Subpatches

Complexity of patches can increase as long as EyesWeb is used to operate in real scenarios. Subpatches offer a way to handle this complexity by managing a set of interconnected blocks as a single object. Different modalities to use subpatches will be provided in the new version of EyesWeb. They all share the concept to manage a complex set of blocks as a single one, but they differ in how multiple subpatches instances are managed and in the visibility scope. The first model is based on the assumption that different subpatches of the same type are disjoint; thus, modifying one subpatch does not alter the other instances of subpatches of the same type. A second model is based on the assumption that different instances of subpatches of the same type do share the subpatch class: modifying a subpatch acts on all instances of such subpatch. Making a comparison with a programming language, the first model is comparable with a macro, which causes the source code to be duplicated wherever the macro is used; the second mode is similar to a function call, where the function code is shared among all function calls.

Another difference is related to the visibility of subpatches: subpatches might be built in the scope of a patch and not exported outside the patch. Such subpatches will be visible only when the owner patch is loaded, and will not be usable outside that patch. Another modality is to export the subpatch in order to make it visible to the whole system. In such a case, all patch shall see such subpatches and referring to them will not result in an error.

3.6 Pins

Pins enable blocks to interconnect one another, or to connect with subpatches. The new version of EyesWeb has brought pins to the level of kernel objects, instead of beings just graphical objects as it happened up to EyesWeb 3.x. Thus, pins can be instantiated in a patch not only when attached to a block (i.e., as inputs, outputs, or parameters pins) but also as standalone objects, or as a facility to specify exported inputs, outputs, and parameters of subpatches.

Standalone pins can be places in the patch and have incoming and outgoing links. Besides providing a facility to avoid redrawing links when a source or destination block is removed, they allow the logical connection of graphically unconnected pins by simply assigning them the same name. Thus, remote parts of the same patch can be connected through pins with the same name without having to draw an actual link.

Another possible use of pins is to specify the exported parameters of subpatches. When a set of blocks is grouped to form a subpatch, not all pins are exported and visible by the users of such subpatch; the pins to be exported can be specified by selecting a subset of the available one; in such subset is it also possible to include pins which are placed in the patch as standalone pins.

3.7 Collections

Collections of different types are included among the objects that kernel can manage natively. The kernel itself is built upon collection facilities. Patches, for instance, contains collections of blocks, datatypes, links, devices, clocks, etc. Besides being used by the kernel, collections can be used by developers e.g. to build complex datatypes basing on the available ones. This simplifies developing higher level datatypes based on the ones available at the lowers levels, as well as a mechanism to link together different types of data; thus, in a way, they provide a basic support to multimodality.

4 Graphical User Interface

The EyesWeb Graphical User Interface has been deeply redesigned, in order to adapt to the new features introduced by the current EyesWeb version, and to provide a more modern look and feel. A first difference with the previous GUI is visible in the Catalog View (the treeview which is by default placed on the left side, and which shows the list of available blocks). Besides proposing the enumeration with the same characteristics as EyesWeb 3.x, it adds a new mode (called Catalog mode) which let users see the distinction of the blocks in disjoint catalogs. Another feature added by the Catalog View is the possibility to filter out some blocks from the list, thus simplifying the process to find out the desired module among all the available ones.

The Patch View, which is the main panel visible in Figure 1, adds zooming grid alignment capability. Moreover, it supports settings the parameters of multiple selected blocks in a single operations. For this purpose, the Properties View, which replaces the previous param dialog, is single instance. This means that only one instance of such view is active at a given time. However, such unique instance is able to show multiple values: when multiple blocks are selected, homogeneous parameters can be managed together, i.e., it is possible to set multiple values at the same time.

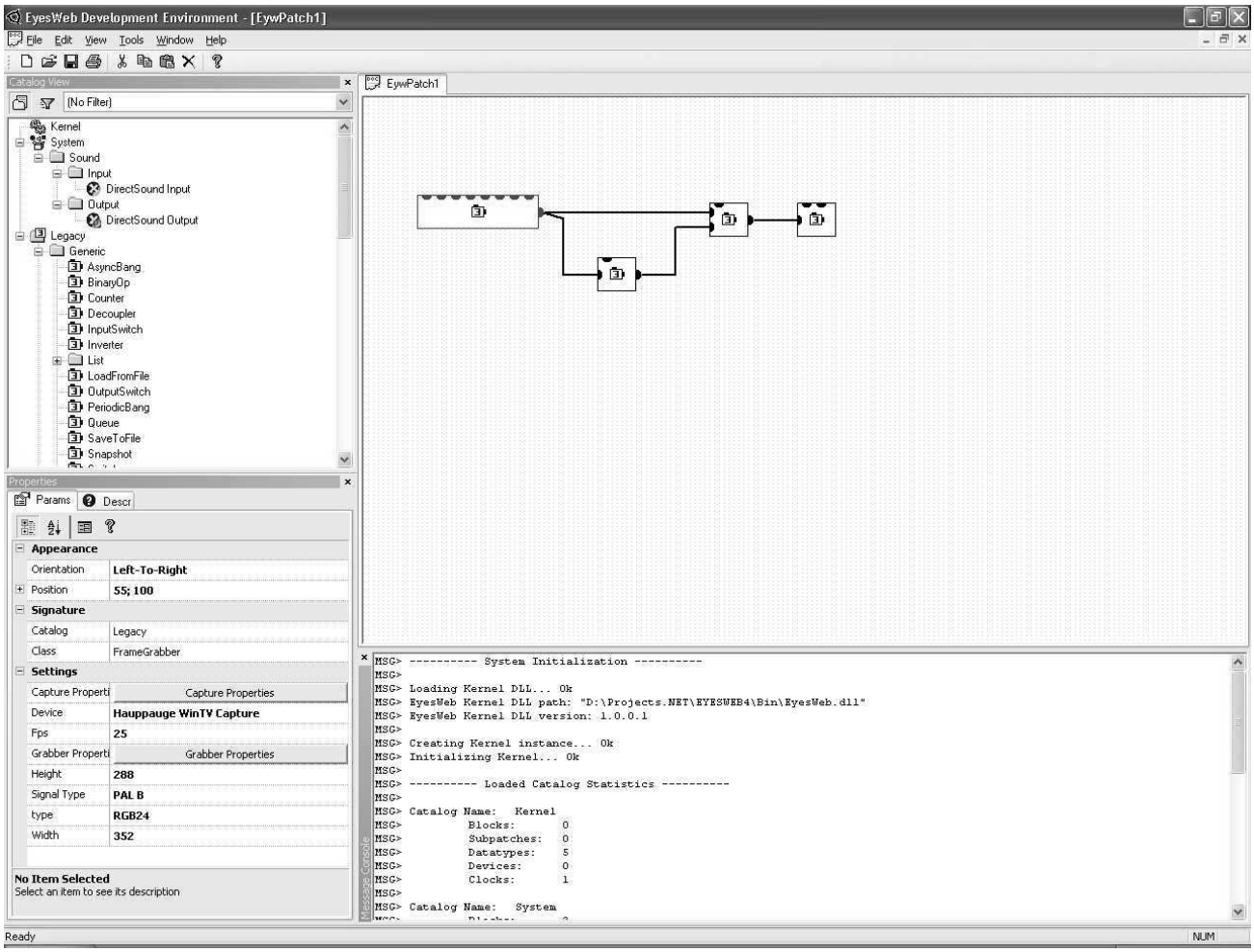

Figure 1. A screenshot of EyesWeb 4.0 at work.

5 Conclusions and future works

The paper has analyzed the upcoming version of EyesWeb, with a particular focus on the requirements that have emerged by the extensive use of the previous EyesWeb versions. EyesWeb was originally conceived to support multimodal human-computer interaction: the upcoming version of EyesWeb tries to enforce this characteristic, while improving the performance in the different application scenarios, and with the perspective of supporting different levels of abstraction in multimodal processing.

The upcoming EyesWeb version is still under development at the time of writing this paper. A first public demonstration will be done at the AISB 2004 convention.

6 References

R. Rowe, *Interactive Music Systems*, MIT Press, 1993.

R. Rowe, *Machine musichianship*, MIT Press, 2001.

J. Chadabe, Electric Sound. *The past and promise of electronic music*. Prentice Hall, 1996.

M. Puckette, Pure Data. Proceedings, International Computer Music Conference. San Francisco: International Computer Music Association, pp. 269-272, 1996.

Cycling74, http://www.cycling74.com

TroikaTronix http://www.troikatronix.com

Meso http://vvvv.meso.net/

A. Camurri, B. Mazzarino, M. Ricchetti, R. Timmers, G. Volpe, Multimodal analysis of expressive gesture in music and dance performances. In A. Camurri, G. Volpe (Eds.), *Gesture-based Communication in Human-Computer Interaction*, LNAI 2915, Springer Verlag, 2004.