### D 3.2 Early prototypes V2 (M14)

<table>
<thead>
<tr>
<th>Grant Agreement nr</th>
<th>644862</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project title</td>
<td>Realtime Adaptive Prototyping for Industrial Design of Multimodal Interactive eXpressive technology</td>
</tr>
<tr>
<td>Project acronym</td>
<td>RAPID-MIX</td>
</tr>
<tr>
<td>Start date of project (dur.)</td>
<td>Feb 1st, 2015 (3 years)</td>
</tr>
<tr>
<td>Document reference</td>
<td>RAPIDMIX-WD-WP3-D3.2.docx</td>
</tr>
<tr>
<td>Report availability</td>
<td>PU - Public</td>
</tr>
<tr>
<td>Document due Date</td>
<td>March 31st, 2016</td>
</tr>
<tr>
<td>Actual date of delivery</td>
<td>April 8th, 2016</td>
</tr>
<tr>
<td>Leader</td>
<td>IRCAM (Deliverable coordination)</td>
</tr>
<tr>
<td>Reply to</td>
<td>Frédéric Bevilacqua (<a href="mailto:Frederic.Bevilacqua@ircam.fr">Frederic.Bevilacqua@ircam.fr</a>) (IRCAM)</td>
</tr>
<tr>
<td>Additional main contributors (author’s name / partner acr.)</td>
<td>All partners</td>
</tr>
<tr>
<td>Document status</td>
<td>Reviewed by PLUX</td>
</tr>
</tbody>
</table>

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N° 644862
EXECUTIVE SUMMARY

This document accompanies the second set of the so-called Early-MIX prototypes at M14, completing the first set Early-MIX prototypes reported at M14. The document is structured as follows. First we recall the role of prototyping in the project, and in particular in the WP Agile Prototyping. Second, we list the Early-Mix prototypes in a synthetic table, indicating the context, technology, platform and authors/contributors. The third section of the document provides with a more detailed description of each prototypes. The last section presents the complete list of Early-Mix prototypes.
BACKGROUND
This deliverable, builds upon D3.1 to document, the second list of early hardware and software prototypes. These prototypes make use of the candidate technologies selected in T4.1, and the documentation is aligned with the guidelines defined in D2.2. The outcomes of this phase will be used in T2.3 for testing early design guidelines.
# TABLE OF CONTENTS

1 INTRODUCTION .................................................................................................................. 5  
1.1 Scope and structure of the document .................................................................................. 5  
1.2 Acronyms and abbreviations ............................................................................................. 5  

2 OVERVIEW OF EARLY-MIX PROTOTYPES ...................................................................... 6  
   Early-MIX Prototypes at M14 .............................................................................................. 6  
   Multimodal Recognition ....................................................................................................... 8  
   (Bio)Signals in Context ......................................................................................................... 10  
   WIML: Web Interactive Machine Learning .......................................................................... 12  
   MaxiLib.js ............................................................................................................................ 15  
   PiPo Plugin in JUCE ............................................................................................................ 18  
   Machine learning software add-ons for OpenSignals (r)evolution ...................................... 20  
   BITalino UART to OSC bridge and WiFi transmission ........................................................... 23  
   Hardware development platform + IDE ................................................................................ 25  
   BITbox ................................................................................................................................... 27  

3 CONCLUSION ......................................................................................................................... 29
1 INTRODUCTION
As stated in the DoW, the main goal of the WP3 (Agile Prototyping) is to develop, in fast cycles, prototypes combining both hardware and software components. These prototypes can be sorted in different categories such as “proof of concept” prototypes that explore new technological solutions, software and hardware prototypes that integrate different components never connected before, or first versions of future commercial products.

During the project, the prototype building and evaluation follow three main cycles:

- Early MIX prototypes (first 14 months of the project)
- Prototype for RAPID-API testing
- API demonstrators for creative domains

The first phase of the Early-MIX prototypes development is divided in two phases, with two deliverables at M8 and M14, respectively. We report in this document on the second phase of Early-MIX prototypes at M14 (the first phase was already reported in D3.1).

1.1 Scope and structure of the document
This document describes the current Early-MIX prototypes existing at M14. This completes the phase of Early-MIX prototypes, and complete the document D3.1.

The goal of the document is two-fold:
1. To give an overview and to point out the links between the prototypes and the current work carried on in WP2 (User Center Designed), and in particular with the results reported in D2.2. This is described in section 3.
2. To provide a concise description (typically 2 pages) of each prototype (Section 4).

1.2 Acronyms and abbreviations
- DoW - Description of Work
- HCI - Human-Computer Interaction
- IML - Interactive Machine Learning
- UCD - User-Centred Design
- MIR - Music Information Retrieval
- IoT - Internet of Things
- OSC - OpenSoundControl protocol
- GMF (Gaussian Mixtures Regression)
- HMM (Hidden Markov models)
- XMM (Multimodal and Hierarchical Hidden Markov models)
- GF - Gesture Follower
- GVF - Gesture Variation Follower
- IRCAM-IoT-board - WiFi board for Internet of Things (prototype)


## 2 OVERVIEW OF EARLY-MIX PROTOTYPES

### Early-MIX Prototypes at M14

We give here an overview of the prototypes built at M14, reported in the table below. Each prototype is further described separately in Section 4, with more detail on the goal and status of each prototype. Videos are available here https://drive.google.com/folderview?id=0B_Qe1G6Umj3RdDRDWU5SN2JMR0E&usp=sharing

Prototypes finalized at M14 (in addition to the ones at M8)

<table>
<thead>
<tr>
<th>Name</th>
<th>Context/Use Case</th>
<th>Technology currently used</th>
<th>Possible integration with other technologies</th>
<th>Platform</th>
<th>Authors</th>
<th>(Potential) Additional Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimodal recognition</td>
<td>User-tailored interaction, Explorative interfaces, Expressive dancing interfaces. Interactive musical experiences.</td>
<td>XMM, r-IOT, Giant Steps Music Engines</td>
<td>Wekinator, RepoVizz, GVF</td>
<td>Desktop/laptop</td>
<td>MTG, IRCAM</td>
<td>MTG, IRCAM</td>
</tr>
<tr>
<td>(Bio)Signals in context</td>
<td>Reactable Running (wearable and mobile devices)</td>
<td>mobile devices, RepoVizz, BITalino</td>
<td>r-IOT, XMM</td>
<td>Mobile &gt; Web/Deskt op &gt; Mobile</td>
<td>UPF, GS, RS</td>
<td>UPF, GS, RS</td>
</tr>
<tr>
<td>WIML: Web Interactive Machine Learning</td>
<td>Interactive Machine Learning through a web interface</td>
<td>XMM</td>
<td>RepoVizz, BITalino, CoSIMA, Wekinator, GVF</td>
<td>Server + mobile</td>
<td>IRCAM, Orbe</td>
<td>IRCAM, Orbe, MTG, PLUX, Goldsmiths</td>
</tr>
<tr>
<td>MaxiLib.js</td>
<td>Intended as a proof-of-principle approach to cross platform API design. Prototype toolchain development for cross-platform audio including web audio - uses an identical API.</td>
<td>Maximilian, Emscripten</td>
<td>Anything</td>
<td>WebAudio and native C++ combined</td>
<td>GS + anyone</td>
<td>Anyone</td>
</tr>
<tr>
<td>PiPo Plugin in JUCE</td>
<td>Testing PiPo processing with OSC connection</td>
<td>PiPo, Juce</td>
<td>Wekinator (simple via OSC)</td>
<td>OSX, can be extended to iOS, Android, Windows,</td>
<td>IRCAM</td>
<td>IRCAM, Roli</td>
</tr>
</tbody>
</table>
### Prototypes (ongoing)

<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Context/Use Case</th>
<th>Technology currently used</th>
<th>Possible integration with other technologies</th>
<th>Platform</th>
<th>Authors</th>
<th>(Potential) Additional Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Machine learning software add-ons for Open Signals (r)evolution</td>
<td>Create aggregated reports and/or extract features from biosignal recordings stored by the user</td>
<td>BITalino, OpenSignals, RepoVizz</td>
<td>r-IOT, Wekinator, GVF</td>
<td>Desktop</td>
<td>PLUX, GS, IRCAM, UPF</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>BITalino UART to OSC bridge and WiFi transmission</td>
<td>Bring OSC and WiFi transmission to the BITalino</td>
<td>Proof of concept with r-IOT</td>
<td>BITalino</td>
<td>IRCAM+ PLUX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Hardware Development Platform + IDE (linked to #8)</td>
<td>Hackers, makers, music technology researchers, tech companies making prototypes</td>
<td>Proof of concept with r-IOT &amp; Energia</td>
<td>Desktop -&gt; to mobile / wearable / IoT</td>
<td>GS, IRCAM, ROLI, PLUX, (possibly other partners)</td>
<td>GS, PLUX, IRCAM</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>BITbox</td>
<td>SDK / toolbox for multimodal feature extraction from raw biosignal data time series</td>
<td>BITalino, OpenSignals</td>
<td>RepoVizz, r-IOT, (possibly the Seaboard - ROLI)</td>
<td>Python, C++</td>
<td>PLUX, (possibly other partners) (possibly other partners)</td>
<td></td>
</tr>
</tbody>
</table>

Finally, we add here for references prototypes, currently developed or planned, linked to future products (not fully reported here, but described in other deliverables)

|   | Wearable Musical Instruments V2                                      | Evolution of previous prototype to test toolchain development Commercial target will include Kurv. | Wekinator, C++, Maximilian                                                               | To be determined, Possibly integrating IML with mobile and embedded systems. | Java, C++, iOS | GS, (possibly other partners) (possibly other partners) |
|   | ORBE-MIX                                                            | Soundwalk, Soundscape, Dynamic map, Connected map, ARG                                  | Waves.js, soundworks, XMM, Collective Soundworks                                          | RepoVizz, Freesound, BITalino, Wekinator                                                 | Server, API, Mobile | ORBE, (possibly other partners) (possibly other partners) |
|   | IML-enabled sound design mobile interface                           | GS, AudioGaming                                                                        | GS                                                                                       | MIX product oriented, Tool for sound designers                                             | iPad | iOS, OSC, Wekinator |

D 3.2 Early prototypes V2
Multimodal Recognition

This prototype targets the creation of interactive generative music experiences based on Interactive Machine Learning (IML)-based gesture recognition.

Contexts and Use Cases
User-tailored interaction, explorative interfaces, expressive dancing interfaces, Interactive musical experiences.

Goals/Questions
- IML for gesture recognition, taking into account trajectory
- IML based on live examples
- IML for Interactive music generation
- Music generation for interactive experiences

Description
IML has been used in sound and music related interactive projects many times. It is usually used to recognize and map user input in form of gestures and poses into parameters of the sound. Wekinator is widely used to recognize discrete user poses and map them into values that be interpolated from the given examples. GVF is traditionally used to map a single example gesture (poses over time) to a sound, and use the performer variations on the gesture to modify the sound playback. This prototype proposes a third approach, one that maps gestures (poses over time) to (interpolated) parameters, in order to control a music generator.

We use Hierarchical Hidden Markov Models (HHMM) from the XMM library to model the input from the user's sensors to the parameters. To do so, we first use some chosen parameters to generate music and record the user making gestures and moving on the music. We repeat this for every set of parameters, each one with a different gesture. We then train the model. At the end the user can make gestures and parameters will be generated to control the music. If the gestures match the trained ones, the parameters will be the ones used to train the model. New sets of parameters, somehow in-between the ones used for training will be generated in case on non-matching gestures.

The music generation process is carried by House Harmonic Filler and Dr.Markov projects. The House Harmonic Filler is an ongoing study of how to achieve chord variations in an interactive and intuitive way. Based on corpus analysis, it allows the user to modify different musical parameters interactively and in real time. Some of these parameters include harmonic rhythm, density, articulation and chordal complexity. Dr.Markov is a drumming agent that uses stylistic knowledge to generate drum patterns. Every generated pattern resembles the style it is based on, while
maintaining its own identity. Post processing controls allow to transform the density of each individual voice, adding or subtracting onsets without disturbing the key elements of the rhythm.

The prototype uses the r-IoT to sense the accelerations of the user. Those are sent through OSC to a PureData (Pd) patch. This patch uses XMM through its python interface and the Pd python external. The same patch controls Pd patches through OSC (\textit{House Harmonic Filler} and \textit{Dr. Markov PD patch through PD signals}). Both music generator patches send MIDI to a VST.

![Figure 1: (left) the R-IoT board, that serve as a base for this prototype. (right) The graphic user interface for the Multimodal Recognition prototype.]

**Technology used**
- Giant Steps House Harmonic Filler (UPF)
- Giant Steps Dr. Markov (UPF)
- XMM interactive machine learning library (IRCAM)
- r-IoT (IRCAM)

**Next steps**
- Use HHMM regression instead of classification
- Change music generator parameters continuously instead of every 64 beats
- Replicable implementation
(Bio)Signals in Context

This prototype combines the results of two previous prototypes (Open Biosignal Repository and Online Repository with Interactive Machine Learning capabilities), with movement data acquisition and classification on mobile devices, in order to create a complete interaction design toolchain for the Reactable Running MIX product.

Contexts and Use Cases
Fitness/Quantified self, multimodal data acquisition, design of new interfaces for musical expression, rapid prototyping tools.

Goals/Questions
- Assisting the development of Reactable’s MIX product
- Multimodal data acquisition “in the wild”
- Tools for multimodal interaction design
- Recognition of body states (walking, running, etc.) through IML

Description
This prototype provides a complete chain of tools for multimodal interaction design, oriented towards the early development of Reactable Systems’ MIX product (Reactable Running). Motion data (accelerometer, gyroscope, magnetometer, altitude), high level data (number of steps, current pace, current, cadence) and classification data using Apple’s CoreMotion Framework are acquired directly on an iOS device and uploaded to RepoVizz using its RESTful API; in an additional step, a video synchronization protocol developed by UPF can be used to automatically align and cut video recordings carried out simultaneously with the physical activity as a visual reference.

This serves several purposes: first of all, data can be recorded “in the wild” while a user is carrying out their daily exercise routine, then this data can be used by the Reactable team to improve the functionality of the engine by having multiple recordings by different users; finally it allows direct comparison between our classification and the one developed by Apple.

Once the data is uploaded on RepoVizz, technology from the Early Prototype V1 “Online Repository with Interactive Machine Learning capabilities” is used to stream the data to a local computer running an instance of Wekinator, in order to train a model that recognizes five different levels of body activity (standing, walking, jogging, running, sprinting), which are the initial events recognized by Reactable Systems’ MIX product, the intelligent music engine.

Wekinator has been extend to export trained models as simple C++ functions which are configured with specific trained data from Wekinator. In this prototype, the exported functions are used in an openFrameworks app which is being compiled directly onto an iOS device. openFrameworks can also compile for Android and Arm/Linux targets.
Technology used

- RepoVizz (UPF)
- Wekinator (GS)
- Apple CoreMotion Framework

Next steps

- Support data acquisition on Android devices (already in progress)
- Integrate the acquisition of biosignal data using BITalino (r)evolution directly interfaced with a mobile device
- Refine and expand Wekinator export to other model algorithms and topographies
- Export model data in a common, JSON format that can be added to the RepoVizz archive alongside of the data that were used for training
- Develop web service version of Wekinator that can be used to train models from RepoVizz data
WIML: Web Interactive Machine Learning

This prototype implement the possibility to use Interactive Machine Learning as a web service. The goal is to use different software modules together with a consistent GUI and an unified API for developers. The users should be able, through web pages, to record and share collections of training sets and trained models, visualize them, fine tune their training data. Furthermore the users should be able to experiment with the different available algorithms and compare their behaviours and results.

Contexts and Use Cases
- Based on the specific requirements for the installation from “Murmure Urbain” by Orbe
- Distributed and Shared Machine Learning

Goals/Questions
- Using Web technologies with C++ libraries for Interactive Machine Learning
- Sharing, dataset and learned models
- Separate learning and performance processes

Description
For each of these use cases we describe a corresponding software architecture, enabling the integration of third-party machine learning technologies.

Client-Server architecture
The server-side software is composed of a classical server (serving pages and implementing websocket connections). This server acts as a host launching various types of machine learning child processes, and as a control hub for all these processes. All the data is stored into a database, can be accessed from the main server code as well as from all the child processes. The clients communicate with this server through a classical web interface, running in a browser. Figure 3 gives an illustration of this architecture. For more advanced applications, technologies such as cordova and node-webkit allow for the creation of client softwares on mobile and desktop platforms that can handle a variety of libraries and protocols. This enables furthermore the use of a large variety of sensors in a direct interaction loop with the interface.

Simple decoder plugin format
We propose to use the IRCAM’s PiPo plugin format to perform “real-time decoding” algorithms derived from machine learning. PiPo has been successfully used as a way to write modules that can process heterogeneous data streams. PiPos can be concatenated to form chains, the resulting chain being a PiPo itself. A PiPo module has an input stream and an output stream, potentially having different sampling rates, number of channels, and frame sizes. It propagates unidirectionally its stream attributes through the chain on initialization, as well as the processed input frames on each process iteration. This architecture is well suited to host machine learning algorithms, especially for recognition tasks with trained models. Moreover, an equivalent JavaScript library (lfop) has been recently developed at IRCAM for a similar use with web technologies.

Example
The prototype, named WIML (for Web Interactive Machine Learning), implements both the Client-Server architecture with a simple decoding algorithm running on web page. In the context of RAPID-MIX, it implements a concrete use case described by Orbe: recognition of human displacement attitudes (walking, wandering, running, etc.) using mobile phones.

This prototype represents a case where the learning and recognition modules are distributed processes (see Figures 3 and 4). The recognition is performed continuously, while different phrases can be recorded, sent to the server, processed by the server, and the new model can be loaded back into the client without any interruption of the recognition process.

In the first version of this prototype, Gaussian Mixture Models from the XMM library are used for instantaneous feature analysis and classification. Training data and likelihoods results can be visualized continuously on the mobile and the servers.

![Figure 3: Distributed architecture for Interactive Machine Learning](image)

![Figure 4: Prototype software architecture.](image)
Technology used

- XMM (IRCAM)
- RepoVizz (UPF)

Next steps

This prototype could be extended to use the other software parts provided by RAPID-MIX partners as other child processes of the server. For each model, a decoder should be written for the client side, though. A deeper comparison of the internals of all the algorithms should be done to improve models compatibility on a low level, and make them potentially exchangeable in a same decoder. More use case should also be gathered and described in order to better define more fluid interaction scenarios.
MaxiLib.js

maxiLib.js is intended as a proof-of-principle approach to cross platform API design. It is a prototype toolchain and examples for cross-platform audio development including web audio. It uses an identical API across C++ and webAudio, so that engineers can develop in C++ native code, and then go on to use identical code inside HTML5 documents whilst preserving 95% of the same code syntax. It demonstrates how to engineer cross platform libraries using emscripten.

Contexts and Use Cases

Designers might work in PD or MAX to prototype devices that uses signal processing tools encapsulated in objects. However, this can be a barrier to commercialisation for embedded and mobile systems. Having original C++ code for such processing objects can make this easier, but requires an engineer. A worse situation is when a designer or engineer wants to implement a signal processing and machine learning toolchain in HTML5 that was realised using native code. This is a recurring use case in the RAPID-MIX target population.

Goals/Questions

It is not enough simply to compile C++ code into JavaScript, as this does not facilitate a unified, cross-platform API, only end-user applications. This is important, as the RAPID-API is intended to support developers. Engineers may waste considerable time porting C++ code to JavaScript. How can this process be improved? How can we increase the speed of development? How can we prevent duplication? What are the pitfalls of engineering a truly cross platform API in this way?

Description

maxiLib.js is a port of the Maximilian audio DSP library to asm.js via emscripten. This allows users to prototype with 95% the same code across C++ and JavaScript. The process of creating such a library has been highly instructive in that it is not straightforward to provide universal API access for both platforms. For example, loading a sample into memory requires access to memory, and the methods for loading audio data into memory in webaudio are vastly different. In this case, we re-engineered a vector processing stream for encapsulating audio data across both C++ and JavaScript with minimal changes to the API. In addition, IFFT processes and advanced feature extraction are not available in webAudio. We were able to easily add such features by re-engineering maximilian to use asm.js - friendly data structures. The result is a powerful DSP library for JavaScript, where developers can take code that works in C++ and use it in real-time in the browser.
The following screenshot shows an Xcode project that creates a synthesiser using maximilian in C++.

```cpp
mixer://we're adding up the samples each update and it makes sense to clear them each time first.
//so this first bit is just a basic metronome so we can hear what we're doing.
currentCount = (int)timer.phase(0); //this sets up a metronome that ticks 8 times a second
if (lastCount != currentCount) //if we have a new timer int this sample, play the sound
    if (voice == 0) {
        voice = 1;
    } else if (voice == 1) {
        voice = 2;
    } else if (voice == 2) {
        voice = 0;
    }
    ADSR[voice].trigger(0, adsEnv.get(0)); //trigger the envelope from the start
    pitch[voice] = voice + 1;
}

//and this is where we build the synth
for (int i = 0; i < 8; i++) {
    ADSRout[i] = ADSR[i].adsr(8 * ADSR[i].trigger()); //our ADSR env is passed a constant signal of 1 to generate the transient.
    LFO1out[i] = LFO1[i].sinebuf(0.2); //this lfo is a sine wave at 0.2 hz
    VCO1out[i] = VCO1[i].pulse(55 * pitch[i], 0.0); //here's VCO1. it's a pulse wave at 55 hz, with a pulse width of 0.6
    VCO2out[i] = VCO1[i].pulse(11 * pitch[i], 0.0); //here's VCO2. it's a pulse wave at 110 hz with LFO modulation on the frequency, and width of 0.2
    VCFout[i] = VCF[i].lores(VCO1out[i] + VCO2out[i], 0.5, 256 * (pitch[i] + LFO1out[i] * 1000), 10); //now we stick the VCO's into the VCF, using the ADSR as the filter cutoff
    mix[i] = mix[i] + VCFout[i] * ADSRout[i]; //finally we add the ADSR as an amplitude modulator
}
output[i] = mix[i] * 0.5; //left channel
output[i] = mix[i] * 0.5; //right channel
```

Now we present the same code as it appears in JavaScript:

```javascript
mixer://we're adding up the samples each update and it makes sense to clear them each time first.
//so this first bit is just a basic metronome so we can hear what we're doing.
currentCount = Math.floor(timer.phase(0)); //this sets up a metronome that ticks 8 times a second
if (lastCount != currentCount) //if we have a new timer int this sample, play the sound
    if (voice == 0) {
        voice = 1;
    } else if (voice == 1) {
        voice = 2;
    } else if (voice == 2) {
        voice = 0;
    }
    ADSR[voice].trigger(0, adsEnv.get(0)); //trigger the envelope from the start
    pitch[voice] = voice + 1;
}

//and this is where we build the synth
for (var i = 0; i < 8; i++) {
    ADSRout[i] = ADSR[i].adsr(8 * ADSR[i].trigger()); //our ADSR env is passed a constant signal of 1 to generate the transient.
    LFO1out[i] = LFO1[i].sinebuf(0.2); //this lfo is a sine wave at 0.2 hz
    VCO1out[i] = VCO1[i].pulse(55 * pitch[i], 0.0); //here's VCO1. it's a pulse wave at 55 hz, with a pulse width of 0.6
    VCO2out[i] = VCO1[i].pulse(11 * pitch[i], 0.0); //here's VCO2. it's a pulse wave at 110 hz with LFO modulation on the frequency, and width of 0.2
    VCFout[i] = VCF[i].lores(VCO1out[i] + VCO2out[i], 0.5, 256 * (pitch[i] + LFO1out[i] * 1000), 10); //now we stick the VCO's into the VCF, using the ADSR as the filter cutoff
    mix[i] = mix[i] + VCFout[i] * ADSRout[i]; //finally we add the ADSR as an amplitude modulator
}
output[i] = mix[i] * 0.5; //left channel
output[i] = mix[i] * 0.5; //right channel
```

Notice how the code is more or less identical. The sound produced is essentially the same, and the performance is surprisingly comparable.
This approach is a promising one for improving the speed and usability of C++ code in the browser. Now that we have this expertise, we can more easily share it with the consortium and use it to facilitate the development of the RAPID-API.

**Technology used**

- Maximilian audio DSP library to asm.js via emscripten

**Next steps**

- Generalization to other C++ libraries
PiPo Plugin in JUCE

The PiPo format, developed by IRCAM, has been identified as a candidate to develop data processing plugins. Precisely, PiPo is a simple plugin API for modules processing streams of multi-dimensional data such as: audio, audio descriptors, or gesture and motion data. We implemented the PiPo plugin in JUCE and made a standalone application in order to demonstrate the validity of this approach in an environment such as JUCE.

Contexts and Use Cases

- Implementing a Processing Plugin format in various programming environments

Goals/Questions

- Testing the PiPo plugin format in JUCE
- Cross-platform PiPo plugin format and JSON description
- Cross-platform JSON description of parameters

Description

The prototype is a standalone application made with Juce. The application receives as input an OSC data streams and output another OSC data streams, using configurable IP addresses and ports. As illustrated in Figure 5, the data is processed using a PiPo chain of processing plugin, which is set through a GUI. The parameters can also be edited, stored or retrieved. The data can be visualized through time profile or text table (based on the IMTR editor).

The interest of such application is to process data with the exact same plugins in any other programming environment that implements the PiPo host. So far, we have PiPo host in Max7 and openFrameworks.
Figure 5: Screenshot of the JuceOSC-Pipo application, showing a basic example with a median filter

**Technology used**

- PiPo (IRCAM)
- MUBU and IMTREditor (IRCAM)
- JUCE (ROLI)

**Next steps**

- Further integration of PiPo plugin imported from Maximilian (GS) and possibly Essentia (UPF).
- Generalization of this approach to other environments.
- Specification of the PiPo format for the RAPID-MIX API.
Machine learning software add-ons for OpenSignals (r)evolution

Currently OpenSignals (r)evolution supports a series of software add-ons for statistical and spectral analysis of multiple biosignal modalities, namely Electrocardiography (ECG), Electromyography (EMG), Electrodermal Activity (EDA), and Respiration data. This prototype aims at the development of additional software add-ons for that incorporate machine learning techniques existing within the consortium for raw biosignal data analysis and reporting.

Contexts and Use Cases
Data exploration, machine learning, biosignal processing, feature extraction, gesture recognition, automated annotation.

Goals/Questions
- Integrate a suite of useful tools for experimentation by interaction designers on a unified platform
- Provide users with a way to annotate recorded data in an assisted way (e.g. detect a gesture or motion pattern within the raw data)
- Facilitate data exploration and system design through automated event detection and classification

Description
OpenSignals (r)evolution has been primarily designed for real-time data recording and visualization of previously recorded data. Data analysis software add-ons have been created to facilitate exploration of raw data by users less experienced in biosignal analysis, however, current add-ons only provide statistical, spectral, and in some cases nonlinear analysis methods (see example on Fig. 6 & Fig. 7 below). Incorporating machine learning tools existing within the consortium will enable users to augment their experience when dealing with biosignals, by providing useful automation and non-supervised data analysis features. We foresee the implementation of automation and assisted data annotation tasks while performing real-time data acquisition; for example by providing training data (e.g. examples of certain gestures, motion, or other actions to be detected), OpenSignals (r)evolution could incorporate interactive machine learning as an aid to detect relevant patterns during data acquisition and automatically annotations to the recorded data. Another potential use of machine learning within OpenSignals (r)evolution includes data exploration, in particular concerning group analysis, by providing a means of detecting common patterns (or differences) in data recorded within a population or multiple data recording sessions of the same individual when performing a certain gesture or expressive interaction activity.
Figure 6: Electrocardiography (ECG) data analysis add-on for OpenSignals.

Figure 7: Statistical, spectral and nonlinear features extracted from ECG data.
Technology used

- OpenSignals (r)evolution (PLUX)
- XMM interactive machine learning library (IRCAM)
- Wekinator (GS)
- (optional) RepoVizz (UPF)

Next steps

- Create proof-of-concept add-on with machine learning algorithms for gesture recognition
- Integrate XMM, Wekinator, and/or other libraries into OpenSignals
- Compile support documentation, usage examples and tutorials
**BITalino UART to OSC bridge and WiFi transmission**

This prototype targets the creation of a set of hardware accessories for BITalino to enable the control and real-time data streaming using the OSC protocol over WiFi. For improved performance, BITalino uses a non-standard communication protocol, which makes its integration within key platforms used in the creative industries realm (e.g. MaxMSP, PureData, JUCE, Wekinator, etc.) cumbersome. Since OSC is currently a widely used protocol, this prototype will constitute an important resource to facilitate the use of multimodal biosignal data in prototyping of interactive expressive technologies.

**Contexts and Use Cases**
Multimodal expressive interaction, design of new interfaces for musical expression, interfacing data acquisition hardware with software platforms, OSC data streaming devices.

**Goals/Questions**
- Interface BITalino with OSC-compliant software
- Create an abstraction layer from the proprietary protocol used on BITalino
- Enables users to access hardware-produced data through the more ubiquitous WiFi transmission channel

**Description**
The BITalino (r)evolution MCU is optimized to receive commands and stream raw data over the UART serial port using a proprietary protocol. To improve the ease-of-use and overall experience when prototyping, Bluetooth 2.0 and Bluetooth Low Energy (BLE) interfaces have been created, together with a set of programming APIs to serve as a more user-friendly abstraction layer to the proprietary protocol. Nevertheless, handling of raw data streams produced by hardware devices in real-time involves specific procedures that constitute a barrier for less experienced users (even through the APIs). As such, this prototype consists of creating a set of hardware tools based on the r-IOT board by IRCAM (Fig. 8), that can complement the range of available interfaces currently available for BITalino (r)evolution by enabling control and acquisition of data from the BITalino (r)evolution MCU and relay the data stream over WiFi using the OSC protocol. WiFi is currently a nearly ubiquitous communication technology widely supported both by computers and mobile devices alike, while the OSC protocol is natively supported by the most used platforms for music, audiovisual, and interaction design (e.g. MaxMSP, PureData, JUCE, Wekinator, etc.).

![Image of r-IOT board](image-url)

**Figure 8: r-IOT board that may serve as a base for this prototype.**
Technology used

- BITalino (r)evolution (PLUX)
- r-IOT (IRCAM)
- (optional) OpenSignals (r)evolution (PLUX)

Next steps

- Implement a firmware for the r-IOT board capable of converting BITalino streams to OSC
- Create a shield or r-IOT redesign to easily interface BITalino (r)evolution with the r-IOT board
- Sharing of usage examples and tutorials
Hardware development platform + IDE

This prototype is focused on providing a system for facilitating rapid prototyping of bespoke music interaction devices using a small prototyping board with sensors, machine learning software, audio DSP tools, and perhaps some e-textile example material. The long term goal is to allow users to create software on a desktop using a very basic IDE, then deploy to a small board, similar in process to the Arduino but focussed on high quality, real-time interaction, audio and machine learning processing.

Contexts and Use Cases
Interactive musical experiences, user-tailored interaction, multimodal expressive interaction, design of new interfaces for creative expression.

Goals/Questions
- Create a programmable hardware base with a user-friendly IDE to enable standalone applications
- Implement libraries focused on high quality, real-time interaction, audio and machine learning processing
- Facilitate the development of bespoke music interaction devices

Description
BITalino (r)evolution is optimized for real-time data streaming using a proprietary protocol. Although BITalino (r)evolution is compatible with popular platforms such as the Arduino and Raspberry Pi, these platforms either have performance limitations for more complex operations or a bulky size when found with better processing units. The goal of this prototype is then to develop a user-programmable system capable of providing a good compromise between diversity of sensors for prototyping, computational power, and compact form factor, for which the r-IOT board can provide a good starting point (Fig 8). The r-IOT is powered by a Texas Instrument 32-bit ARM Cortex chipset, which is compatible with TI’s code composer and Energia IDE (which is a port of the Arduino IDE for TI processors). As such, this enables the creation of a module to be used in the creation of standalone programmable devices.
Technology used

- BITalino (r)evolution (PLUX)
- r-IOT (IRCAM)
- (optional) JUCE (ROLI)

Next steps

- Implement a firmware for the r-IOT board capable of acquiring BITalino streams
- Create a shield or r-IOT redesign to easily interface BITalino (r)evolution with the r-IOT board
- Develop libraries to enable high quality, real-time interaction, audio and machine learning processing
BITbox

This prototype targets the creation of a toolbox that provides a feature extraction layer through which users can seamlessly derive useful information from the raw time series data (e.g. muscle activations, electrodermal activity events, heart rate, etc), further enhancing the overall user experience in new application domains, namely those pertaining creative industries.

Contexts and Use Cases

Biosignal acquisition, rapid prototyping of wearables, educational support, design of new interfaces for musical expression

Goals/Questions

- Low-cost and easy-to-use access to software for biosignal feature extraction
- Enabling anyone to easily use events and features derived from time series in their projects
- Providing a user-friendly interface to deal with raw biosignal data

Description

From the User-Centered Design actions and workshops developed to date within the RAPID-MIX project, we have learned that users can rapidly create hardware prototypes to acquire biosignals using BITalino (r)evolution and can easily record raw data using OpenSignals (r)evolution and the existing suite of APIs. However, they often have difficulty in extracting features from that data. This prototype aims at the creation of a Software Development Kit (SDK) / toolbox that can enable users to easily extract useful features from raw data; features include muscle activations from Electromyography (EMG) sensor data, galvanic skin response events from Electrodermal Activity (EDA) sensor data, heart rate from Electrocardiography (ECG) sensor data, among others. The goal is to provide users with little or no experience in the field of biosignals with much needed prototyping tools to deal with the raw data.
Figure 11: Example Python offline code for peak detection and heart rate calculation from raw ECG data, in which dictionaries are used to make function input/output parameters more intuitive and human readable.

```python
def process_ecg(signal, sampling_rate):
    peaks = find_peaks(signal)
    heart_rate = calculate_heart_rate(peaks, sampling_rate)
    return heart_rate
```

Figure 12: Output of the code snippet listed above.

**Technology used**

- BITalino (r)evolution (PLUX)
- OpenSignals (r)evolution (PLUX)
- (optional) RepoVizz (UPF)
- (optional) WiFi to OSC bridge (IRCAM)

**Next steps**

- Implement elementary feature extraction algorithms
- Create proof-of-concept “BITbox” code base
- Support the OpenSignals (r)evolution file format and RepoVizz access
- Sharing of usage examples and tutorials
3 CONCLUSION

In conclusion, the current development of the early RAPID-MIX Prototypes addresses the expected results as stated in the DoW by providing both novel hardware prototypes and software prototypes implementing state-of-the-art interaction analysis techniques. As already noted in D3.1, there are obvious overlaps between some of these prototypes, which we did not try to avoid at this stage. On the contrary, our approach was to let emerge common goals/questions that could arise from different technologies or use cases. This convergence of questions will seed the foundation of the development of the RAPID-MIX API.

The next steps concern the refinement of the current prototypes, their evaluation and incorporation in MIX products. These first set of prototypes will feed a new series of UCD activities.