TOBI iC

Definition, implementation and scenarios

Michele Tavella

michele.tavella@epfl.ch

October 23, 2012

Contents

1 Introduction

The TOBI iC is designed by TOBI's WP5 and WP8 members, while its implementation is mantained by EPFL. iC defines the way classifiers in the hBCI transmit their outputs to other modules. The core component of iC is the "iC message". An iC message is the atomic unit used by the hBCI to transmit classifier outputs. An iC message can encode the outputs of multiple classifiers, regressors etc. As today, iC is available as a C++ library (Section 4.3) and as a MEX interface (Section 4.4). Under this perspective it is safe to assume that only data with a high information content but with a reduced transmission rate have to be handled by this interface.

1.1 Disclaimer

The goal of this document is to make users familiar with hBCI designs including iC. The document is written in an easy format that should be familiar to most of the BCI community, thus not only addressing the hBCI-ninjas within the TOBI consortium or the code-monkeys in your lab. This document is shipped with the iC library libtobiic and it has to be considered an evolving text that will be continuously updated. Although it contains some references and examples based on libtobiic and mextobiic, this document does not replace the Doxygen or Matlab documentation shipped with libtobiic and mextobiic respectively.

1.2 Acknowledgment

This work is supported by the European ICT Programme Project FP7-224631, TOBI: Tools for Brain-Computer Interaction. This document only reflects the authors' views and funding agencies are not liable for any use that may be made of the information contained herein.

2 iC and the hBCI

Before moving to the design details, it is important to highlight the scenarios in which iC is generally used. For this reason, two examples are presented. The first example shows how iC can be used in a context in which two different classifier outputs need to be fused to provide a unique probability output later sent to higher-level modules (Section 2.1). In a second example we describe how iC can be used to merge the output of two classifiers into a unique output (Section 2.2).

The difference between the two examples is subtle. In the first example, we fuse the outputs of two classifiers in a single one. The fused probabilities are then sent to higher level modules. In the second example, the output of two classifiers are merged together and transmitted in the same iC message as two separate probabilities.

2.1 EEG and EMG fusion

For this first example, we will use a BCI that allows users to control a device both using motor imagery and muscular activity. The hBCI acquires to sources of biological signals: EEG at 512Hz and EMG at 2048Hz. Spectral components

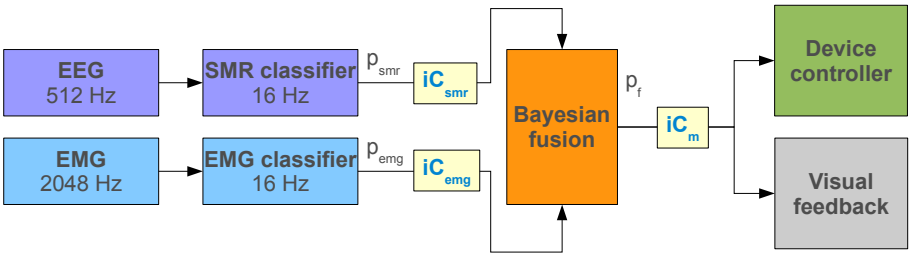

Figure 1: The SMR and EMG classifier output p_{smr} and p_{emg} probabilities, that are transmitted to the fusion block in two iC messages: iC_{smr} and iC_{emg}. The fusion emits a probability that p_f, transmitted via iC_f.

are then extracted from EEG at 16Hz and classified, while EMG activity is classified directly in the time domain and a probability output is provided at 16Hz.

EEG and EMG activity are classified by two different classifiers, as shown in Figure 1. The two probability outputs, p_{smr} and p_{emg}, are sent in two different iC messages (iC_{smr} and iC_{emg}) to the fusion module. The fusion module outputs a single iC message (iC_f) containing the fused probability $p_f = f(p_{smr}, p_{emg})$.

Two points need to be highlighted. First, the classifier output is generally much slower than the rate biological signals are acquired. Secondly, the output of a classifier can reach a larger set of modules, for which disparity needs to be taken in consideration. The latter claims motivated the use of TCP/IP communication and XML format for transmitting and encoding the messages (Section 4).

2.2 EEG and ErrP fusion

In the previous example we used the case of EEG and EMG fusion to showcase a typical example of iC use within the hBCI design. Before we were fusing two probabilities in one. From an iC message perspective, the fusion module receives two distinct messages, extracts the probabilities, fuse them together and outputs a message with the result of the fusion.

Another possibility might require the fusion to receive multiple iC messages and combine them together, as in the case of the EEG/ErrPs BCI shown in this example.

Similarly as before, the fusion receives two probabilities p_{smr} and p_{errp}. The two probabilities are received as two iC messages, iC_{smr} and iC_{errp} respectively. This time the fusion module extracts the content of the two input messages and merges is in a single message, iC_m, containing both received probabilities (p_{smr} and p_{errp}). The internal structure of iC_m is displayed in Figure 3.

3 Structure of an iC message

It should now be clear to the reader that iC messages are used to have classifiers communicate with higher level modules. It should also be clear that iC messages

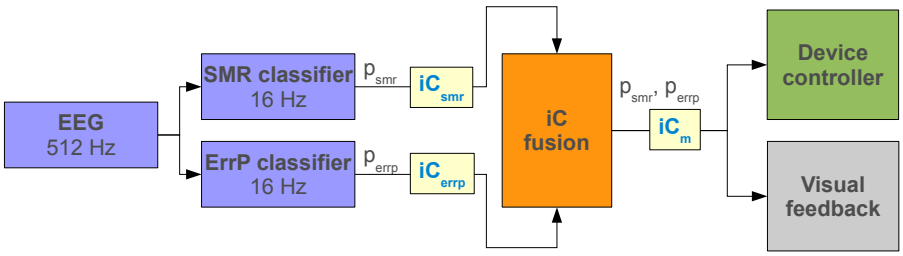

Figure 2: The SMR and ErrP classifier output p_{smr} and p_{errp} probabilities, that are transmitted to the fusion block in two iC messages: iC_{smr} and iC_{emg}. The fusion emits a probability that p_m, transmitted via iC_m.

constitute the atomic communication unit of TOBI iC. Finally, in the two examples we showed how different fusion modules can manipulate the content of the iC messages in very different ways.

What has not been described yet is the internal structure of an iC message, but specially the fact that iC can be used directly within your libraries and programs to describe classifiers, class labels and class values. In fact, iC is not only a way to encode/decode messages to be streamed in the hBCI pipeline. Its internal auto-configurable structures make iC the perfect solution to handle whatever is related to statistics and classification within any BCI design. Furthermore, thanks to the TOBI standards, it makes it possible to design plug-and-play hBCI modules (This is dedicated to WP8 people, but we need to define the standard somehow).

Figure 3 depicts the internal structure of the iC message iC_m from the example of Section 2. The diagram provides an ideal schematic for the topics introduced in this Session. For this reason, we encourage the reader to refer to it throughout the development of the following paragraphs.

3.1 iC messages

An iC message has to be considered as a set of iC classifiers and two attributes: a frame number and a iC version tag.

When the acquisition feeds in the hBCI pipe a data frame (i.e. EEG, EMG, etc.), it sets its frame number. Frame numbers are used withing the hBCI design to make sure data in different pipelines are aligned with each other. Not only: frame numbers become really handy to debug the whole loop. The two last claims become stringent requirements for distributed designs, in which different modules run on different processes or even machines.

Under this perspective, when a classifier receives some RAW data and outputs a probability, it must also set the frame number in the iC message so to convey the acquisition timing information to the higher level modules.

The version tag contains the specific version of TOBI iC. In this way, different modules that communicate by means of iC, can know if they have all the capabilities required to decode a received iC message.

As stated few lines ago, apart from the frame number and the version tag, an iC message has one or more classifiers, described in the next Section.

3.2 iC classifiers

An iC classifier is, by definition, a set of iC classes. So, if you want, the iC message is like an onion, where each layer stores some kind of information about a classifier output. More in depth the layer is, more low-level the information stored wherein will be.

Each iC message stores multiple classifiers using an hash table. An hash table is a data structure in which a key (i.e. "red", "green", "blue") is associated to a value (i.e. "FF0000", "00FF00", "0000FF"). In iC, the key of the hash table is the classifier name, and the value is an iC classifier.

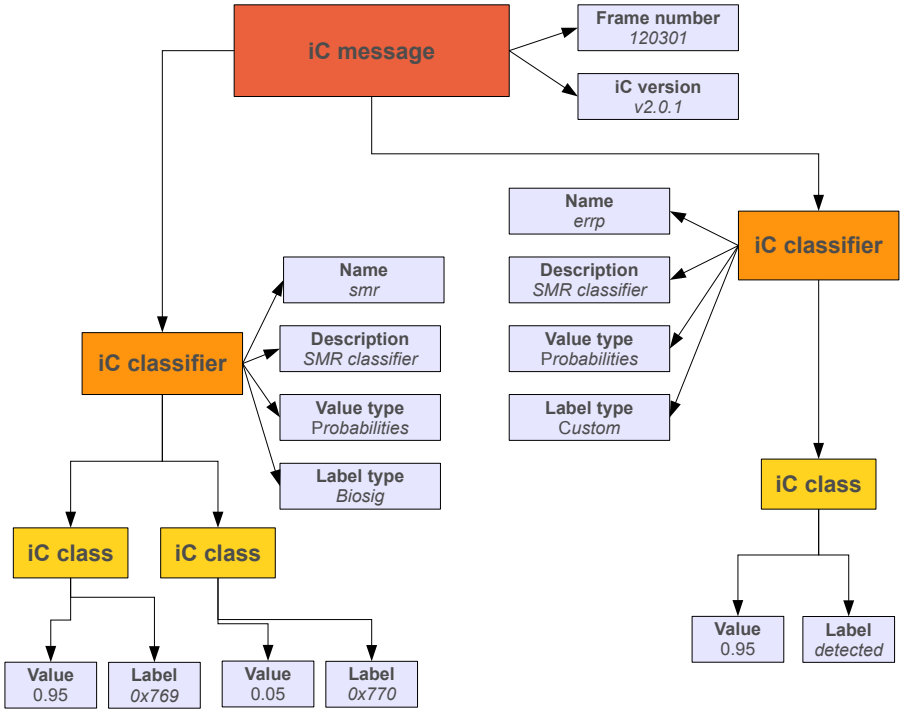

Figure 3: Internal structure of the iC message iC_m from Section 2. The light blue boxes contain the attributes for iC message, iC classifiers and iC classes.

An iC classifier has four distinct attributes:

1. **Name**: the name of the classifier (i.e. "smr", "emg" or "errp"). This attribute is mandatory and acts as a primary key for the hash table. Within an iC message, the name of an iC classifier must be unique.

2. **Description**: the description of the classifier (i.e. "SMR classifier", "EMG classifier" or "ErrP classifier"). This attribute is not mandatory.

3. **Value type**: the value type attribute describes what kind of output is emitted by the classifier. Possible value types are probabilities or regression coefficients.

4. **Label type**: the label type attribute encodes the type of the class labels. For example, an "smr" classifier might classify right versus left hand or tongue versus both feet motor imagery. This four kinds of motor imagery are labeled differently. The label type defines which kind of label it will be used for the different kinds of classes. Possible values are custom labels (i.e. "right_hand_mi", "left_hand_mi") or Biosig labels (i.e. "0x769", "0x770").

3.3 iC classes

An iC class is the basic unit of an iC classifier. It stores a class value and a class label. For example, the "smr" classifier encodes values as probabilities and uses Biosig labels for two classes: right hand and left hand motor imagery. The class values could be, for example, 0.05 and 0.95, while the class labels would be "0x769" and "0x770".

4 Implementation

TOBI iC is implemented in C++ and uses GNU Autoconf and Automake to configure and build the library. The library is platform independent and can be easily compiled under windows using MinGW + MSys. The University of Glasgow maintains the Python bindings, while QualiLife maintains the .Net version. The only external dependency of libtobiic is RapidXML.

A MEX interface for Mathworks Matlab has been written using MWrap. To build the MEX interface under Microsoft Windows, it is necessary to use MinGW + MSys together with GNUMex and Mwrap. A precompiled version for Microsoft Windows 32bit is available on the TOBI SVN.

4.1 Communication

As today iC is not shipped with a transport layer (i.e. TCP/IP, UDP) because different partners within the TOBI consortium use already different networking stacks. If you are not expert in coding and you want to evaluate iC, we strongly recommend to start with the MEX interface and eventually using jtcp or judp for sending data on the network.

4.2 Serialization

Up to now we always said that iC messages are exchanged between different modules. Still, we never addressed specifically how this messages are encoded and decoded for the transmission. Similarly to what has been said in Section 4.1 for the communication, different groups might require different solutions for encoding/decoding iC messages. For this reason the library provides a template class named *ICSerializer* on top of which is possible to implement any serialization method.

As today the consortium agreed on keeping the complexity of high-level communication as low as possible, thus choosing XML (using RapidXML as a back-end) for the serialization and de-serialization of the iC messages. Readers interested in this topic will find all the necessary informations in Section 5.

Some examples of serialized iC messages are available in Section .

4.3 libtobiic

The C++ library is distributed with a Doxygen API documentation and several examples. It can be downloaded from here: `http://`.

4.4 mextobiic

The MEX interface is distributed with the standard Mathworks Matlab documentation and several examples. It can be downloaded from here: `http://`.

4.5 Limitations of mextobiic

Although the Matlab interface provides all the means to manipulate the hierarchy of the iC structures, it does not allow the user to perform complex automated operations to combine and merge messages together. This is evident in Section 5, where fusion is handled manually. As today it is still unclear to what extent this functionality will be implemented in the core library and exposed in the MEX interface. Still, the libtobiic API is ready for all possible message manipulations, and in C++ it is trivial to perform such kind of operations. Under a usability perspective, the maintainers will add soon some extra functionality to support the automated merging and splitting of iC messages.

5 Examples

The examples in the following Sections implement and comment, line by line, what has been described in Section 2.1 and 2.2. The examples here provided are coded in Matlab.

5.1 EEG and EMG fusion

The code for the first example is included directly in this document. Still, the reader should be aware that the example is distributed with mextobiic. The comments are provided as in the example.

To help the reader, these are the main functional blocks within the provided example:

- Lines 19-35: configuration of the SMR iC message iC_{smr}, classifier and classes.
- Lines 38-54: configuration of the EMG iC message iC_{emg}, classifier and classes.
- Lines 57-85: configuration of the Fusion iC message iC_f, classifier and classes both for the inputs and for the output.
- Lines 91-172: simulated hBCI loop (from the classifier output to the fusion output only).
  - Lines 92-98: setting of frame number
  - Lines 100-110: setting of the class values
  - Lines 112-139: serialization and de-serialization of the iC messages
  - Lines 141-153: fusion of the probabilities emitted by the classifiers
- Lines 174-185: de-allocation of the iC objects

The XML messages, resulting from the serialization of the different iC messages, can be found in Section 5.1.2, 5.1.1 and 5.1.3.

```
%   Copyright (C) 2010 Michele Tavella <michele.tavella@epfl.ch>
%
%   This file is part of the mexctobiic wrapper
%
%   The libndf library is free software: you can redistribute it and/or
%   modify it under the terms of the version 3 of the GNU General Public
%   License as published by the Free Software Foundation.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You should have received a copy of the GNU General Public License
%   along with this program.  If not, see <http://www.gnu.org/licenses/>.

clear all;

% Create a structure for the SMR classifier, with
% - an iC message
% - an iC serializer
% - a buffer to store the serialized messages
SMR.message = icmessage_new();
SMR.serializer = icserializerrapid_new(SMR.message);
SMR.buffer = '';
% Add a classifier named 'smr' to SMR.message, using:
% - 'biosig' as label type
% - 'prob' as value type.
icmessage_addclassifier(SMR.message, ...
    'smr', 'SMR classifier', ...
    icmessage_getvaluetype('prob'), icmessage_getlabeltype('biosig'));
% Add two classes to the 'smr' classifier with labels '0x769' and '0x770' and
% values 0.50
icmessage_addclass(SMR.message, 'smr', '0x769', 0.50);
icmessage_addclass(SMR.message, 'smr', '0x770', 0.50);


% As before, create a structure for the EMG classifier, with
% - an iC message
% - an iC serializer
% - a buffer to store the serialized messages
EMG.message = icmessage_new();
EMG.serializer = icserializerrapid_new(EMG.message);
EMG.buffer = '';
% Add a classifier named 'emg' to EMG.message, using:
% - 'biosig' as label type
% - 'prob' as value type.
icmessage_addclassifier(EMG.message, ...
    'emg', 'EMG classifier', ...
    icmessage_getvaluetype('prob'), icmessage_getlabeltype('biosig'));
% Add two classes to the 'emg' classifier with labels '0x300' and '0x301' and
% values 0.50
icmessage_addclass(EMG.message, 'emg', '0x300', 0.50);
icmessage_addclass(EMG.message, 'emg', '0x301', 0.50);


% Create a structure for the Fusion.
% We need an SMR and EMG inputs, and the fusion output, and for each one we
% allocate:
% - an iC message
% - an iC serializer
% - a buffer to store the serialized messages
%
% SMR input
Fusion.SMR.message = icmessage_new();
Fusion.SMR.serializer = icserializerrapid_new(Fusion.SMR.message);
Fusion.SMR.buffer = '';
% EMG input
Fusion.EMG.message = icmessage_new();
Fusion.EMG.serializer = icserializerrapid_new(Fusion.EMG.message);
Fusion.EMG.buffer = '';
% Fusion output
Fusion.F.message = icmessage_new();
Fusion.F.serializer = icserializerrapid_new(Fusion.F.message);
```

```
75  Fusion.F.buffer = '';
76
77  % Add a classifier named 'smr+emg' to Fusion.F.message, using:
78  % - 'custom' as label type
79  % - 'prob' as value type.
80  icmessage_addclassifier(Fusion.F.message, ...
81      'smr+emg', 'SMR+EMG fusion', ...
82      icmessage_getvaluetype('prob'), icmessage_getlabeltype('custom'));
83  % Configure the classes, this time using 'right' and 'left' as custom labels
84  icmessage_addclass(Fusion.F.message, 'smr+emg', 'right', 0.50);
85  icmessage_addclass(Fusion.F.message, 'smr+emg', 'left',  0.50);
86
87  % Later on we will need a figure to plot the fused probabilities
88  figure(1);
89
90  % We will simulate 100 iterations of the BCI loop
91  for i = 1:100
92      % Simulate we are running at 16 Hz
93      pause(1.00/16);
94
95      % Imagine the SMR and EMG classifier are running in sync.
96      % They would have the same frame number
97      icmessage_setfidx(SMR.message, i);
98      icmessage_setfidx(EMG.message, i);
99
100     % Generate a random SMR probability and set the values for the classes
101     % of the 'smr' classifier
102     p = rand();
103     icmessage_setvalue(SMR.message, 'smr', '0x769', p);
104     icmessage_setvalue(SMR.message, 'smr', '0x770', 1-p);
105
106     % Again, generate a random EMG probability and set the values for the
107     % classes of the 'emg' classifier
108     p = rand();
109     icmessage_setvalue(EMG.message, 'emg', '0x300', p);
110     icmessage_setvalue(EMG.message, 'emg', '0x301', 1-p);
111
112     % Ok, we need a bit of imagination here.
113     % Imagine the SMR and EMG classifers serialize their iC messages and send
114     % them via network to the fusion.
115     %
116     % This is how you serialize the messages, writing the XML messages in the
117     % buffers.
118     SMR.buffer = icmessage_serialize(SMR.serializer);
119     EMG.buffer = icmessage_serialize(EMG.serializer);
120     % When the Fusion module receives the serialized messages, it would
121     % store it in its internal buffers
122     Fusion.SMR.buffer = SMR.buffer;
123     Fusion.EMG.buffer = EMG.buffer;
124     % And then it would deserialize the buffers to the appropriate iC messages
125     icmessage_deserialize(Fusion.SMR.serializer, Fusion.SMR.buffer);
126     icmessage_deserialize(Fusion.EMG.serializer, Fusion.EMG.buffer);
127     % When the Fusion receives messages, it must be sure they represent the
128     % output of two classifiers that classified the very same frame.
129     % So, we first extract the frame number and then we check if it is the
130     % same.
131     % Keep in mind that in this example the frame number will always be the
132     % same.
133     fnEEG = icmessage_getfidx(Fusion.SMR.message);
134     fnEMG = icmessage_getfidx(Fusion.EMG.message);
135     if(fnEEG == fnEMG)
136         icmessage_setfidx(Fusion.F.message, fnEEG);
137     else
138         disp('Error!');
139     end
140
141     % At this point the Fusion fuses the 'smr' '0x769' class value with the
142     % 'emg' '0x300' one, and puts the resulting value in 'smr+emg' 'right'.
143     % Sometimes when I refer to this example I mention
144     % "Bayesian fusion", but here I simply do averaging.
145     icmessage_setvalue(Fusion.F.message, 'smr+emg', 'right', ...
146         0.5 * icmessage_getvalue(Fusion.SMR.message, 'smr', '0x769') + ...
147         0.5 * icmessage_getvalue(Fusion.EMG.message, 'emg', '0x300'));
148     % The same is done for the other class. Keep in mind this is just an
```

```
149     % example to show you how to use the methods, it woud be a bit redundant
150     % to perform this operation in real life.
151     icmessage_setvalue(Fusion.F.message, 'smr+emg', 'left', ...
152         0.5 * icmessage_getvalue(Fusion.SMR.message, 'smr', '0x770') + ...
153         0.5 * icmessage_getvalue(Fusion.EMG.message, 'emg', '0x301'));
154
155     % At this point we make some people happy by plotting the fused
156     % probability output.
157     bar([icmessage_getvalue(Fusion.F.message, 'smr+emg', 'right') ...
158          icmessage_getvalue(Fusion.F.message, 'smr+emg', 'left')]);
159     ylim([0 1]);
160     xlim([0 3]);
161     set(gca, 'XTickLabel', {'Right', 'Left'});
162     title(['Probabilities after fusion — Frame ' ...
163         num2str(icmessage_getfidx(Fusion.F.message))]);
164
165     % Imagine we have to send the fused probabilities to another module.
166     % You need first to serialize the message and place the XML output in the
167     % appropriate buffer.
168     Fusion.F.buffer = icmessage_serialize(Fusion.F.serializer);
169     % Then, we display the buffer.
170     disp(Fusion.F.buffer);
171 end
172
173 % Let's deallocate all the serializers...
174 icserializerrapid_delete(SMR.serializer);
175 icserializerrapid_delete(EMG.serializer);
176 icserializerrapid_delete(Fusion.SMR.serializer);
177 icserializerrapid_delete(Fusion.EMG.serializer);
178 icserializerrapid_delete(Fusion.F.serializer);
179 % ... and all the messages
180 icmessage_delete(SMR.message);
181 icmessage_delete(EMG.message);
182 icmessage_delete(Fusion.SMR.message);
183 icmessage_delete(Fusion.EMG.message);
184 icmessage_delete(Fusion.F.message);
```

5.1.1 XML message for the SMR classifier

The serialized version of an iC message for the SMR classifier is here provided.

```
1 <tobiic version="0.1.1.0" frame="1">
2     <classifier name="smr" description="SMR classifier" vtype="prob" ...
          ltype="biosig">
3         <class label="0x769">0.743132</class>
4         <class label="0x770">0.256868</class>
5     </classifier>
6 </tobiic>
```

5.1.2 XML message for the EMG classifier

The serialized version of an iC message for the EMG classifier is here provided.

```
1 <tobiic version="0.1.1.0" frame="1">
2     <classifier name="emg" description="EMG classifier" vtype="prob" ...
          ltype="biosig">
3         <class label="0x300">0.392227</class>
4         <class label="0x301">0.607773</class>
5     </classifier>
6 </tobiic>
```

5.1.3 XML message for the Fusion module

The serialized version of an iC message for the Fusion module is here provided.

```
<tobiic version="0.1.1.0" frame="11">
    <classifier name="smr+emg" description="SMR+EMG fusion" vtype="prob" ...
         ltype="custom">
        <class label="left">0.654274</class>
        <class label="right">0.345726</class>
    </classifier>
</tobiic>
```

5.2 EEG and ErrP fusion

This example follows what already seen in Section 5.1. The only difference is the merging of the classifier outputs and the configuration of the iC message emitted by the fusion (Lines 141-148 and 55-80 respectively). Furthermore, the reader might be interested in comparing the XML output in Section 5.2.1 with the one in Section **??**.

```
%   Copyright (C) 2010 Michele Tavella <michele.tavella@epfl.ch>
%
%   This file is part of the mexctobiic wrapper
%
%   The libndf library is free software: you can redistribute it and/or
%   modify it under the terms of the version 3 of the GNU General Public
%   License as published by the Free Software Foundation.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You should have received a copy of the GNU General Public License
%   along with this program.  If not, see <http://www.gnu.org/licenses/>.

clear all;

% Create a structure for the SMR classifier, with
% - an iC message
% - an iC serializer
% - a buffer to store the serialized messages
SMR.message = icmessage_new();
SMR.serializer = icserializerrapid_new(SMR.message);
SMR.buffer = '';
% Add a classifier named 'smr' to SMR.message, using:
% - 'biosig' as label type
% - 'prob' as value type.
icmessage_addclassifier(SMR.message, ...
    'smr', 'SMR classifier', ...
    icmessage_getvaluetype('prob'), icmessage_getlabeltype('biosig'));
% Add two classes to the 'smr' classifier with labels '0x769' and '0x770' and
% values 0.50
icmessage_addclass(SMR.message, 'smr', '0x769', 0.50);
icmessage_addclass(SMR.message, 'smr', '0x770', 0.50);


% As before, create a structure for the ErrP classifier, with
% - an iC message
% - an iC serializer
% - a buffer to store the serialized messages
ErrP.message = icmessage_new();
ErrP.serializer = icserializerrapid_new(ErrP.message);
ErrP.buffer = '';
% Add a classifier named 'errp' to ErrP.message, using:
% - 'biosig' as label type
% - 'prob' as value type.
icmessage_addclassifier(ErrP.message, ...
    'errp', 'ErrP classifier', ...
    icmessage_getvaluetype('prob'), icmessage_getlabeltype('custom'));
% Add two classes to the 'errp' classifier with label 'detected' and value 0.00
icmessage_addclass(ErrP.message, 'errp', 'detected', 0.00);


% Create a structure for the Fusion.
```

```
% We need an SMR and ErrP inputs, and the fusion output, and for each one we
% allocate:
% — an iC message
% — an iC serializer
% — a buffer to store the serialized messages
%
% SMR input
Fusion.SMR.message = icmessage_new();
Fusion.SMR.serializer = icserializerrapid_new(Fusion.SMR.message);
Fusion.SMR.buffer = '';
% ErrP input
Fusion.ErrP.message = icmessage_new();
Fusion.ErrP.serializer = icserializerrapid_new(Fusion.ErrP.message);
Fusion.ErrP.buffer = '';
% Fusion output
Fusion.M.message = icmessage_new();
Fusion.M.serializer = icserializerrapid_new(Fusion.M.message);
Fusion.M.buffer = '';

% Add a classifier named 'smr' and one 'errp' to Fusion.M.message, using
% the correct label and value types.
icmessage_addclassifier(Fusion.M.message, ...
    'smr', 'SMR classifier', ...
    icmessage_getvaluetype('prob'), icmessage_getlabeltype('biosig'));
icmessage_addclassifier(Fusion.M.message, ...
    'errp', 'ErrP classifier', ...
    icmessage_getvaluetype('prob'), icmessage_getlabeltype('custom'));
% Configure the classes
icmessage_addclass(Fusion.M.message, 'smr', '0x769', 0.50);
icmessage_addclass(Fusion.M.message, 'smr', '0x770', 0.50);
icmessage_addclass(Fusion.M.message, 'errp', 'detected', 0.00);

% Later on we will need a figure to plot the fused probabilities
figure(1);

% We will simulate 100 iterations of the BCI loop
for i = 1:100
    % Simulate we are running at 16 Hz
    pause(1.00/16);

    % Imagine the SMR and ErrP classifier are running in sync.
    % They would have the same frame number
    icmessage_setfidx(SMR.message, i);
    icmessage_setfidx(ErrP.message, i);

    % Generate a random SMR probability and set the values for the classes
    % of the 'smr' classifier
    p = rand();
    icmessage_setvalue(SMR.message, 'smr', '0x769', p);
    icmessage_setvalue(SMR.message, 'smr', '0x770', 1-p);

    % Again, generate a random ErrP probability and set the values for the
    % classes of the 'errp' classifier
    p = rand();
    icmessage_setvalue(ErrP.message, 'errp', 'detected', p);

    % Ok, we need a bit of imagination here.
    % Imagine the SMR and ErrP classifers serialize their iC messages and send
    % them via network to the fusion.
    %
    % This is how you serialize the messages, writing the XML messages in the
    % buffers.
    SMR.buffer = icmessage_serialize(SMR.serializer);
    ErrP.buffer = icmessage_serialize(ErrP.serializer);
    % When the Fusion module receives the serialized messages, it would
    % store it in its internal buffers
    Fusion.SMR.buffer = SMR.buffer;
    Fusion.ErrP.buffer = ErrP.buffer;
    % And then it would deserialize the buffers to the appropriate iC messages
    icmessage_deserialize(Fusion.SMR.serializer, Fusion.SMR.buffer);
    icmessage_deserialize(Fusion.ErrP.serializer, Fusion.ErrP.buffer);
    % When the Fusion receives messages, it must be sure they represent the
    % output of two classifiers that classified the very same frame.
    % So, we first extract the frame number and then we check if it is the
```

```
130     % same.
131     % Keep in mind that in this example the frame number will always be the
132     % same.
133     fnEEG = icmessage_getfidx(Fusion.SMR.message);
134     fnErrP = icmessage_getfidx(Fusion.ErrP.message);
135     if(fnEEG == fnErrP)
136         icmessage_setfidx(Fusion.M.message, fnEEG);
137     else
138         disp('Error!');
139     end
140
141     % At this point the Fusion merges the 'smr' and the 'errp'  messages
142     % in a single one.
143     icmessage_setvalue(Fusion.M.message, 'smr', '0x769', ...
144         icmessage_getvalue(Fusion.SMR.message, 'smr', '0x769'));
145     icmessage_setvalue(Fusion.M.message, 'smr', '0x770', ...
146         icmessage_getvalue(Fusion.SMR.message, 'smr', '0x770'));
147     icmessage_setvalue(Fusion.M.message, 'errp', 'detected', ...
148         icmessage_getvalue(Fusion.ErrP.message, 'errp', 'detected'));
149
150     % At this point we make some people happy by plotting the fused
151     % probability output.
152     bar([icmessage_getvalue(Fusion.M.message, 'smr', '0x769') ...
153          icmessage_getvalue(Fusion.M.message, 'smr', '0x770') ...
154          icmessage_getvalue(Fusion.M.message, 'errp', 'detected')]);
155     ylim([0 1]);
156     xlim([0 4]);
157     set(gca, 'XTickLabel', {'0x769', '0x770', 'detected'});
158     title(['Probabilities after merging — Frame ' ...
159         num2str(icmessage_getfidx(Fusion.M.message))]);
160
161     % Imagine we have to send the merged probabilities to another module.
162     % You need first to serialize the message and place the XML output in the
163     % appropriate buffer.
164     Fusion.M.buffer = icmessage_serialize(Fusion.M.serializer);
165     % Then, we display the buffer.
166     disp(Fusion.M.buffer);
167 end
168
169 % Let's deallocate all the serializers...
170 icserializerrapid_delete(SMR.serializer);
171 icserializerrapid_delete(ErrP.serializer);
172 icserializerrapid_delete(Fusion.SMR.serializer);
173 icserializerrapid_delete(Fusion.ErrP.serializer);
174 icserializerrapid_delete(Fusion.M.serializer);
175 % ... and all the messages
176 icmessage_delete(SMR.message);
177 icmessage_delete(ErrP.message);
178 icmessage_delete(Fusion.SMR.message);
179 icmessage_delete(Fusion.ErrP.message);
180 icmessage_delete(Fusion.M.message);
```

5.2.1 XML message for the Fusion module

The serialized version of an iC message for the Fusion module is here provided.

```
1 <tobiic version="0.1.1.0" frame="100">
2     <classifier name="errp" description="ErrP classifier" vtype="prob" ...
          ltype="custom">
3         <class label="detected">0.953813</class>
4     </classifier>
5     <classifier name="smr" description="SMR classifier" vtype="prob" ...
          ltype="biosig">
6         <class label="0x769">0.520190</class>
7         <class label="0x770">0.479810</class>
8     </classifier>
9 </tobiic>
```