International Journal of Bioelectromagnetism *www.ijbem.org*
Vol. 9, No. 3, pp. 172- 184, 2007

Optimization of Brain Computer Interface systems by means of XML and BF++ Toys

Lucia Rita Quitadamo[a], Maria Grazia Marciani[ab], Luigi Bianchi[abc]

[a] *Department of Neuroscience, "Tor Vergata" University,Viale Oxford 81, 00133 Rome, Italy*
[b] *Fondazione Santa Lucia IRCCS - Neurofisiopatologia, Via Ardeatina 306, 00179 Rome, Italy*
[c] *Centro di Biomedicina Spaziale, "Tor Vergata" University, Via della Ricerca Scientifica snc, 00133 Rome, Italy*

Correspondence: Luigi Bianchi, Department of Neuroscience, "Tor Vergata" University, Viale Oxford 81, 00133 Rome, Italy.
E-mail: luigi.bianchi@uniroma2.it, phone: +39-06-20904728, fax: +39-06-20902106.

Abstract. The optimization of Brain Computer Interface systems is of great importance for the purpose of making them more usable and adjustable according to the needs of the end users. However, when evaluating their performances, it is evident the lack of a standard metric and of a common way to describe or represent the behavior, characteristics and data relative to the functional modules that compose them. The need of sharing data virtually everywhere and of making them usable by every researcher has inspired the work described in this paper: a set of tools, the BF++ Toys, which simulate and optimize the behavior of BCI systems, were implemented. They made wide use of the XML technology for describing and documenting all the main entities involved in BCI. Finally it will be shown how BF++ Toys and XML represent a versatile and reliable mean for the purpose of optimizing BCI systems.

Keywords: BCI; BF++ Toys; XML; Optimization; File Formats.

1. Introduction

Many people in the world are affected by severe neuromuscular impairments, which make them lose the control on their muscular voluntary activities thus isolating them from the environment.

Brain Computer Interface (BCI) systems try to facilitate for these people the communication of their intents by translating some electrophysiological signals into commands towards external peripherals without making use of the classical pathways of nerves and muscles [Wolpaw et al., 2002].

These systems are formed by several interconnected modules, which represent some entities pertaining different disciplines such as psychology, engineering, rehabilitation, informatics, neurophysiology, computer science, etc. For this reason, there are scientific works which deal with just a subset of these modules. In some cases they are focused on the implementation of new classifiers [Neuper et al., 2005], some others on the definition of new experimental protocols [Yoo et al., 2004], some others on the development of systems that can be used out of the various research laboratories [Vaughan et al., 2006].

Unfortunately, most of the research groups involved with BCI "do" it in different ways according to not only the systems and platforms used but also to data file formats. Moreover, these last are relative not only to the electrophysiological processed data (e.g. EEG, ERP) but also to configuration settings (e.g. feedback rule, application behavior) and performance data (e.g. confusion matrices). This fact represents an obstacle in the data and tools exchange among different laboratories, as many data conversion tools need to be implemented and maintained. Moreover, it is practically impossible to simulate the behavior of a system and to optimize it by assembling modules from different groups as a unique way for describing their characteristics. A typical situation is a group which is involved in the implementation of spellers that would like to simulate the performances of systems built by assembling their developed modules with other ones available from the literature.

A recent study [Bianchi et al., 2007] has demonstrated that under certain conditions it is possible to reliably predict the behavior of a system built by assembling different modules if a well-defined and relatively simple description of them is available. This will allow the optimization of many BCI systems without the need of really building them: a way to break the interdependence of the modules and thus of the various research groups.

To be accessible to a wide audience, however, it is necessary to provide an efficient and extensible file format platform, suitable to fulfill the needs of virtually any scientist and to provide tools that are able to handle the stored information.

Here we describe file formats based on the XML technology for storing some entities frequently encountered among the BCI community as well as some free tools (BF++ Toys) that use them and that were developed to optimize the performances of complete systems and to document them. This will allow the dissemination of resources and the improvement of the performances of the systems based on the proposed platform. Finally, as new BCI modules will be described, every researcher could immediately determine if and how it could improve the performances of his own systems.

The entire described file formats specifications and tools can be freely downloaded at www.brainterface.com.

2. Material and Methods

A BCI system is usually formed by two main functional blocks (Fig. 1): the Transducer (TR) and the Control Interface (CI) [Mason et al., 2005].

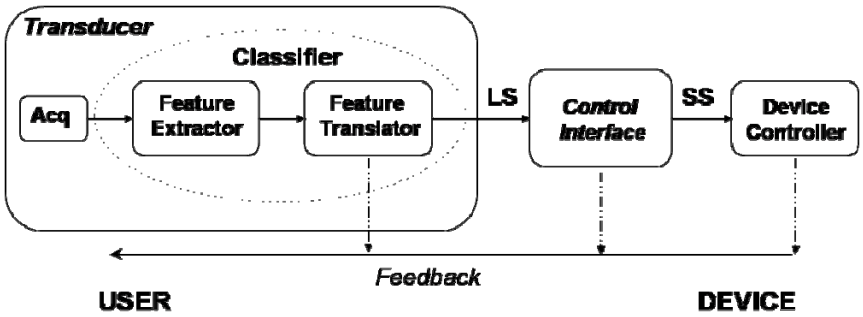

Figure 1. *Simplified functional model of a BCI system.*

The TR on its turn is composed by the acquisition stage, which deals with the electrophysiological signals, and by the Classifier, formed by the Feature Extractor and the Feature Translator, whose task is to extract the features of interest from the signals and to translate them into a logical symbol (LS) which belongs to the classifier logical alphabet (LA). The LS, in general, has no semantic meaning but is just a mapping with some subject's performance (e.g. the imagination of a cube rotation [Millan and Mouriño, 2003], the reaching of a cursor on a target [Wolpaw and McFarland, 2004], the selection of a row in P300 virtual keyboard task [Sellers et al., 2006], etc…). The CI encodes sequences of LSs and turns them into semantic symbols (SS) that belong to a semantic alphabet (SA) that can be used to drive an external peripheral (e.g. virtual keyboards, robots, neuroprostheses, etc…). As an example, if we have a logical alphabet of four symbols α, β, γ, δ the CI can implement an encoding for a virtual keyboard application that associates the sequence ααα to the semantic symbol A, then ααβ to the semantic symbol B and so on (usually one of the LSs is reserved for the UNDO key). In this way 27 semantic symbols can be encoded with 3 LSs-long sequences so that the 26 characters of the English alphabet plus the space one can be mapped. Because each SS has its own probability to occur (e.g. the 'E' is more frequent than the 'Z' in the English language) and because the encoding strategy determines how many times a logical symbol appears for each SS also LSs have in general different probabilities of occurrence. In case of a classifier that exploits different performances according to the current LS (e.g. a mental task is classified more successfully than another one), then the adoption of different encodings will result in different LSs probability of occurrence and then in different overall system performances. It is evident that in general the worst-classified LS should occur less frequently to reduce the error-rate. This can be done by choosing the proper encoding.

The aforementioned modules, TRs and CIs, need some metrics which describe their characteristics and how they match. However classical metrics (Mutual Information, Entropy, bit-rate) do not provide the necessary information for the evaluation of the performances of a whole system as they do not consider how TRs and CIs adapt themselves and how errors should be corrected. This is why a new metric has been proposed [Bianchi et al., 2007] for the evaluation and optimization of the performances of BCI systems which combines the information about the entities previously described (LA, SA, Encoders) with those derived from a new component, the Extended Confusion Matrix (ECM).

The ECM is used to evaluate the performances of a TR by storing the LSs asked for the classification and those actually classified, including the cases in which the TR abstains from decisions [Pietraszek, 2005]. An ideal ECM (no errors and no abstentions) should be a diagonal matrix.

An example of ECMs is given in Fig. 2: in both the matrices (a, b) the rows correspond to the asked symbols, the columns to the classified ones and the last column indicates the number of abstentions (Abst). Anyway they differ for the distribution of errors: in fact, in the first ECM the best-classified symbol is the first one, while in the second one the error distribution is inverted and the best classified symbol is the last one.

	α	β	γ	δ	Abst
α	393	4	4	4	45
β	8	381	8	8	45
γ	16	16	357	16	45
δ	32	32	32	309	45

a)

	α	β	γ	δ	Abst
α	309	32	32	32	45
β	16	357	16	16	45
γ	8	8	381	8	45
δ	4	4	4	393	45

b)

Figure 2*. Example of two different ECMs characterized by the same mean error and abstention rates (both 10%). In a) the best-classified symbol is the first one and the errors increase with the row index with a power of 2 function, while in b) the best-classified symbol is the last one and the errors are distributed in the reverse order.*

This example shows what has been previously mentioned about the optimization of BCI systems based on the choice of the proper encoding: in the first case the choice of an encoding which maximizes the occurrence of the 'α' symbol represents a simple optimization strategy, while in the second case the occurrence of the same 'α' symbol should be minimized. In other words the performances of a system can be optimized by choosing the proper encoding for a given TR according to the systems requirements and the final applications.

By associating a cost to each of the errors and the abstentions of an ECM and by evaluating their influence on the information rate of the system (which is reduced with respect to the ideal case), an Expected mean Selection Cost ($\overline{ESC}$) that represents the number of classifications required to generate a correct logical symbol can be computed.

Finally the Efficiency of a system can be defined as:

$$Eff_{SYS} = \frac{1}{\overline{L_{CW}} \cdot \overline{ESC}} \tag{1}$$

where $\overline{L_{CW}}$ is the mean codeword length; in general this means that a system characterized by a lot of errors and abstentions and thus by great costs, will produce a lower efficiency and a longer output encoded sequence; hence the necessity of creating encoders which try to minimize the $\overline{ESC}$.

As it has been demonstrated [Bianchi et al., 2007], it is possible to predict a collection of CIs, given a collection of TRs. Also the description of how errors should be handled can be predicted. In particular, what will be the performances of all the system that is possible to build by combining all the TRs with all the CIs can be clearly used to optimize BCI systems because it will be sufficient to choose the most efficient combination of TRs and CIs.

All the entities used for the description of TRs and CIs need a file format easily understandable and handable both for storage necessity and for easily managing the simulations for the computation of the performances indicators.

In addition, as everyone could be interested in the evaluation of BCI systems and in the comparison of the obtained results, it is of primary importance to find a unique way for representing data, a fact that can be also fundamental for the optimization of the entire system.

An excellent mean for doing that is XML (eXtensible Markup Language). We stored data (ECMs, encoders, LAs, SAs, etc.) in XML files in order to allow fast and simple interchanging of them, even on Web, and at the same time a robust and customizable way of formatting them.

2.1. Introduction to XML

XML is a set of technologies for creating personal markup languages, which makes use of tags. It describes data and distributes them in a format, which is independent from the platform. This independence comes from the fact that XML does not use a specific language: in fact, XML tags are not predefined, so that everyone can write his own personal tags [Box et al., 2000].

XML is very similar to HTML (Hyper Text Markup Language) but it is not its substitute as its aim is different: XML was designed for storing and exchanging data, while HTML was created for showing data in a format easily readable even if hardly adjustable during time.

One of the main advantages of XML is that it is extensible: one can create new tags and new document structures and add or remove elements without affecting the overall structure of the document and thus without breaking the backward compatibility with existing tools. This means that one can add new tags to an XML file and that all the software that is unaware of them will continue to work. In this way, it is very comfortable sharing data across different laboratories as every researcher is free to extend them with the results of an analysis without disturbing the activity of the other ones that can receive the new data and continue to work with the previously developed tools.

Another great advantage of XML is its *portability*: the user defines tags and attributes with no need of special libraries for reading them. XML files do not need any compilation and special software for being interpreted, even if this is allowed and sometimes preferred. This technology is also cross-platform and compatible with a lot of browsers, such as the latest releases of Internet Explorer, Firefox, Opera, etc.... This makes XML the ideal choice for data exchange between different software and hardware platforms and the basic technology for the communication world.

The essential characteristic of XML is that data are independent. The content of a file is kept separate from its presentation, so that one can store the content in an XML file only once and then extract and visualize it in the desired format (according to the final format, there is an appropriate XML technology that allows its generation, called XSLT).

As XML lets the information be transmitted precisely and smartly, it has easily become the universal method for communication on the Web and beyond.

2.2. Stroring BCI data: XML basics

We used XML for storing LAs, SAs, ECMs and Encoders generated by means of the BF++ Toys. In Listing 1 the XML code for Fig. 2a is reported.

```
<?xml version="1.0" encoding="UTF-16"?>
<DOC>
    <Type Name="Subject_01" SubType="Session_01"/>
        <ECM>
            <Row>
                <In>α</In>
                <C1>393</C1>
                <C2>4</C2>
                <C3>4</C3>
                <C4>4</C4>
                <Abstentions>45</Abstentions>
            </Row>
            <Row>
                <In>β</In>
                <C1>8</C1>
                <C2>381</C2>
                <C3>8</C3>
                <C4>8</C4>
                <Abstentions>45</Abstentions>
            </Row>
            <Row>
                <In>γ</In>
                <C1>16</C1>
                <C2>16</C2>
                <C3>357</C3>
                <C4>16</C4>
                <Abstentions>45</Abstentions>
            </Row>
            <Row>
                <In>δ</In>
                <C1>32</C1>
                <C2>32</C2>
                <C3>32</C3>
                <C4>309</C4>
                <Abstentions>45</Abstentions>
            </Row>
        </ECM>
</DOC>
```

Listing 1

In the first row of the code the xml version has to be declared; it is not part of the document closely, but it is necessary for the definition of the standard used. Then the list of the elements with their relative attributes succeeds. Usually elements have an opening tag and a closing tag which encase the element content [Castro, 2000]; for example, the expression *"<name>content</name>"* represents the

non-empty element “name” with content “content”. Alternatively, if an element contains attributes only and no content it is said *empty* and can be represented with a single opening and closing tag, such as <name attribute1= ”attribute1” attribute2= ”attribute2”/>. Examples of this last format will be present in Listing 2.

The tree structure in Listing 1 makes it easy to recognize the elements of the file, their attributes and their correlations: there is a unique root element, <DOC>, which contains all the other elements; then two child elements, <Type> which contains some explicative information (Name and Subtype attributes) and ECM which contains the data of interest. The rows of the ECM are stored in the <Row> element which is replicated four times and contains a child element <In>. This child element gives us the information about the LS used for the coding, four elements <Ci>, i=1, 2, 3, 4, which contain the elements of the matrix ($ECM[i, j]$) and an element <Abstentions> which represents the number of the undetermined cases of the classifier. From the tree structure, it can be seen how easy is to add new tags without losing the compatibility with the entire document.

An XML document is considered *well-formed* if it is syntactically correct with regard to the rules previously mentioned.

2.3. Definition and validation of BCI data: XML Schemas

The necessity of having a robust representation of the data is satisfied with XML Schema, which is an XML-based language that gives the opportunity of controlling the content of elements and the coherence of the document by means of schemas. Schemas are not necessary but they can be used to validate a file: for example, you can control that data are expressed in the correct form (string, decimal, float, integer, date, etc.) and that the elements of the document succeed in the correct order. There are many programs for validating an XML file; in our research, Altova XMLSpy2006 was used.

We will now report the code of a schema that can be used for validating ECMs with up to 8 classes.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="DOC">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Type">
                    <xsd:complexType>
                        <xsd:attribute name="Name" type="xsd:string"/>
                        <xsd:attribute name="SubType" type="xsd:string"/>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="ECM">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Row" minOccurs="2" maxOccurs="8">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="In" type="xsd:string"/>
                                        <xsd:element name="C1" type="xsd:nonNegativeInteger"/>
                                        <xsd:element name="C2" type="xsd:nonNegativeInteger"/>
                                        <xsd:choice>
                                            <xsd:sequence>
                                                <xsd:element name="C3" type="xsd:nonNegativeInteger" minOccurs="0"/>
                                                <xsd:element name="C4" type="xsd:nonNegativeInteger" minOccurs="0"/>
                                                <xsd:element name="C5" type="xsd:nonNegativeInteger" minOccurs="0"/>
                                                <xsd:element name="C6" type="xsd:nonNegativeInteger" minOccurs="0"/>
                                                <xsd:element name="C7" type="xsd:nonNegativeInteger" minOccurs="0"/>
                                                <xsd:element name="C8" type="xsd:nonNegativeInteger" minOccurs="0"/>
                                            </xsd:sequence>
                                        </xsd:choice>
                                        <xsd:element name="Abstentions" type="xsd:nonNegativeInteger"/>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Listing 2

In the schema we have to individuate a type for all the elements. The elements DOC, Type, ECM and Row are indicated as complex types because they contain other elements and some attributes. The element Row contains a child element, named “In” (<xsd:element name="In" type="xsd:string"/>),

which indicates the LS and two elements (C1 and C2) which correspond to the minimal dimension of an ECM (a classifier has to distinguish between two classes at least).

In Fig. 3 the graphical view of the schema, generated with Altova XMLSpy 2006, is reported.

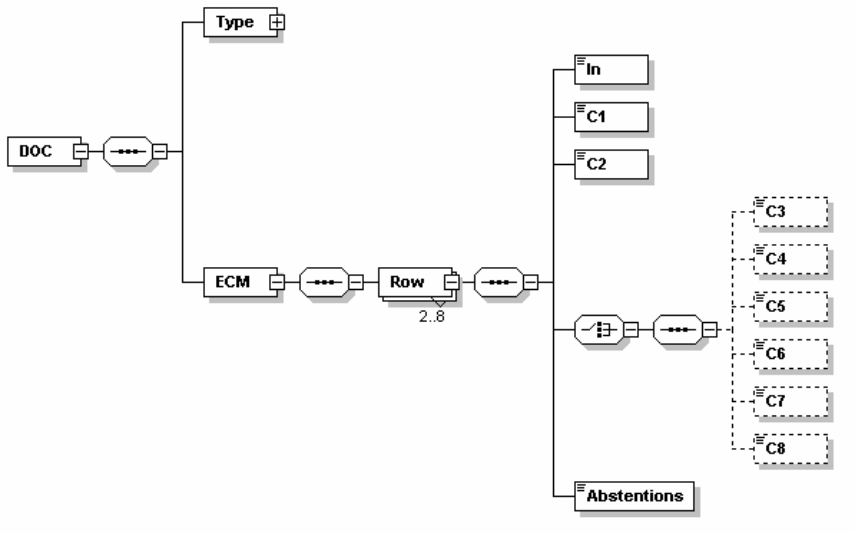

Figure 3. *Graphical representation of the schema relative to the ECM.*

It is important to stress the fact that XMLSpy does not have any previous knowledge about the entity “Extended Confusion Matrix" and its meaning; this is the reason that makes XML files self-documenting, that is, an XML file contains all the necessary information within itself. In case of some elements are added to the xml document, a new schema can be created for validating them.

If all the elements and attributes of the document respect the definition provided by a schema the XML file is said to be *valid* with respect to that schema.

2.4. Manipulating ECMs files:XSLT and XPath

XML contains two other technologies for interacting with XML files: XSLT (Extensible Style Language Transformation) and XPath. XSLT provides a flexible, powerful language for transforming XML documents into something else such as an HTML document, another XML document, a Portable Document Format (PDF) file, a Scalable Vector Graphics (SVG) file (see next paragraph), a Java code, a JPEG file, etc. In particular, you write an XSLT style sheet to define the rules for transforming an XML document and the XSLT processor does the work.

XPath, instead, is a routine that is used for describing the nodes of the tree structure by indicating their position in the XML document. With XPath you can also use functions and expressions for selecting a group of nodes and making mathematical operations on them.

An example of the capabilities of transformations has been shown in Fig. 2, where ECMs stored in an XML file have been converted in a tabular HTML format.

With some operations and with the use of XPath expressions – e.g. if you refer to the ECM file reported in Listing 1 and you want to select all the child elements of the first Row element, you must use the expression “DOC/ECM/Row[1]/*” - we can visualize on a browser all the data as matrices and even calculate some metrics.

2.5. Creating ECMs graphical representations: SVG

Scalable Vector Graphics uses the XML grammar for visualizing data in a graphical vectorial form. The source can be an XML file that, by means of an XSL transformation, could be transformed in an image; this image is not coded as a group of pixels (as in bitmap-raster images) but through pictures and curves combined with each other.

SVG can handle vectors, text and animations. Being constituted by simple text, SVG provides a great flexibility of access to the contents of the file, as text can be easily indexed, selected, searched and extracted.

SVG files have a great “scalability”, as the source file can be dimensioned and viewed in any scale; this means that the image resolution is not dependant from the device used. Also, SVG has a great compatibility with HTML (in fact it is integrated in Web pages) and, being an open standard, it can be used by everyone and, therefore, improved.

We used SVG for describing ECMs data: we built a pie chart, which represents the percentage of correct, undetermined and misclassified classifications.

Figure 4 represents the image we obtained applying the transformation to the ECMs previously reported in which you can see the chart and the legend for it.

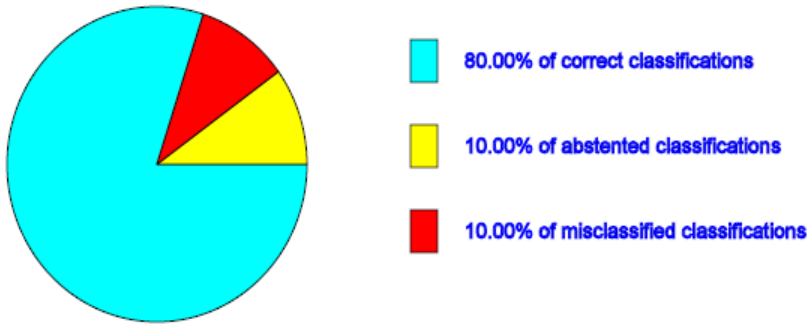

***Figure 4**. Synthetic graphical representation of an ECM in SVG format generated by means of an XSL transformation of an XML file representing an ECM.*

2.6. Creating new ECMs from a schema: XForms

XForms (XML forms) is an XML-based technology for the definition of forms in XML documents. Forms are a wide spread mean for communicating data to an application in Internet environment and they allow the description of fields where the users can insert data to be stored or sent to an application staying on a Web server. XForms has been developing with the aim of furnishing a more flexible and more advanced technology than HTML for the definition and handling of forms.

One of the main characteristics of XML, which totally transfers to XForms, is the separation between the content and the representation of data; so, by means of XForms one can define forms, which are independent from the application and the devices which they will be visualized on. Data inserted within a form will be transferred to the server in an XML format, using a structure based on form data model.

We used XForms for creating graphical interfaces constituted by fields in which one can insert data relative to a generic ECM. The source file was the schema we used for validating ECM XML file. We used XFormation 2.1 Software which is able to generate XHTML fragment (XHTML is an HTML standard written following XML strict rules) from a schema (it is logical that, having a schema which specifies the structure of an XML file and the data type which can be inserted, one can immediately create the fields of the forms). Then you can add models and styles to the document and, mainly, instances, which allow the definitions of the frame of the XML file which will contain the data inserted from the external interface. You can also give data a tabular view and create *reset* and *submit* buttons for clearing the fields or saving the data.

The file created from the schema is saved with .xhtml extension and given as input to XSMILES that is a JAVA-based XML browser, which runs on many embedded devices and which allows the visualization of the forms created and the validation of the data inserted by the user, by means of a schema.

XForms is the nth technology provided by XML that will lead to the publication of the files obtained with BF++ Toys.

2.7. BF++Toys

BF++ Toys are a set of software tools that are able to handle the information relative to the previously described entities and that can help to develop and optimize BCI systems. They have been developed in C++ and use classes from the BF++ Framework [Bianchi et al., 2003], such as those to define and handle Alphabets, Symbols, Encoders, Transducers, etc. and to read and write them in XML format. Their (open source) nature is to be extensible, so that every research laboratory can use them and contribute to their development. Actually, they are formed by four main applications: the ECM Generator, the Encoder Generator, the Simulator and the Optimizer. Here an overview of their main characteristics is reported, while a more accurate description is provided at http://www.brainterface.com.

The ECM Generator

Apart from the XForms based technology, which allows to manually fill and create an ECM, and apart from those automatically generated by means of the BF++ framework, the ECM Generator provides a way for simulating the behavior of various transducers with different error distributions. This is achieved by means of an extensible hierarchy of C++ classes, so that one can easily create his own errors and abstentions distribution laws. In Listing 3 all the code necessary to implement a generator responsible for the construction of an ideal (identity) ECM is reported. It is evident how easy it is to create a new generator: it is sufficient to populate and return a matrix object (mat_) whose size is determined by the size of the LA (m_LSA in the listing) and all of the BF++ Toys can load and save ECMs, compute metrics, perform simulations, etc…

```
class ConfMatGenIdeal : public ConfMatGenT
{
 public:
      ConfMatGenIdeal() : ConfMatGenT ("Ideal") {}

  protected:
  virtual EasyMat<MatType> DoGenerate(constLogicalAlphabet* m_LSA)      {
         EasyMat<MatType> mat_;//a matrix of doubles

         if (m_LSA)
            mat_.ReSize(m_LSA->size(), m_LSA->size() + 1);

         for (size_t r = 0; r < mat_.Nrows(); r++)
            mat_[r][r] = 1;

         return mat_;
      }
};
```

Listing 3

In Fig. 5 it is represented a screenshot of the Toy responsible to generate and visualize the ECM of Fig. 2a.

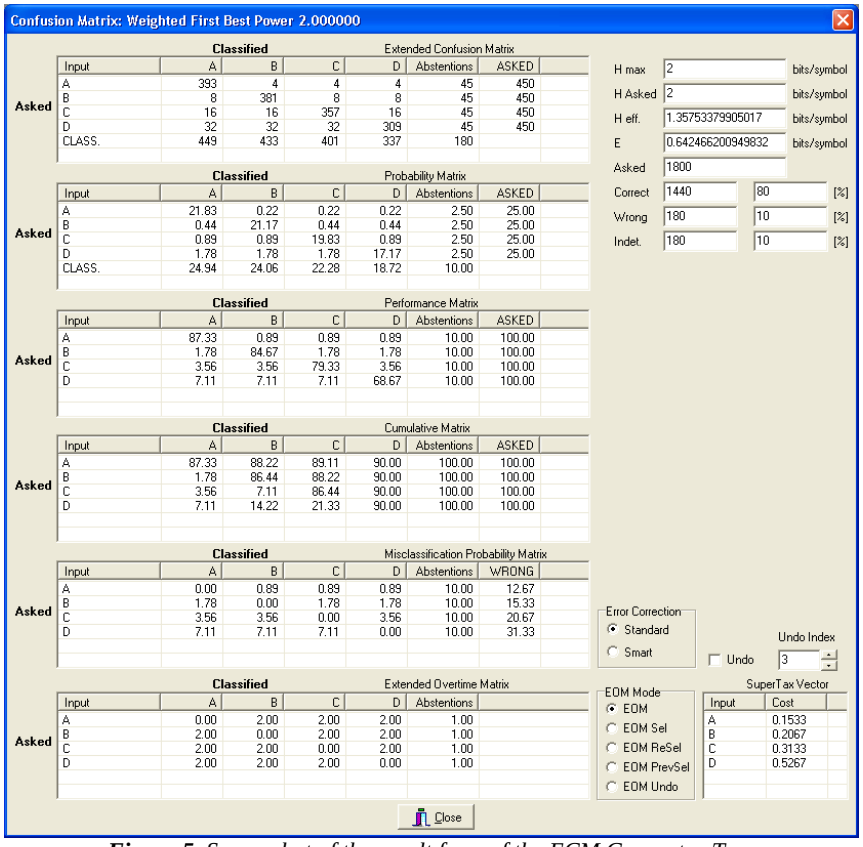

Figure 5. *Screenshot of the result form of the ECM Generator Toy.*

The following six matrices are represented from top to down:

1) the ECM, augmented by a row and a column: in the last row the number of times that a LS was classified (correctly or not) is represented, while the last column represents the number of times a LS was asked.
2) the Probability Matrix, in which each cell represents the corresponding ECM cells with the values expressed in percentage.
3) The Performance Matrix, in which the probability of a classification once a LS was asked is reported.
4) The Cumulative Matrix, used internally by BF++ Toys.
5) The Misclassification Probability Matrix, in which in each cell the probability of occurrence of an error or an abstention is reported.
6) The Extended Overtime Matrix, in which the number of selections necessary to correct a mistake is reported.

The Encoding Generator

The Encoding Generator is a Toy that creates an encoding from a LA and a SA. In a similar way, to the ECM Generator it is very easy to add new generation strategies to the BF++ Toys. Simple metric is computed and instantly one can see what the probability of occurrence of a logical symbol is. In Fig. 6 an encoding for the English alphabet is reported. For each semantic symbol (the 26 characters plus the space one) the probability of its occurrence (see the Prob. column in the left grid) is described. Then the LS Seq. column indicates the encoding for each of the SS and in the third column, the Code Length is reported that in this case is fixed and equal to 3. From this information, it is automatically computed the probability of occurrence of each LS and a pie chart is built to visually report this. With this encoding the α, β and γ LSs occur respectively with a probability of 29.770%, 29.387% and 40.843%.

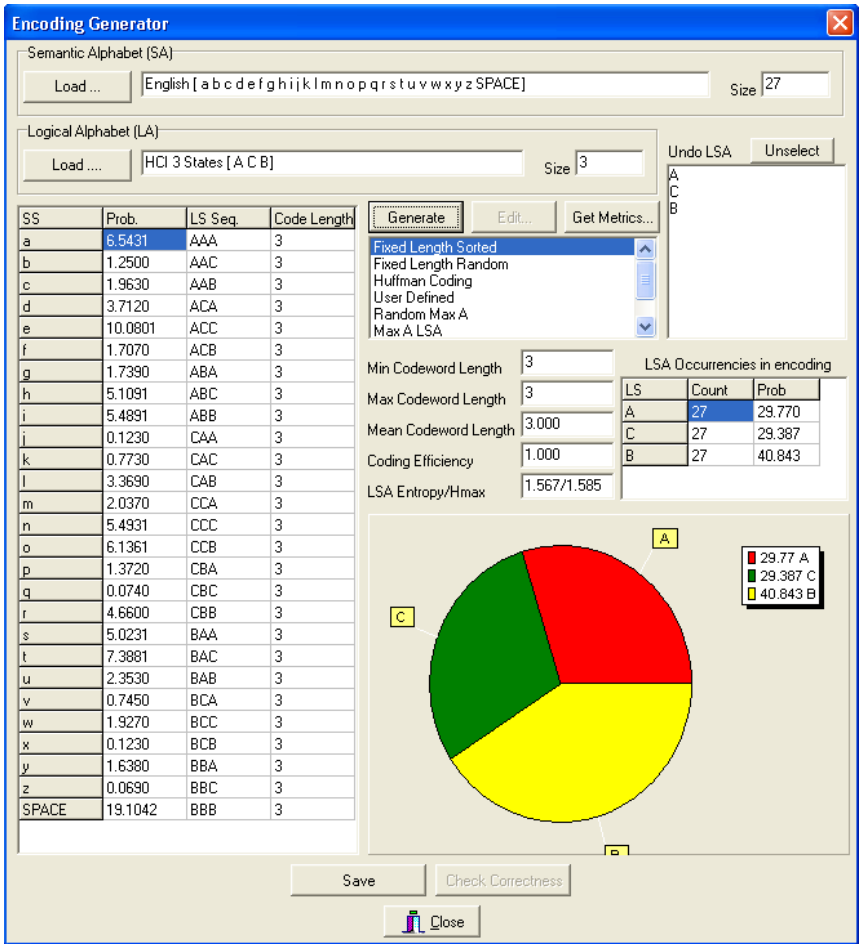

Figure 6. *Screenshot of the main form of the Encoding Generator Toy. See text for details.*

Simulators

Simulators are BF++ Toys that are responsible to generate realistic encoded LSs sequences in a copy spelling task given an encoding and an ECM. In Fig. 7 the simulation of the encoding of the

sentence “HELLO WORLD” (bottom left edit box) is reported: the encoded simulated sequence which also takes into account errors (symbol ‘δ’) and abstentions (symbol ‘?’) is reported in the bigger edit box on the right of the form. In the edit box below it the correct encoded sequence of LS is reported.

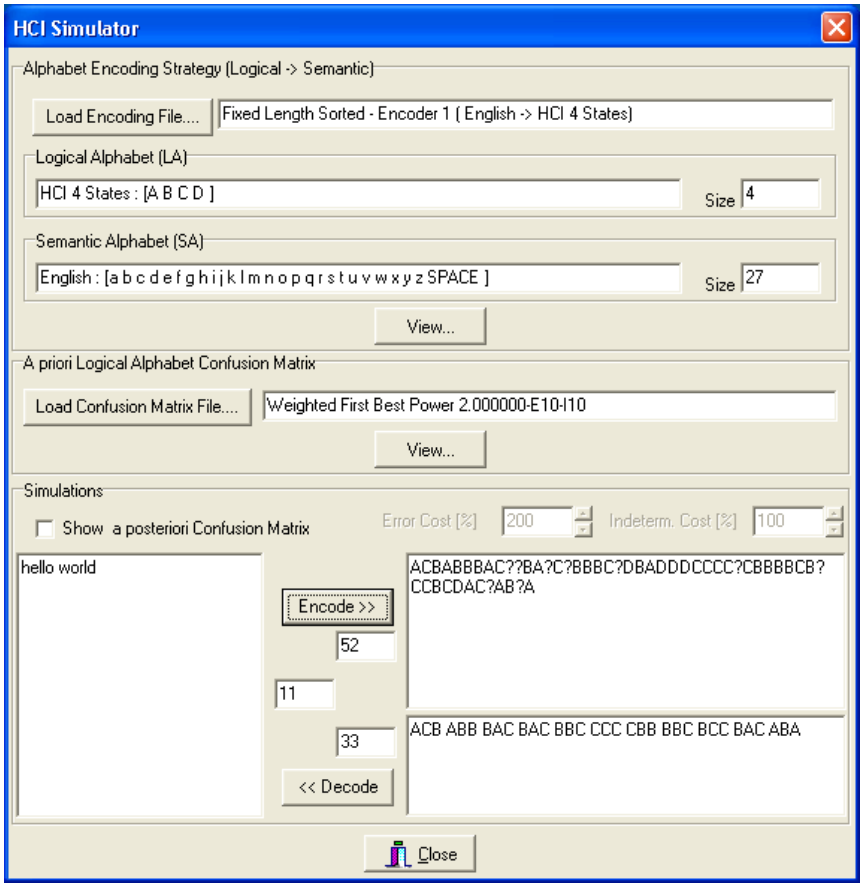

Figure 7. *Screenshot of the main form of one of the simulators. See text for details.*

Other simulators belonging to the BF++ Toys family perform more complex operations, such as encoding the same message with many different encoders or with many different ECMs to allow a comparison among them. External ASCII files can also be used as SS sequences to encode entire books or even collection of books.

Optimizers

BF++ Simulators are a reliable way to identify the best combination of TRs (described by ECMs) and CIs (described by encoders) in a virtual keyboard application. However, in order to have consistent results, it is necessary to encode long semantic sequences. This could require several hours of processing time on a Desktop PC if the number of encoders or ECMs to be tested is huge. A better solution is to adopt the metric described in [Bianchi et al., 2007], which requires just few seconds to determine which TR and/or CI to choose to build the best system. In a typical situation one can choose which encoder should be assembled to a TR to build the best system. The Optimizer Toy is devoted to this: it computes the efficiency of a system built by combining a TR with a set of encoders and by performing all the possible permutations among the LSs of the LA. This produces the list that appears in the bottom part of the form of Fig. 8: the various combinations indicate that the best encoder to associate to the ECM selected in the top grid of the form is the Encoder 7, first permutation, with the δ symbol associated to the Undo key. This system will require on average the selection of 3.614 LSs to generate a correct SS. In an ideal system, that is a system that makes no errors, just 3 LSs are necessary.

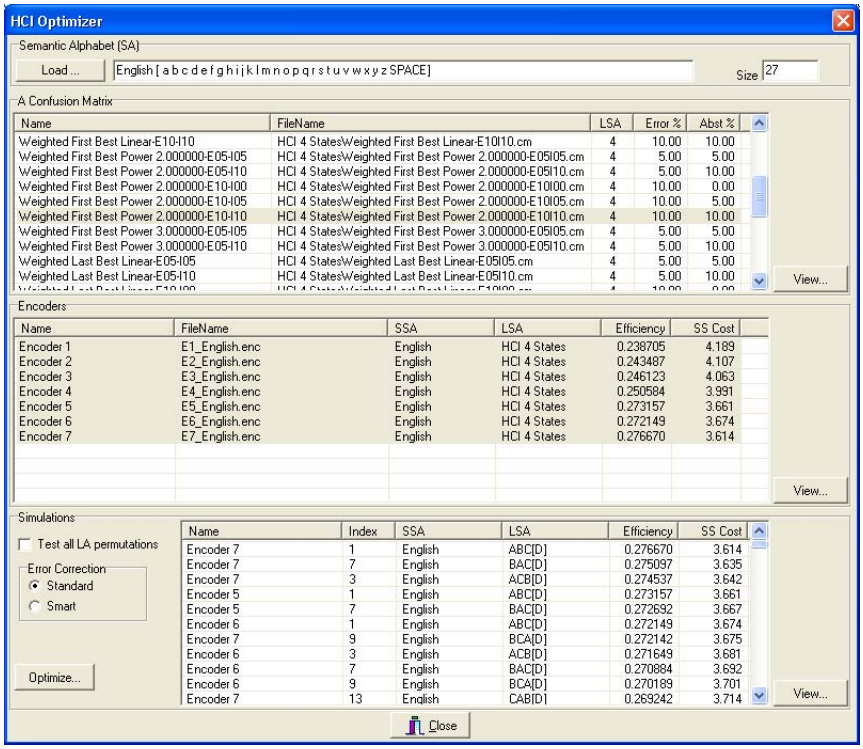

Figure 8. *Screenshot of the main form of one of the optimizers. See text for details.*

A more practical way to optimize a system is to start from the subject's performances, that is a real ECM and to find which encoder should be bound to it in order to obtain the best system. For this reason a new optimizing BF++ Toy (see Fig. 9) was developed which works in this way: once one has identified the alphabets and provided the ECM, then a huge number of encodings are randomly generated and the overall system metric is computed. Less than 1 ms is necessary to generate an encoder and to evaluate the system which uses it (on a PC equipped with a 2.4 GHz Pentium 4), so that about a hundred of thousands of different encoders and systems can be evaluated in less than two minutes. For example, for the ECM described in Fig. 9 we started with an encoder which required 4.698 LS selections to generate a correct SS, but we were able to reduce it to 4.02 in less than two minutes after using this optimizer, thus boosting the performance of a whole system by more than 15%. It is important to underline, however, that the improvement depends on the error distribution over the ECM: the more uniform it will be, the lower the improvement will also be.

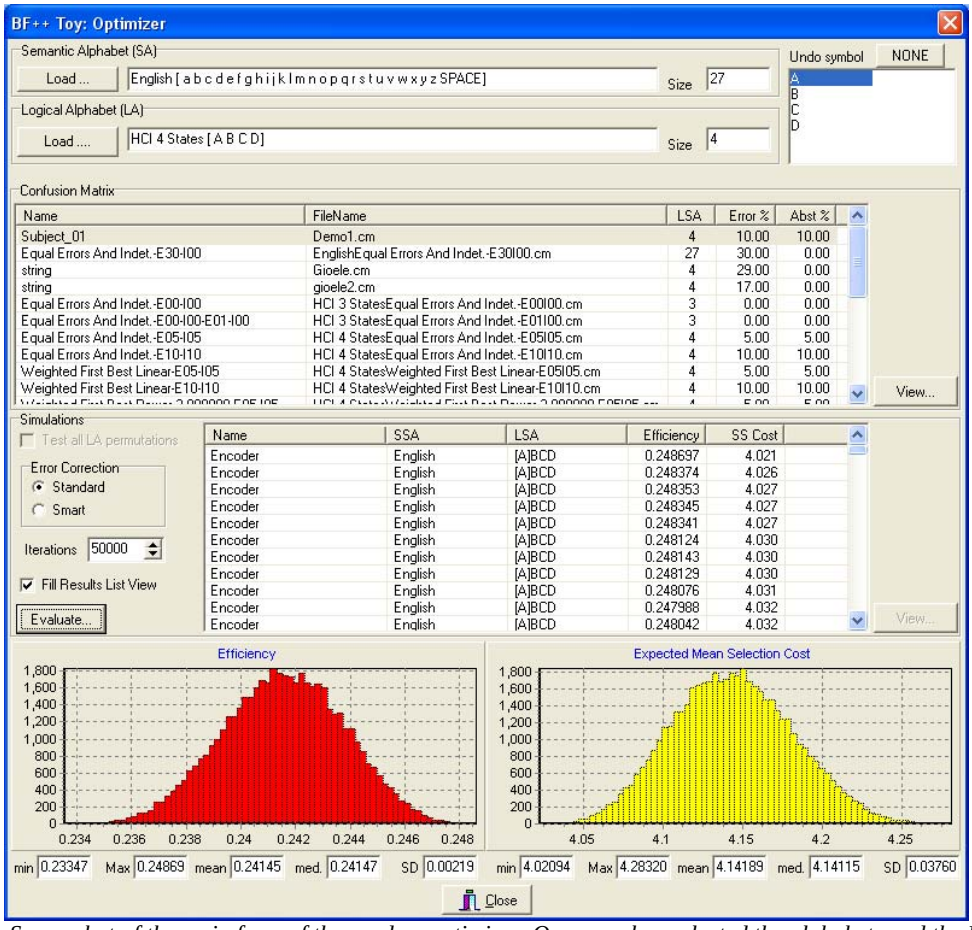

Figure 9. *Screenshot of the main form of the random optimizer. Once one has selected the alphabets and the ECM he can start to generate random encoders. On the yellow histogram one can see the expected mean selection cost distribution, that is the average number of LS selections necessary to generate a correct SS. The lower the value, the better the system. The red histogram, instead, represents the efficiency value, and in this case higher values represent better systems.*

3. Conclusions

The use of XML and BF++ Toys for optimizing BCI systems has great potentialities as they represent a quite innovative and strictly technical approach to the resolution of the major problems affecting these systems. The possibility of sharing tools, data, software and file formats and of evaluating and optimizing the performances of BCI systems by means of a reliable metric is vital for BCI research field as it involves a lot of groups which work with different aspects of the systems and is intended for a lot of people with different capabilities and necessities.

XML and BF++ Toys represent the ideal purpose for the improvement of these systems thanks to their great flexibility, easy understanding and their compatibility with the most common platforms. They can be used, shared and adapted by everyone according to their own personal requirements in order to make BCI research a more standard and spreadable reality.

In conclusion, XML and BF++ Toys allow the evaluation of the indicators needed for the characterization and optimization of the performances of BCI systems. Thus the comparison among different systems is important for the choice of the system which best fits the needs of the final user who usually is a person with minimal residual autonomy and who can communicate only with an interface which suits to him to the highest degree.

Acknowledgements

This work was partially supported by the DCMC (Disorders of Motor and Cardio-respiratory Control) Project of the Italian Space Agency and by the COST BM0601 Action “Neuromath”; it is also dedicated to the memory of Renato Grasso and Aleksandar Kostov.

References

Bianchi L, Babiloni F, Cincotti F, Salinari S, Marciani MG. Introducing BF++: a C++ framework for cognitive bio-feedback systems design. *Methods of Information in Medicine*, 42(1):104-110, 2003.

Bianchi L, Quitadamo LR, Garreffa G, Cardarilli GC, Marciani MG. Performances evaluation and optimization of Brain-Computer Interface systems in a copy spelling task. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 15(2): 207-216, 2007.

Box D, Skonnard A, Lam J. Essential XML. Beyond Markup. Addison-Wesley, Indianapolis, 2000.

Mason SG, Moore Jackson MM, Birch GE. A general framework for characterizing studies of Brain Interface technology. *Annals of Biomedical Engineering*, 33(11): 1653-1670, 2005.

Millàn J, Mouriño J. Asynchronous BCI and local neural classifier: an overview of the adaptive Brain Interface project. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2): 159-161, 2003.

Neuper C, Scherer R, Reiner M, Pfurtscheller G. Imagery of motor actions: differential effects of kinesthetic and visual-motor mode of imagery in single-trial EEG. *Cognitive Brain Research*, 25(3): 668–677, 2005.

Sellers EW, Krusienski DJ, McFarland DJ, Vaughan TM, Wolpaw JR. A P300 event-related potential brain-computer interface (BCI): the effects of matrix size and inter stimulus interval on performance. *Biological Psychology*, 73(3): 242–252, 2006.

Vaughan TM, McFarland DJ, Schalk G, Sarnacki WA, Krusienski DJ, Sellers EW, Wolpaw JR. The Wadsworth BCI research and development program: at home with BCI. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 14(2): 229-233, 2006.

Wolpaw JR, Birbaumer N, McFarland DJ, Pfurtscheller G, Vaughan TM. Brain-computer interfaces for communication and control. *Clinical Neurophysiology*, 113(6): 767-791, 2002.

Wolpaw JR, McFarland DJ. Control of a two-dimensional movement signal by a non invasive brain-computer interface in humans. *Proceedings of the National Academy of Science of the United States of America*, 101(51): 17849-17854, 2004.

Yoo SS, Fairneny T, Chen NK, Choo SE, Panych LP, Park H, Lee SY, Jolesz FA. Brain-Computer interface using fMRI: spatial navigation by thoughts. *Clinical Neuroscience*, 15(10): 1591-1595, 2004.