Conceptual Interactive Learning Tools Based on Computer Simulators

Damjan Zazula, Bogdan Viher, Dean Korošec, Enis Avdičauševič, Mitja Lenič, Božidar Potočnik
University of Maribor, Faculty of Electrical Engineering and Computer Science
Smetanova 17, 2000 Maribor, Slovenia
Phone: + 386 62 220 7480, Fax: + 386 62 211 178
E-mail: zazula@uni-mb.si

Keywords: modelling and simulation, conceptual learning, client-server, Java applets, electromyography

Edited by: Rudi Murn

Received: December 3, 1998 **Revised:** July 19, 1999 **Accepted:** August 18, 1999

An existent computer-based simulator may serve double purpose: simulating the basic phenomenon that was conceptually modelled by the simulator, and offering its inherent teaching characteristics at the same time. In this paper, we describe shortly the basic concepts of a simulator of electromyographic signals (EMGs), which was, first, written for a single-user environment in C++ and later ported to Java and interactive network environment. At this stage it became natural that the Java application was upgraded by the features of a teaching and learning tool. Such an integration was possible in a client-server approach providing the network users with a compact facility for generating the EMG components and signals, and learning about the electro-physiological phenomena in parallel.

1 Introduction

The world-wide accessibility of the Internet applications has recently opened a new prospective of the computer-supported processing and dissemination of information. Interaction has become a basic rule moving frontiers out of the local computer, out of the local community. At the same time, the philosophy of security and protection drew the major attention. New concepts have been developed, one of the most outstanding being the interactive applications in Java [6], and the corresponding tools and approaches, for example the client-server connections, three-tier architecture, and applets.

The simulator of electromyographic signals (EMGs) was under the name of *EmgSim* [3] developed in the System Software Laboratory at the Faculty of Electrical Engineering and Computer Science in Maribor. Initially, it was written in C++ for the Windows environment. Later on, we ported it into Java as an Internet application, built in the client-server mechanism, and even added some features being characteristic for the computer-assisted educational tools. Thus, we developed an integrated environment that may contribute at different levels:

- it generates artificial EMG signals whose building blocks are known in details; therefore, it plays a role of a reference to the EMG decomposition techniques [4, 5, 1],
- it enables experimenting with the effects of different electro-physiological parameters on the needle- and surface-recorded EMGs [7], and
- it also runs as a teaching/learning tool of the electro-physiology of muscles.

The approach will be presented briefly in this paper. Section 2 describes conceptual model of the *EmgSim* simulator, Section 3 speaks about the transformation into an Internet application, whilst the modifications introducing the educational elements are revealed in Section 4.

2 Simulations of electromyographic signals

Contractions of muscles are accompanied by electrical activities that may be measured as an EMG, invasively by needle electrodes or non-invasively by surface electrodes. Physiologically, muscles consist of the so called fibres. Each fibre is innervated by a motoneuron transmitting the triggering electric pulses that cause the contraction of a fibre. In fact, one nerve innervates several fibres that are, consequently, contracted at the same time and acting as a primary unit of force – called a motor unit (MU). Several MUs then build up the whole muscle.

Looking at this structure from the electrical point of view, the activation potentials flow down from the motoneuron. It triggers the corresponding single fibres, which affects them with a charge spreading from the innervation zone towards the tendons. This travelling charge is measured as a single-fibre potential (SFP). All the SFPs of one MU sum up into a motor-unit potential (MUP). Several MUPs are finally superimposed and, thus, form the observed EMG (Figure 1)[7].

Our simulator conceptually follows the electro-physiology. Therefore, it begins first with building up a muscle. Several parameters may be specified in this stage [3], like the distribution of single fibres in the MUs, the

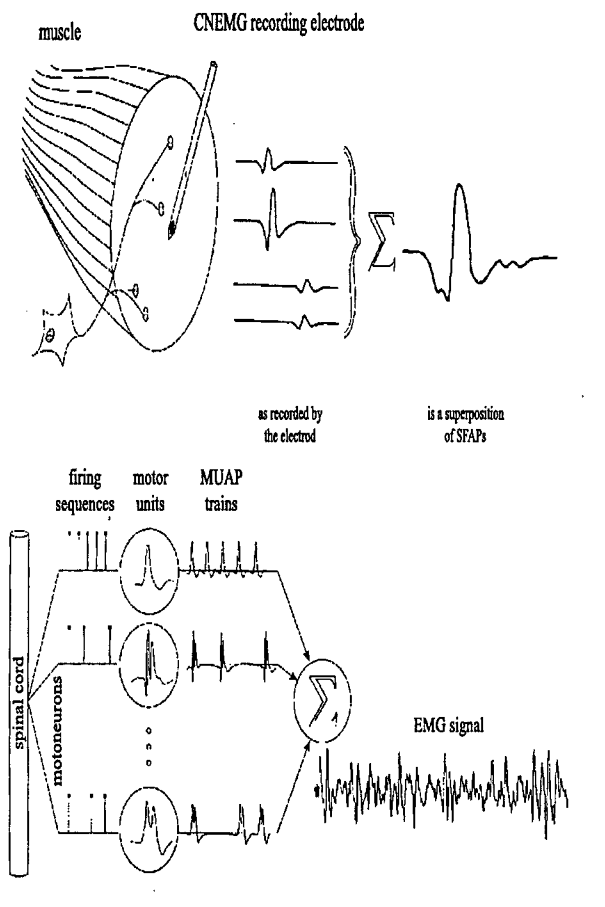
muscle
CNEMG recording electrode
as recorded by the electrod
is a superposition of SFAPs
firing sequences
motor units
MUAP trains
spinal cord
motoneurons
EMG signal

Figure 1: The physiology of a muscle and sources of electrical activity – top, a model of the muscle – bottom.

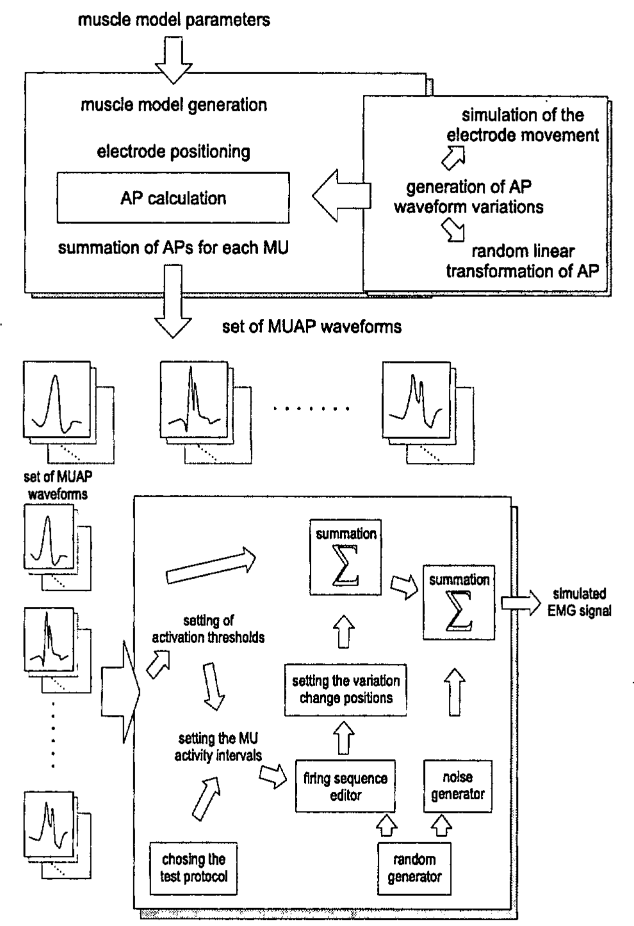
muscle model parameters
muscle model generation
electrode positioning
AP calculation
summation of APs for each MU
simulation of the electrode movement
generation of AP waveform variations
random linear transformation of AP
set of MUAP waveforms
set of MUAP waveforms
summation
summation
setting of activation thresholds
setting the variation change positions
setting the MU activity intervals
firing sequence editor
noise generator
chosing the test protocol
random generator
simulated EMG signal

Figure 2: Principle of operation of the EMG simulator.

distribution of MUs in the muscle, the features of the fibres (their diameters), the triggering instants, etc. The principle of operation is depicted in Figure 2.

3 The EMG simulator as an application in Java

We have stated multiple advantages of today's interactive network applications. The most important in our case are flexibility and accessibility supported by intrinsic levels of control, security and protection. This is exactly what was needed for our EMG simulator to become widely available application on the Internet. Actually, the client-server link seems ideal for this purpose. On the server site, the modelling and simulation part runs with all the calculation routines, data bases, control, security and protection. Clients enter the server from their machines on a platform-independent principle (see Figure 3). If they have access rights, they may send in their requests, i.e., they may run the EMG simulator at different stages. Only the user-interface code is transferred to their site, where it is interpreted and supports the windows and menu environment.

However, as all the computation is done within the server, there is a lot of data that must be transmitted to the users in certain stages of simulation. This could degrade the simulator performance and its responsiveness, that's why a special data communication protocol was developed between the server and the clients.

The characteristics of our Java realisation, as well as the client-server communication, will be described in the following paragraphs.

3.1 Realisation of the server and client

The client software that runs on a user's machine in Java would need more time to run the simulation. Therefore, we divided the simulator in two parts: the client written in Java for user interface, and the server written in C++ (native code) for the calculation of the simulation. The client performs simple calculations and provides user interface. The server performs all the time critical calculations.

The client-server architecture has been used to speed up the simulation. The basic problem is communication over the network. If all the clients were connected to the same instance of the server, an internal server mechanism would have to keep trace and status of every action of individual clients. This would make the server code rather complicated and vulnerable. Therefore, it was worthwhile to sacrifice a part of the server's performance by starting a new

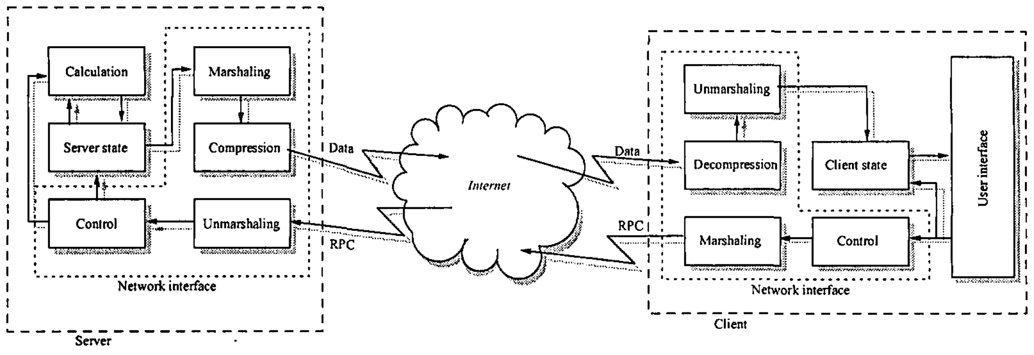

Figure 3: The simulator structure in Java: a client-server connection.

server instance for every new client, and even avoiding the solution with threads for the same reason.

The simplest solution for the communication would be remote procedure calls (RPC), where there would be no need to track the status of simulation, i. e., the stage of its execution. The client would inherently employ the remote functions as demanded in the successive steps by the user. However, Java programming does not support RPCs, it is based on remote method invocation (RMI), which lacks tools for standard serialisation of data and actions.

As mentioned, our solution to the server was starting new instances for every new client. The server incorporates special communication routines – deamons that use standard inputs (STDI) and outputs (STDO). The STDO transfer of data is activated by detection of the ENTER code. One inconvenient behaviour of the deamon is that, after the transfer is finished, terminates the connection. The action is natural in the multi-client environment, however, in our solution it is disturbing. A client and its instance of the server should stay connected and on-line all the time. To circumvent the problem of the communication channel termination, all the server messages are sent out by flushing.

On the other hand , clients call the server functions by simple strings of the following form:

<function name> <arguments>.

As the clients are Java applets, they run under multithreading principle. The server on the other side, which is a C program, is not able to synchronize with the order of precedence of incoming messages. Therefore, the clients communication is synchronized with the server using an interval monitor structure which prevents sending a message before the previous one has been completed.

To reduce the amount of network communication, the client and the server must have a synchronized copy of the state of their common actions. Another problem is data format. For example, Java uses IEEE 745 for representing floating point arithmetics. The server could also run on a Motorola, MIPS, SPARC or Intel-based machine that use different floating-point representations. Therefore, the data format for the network should be in a machine independent format. That is done by marshaling/unmarshaling of the data sent or received over the network (see Figure 3).

Hence, the data is serialised and transfered in either direction in strings containing also all the necessary information about the original data structures (tables, vectors, etc.) and types (integers, floating point values, etc.).

The amount of data sent from a client to the server is rather low. On the other hand, in the stage generating the EMG singal the server have to send out about 3 M bytes of data. It is, therefore, of crucial importance that the transfer is coded. We implemented run-time length code compression based on nibbles. The compression rate achieved is 38%, on overage.

Figure 4 depicts class-hierarchy diagram considering the client-server interconnection. Solid lines in the figure show the hierarchical dependence, whereas dotted lines indicate the information flow. Only the part depicting the client can acctually be shown in the form of class hierarchy. The server has been coded in C++, so the class hierarchy is not applicable in the same way. Nevertheless, the user interface in Figure 4 consists of two separated branches: one deals with user interaction, the other with data visualisation. The MUDialog box stands for the parameter input on the motor units, and the SPShapeDialog box accepts parameters on the action potentials. On the other hand, the EMGPanel box deals with the visualisation of the muscle structure, while the MUAPsPanel takes care of displaying the motor-unit action potentials, i.e., the EMG signals.

4 Upgrading to educational tool

Once having a conceptual model in the form of a simulator, it is just a step ahead to upgrade it for educational purposes. Although a reversed order may seem more normal, i.e., to build a teaching tool and then to include the simulation sessions as explanations, we haven't found any drawbacks in our approach. Moreover, it offers a two-level construction that, on the first level, acts as an extended help for the users of our simulator, on the next level, however, it enables self-evaluation. The latter is done in a typical learning cycle: a piece of learning material is provided, with explanations, demonstrations, graphics, diagrams, etc., then questions about the topic are asked. The answers are scored automatically and the results direct the user either over to the next topic or suggest repetition of some previous chapters (potentially, even some additional pages for basic comprehension may be inserted).

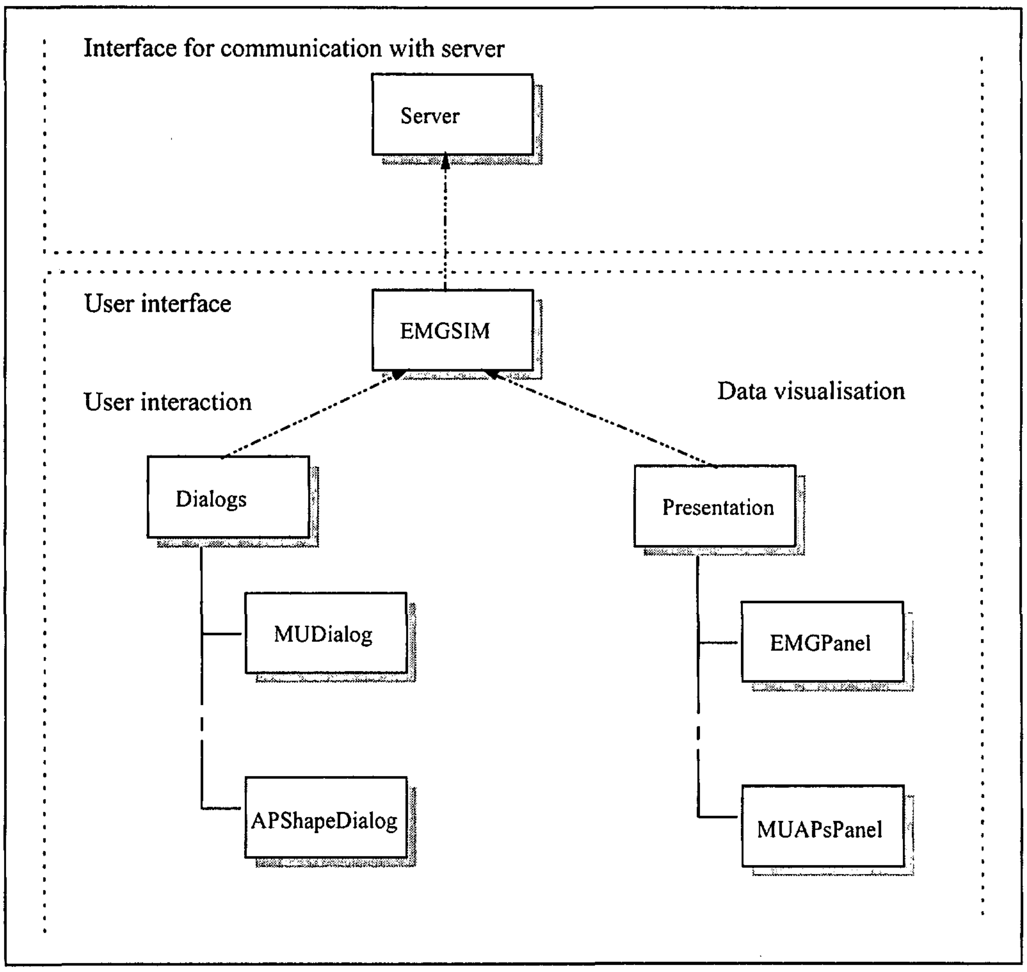

Figure 4: Class hierarchy: the client-server connection.

The advantage of having chosen the simulator as a driving engine for the learning subroutines certainly is reactivation of the same learning modules at any stage of the simulation run, as depicted in Figure 5.

In order to extend the Internet EMG simulator as in Figure 3, special program exits were inserted into the code of Java realisation. Their role can be compared to masked procedure calls: the application runs with all of its functions even if program exits are not activated. When activated, they trigger certain parallel actions which are understood as auxiliary to the main operation of the application.

The most important and instructive notions and steps in the simulation are elaborated additionally in separate pages of explanations an teaching material. The links are realised in terms of program exists. Each program exit has a corresponging parameter containing the address of an HTML document which will be displayed upon a call initiated by the user pressing a button to enter the teaching/learning procedure.

There is another parameter that has special meaning. With this parameter we can switch between two types of the program exits. The first type is browser independent and it is implemented with an applet context. As we know, Java 1.1 is still not fully supported by browsers. Therefore we decided to implement additional program exits bound to Netscape's Internet browser. In this way, we can use appletviewer which supports Java 1.1 to run our applet, and the Netscape's browser to view educational documents. However, similar solution would be possible with Internet Explorer as well.

Every such a program exit calls the indicated HTML document that may contain any necessary educational/explanation contents, supported by all the available relevant information, either locally or over the Internet connections. This may include texts, voice, static images, tables, graphs, or movies.

Afterwards, a self-evaluation procedure is entered by a special button. It consists of three stages:

- questions on the topics presented,
- answers typed in the pre-prepared forms, and
- an automatic evaluation of the answers.

4.1 Concept of the question-answer engine

The first two stages are more or less straightforward. The questions are sequentially numbered and insertet at the end of the HTML pages that are activated through the simulator

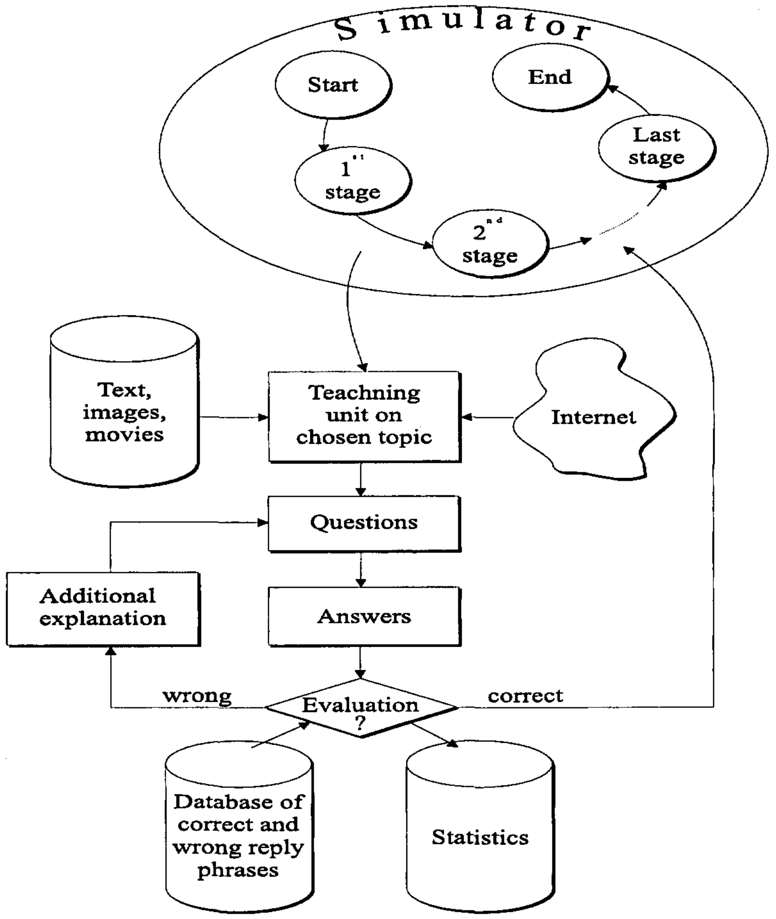

Figure 5: Block diagram of the simulator upgrading to a teaching/learning tool.

program exits. Dialog windows followning the questions receive the typed in answers. The entered answer links to an evaluation CGI (Common Gateway Interface) script directly from HTML. Such scripts enable execution of applications being called from the HTML documents. For the time being, our question-answer engine classifies answers as false or correct only. This is done with the help of a database containing sets of the words and phrases of possible correct and false answers in the following format:

<sequential number of the question>:< elemnts of possibly correct answers>:< elements of possibly wrong answers>.

The computer-assisted evaluation of the answers is, however, not an easy task. If we wanted to have it universal and complete, we would need a thorough base of grammatical, syntactical and semantic rules for every teaching/learning language. At the present stage of development, we have simplified these demands considerably.

The answers are parsed to the individual words. The words and word groups are, afterwards, compared to two lists of phrases. These lists are linked with the corresponding question. One list contains typical correct words and phrases, the other typical wrong words and phrases. According to the score achieved in both lists, every answer is treated either correct or wrong.

For each question, the user/teacher has to prepare and enter a target path (the document which contains the question), a question, a list of correct answers, a list of wrong answers, and a path to the next question in chain. This is realised by another CGI script. The target path is an absolute path in the file system, the path of the next document in chain is relative to the HTTP server. A blank field for the next question terminates the chain. After having been submitted, a new record is written to the database and the target document is generated. Each question has unique ID (a sequential number) which links it to corresponding record in the database. The generated document contains the question and a field for answer. Submitted answer is evaluated and a new document is generated, containing the evaluation score and a link to the next question in chain, although the entered answer may have been incorrect.

5 Conclusion

The described solution could mean a universal approach to upgradings of network versions of simulators. The necessary change in the simulator code is minimum, actually it means only inserting a simple routine to load HTML documents for additional explanation and activating the question-answer engine. The latter runs quite independent of the application and, therefore, can be developed and treated stand alone once for all the applications using it.

Acknowledgement

This work was supported by an HP Labs donation of workstations and by a grant of the Slovenian Ministry of Science and Technology, Contact No. L2-8834, to the Centre for Interactive Network Applications at the Faculty of Electrical Engineering and Computer Science in Maribor.

References

[1] Korošec D., Martinez C., Zazula D., "Parametric modelling of EMG signals," *IEEE EMBS*, Amsterdam 1996, 2 pp.

[2] Roeleveld K., *Surface motor unit potentials; the building stones of surface electromyography*, PhD Thesis, Katholieke Universiteit Nijmegen, The Netherlands, 1994.

[3] Viher B., Korošec D., Zazula D., "Computer simulator of electromyographic signal," *CBMS '96*, Ann Arbor 1996, pp. 47-52.

[4] Zazula D., Korže D., Šoštarič A., Korošec D., "Study of methods for decomposition of superimposed signals with application to electromyograms," in: Pedotti et al. (Eds.), *Neuroprosthetics*, Springer Berlin 1996, pp. 377-389.

[5] Zazula D., Korošec D., Šoštarič A., "Parametric decomposition of the SEMG," *Proceedings of the SENIAM Workshop*, Nijmegen, The Netherlands 1998.

[6] Horstmann C. S., Cornell G., *Core Java Volume II - Advanced Features*, Sun Microsystems Press, USA, 1998.

[7] Stalberg E., Trontelj J. V., *Single Fiber Electromyography*, Raven Press, New York, 1979.