Intelligent Medical Instruments

Zoltán Papp, Gábor Péceli, Balázs Bagó, and Béla Pataki
Department of Measurement and Instrument Engineering
Technical University of Budapest, Hungary

NOWADAYS, the designers of computerized medical instruments attempt to extend the performance of the measuring systems with data processing capabilities towards intelligent reactions. These intelligent instruments can help in monitoring, in measurement supervision, and in making diagnoses. This paper, based on the authors' experiences in the development of intelligent EEG recorders and analyzers, introduces those system design methods that can contribute to the resolution of the rather complex requirements in the field of medical instrumentation. These requirements involve the problems of:

- Efficient implementation of conventional data processing algorithms
- Efficient implementation of knowledge-based data processing algorithms
- Event-driven real-time operation
- Integration of problem-oriented user-friendly user interface, and
- The interactions of the above elements.

In this paper, both conceptional and implementational aspects are investigated and illustrated by some prototyping examples.

FUNCTIONAL AND IMPLEMENTATIONAL EXPERTISE

In order to perform successful measurements, detailed knowledge of the specific application field and the particular measurement methods are prerequisite. To develop efficient measuring instruments of any kind, knowledge of the user's requirements and a quite general engineering background are also needed. The two basic constituents of this latter requirement are the FUNCTIONAL (measurement theoretical) and the IMPLEMENTATIONAL (measurement technological) EXPERTISE [1]. The background for the functional expertise lies in measurement theory, which exposes the general properties of the measurement process. This kind of expertise covers the specification of the measurement environment, information processing, and the evaluation of results. The implementational expertise relies on metrology and the measurement technology valid for the specific domain of application.

The key issue of intelligent measuring system design is the *formulation and efficient implementation of the expertise required.* The complexity of this design procedure depends mainly on the measurement problem at hand. In the field of biomedicine, this complexity is considerably high, and asks for rather careful and systematic design approaches. These system design methods serve:

- Efficient implementation of conventional data processing algorithms
- Efficient implementation of knowledge-based data processing algorithms
- The efficient, event-driven real-time operating environment, and
- The realization of problem-oriented user interface.

The conventional data processing algorithms (i.e., estimations, decision schemes, etc.) can relatively easily be implemented using well-developed methods. The knowledge-based data processing algorithms are based on non-formal heuristic knowledge originating mainly in professional intuition and experience, but nevertheless very useful in solution searching or for solutions with excessive demand for operational time and resources.

Regarding the implementation of heuristic knowledge, the following three problems are to be considered:

- Representation of knowledge
- Manipulation of the knowledge base
- Integration of the knowledge base with the numerical data base and the algorithms within the unified real-time control structure of the measuring system [2].

The first two problems traditionally belong to the research field of artificial intelligence [3], while the third one is an emerging chapter of measuring system design. The implementation and manipulation of the knowledge base, in general, is a task with significant time and resource requirements. Therefore, in real-time measuring systems its usage is limited by the built-in processing capability of the measuring devices.

In the following two sections, possible alternatives are presented; first for the implementation of conventional data processing and later for the knowledge-based data processing in measuring systems operating in a real-time environment. The topic of the problem-oriented user interface is outlined in the subsequent section. The paper ends with a short presentation of an intelligent EEG recorder development by the authors.

FIXED VERSUS RUN-TIME PROGRAMMABLE SOFTWARE ARCHITECTURES

One of the first experiences of the authors in the field of medical instrumentation, nearly ten years ago, was the development of a family of somewhat intelligent medical instruments for electrophysiological data recording and analysis. This family of instruments can be characterized by multichannel data acquisition, feature extraction, parameter estimation, and decision making capabilities, which are connected to processing real-time events and controlled through a problem oriented operator interface. The implementation of these functions requires considerable effort. However, similarities in data acquisition and signal processing, the nature of concurrencies, and real-time operation promise analogies utilizable in the design procedure—even of somewhat different instruments. This analogy is also valid for operator interface, where convenience includes the very same "front panel philosophy" to be applied for instruments of the same field. These encouraging facts have initiated steps towards standardization in several respects, such as data acquisition, signal processing, real-time structures, and specification methodology. By extracting the common elements, a quite general structuring of the software system is possible, which results in a framework useful as a starting point for further development of instruments of similar types.

In our experience, a three-level software module structure [4] can be used. The function-independent realization of the real-time control structure provides an instrument "frame" on which real-time measuring instruments of various functions can be built up. For the sake of wide range applicability, the following requirements were raised against the realization:

- In the real-time structure, the elements determining the real-time characteristics of the instrument should be separated (e.g., priority dependent scheduling algorithms). These elements should be transformed into a form in which real-time requirements can be easily expressed.
- The real-time structure should be made modular. This

0739-5175/88/0600-0018$1.00

provides "tailorability"; in realization of a given instrument certain elements can be omitted or new elements can be "pasted." In the course of designing module interfaces, an effort should be made to maximize information hiding [5].

- The module structure should be transformed into a three level structure:
  - —modules for realization of real-time control structures and its tables
  - —modules for realizing measuring function and user interface
  - —general library modules.

Of course, there are technological aspects of the realization of module structure. As a minimal requirement, the language applied should incorporate tools for modular programming. In this case, the function related to the real-time operation can be accomplished by the explicit calling of a real-time operating system or a run-time operation system. Using a high level real-time language, the realization can be simplified because the elements of the access graph (process, resource, protected resource, synchronization, access rights) correspond to the elements of the language (process type, module, interface module, procedure declaration/calling) [6]. The structure modularized according to foregoing requirements is shown in Fig. 1 (the figure does not contain all the legal accesses). The function dependent modules are shaded in the figure. On the base of naming and access rights, the elements of the access graph can be identified.

The main features of the module structure are:

- The uppermost level contains only the "frames" of the COMMAND DECODER and M1 ··· Mm processes (measuring chain drivers). Having started, they immediately enter the second level, where the function dependent procedures (PARSER, and M11 ··· Mmk) are realized. (Of course, inside the level additional decompositions can be performed.)
- The server processes can be utilized via the MEASurement CONTROL interface module. In order to provide universal applicability, the MEASurement CONTROL module is table driven; the CONTROL TABLE contains parameters defining the specification of real-time operation (e.g., priority of measuring processes, type of resources, restrictions of its utilization, etc.).
- The server processes handle special hardware units, so their algorithms depend on the measuring functions to be implemented.
- The lower level contains standardized library modules that supply implementing instrument dependent algorithms.

Using the module structure presented here, the implementation of a real-time signal processing instrument with interactive user interface is simplified into sequential programming.

A simplified version of the above discussed software architecture was used first in the course of the development of an EMG analyzer and later in that of an EEG signal analyzer. Experiences of these instrument developments have verified the applicability of the software architecture presented. The modularized system is easy to manage, both in the module implementation phase and in the system integration phase. One of the most advantageous feature is that the various user interfaces and measuring data processing functions can be implemented without any modification of the instrument

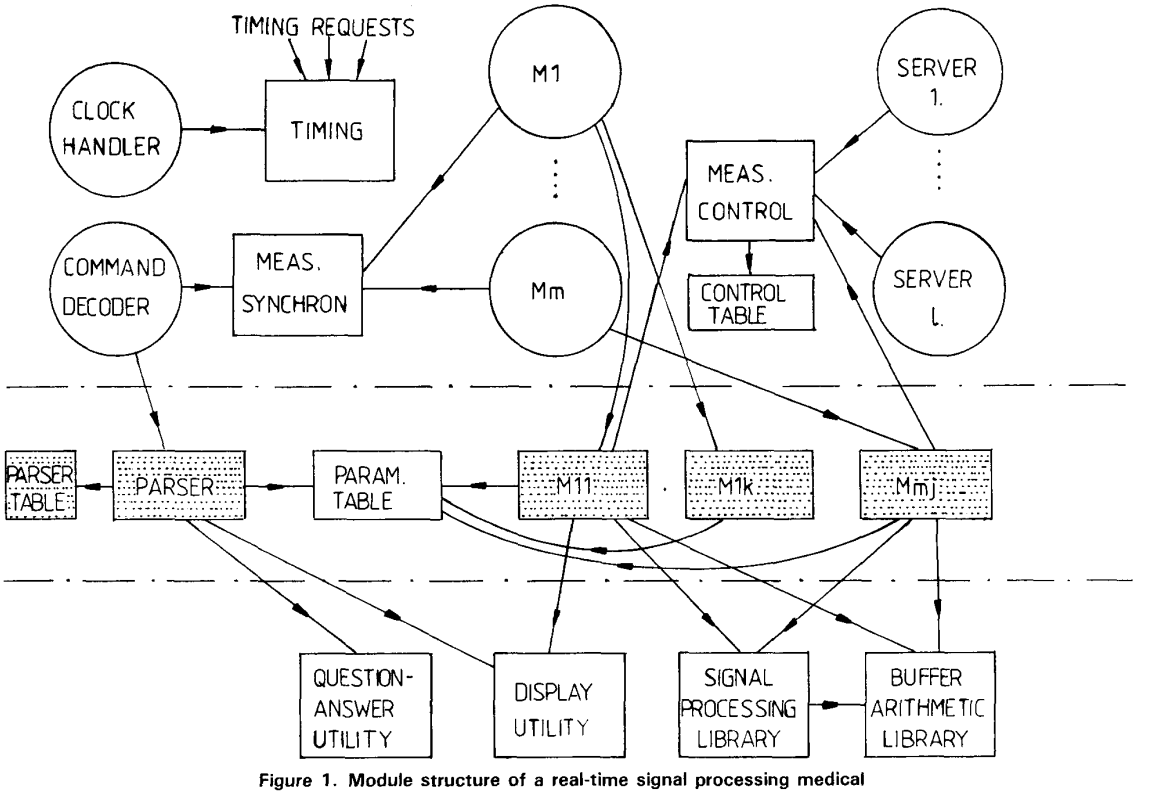


Figure 1. Module structure of a real-time signal processing medical instrument. Circles indicate process; squares indicate software modules.

"frame". Utilizing the frame of the EMG analyzer and the library modules, the time needed for developing the EEG analyzer shortened to a quarter of the time needed for the original EMG analyzer.

Fixed software architectures, however, have several shortcomings when the flexible programmability of the measuring device is also required. In our experience, the development of a general purpose signal processing system showed that conventional command languages support neither the introduction of additional procedures and data structures nor the modification of their internal real-time behavior. A thorough investigation of these shortcomings shows that the solution of the following problems is required:

- *Balance problem:* Rather contradictory demands are to be satisfied by a signal processing programming environment in the different phases of system development [7]. The possible solution of the balance problem is obviously a compromise among different requirements. However, without supporting this by appropriate conceptual and implementational tools, the result will be time consuming and occasional.
- *User interface problem:* Programmable signal processing systems require (in a sense) multi-tasking and a real-time operating environment, which support event-driven experiment control and interactivity with the user. The problem is that, on the one hand, it would be desirable to provide high level user interfaces, while on the other hand, high level tools may limit the appropriate application of the system facilities, which is not tolerable by the user. However, if the user interface is solved by providing a lower level programming language, possibly having real-time facilities, the user will have a programming task of higher complexity, which may easily reach the limits of his own programming ability.
- *System interface problem:* The typical (documented) interfaces of the traditional signal processing development system are shown in Fig. 2. Difficulties arise if an existing signal processing system has to be interfaced with other programming languages or programs (e.g., expert systems).
- *Signal representation problem:* Signal processing involves—besides numerical processing—some symbolic processing too (e.g., physical units transformation, derivation accounting), but traditional tools facilitate mostly the numerical side of signal processing.

As an attempt to give a solution to the above problems, we developed an experimental programming environment for signal processing (PESP), which gives a multi-level approach to the design and implementation of various data acquisition and signal processing programs and measuring devices. The conceptual base of PESP is a description language for signal processing together with an experiment control mechanism

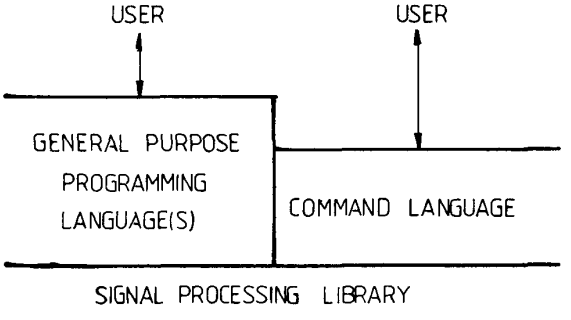


Figure 2. Typical interfaces of traditional signal processing development systems.

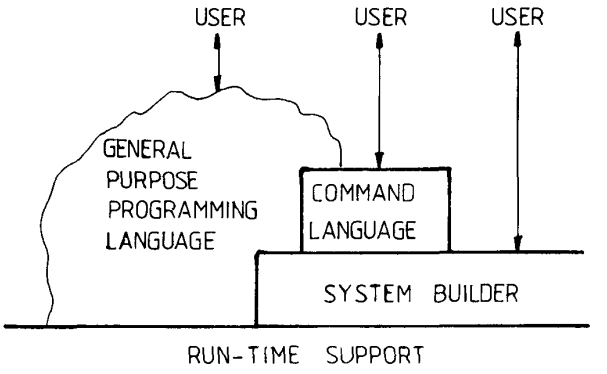


Figure 3. Components of an experimental Programming Environment for Signal Processing (PESP).

[8]. The implementational components of PESP are shown in Fig. 3.

The *run-time support* level constitutes background for the execution of the defined signal processing activity in a real-time environment.

The *system builder* provides symbolic interface for the provisions of the system. Its task is threefold:

- The system builder provides tools for building a network from the primitives of the languages. Using this facility you can create signal processing instances from processing element types and connect them together by means of signals.
- The system builder realizes multi-level abstraction; a subnetwork can be "boxed-up" and after that this item can be used as a higher level language "primitive" (a new processing element type).
- The system builder facilitates the modification/expansion of the language primitives. A user written signal processing function can be "merged" with the existing library, after that, it appears as a new language primitive.

Based on the system builder, different types of *command languages* (shells, etc.) can be constructed. A very advantageous feature of a well-defined symbolic interface is that its syntax can easily be converted to another one (e.g., graphic representation, dedicated keyboard).

A *general purpose programming language* can be integrated into this signal processing frame. A signal processing program written in that language can communicate with the implementational components of signal processing on different levels. It can directly use the element of the run-time support level (by means of function calls). It should be pointed out that in this organization even a sequential programming language is suitable for realizing parallel signal processing chains. If the programming language used communicates with the implementational components through the symbolic interface, the program has to create the symbolic description of the required signal processing architecture and pass it to the system builder, which after interpretation, builds up the architecture.

The operation of a signal processing program is based on accepting, producing, and processing signals according to the signal flow graph, which is a real-time and, in case of parallel branches, parallel processing task. To insure appropriate run-time behavior, signal processing is executed on the basis of an intermediate code. After defining the structure of the system, all the checkings and transformations of the numerical and symbolical properties of the signals can be carried out in advance, before starting the system. Computa-

tional power can thus be concentrated on pure data processing manipulations, which is not typical in conventional command languages for signal processing. The *run-time support* gets the description of the signal processing system in the form of an intermediate code from the system builder, and carries out all the real-time processing. The run-time support consists of three parts.

The *executive* reads the intermediate code, does the scheduling, tests the input signals if they are ready for processing or not, handles the control signals, and carries out the experimental control, the testing, and the debugging operations.

The extendable *signal processing library* contains all the bodies of the processes. Each process is build up from three routines: a data processing routine, a routine for testing and transforming signal description tables, and a routine for testing and transforming symbolic properties. (These last two parts, in fact, could be one routine, but the implementation of the numerical and symbolical computation is supported by different tools. Here they are written in different languages.)

The *utilities* consist of four main libraries: operating system interface, real-time kernel, signal processing utilities, and symbolic package.

The experimental version of the PESP was implemented on an IBM PC/AT computer using C and LISP languages.

Signal processing systems developed using the programming environment described here can have high-level, symbolical user interfaces well suited to the requirements of the application area and the operator, both in research and routine measurement situations.

INSTRUMENTS WITH REAL-TIME KNOWLEDGE BASED DATA PROCESSING

These instruments can be considered as straight-forward extensions of instruments dedicated to "conventional" real-time analytical data processing. In this case, the analytical data processing chain is followed by a fact generator and a (typically forward chaining) inference engine.

As an early experience, we designed an intelligent EEGer instrument assistant [13], which involves and uses some measurement technological knowledge. The results of the experience were:

- It is difficult to build a rule base using two-valued facts because of the low level expressing power.
- Since these two-valued facts are produced by the fact generator, the modification of the knowledge base (rule base) usually also induces the modification of the fact generator.
- As a consequence of the varying instrument environment, the inference engine should have fact (value) retraction capability.
- For the sake of focusing the searching, meta-rules should be used.
- Keeping the run-time on a considerably low level, rule compilation should not be put aside.

The updated version of the instrument mentioned above has the following rule structure.

```
RULE <id> AND|OR <relation> <relation>...
          CONCLUSION <attribute setting>
                     <attribute setting>...
          ACTION_SET <actions to be done in case of
                      rule firing>
          ACTION_RET <actions to be done in case of
                      retracting>
```

where ⟨relation⟩ is a relational expression on attribute values. There are two special actions (INCLUDE, EXCLUDE) by which the active set of rules can be controlled (this is the way to formulate meta-rules). The rule compiler determines the static dependency among attributes, relations, and rules. Using this information in the code generation phase, the number of rule matchings can be minimized and the propagation of the value retraction can be controlled. Figure 4 shows the dependency of the following rule set.

```
RULE r1 AND <relation containing attribute a1>     ; rel1
            <relation containing attribute a2>     ; rel2
        CONCLUSION <setting attribute a4>
                   <setting attribute a5>


RULE r2 OR  <relation containing attribute a1>     ; rel3
            <relation containing attribute a2>     ; rel4
            <relation containing attribute a3>     ; rel5
        CONCLUSION <setting attribute a5>
```

Arrows show the propagation of attribute value setting/retracting. A piece of code is generated:

- for each relation to determine the truth value of the relation
- for each rule to propagate attribute value setting
- for each rule to propagate attribute value retraction.

The appropriate invocation of the individual code segments is controlled by a small run-time system, which realizes the forward chaining control paradigm.

Our rule compiler can generate C language or LISP code. The run-time system is written in C.

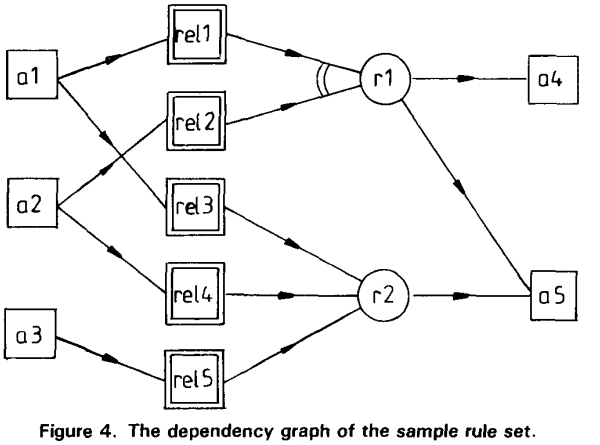


Figure 4. The dependency graph of the sample rule set.

PROBLEM-ORIENTED USER INTERFACE

The main purposes of the user interface are to determine/control the operation of the instrument (measuring system) and to present the results of the data processing in a user-friendly way (i.e., to support the result interpretation by means of showing the result set of various data/signal processing activities in a different point of view).

The effects of the former to the real-time operation and to the architecture of the instrument are presented in [4] and [8]. Here we deal with the consequences of the latter.

The model of the system measured (investigated) can be considered in various aspects and at different depths. This model should be reflected in the user interface. A given system parameter (which can be a result of an analytical signal processing activity or an inference process) may

belong to several aspects and may deliver information at different depths (Fig. 5). As a consequence the information inquiry should be organized according to the system model. In this type of user interface architecture, the demand-driven data acquisition and processing operation mode can easily be integrated [11]. This approach supports the economical resource management, too. It should be mentioned that the user interface presented can be extended with knowledge-based components in a straightforward way (e.g., expert system using backward chaining reasoning paradigm for automatic result interpretation). In this case, the result-set constitutes the dynamic data base (facts) for the interference system. This extension requires the modification of the system model component only.

INTELLIGENT EEG RECORDING

Our latest development in the course of medical instrument design is an EEG recorder having "measurement technological intelligence," which is comparable with the technological knowledge of a well-trained human assistant. The purpose of the development was threefold:

- Automation of the recording (it is especially advantageous during the long-time—e.g., sleep—analysis periods)
- Increasing the reliability of the evaluation by means of good quality recording, and
- Supporting the physician in the long record evaluation by drawing his/her attention to the possibly abnormal record intervals.

The developed device is composed of two basic units. One of them is a traditional, although microprocessor controlled, programmable EEG recorder with remote control facility. The other unit is the supervising analyzer receiving the signals from the recorder (Fig. 6). During the analysis, the following items are closely monitored:

- Amplitude conditions of the channels (average intensity, trends)
- Power density spectra in the common EEG bands (δ, θ, α, $\beta 1$, $\beta 2$)
- Noise spectrum of the channels.

The results of the measurement are bound to a set of attributes (e.g., representing the spectral component at the line frequency, the average intensity in a certain EEG channel, etc.). This data base can be extended with the information resulting from other sources; non-analogue signals are also generated by the recorder and are directly convertable into two-valued attributes (e.g., paper is out, some channels overdriven, etc.). These attributes form the situation specific data structure, which the inference system operates on. The operation of the inference system is based upon the heuristic knowledge constituted from attribute relations (facts) and IF-THEN rules. It results in possible changing of the operational parameters of the recorder and/or sending messages to the operator.

The rule manipulating strategy of the inference system is the common forward-chaining method [3]. The inference process is induced by changing of a relation value and it continues until it runs out of the applicable rules. If no more rules are found, the system stops and will start again only if some new facts appear. For the sake of run-time efficiency, the direct code implementation was chosen instead of the more common interpretative one. In certain situations it is necessary to delay the inference process (e.g., to wait until the filter transients settle). This problem was solved by the introduction of "time-dependent" facts, giving exactly the same event-controlled structure of the inference system as in the case of the facts coming directly from the measurement results.

The software of the instrument was written by means of the real-time programming technology developed at the Department of Measurement and Instrument Engineering, Technical University of Budapest [12]. Approximately 80 percent of the program was written in this high level language; the time critical parts were written in structured

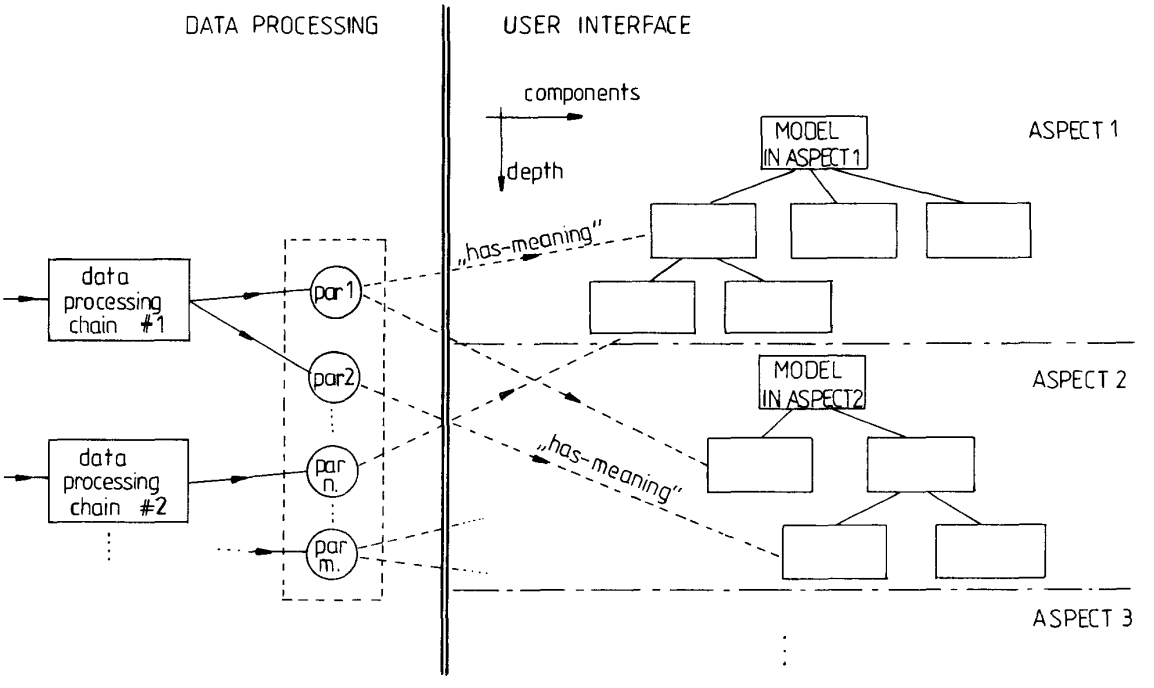


Figure 5. Linking user interface to data processing chains.

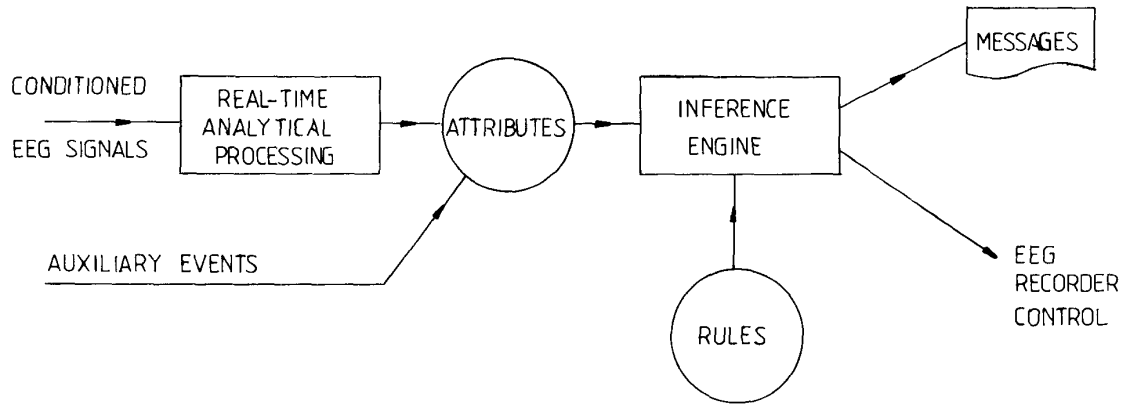


Figure 6. The signal processing scheme of the intelligent EEG recorder.

assembly language. The whole program required 16 kbytes (3 kbytes of this was for the rule base itself).

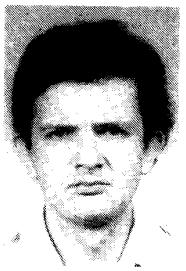
CONCLUSIONS

In this paper, some possible alternatives of system design methodologies have been presented, which in the authors' experience seem to be useful in the development procedure of fairly complex medical instruments. In addition to analytical and knowledge-based data processing, the requirements of the real-time operation are investigated with the intention of finding admissible compromises to solve the implementation problem.

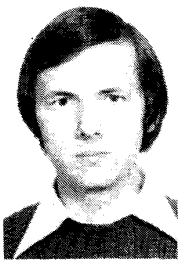
REFERENCES

1. **Sztipánovits J, Bourne JR:** Design of Intelligent Instrumentation. *Proc of the First Conference on Artificial Applications.* Denver, pp. 490–495, 1984.
2. **Sztipánovits J, Bourne JR:** Architecture of Intelligent Medical Instruments. *Proc of the Annual Conference of the IEEE Engineering in Medicine and Biology Society.* Chicago, pp. 1132–1136, 1985.
3. **Rich E:** *Artificial Intelligence.* McGraw-Hill, 1983.
4. **Papp Z, Bagó B, Dobrowiecki T, Péceli G:** Software Architecture of Real-Time Medical Instruments. *Proc of the Seventh Annual Conference of the IEEE Engineering in Medicine and Biology Society.* Chicago, pp. 1137–1142, 1985.
5. **Parnas DL:** On the Criteria To Be Used in Decomposing Systems into Modules. *Comm of the ACM,* 15(12):1053–1058, 1972.
6. **Wirth N:** Toward a Discipline of Real-Time Programming. *Comm of the ACM,* 20(8):577–583, 1977.
7. **Kopec GE:** The Integrated Signal Processing System ISP. *IEEE Trans on Acoustics, Speech and Signal Proc,* 33(4):842–851, 1984.
8. **Bagó B, Papp Z, Péceli G, Reguly Z:** A Multi-Level Signal Processing System. *Proc of the Eighth Annual Conference of the IEEE Engineering in Medicine and Biology Society.* Dallas-Fort Worth, pp. 825–828, 1986.
9. **Forgy CL:** Rete: Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence,* 19:17–37, 1982.
10. **Doyle J:** Expert Systems and the "Myth" of Symbolic Reasoning. *IEEE Trans. on Software Engineering,* 11:1386–1390, 1985.
11. **Biegl C, Karsai G, Sztipánovits J, Bourne J, Harrison C, Mushlin R:** Execution Environment for Intelligent Real-Time Instruments. *Proc of the Eighth Annual Conference of the IEEE Engineering in Medicine and Biology Society.* Dallas-Fort Worth, pp. 807–810, 1986.
12. **Bagó B, Gerhardt T, Karsai G, Papp Z, Péceli G:** Software Tools for Medical Instruments. *Proc of the Seventh Annual Conference of the IEEE Engineering in Medicine and Biology Society.* pp. 1143–1147, 1985.

Zoltán Papp received the B.Sc., M.Sc. and university doctoral degree in electrical engineering and measurement theory from the Technical University of Budapest, Hungary, in 1978, 1980, and 1984, respectively. He is an assistant professor at the Technical University of Budapest, Department of Measurement and Instrument Engineering (from 1982), and teaches measuring system design in undergraduate and postgraduate courses. He is author and coauthor of more than thirty technical articles and coauthor of a book. His research interests include real-time programming, measuring system design and application of artificial intelligence techniques in measurement technology.



Gábor Péceli received the diploma in electrical engineering (1974) and university doctorate (1979) from the Technical University of Budapest, Hungary, as well as a candidate degree in technical sciences from the Hungarian Academy of Sciences in 1985. He is an associate professor in the Department of Measurement and Instrument Engineering, Technical University of Budapest, and teaches circuit theory, electronic circuits, electronic measuring instruments, and digital signal processing. His research interests include network theory, analog and digital filtering, theoretical background of digital signal processing, and implementation of signal processing algorithms.



Balázs Bagó received the diploma in electrical engineering (1980) and university doctorate (1984) from the Technical University of Budapest, Hungary. He is an assistant lecturer in the Department of Measurement and Instrument Engineering at the Technical University of Budapest (from 1982) and teaches electronical instruments, measuring system design, and real-time programming. His research interests include measurement theory, microprocessor-based instrumentation, parallel computing, concurrent, real-time and system programming.

Béla Pataki received the M.Sc. degree in electrical engineering and measurement theory from the Technical University of Budapest, Hungary, in 1978. He worked with the Works for Electronic Measuring Gear, Budapest. In 1982 he joined the staff of the Department of Measurement and Instrument Engineering at the Technical University of Budapest, as an assistant lecturer. His research interests include measurement theory, microprocessor-based instrumentation, and digital signal processing.