Towards an Intelligent Problem-Solving Environment for Signal Processing

Chandra Shekhar Sabine Moisan Monique Thonnat

June 15, 1993

1 Introduction

Signal processing (SP) is an engineering discipline that has contributed greatly to technological progress in the 20th century. It has evolved considerably during the past few decades, and continues to evolve rapidly. Signal processing has applications in a wide variety of fields, such as meteorology, geology, medicine and robotics. As the need grows for more effective solutions to problems in these application domains, SP specialists have been coming up with more and more sophisticated techniques. These techniques, if used correctly, can provide solutions that are more robust, more efficient, more adaptive, and less expensive than previously used methods. Unfortunately, they also tend to be more difficult for a non-specialist to utilise, and consequently, the real users of signal processing methods, who are often non-specialists in SP, prefer to use methods that are simpler, although less effective. There is thus a wide gulf between the specialists and end-users of signal processing.

The complexity of certain techniques of signal processing is not the only difficulty encountered by its non-specialist users. The task of solving a signal processing problem may involve complex decision making at various levels, for which SP expertise may be necessary. Very often a single algorithm cannot solve a given problem, and several steps are involved in going from the raw data to the final results. Thus a number of stages of processing may have to be linked and scheduled to achieve the desired signal processing objective. At each of these stages, there may be a number of possible methods, from which the most appropriate one has to be selected. Each method may be implemented in a different way, using a different programming language and environment. A method may have one or more parameters, which have to be initialized before execution. Often, due to uncertainty in the data and in the problem model, it is not possible to predict beforehand if a given algorithm will produce the desired result for a certain parameter setting. It may be necessary to start with a rough guess of the parameters, execute the algorithm, examine the results, and if necessary, modify the parameter values, and repeat the procedure until results of the desired quality are obtained. If all this has a very heuristic flavour, it only expresses the reality.

A final difficulty faced by the user of signal processing techniques is that although many good libraries of signal processing programs exist, the algorithms needed for an application may not be available in any single one. Usually, the user commits himself to a specific library or environment, and has great difficulty in incorporating other programs or libraries into his set-up.

It is obvious, from the preceding discussion, that a transfer of expertise from the SP specialist to the SP user is required. This may be done in various ways, as shown in Fig. 1. The direct and constant supervision of the signal processing specialist, as in Fig. 1(a), although very effective, is usually impractical for economic and other reasons. A better solution is to design knowledge-based systems (KBS) for the application domain, incorporating the knowledge of the SP specialist as well as the domain specialist. (The word "domain" here refers to the domain of application, which may be manufacturing, medicine, geology, etc.) The services of the SP specialist are needed only in the development and periodic upgrading of the KBS. While such knowledge-based systems are of great help to non-specialists in signal processing, they are currently being developed using off-the-shelf AI tools that do not incorporate any *deep knowledge* of SP (Fig. 1(b)). A great amount of initial effort is involved in adapting the tools to the application. A further limitation of this approach is that the resulting system is usually dedicated to a particular problem, and cannot be readily used for other problems of a similar nature.

The next logical step, therefore, is the development of AI tools *dedicated* to SP problem-solving (Fig. 1(c)). These tools should function at the level of the signal processing expert, enabling him to express his knowledge in a convenient and natural manner. The purpose of this paper is to propose a dedicated AI architecture for the

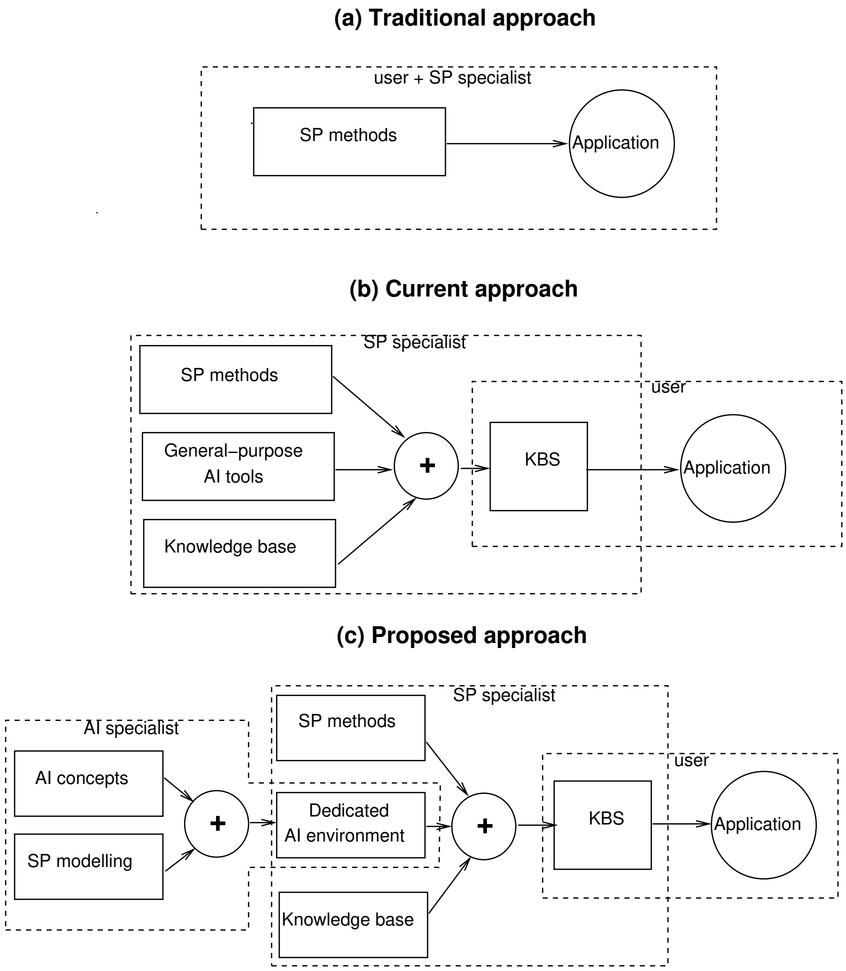

Figure 1: The different approaches to building a signal processing application

automation of SP problem-solving to the maximum extent possible, by imitating the strategies of a signal processing specialist.

We use an AI approach, focusing on the analysis of the different problem-solving tasks involved in signal processing and understanding. We clearly distinguish between signal processing and signal understanding, as being two different but co-operating high-level tasks (Fig. 2). In the first task, the knowledge is related to signal processing methods, such as filtering, sampling, etc. and "textbook concepts" such as signals, systems, noise, etc. This task is for the most part independent of the application domain. The second task, on the other hand, is highly dependent on the application domain, and deals with high-level models of the objects and events of that domain. The two tasks also differ in the type of reasoning used. In signal processing, the reasoning is mostly in the choice and tuning of appropriate methods for a given SP goal. In signal understanding, the reasoning is typically employed in matching the available evidence with a model, or in the accumulation of evidence to support a hypothesis. Although an SP application will involve the cooperation of both tasks, it is our opinion that the separation of the two tasks will result in more appropriate frameworks for each.

The AI architecture we propose in this paper incorporates the strategies used by the signal processing expert for problem-solving. It addresses many the difficulties with the use of SP methods discussed in the introduction. It provides, above all, an effective and convivial means of developing autonomous signal processing chains for various applications. Our proposed approach builds on experience gained in developing the AI environment OCAPI [2, 3], which was originally developed for the automatic supervision of image processing tasks. It has been used since then in a number of applications, including the identification of galaxies, traffic image analysis, and is currently being used for the automation of scientific calculations.

2 Related work

The use of AI methods for signal processing is a relatively recent phenomenon. Much of the early work in this direction was in developing expert systems for specific signal and image processing applications (e.g. [14, 7]). These made extensive use of the constraints of the domain, and were concerned mostly with the data interpretation as-

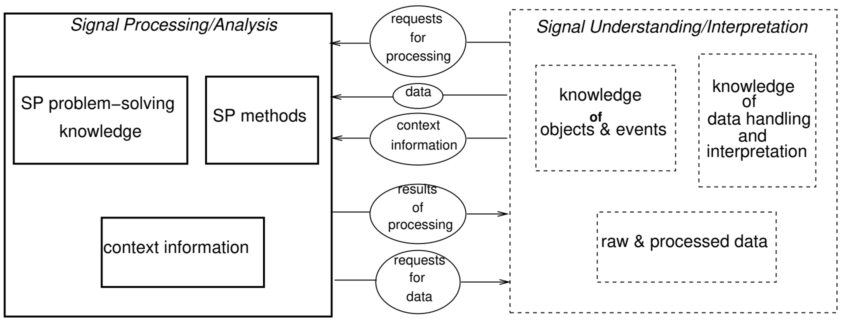

Figure 2: Signal processing vs. signal understanding

pects. More recently, many expert systems have been developed for specific signal processing tasks, rather than for specific applications (e.g [5, 10, 6, 13]). The work in [13] (see also [4]) is notable in that although it is currently restricted to the problem of system identification, it aims at creating a much wider platform for the development of signal processing applications. Their sophisticated mechanisms for the automatic supervision of programs from a large library should be very useful for a number of SP tasks and applications. However, their system is somewhat unwieldy since it is implemented on a general-purpose expert system shell.

As discussed in the introduction, expert systems built using general purpose software have some drawbacks, and there is hence a need for dedicated software tools. There are many examples of software environments dedicated to signal processing, such as SIGNAL [9], QuickSig [11] and ADE [12]. These, while having excellent facilities for algorithm design and simulation, do not provide the supervision mechanisms required in a complex signal processing application.

To remedy this situation, Nawab and Lesser [15] have developed an architecture for the integration of the processing and understanding of signals (IPUS), implemented on a special blackboard shell called RESUN (REsolving Sources of UNcertainty). Their emphasis is on the integration of signal processing and understanding, which is in contrast to the philosophy of this paper as outlined in the introduction. A major contribution of their work is a typology of SP "distortions" and "discrepancies", which are approximately defined to be, respectively, the possible mistakes that can be made in choosing a signal processing algorithm for the given task, and the errors that result in each case.

Dawant and Jansen [1] take a slightly different approach. They recognize the need to separate the domain-independent (i.e. signal processing) and domain-dependent (i.e. signal understanding) parts of the knowledge. In their architecture, they have a separate knowledge-base for each, and the two are linked by a coupling mechanism. Like [15] they too use a blackboard framework to provide for opportunistic control, for both tasks (processing and interpretation). Although data-driven control may be appropriate for signal interpretation, for signal processing it is more appropriate to have an architecture with structured, rather than opportunistic control. The necessary flexibility, robustness and adaptiveness to the data can be ensured by sophisticated mechanisms for planning and execution control, as presented in this paper.

3 Signal processing fundamentals

There are two fundamental concepts in signal processing, that of a *signal*, and that of a *system*. A signal is a flow of data, a simple example of which is a 1-D time-sequence of real numbers. A system, or SP algorithm or SP method, takes as input a signal, called the *input signal*, and transforms it into another, *the output signal*.

Signals and systems may be continuous or discrete. Signals may be deterministic or stochastic. A special type of stochastic signal is the *noise* that corrupts every physical signal. A continuous signal may be converted into a discrete-time signal by sampling. In addition, the values of the signal at the sampling instants may also be quantized and coded. A system can be of various types, and of various degrees of complexity. Most signal processing literature deals with linear, time-invariant systems, which have the fundamental property of preserving the frequency of a sinusoidal input. They are the easiest to characterize and to use in the design of complex

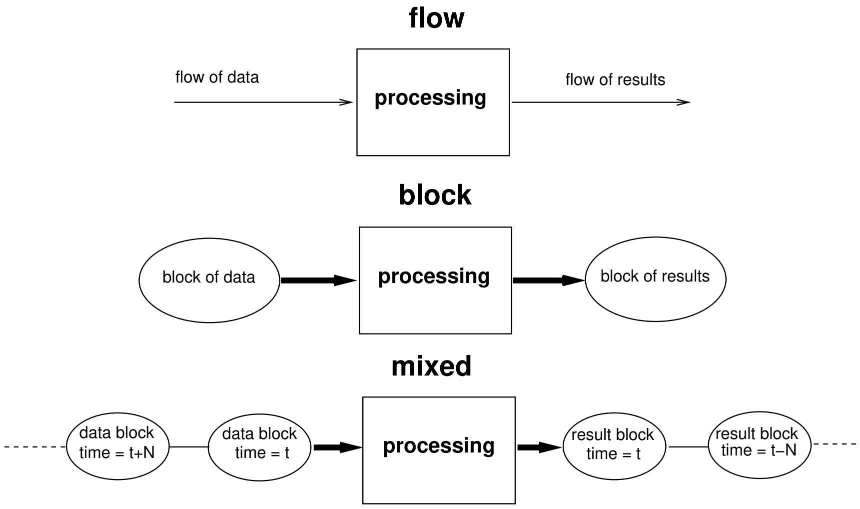

Figure 3: The different types of systems

systems. However, many operations, such as demodulation and sampling, require the system to be nonlinear. Systems can also be characterized by the manner in which they process the signal (Fig 3). We describe systems in which there is a continuous in-flow and out-flow of data, with a new output value being generated as and when each new input datum becomes available, as flow-type systems. Common examples of this are recursive filters, adders, multipliers, etc. In contrast, systems such as transform operators which take as input a block of signal data, and produce a block of output data, are said to be *block*-type systems. Hybrid systems which process blocks of data continuously are called *mixed* systems. Systems may also be looked upon as signal processing *algorithms* [15], which may have one or more adjustable *parameters*.

A signal processing application typically involves several stages of processing, of various types (Fig. 4). The first stage, transduction, is the creation of the signal data from information that may be in other physical forms such as an electromagnetic wave or the motion of a physical object. The next step, filtering, is the processing of this signal to give it the properties that will make it suited for the next stage, signal analysis. Feature extraction is the determination of

∞

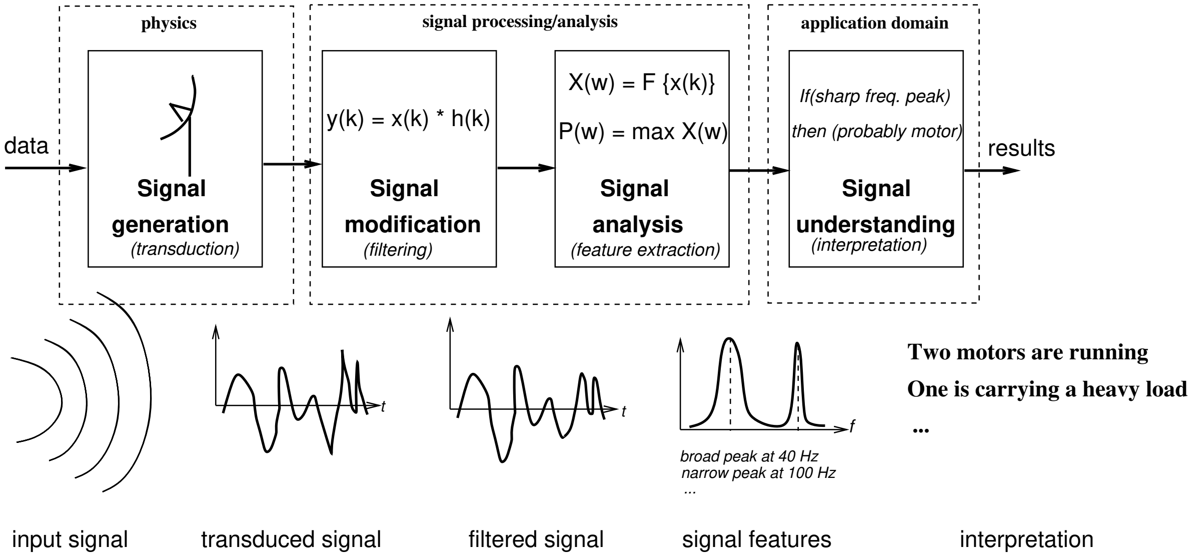

Figure 4: The different stages of a signal processing application

various features of interest from the signal, such as frequency content and large amplitude variations. These features are then used in the final stage of processing, interpretation, to interpret the signal.

Figure 4 and the description in the previous paragraph give a very simplified picture of the stages of processing in a typical SP application. In reality, the building blocks of the application may be far more numerous and complex, and there may be various feedback and feedforward connections present. However, Fig. 4 does illustrate the different types of knowledge (physics, signal processing and signal understanding) involved. Although the boundaries between these areas may often be fuzzy, the expertise required in each area is often of a very different nature from the others. In this paper we restrict ourselves to the parts of the SP chain (i.e. the processing and analysis aspects) which traditionally fall within the boundaries of signal processing.

4 A problem-solving model for signal processing

In this section we examine the strategies used by a signal processing specialist in solving a given problem. We are not interested in the *design* of SP algorithms—we assume that the necessary programs or libraries of signal processing methods are already available. We roughly classify the specialist's tasks as follows:
(1a) to select, schedule and link signal processing modules;
(1b) to initialize SP modules, to execute them, to monitor their performance, and to handle failures; and
(2) to handle data, to initiate processing and to interpret results.
All the above tasks are not, in general, done by the same specialist. Tasks (1a) and (1b) are usually performed by the signal processing specialist, using problem-solving expertise that is to a large extent independent of the domain of application. Task (2), which is normally performed by the domain specialist, involves a different type of expertise from that required for the other tasks.

In this paper, we restrict ourselves to the part of the problem solving that relates to the processing and analysis, i.e. to the tasks (1a) and (1b). Corresponding to these tasks, two forms of reasoning used by the expert can be identified: reasoning for *planning* and reasoning for *execution control*. This is shown in Fig. 5.

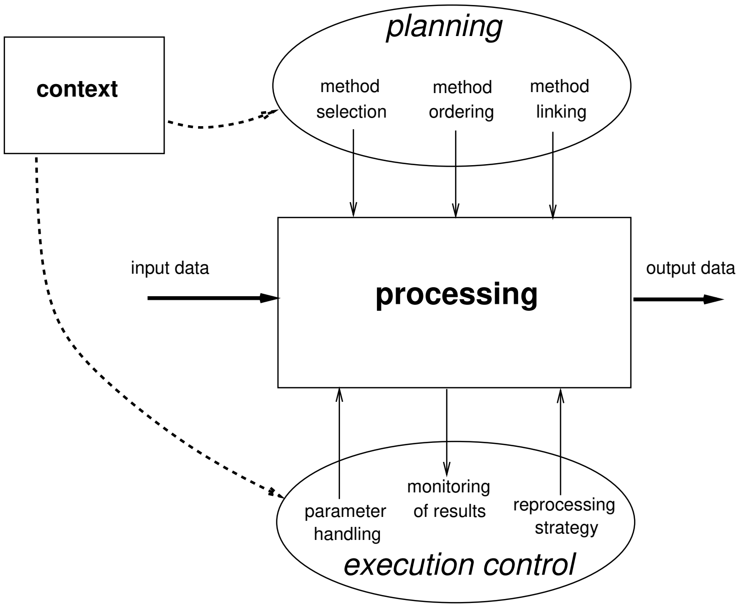

Figure 5: The basic problem-solving model

It is evident that a number of decisions have to be made at various times, for planning as well as for execution control. What distinguishes signal processing from classical AI application areas such as game-playing is the complex nature of the information to be processed. It is not possible to define exhaustively all the states of the world, nor is it possible, in general, to predict the results of an SP task for a given input. Classical AI planning, applicable when the initial state, the goal state and state transformations can be clearly defined (such as in the Monkey and Bananas problem), is evidently inadequate for the task. Further, there is a natural ordering to the signal processing tasks to be performed in solving a given problem, and this fact should be used in selecting a planning strategy. Planning, therefore, has to be based on what SP tasks do, rather than on the states of the world, and has to be interleaved with the execution.

During execution control, the main tasks are parameter setting, performance monitoring and failure handling. A typical signal processing application involves a number of interconnected systems, each of which performs certain basic SP operations. Each of these operations may involve a number of parameters. In order to obtain the desired results, the specialist selects suitable initial values for the parameters, as well as strategies for modifying them if the need arises. The specialist decides the criteria for evaluating the performance of each module. These criteria are based on numerical or symbolic analysis of the module's output. In the event of unsatisfactory performance (failure), he decides on the strategy to be followed (such as reprocessing or replanning).

5 The proposed architecture

Based on the preceding discussion, we propose a software architecture for intelligent problem solving, dedicated to problems in signal processing, as motivated in the introduction and in Fig. 1(c). It is a KBS development tool which uses frames and production rules as knowledge representation schemes. Its goal is to enable the automation of the problem-solving to the maximum extent possible, by imitating the strategy of the signal processing specialist. It provides mechanisms for planning and control of execution appropriate for SP problem-solving, in addition to the symbolic problem representation facilities available in most popular signal processing languages. The main components of this architecture, and the relationships between them, are shown in Fig. 6. The functions of each component are explained in this section.

Problem description

Signal processing problems, as mentioned earlier, are described in terms of signals and systems. Signals are the data on which the processing is done, and systems are the methods used for processing the signals. The architecture provides for a basic signal data type, with sub-types such as sequence of complex numbers, sequence of integers, etc., and simpler data types such as symbols, symbol sets, floating point numbers, etc. The signal data type is represented in the form of a frame, which contains, in addition to the data, other information about it, such as data formats, history, storage space used, etc. A signal can have one or more channels, giving it a concurrent aspect apart from its temporal nature. More complex signal types, such as image sequences, with their rich spatio-temporal structure, can also be represented as variants of the basic signal data type. Signals may be accessed either in the block mode or in the

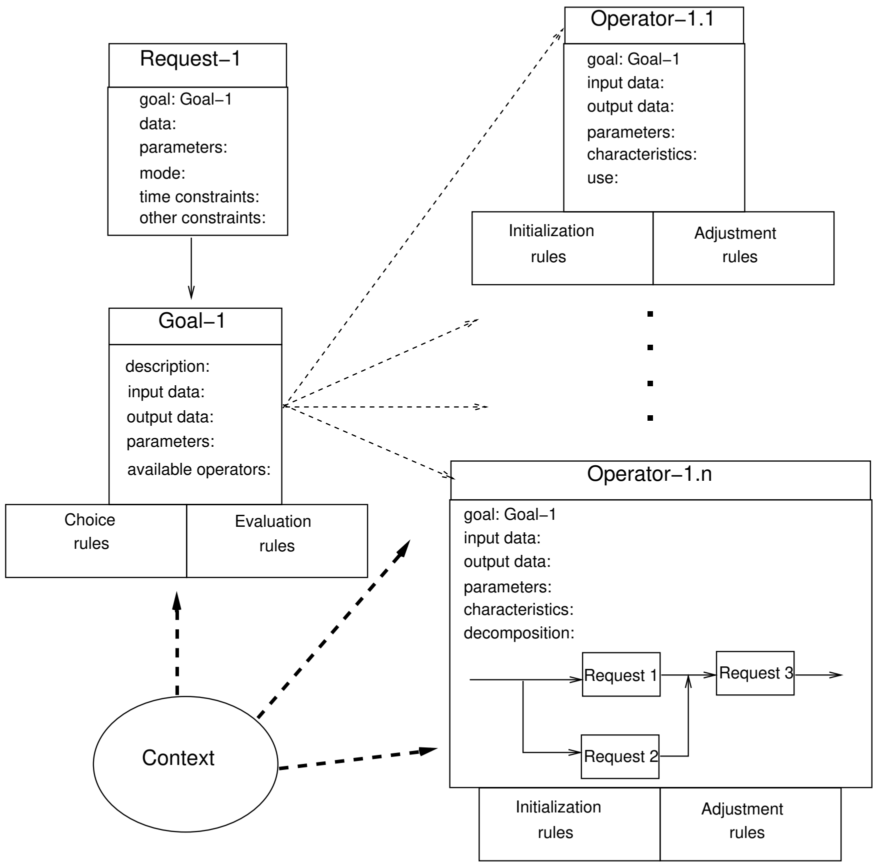

Figure 6: Relationships between the various entities

flow mode, depending on the way in which they are generated and stored, and on the way in which they are processed.

Knowledge about signal processing methods is expressed at two levels of abstraction. A *goal* is the abstract form of an SP functionality, which is realized in a more concrete form by one or more *operators* corresponding to it. An operator may be either a simple one, corresponding to an executable program, or may be a complex one, with a *decomposition* into sub-parts. A *request* consists of the signal data and the type of processing to be done on them, in addition to parameters, processing mode and other constraints. We provide for three processing modes in requests, block, flow and mixed, corresponding to the three system types shown in Fig. 3. All available contextual information about the problem, such as user specifications, type of sensors used, etc, is stored in the *context*. Goals, operators, requests and context are represented as frames. This is shown in Fig. 6, along with the relationships between them.

Planning

The basic planning mechanism provided is *hierarchical script-based planning* [8]. A plan is a set of steps to attain the desired signal processing objective. It is similar to the block schematics often used in the design of SP algorithms, but is different in two ways: (a) **plan hierarchy**: each "box" in the plan can be a complex SP tasks, with its own plan and (b) **plan abstraction**: the components of the plan represent abstract SP tasks (goals), and not specific algorithms or programs. During execution, *choice rules* are used for selection of the operator best-suited for the task at hand. They are of the form:

if

<the data have a property x>
or <context field f has a value y>
or <the request has a constraint z>

then

<chose operator "a">
or <chose an operator with characteristic b>

(The above shows only some of the possible types of premises and actions in a choice rule. Many other types of premises and actions are possible, for choice as well as for the three other kinds of rules described later.)

Skeletal plans are physically stored in complex operators in the form of decomposition diagrams, consisting up of one or more sub-tasks connected using series, feedforward and feedback links. These sub-tasks are treated as requests (and not goals or operators) since the data flow between the sub-tasks of a complex operator is known.

Execution control

Execution control, in this architecture, can be described as the task of executing a plan, making it work on the given data and in the given context. *Initialization rules* are used to set the initial values of the parameters. They are of the form:

if
<the data have property x>
and <the field f of the context has value v>
then
<set parameter p to value f(v)>

The performance of a signal processing request after execution is analyzed using *evaluation rules*, which are of the form:

if
<the output x does not satisfy criterion c>
then
<declare that quality q of the results is unsatisfactory>
<declare that the execution was a failure>

The re-processing strategy consists of *adjustment rules* for operators, of the form:

if
<the quality q of the results was declared unsatisfactory>
then
<adjust parameter p using method m>

Planning and execution control are interleaved in the sense that choice rules may be re-applied if operator adaptation does not produce the desired results. The different kinds of rules and their relationships to the other objects in the architecture are shown in Fig. 6.

The application of the rules for execution control to a request may be done in different ways, depending on the nature of the request and other constraints. The following four modes of execution control are possible:

1. Continuous: For a request of type flow or mixed, this means that the results are monitored after each execution. This mode, although the most effective, is the most expensive computationally, and may not be needed or even possible for all requests. For a request of type block, this mode may be used if the quality of the result is very important.

2. Periodic: For a request of type flow or mixed, this means that the results are monitored after each N executions. The previous mode is a special case with $N = 1$. This mode is not applicable to block-type requests.

3. Absent: In this mode there is no monitoring of the results of execution. This is applicable to requests for which execution control is not possible due to time constraints, or which are so straightforward that no execution control is needed.

4. Event-based: In a real-time application, there may be situations where the occurrence of an external event may signal the need to check the execution of one or more requests, which may otherwise run unmonitored. In such cases, the execution control of the requests is event-based. This mode may be used in conjunction with the periodic mode.

Building a knowledge base

In order to solve a given problem or set of problems the specialist's knowledge about the problem and the methods is expressed using the mechanisms discussed in the preceding subsections.

The first step is the building of *task blocks*, which are combinations of goals and the associated operators. Corresponding to each problem type, a goal instance is created. The specialist may have one or more different ways of fulfilling a chosen goal, and for each one an operator instance is created and linked to the goal. For a simple operator, the necessary information about the program which corresponds to it is entered into its "use" field. If the operator is a complex one, with a decomposition into one or more parts, task blocks have to be created for each sub-problem type, unless they have already been created. The procedure of defining task blocks is done for all problems and all sub-problems, until each simple operator is linked to a program, and task blocks have been defined for each sub-problem of every complex operator.

The ensemble of the task blocks of a problem constitute the static part of the knowledge base. The next step in the creation of the rules which express the strategy of the specialist in executing a plan. This is a more difficult phase than the previous one, but is crucial to the success of the entire approach to problem-solving. It is difficult because, as in many other engineering disciplines, the specialist's reasoning may not often be easily expressible in the form of clear-cut rules. It should also be borne in mind that multiple rules will often be required to provide the required depth of reasoning. These inherent difficulties notwithstanding, experience has shown that it is possible to express the specialist's reasoning in a concrete fashion, provided the representation mechanisms have been correctly chosen.

The architecture presented in this paper enables the systematic expression of the "rules of thumb" and the "approximate reasoning" which are crucial to succesful problem-solving. This is done by means of the four types of rules discussed earlier. A mixture of numeric and symbolic reasoning may be needed for all types of rules. Choice rules, being the easiest, are defined first. They use the values of context fields, data definitions, constraints in the request, etc. to select an operator from among the choices available. Initialization rules are defined next. These may be somewhat more difficult, in that they have to formalize the "rough initial guesses" that the specialist makes before starting an signal processing task. Adjustment rules are then defined for operators which have adjustable parameters. Step sizes for parameters have to be carefully chosen so that the change in the behaviour of the algorithm is neither too sudden nor too gradual. Finally, evaluation rules are defined. This is a rather difficult task since the question "Are the results good enough?" is often subjective, and appropriate quality measures may not be readily available. However, a close examination of the specialist's strategy often reveals hidden reasoning capable of being expressed in concrete terms as evaluation rules.

Problem-solving using the proposed architecture

The previous subsection describes how to build a knowledge base for an application. Here we describe how an application problem is solved in our architecture, using this knowledge base and the available methods.

The solving of a problem starts with the creation of a request, which states the goal required, the data on which this goal is to be performed, and the context in which the problem is being solved. (The goal would have already been defined while creating the knowledge base.) Using choice rules, the available operators for the goal are rank-ordered. The best operator is selected, and executed on the given data after the operator's initialization rules have been applied. If the operator is a simple one, this coresponds to the execution of the corresponding program. If it is a complex one, requests for its sub-tasks are created, and a tree of requests is created. Requests are chosen from this tree and executed depending on their sequencing. If the mode of execution control calls for it, the results of executing a request are judged using evaluation rules, and in the event of a failure, either the same operator is re-executed after its parameters have been adjusted using the relevant rules, or the next best operator is applied. In the event of successful execution, the next request in the tree is selected for execution. This continues, and the execution terminates when the tree of requests is empty.

6 Conclusions

We have proposed an approach to the development of an intelligent environment for signal processing. Currently existing AI tools are not directly applicable to this field, since they do not capture the fundamental aspects of problems in it. Our main contributions are (a) the clear separation of the two types of knowledge involved in SP problem-solving: processing knowledge and interpretation knowledge, (b) the development of a basic problem-solving model for signal processing, and (c) a proposed architecture for SP problem-solving based on this model. The architecture OCAPI [3], which was developed primarily for supervision of image processing tasks, has many of the features of the proposed architecture, but is not adapted to handle the essentially temporal nature of SP tasks. Future work will involve the extension of the existing OCAPI implementation to handle signal data, to process tasks in flow and mixed modes, to handle different modes of execution control, etc.

References

[1] B.M.Dawant and B.H.Jansen, "Coupling numerical and symbolic methods for signal interpretation," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 1, pp. 115–124, 1991.

[2] V. Clément and M. Thonnat, "Supervision of perception tasks for autonomous systems: the OCAPI approach," in *AIS'92 3rd Annual Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems*, (Perth, Western Australia), pp. 203–212, July 1992.

[3] V. Clément and M. Thonnat, "A knowledge-based approach to the integration of image processing procedures," *CVGIP: Image Understanding*, Vol. 57, pp. 166–184, Mar. 1993.

[4] L.V.R. de Arruda, *Etude d'Algorithmes ...System a Base de Connaissance pour l'identification*, Ph.D. dissertation, University of Nice, Sophia-Antipolis, France, Nov. 1992.

[5] J.E.Larsson, "An expert system for frequency response analysis," in *IFAC 11th Triennial World Congress*, vol. 4, (Tallinn, Estonia), pp. 41–45, 1990.

[6] J.E.Larsson and P. Persson, "An expert system interface for an identification program," *Automatica*, Vol. 27, No. 6, pp. 919–930, 1991.

[7] J.G.Cleary *et al.*, "Knowledge-based systems for the interpretation of seismic data," in *Coupling Symbolic and Numerical Computing in Expert Systems* (J.S.Kowalik, ed.), pp. 231–246, North-Holland, 1986.

[8] J.Hendler *et al.*, "AI planning: systems and techniques," *AI Magazine*, pp. 61–67, 1990.

[9] P. le Guernic *et al.*, "Programming real-time applications with SIGNAL," *Proceedings of the IEEE*, Vol. 79, No. 9, pp. 1321–136, 1991.

[10] M.Haest *et al.*, "ESPION: An expert system for system identification," *Automatica*, Vol. 26, No. 1, pp. 85–95, 1990.

[11] M.Karjalainen and T. Altosaar, "QuickSig: An expert workstation for numeric and symbolic signal processing," in *Signal Processing IV: Theories and Applications* (J.L.Lacoume *et al.*, eds.), pp. 1197–1200, North-Holland, 1988.

[12] M.M.Covell *et al.*, "Computer-aided algorithm design and rearrangement," in *Symbolic and Knowledge-based Signal Processing* (A.V.Oppenheim and S.H.Nawab, eds.), pp. 31–87, Prentice Hall, 1992.

[13] P.Senges *et al.*, "A KBS for CAD and supervision of control systems," in *Proc. First European Control Conf.*, (Grenoble, France), pp. 126–131, July 1991.

[14] S.D.Campbell and S.H.Olson, "An expert system for weather radar interpretation," in *Coupling Symbolic and Numerical Computing in Expert Systems* (J.S.Kowalik, ed.), pp. 329–348, North-Holland, 1986.

[15] S.H.Nawab and V. Lesser, "Integrated processing and understanding of signals," in *Symbolic and Knowledge-based Signal Processing* (A.V.Oppenheim and S.H.Nawab, eds.), pp. 251–285, Prentice Hall, 1992.