CODEN: LUTFD2/(TFRT-7315/1-12/(1986)

Expert Systems for Process Control

Karl-Erik Årzén

Department of Automatic Control
Lund Institute of Technology

May 1986

Department of Automatic Control **Lund Institute of Technology** P.O. Box 118 S-221 00 Lund Sweden	*Document name* INTERNAL REPORT
	Date of issue May 1986
	Document Number CODEN: LUTFD2/(TFRT-7315)/1-12/(1986)
Author(s) Karl-Erik Årzén	*Supervisor*
	Sponsoring organisation STU

Title and subtitle

Expert systems for process control

Abstract

In this paper an expert system is used as a part of a feedback controller which consists of algorithms and logic. The logic is often the dominating part, particularly when different control functions are combined. This logic can be implemented in an expert system. The task of the expert system is to orchestrate the application of different numerical algorithms to the controlled plant in real time. This problem contains element of both monitoring and planning. A prototype implementation is described and experiments with this system are evaluated. A real time expert system architecture is outlined based on experiences from the experiments.

This paper was presented at the 1st International Conference on Applications of Artificial Intelligence in Engineering Practice, Southampton UK, April 1986.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title	*ISBN*

Language English	*Number of pages* 12	*Recipient's notes*
Security classification		

EXPERT SYSTEMS FOR PROCESS CONTROL

Karl-Erik Årzén

Dept. of Automatic Control, Lund Inst. of Technology,
Box 118, S-221 00 Lund, Sweden.

Abstract: In this paper an expert system is used as a part of a feedback controller which consists of algorithms and logic. The logic is often the dominating part, particularly when different control functions are combined. This logic can be implemented in an expert system. The task of the expert system is to orchestrate the application of different numerical algorithms to the controlled plant in real time. This problem contains elements of both monitoring and planning. A prototype implementation is described and experiments with this system are evaluated. A real time expert system architecture is outlined based on experiences from the experiments.

1. Introduction

Process control is an area where expert systems have several natural applications at different levels. An expert system can be used for global alarm and performance analysis at the plant level (Moore et. al. 1984; Yamada and Motoda 1983). At this level the expert system is used on top of an existing process control system. It is mainly used as a tool for the operator. The expert system gives information and advice to the operator. It may tell which set point changes that are needed to increase overall productivity or, in the case of an alarm, where the initial fault has occurred.

In this paper the expert system is instead used as an element of the feedback loop in a single controller. The expert system is no longer merely used as an operator tool. Instead it affects the controlled plant directly. This imposes severe demands on the completeness and consistency of the knowledge used in the system. On the other hand the domain of knowledge; single loop control, is much smaller. This approach will from now on be called Expert control. The idea was first proposed in (Åström and Anton 1984). An expert system has been used by (Sanoff and Wellstead 1985) at the single loop level for control of a self-tuning regulator but still only as an operator tool. The automatic control scene is described in this chapter. The consequences of using an expert system approach are discussed in chapter 2. Chapter 3 describes a prototype forward chaining implementation. The experiments performed with this prototype are described in chapter 4. The results of the experiments are discussed in chapter 5 where an outline of a system better suited for this application is given. The need for an expert system approach increases with increasing problem complexity. Multi-loop or multivariable systems are thus good candidates. The approach taken here, however, has been to start with a simpler SISO problem. This is no serious limitation since the structure of the implementation is the same as in the more complex case.

The scene of automatic control of today is divided into two parts. The theoretical arena where almost all research is done is focused on algorithms and not so much on their practical implementation. Complex algorithms exist, e.g. adaptive algorithms (Åström 1983), that each work well for a certain class of control problem. Common to them are that they often need much a priori information about the controlled system e.g. system order, number of time delays etc, to work well. These algorithms are seldom used in practice except for some pilot projects where much modelling and analysis must be done before the algorithms can be applied.

The other arena is the industrial control practice. The controllers used in industry are simple, PID or more often PI algorithms. The emphasis is more on the safety and ease of operation and on the combination of several loops. The controller should do well during startup and under abnormal operating conditions. It should be possible to safely switch between manual and automatic control etc. General Direct Digital Control (DDC) packages that contain the tools for achieving these goals are available and can comparatively easy be put into operation.

Most algorithms need a safety jacket of logical conditions to work well; see (Clark 1981 and Isermann 1982). These logical networks are often the dominating part of the code of the controller. They are difficult to modify and make the code less readable. Heuristic knowledge of different kind is difficult to include in existing controllers in a natural way. An typical example is the knowledge and rules of thumb the operator has gathered about the actual plant based on long experience of its operation.

2. Expert Control

The idea of expert control is to incorporate a rule based expert system in a feedback control system. The task of the expert system is to orchestrate the application of numerical algorithms to the physical process. The numerical algorithms can be divided into three groups: control algorithms, identification algorithms and supervision or monitor algorithms. The control algorithms may be of varying complexity, from simple PI or relay controllers to more complicated optimal or pole-placement algorithms. The identification algorithms could range from simple algorithms for estimation of e.g. static gain to more complex model based algorithms such as the Least-Squares algorithm (Åström and Wittenmark 1973). The supervision algorithms should detect such things as static errors, alarm level crossings, ringing in the controller output, instability etc. The expert system should decide in which order the algorithms should be applied and their parameter setting. The result of the application of one algorithm increases the knowledge about the physical plant and affects the application of further algorithms. Knowledge of the physical plant is thus successively built up during the operation of the system. The overall goal is to have as good performance as possible of the controlled plant. Another goal is to be able to extract process knowledge from the system.

The combination of an expert system with numerical algorithms for control purposes has many interesting aspects. One obvious advantage is the division between numerical algorithms and branching logic or execution flow logic. The algorithms are coded in as pure form as possible and the logic is implemented as rules in the expert system. This will improve the readability of the code and simplify the development and maintenance of the program. This is particularly important since it will enable persons without programming experience, for instance process operators, to understand and manipulate the program. An expert system implementation will also make it possible to change the logic interactively. It also gives the possibility to interactively interrogate the system by using the explanation facilities of the expert system.

Another aspect is the ease of implementing the safety nets needed for an practical controller. The complexity of the safety net increases with the complexity of the system.

Large nets of logic can be systematically implemented in an expert system. This can be used to make practically usable implementations of complex control algorithms that require large safety nets, e.g. adaptive algorithms.

It is sometimes desirable to use more than one numerical algorithm. Filters on the measured input, different identification algorithms in self tuning controllers, one controller for operation in steady state and one very robust controller for startup are some examples. The combination of algorithms imposes additional requirements on the correctness of the corresponding logic. The usual way to implement this is either to use programmable logic combinated with function block programming or to write a special purpose program. Most control systems based on function blocks have a comparatively low level user interface. The programmer works with logical or analog signals and function blocks with these as inputs and outputs. The syntax is fixed. It is difficult to work hierarchically or to introduce abstractions. Furthermore it is difficult to change the structure on line. In many systems the controlled plant must be stopped and the system must be recompiled. Another possibility is of course to write a special purpose program. This program will however soon be filled with logic statements, difficult to understand and maintain. The basic idea, to combine several algorithms, is however, a valuable asset that is not exploited in todays control systems. A combination of control algorithms each well suited for its purpose is a very good alternative to the nonexisting ideal control algorithm that can handle all possible cases of parameter variations, disturbances, system structures etc. One implication that points in this direction comes from the area of image analysis. When a digital image is analyzed the natural way to do is to use many different algorithms each one specialized on one aspect of the analysis. Examples are feature extracting algorithms of different type and algorithms for image understanding. Startup of adaptive algorithms is a special case of combination of algorithms where expert control might be useful. The expert system can use simple robust algorithms to collect some of the a priori information that the adaptive algorithm needs to work properly.

The last aspect of expert controllers is their implications for the user interface. Todays controllers do not have the capability to fully exploit the a priori information available about a physical plant nor to collect such knowledge from the operation of the system. The knobs on a controller are usually used to set controller specific parameters, e.g. regulator gain, integration and derivation times in a PID controller. On some controllers it is possible to use the knobs to specify the performance of the closed loop system, e.g. bandwidth or overshoot. In both cases this quantitative information might be somewhat unnatural to the operator. With an expert controller it is, however, possible to allow the operator to input plant specific information of a qualitative nature. Process operators always have some information about the physical process of this kind. Examples may be estimates of dominant time constant and gain, if the process has integral action or not, the nature of nonlinearities etc. It is also possible to implement a priori information of a more procedural kind. The process operator may have a certain mode of procedure or rules of thumb each time a larger change of production is performed. This can often be expressed as a set of rules.

One possible use for an expert control system apart from using it as an actual industrial controller is to use it as a testbench. The incremental expert system environment can be used for experiments with, and rapid prototyping of, new control structures.

From an AI point of view this expert system application contains elements of planning and a monitoring. The expert system should plan how the different algorithms should be applied to the controlled process and it should monitor both the plan execution and the actual control. The real time aspect of combining an expert system with a controller is another thing to take into consideration. The numerical algorithms and the expert system must be implemented as separate parallel processes. The reason for this is the different timescales that the two parts operate in. The response time of the

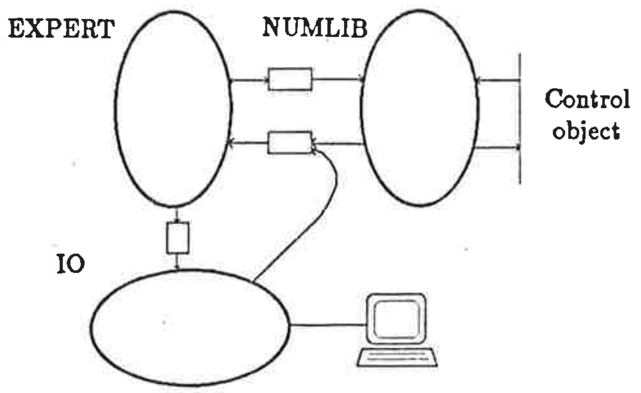

Fig. 1: Process structure. The ellipses represent processes and the rectangles represent mailboxes.

numerical algorithms must match the demands from the physical process while the expert systems rule interpretation is a comparatively slow process. The response time for the expert system can thus be an order of magnitude slower than that of the numerical control algorithm. This poses restrictions on the plants that can be controlled. The fact that most existing expert systems are not developed for use in real time applications is a problem. There are some exceptions like HASP/SIAP (Nii et. al 1982), but most systems are used in an off-line mode.

3. A Prototype Implementation

A prototype implementation of expert control will be described in this chapter. The implementation is done on a VAX 11/780 running VMS. The expert system is implemented in Lisp and the numerical algorithms in Pascal. The reason for choosing Lisp is mainly that it is the traditional AI language with a variety of expert system development tools available. The choice of Pascal was made for convenience because much of our coding of ordinary controllers is done this way. Different algorithms are thus easily available. The Lisp used is the Unix dialect Franz Lisp (Foderaro and Sklower 1981). The software package EUNICE (Kashtan 1982) is used to create a Unix environment under VMS. The expert system part is written in the pure forward chaining expert system shell OPS4 (Forgy 1979). The reason for choosing this was simply the fact that it was available to us and that a prototype was needed quickly to test the basic ideas.

Concurrency is obtained by using the facilities of VMS as a real time operating system. The parallel processes are implemented as VMS subprocesses. The system has three different processes. The numerical algorithm process called NUMLIB is implemented in Pascal. The expert system called EXPERT is implemented in Lisp and OPS4. The third process is the operator interface. This is also implemented in Lisp and is called IO. A process graph of the system is shown in Fig. 1.

Since the numerical algorithm process is the most time critical, it has the highest priority. The processes communicate through VMS mailboxes. A mailbox is a queue where messages can be read or written. An attempt to read from an empty mailbox will cause the calling process to be halted until a message arrives. Halting can be prevented by checking how many messages there are in the mailbox before reading from it. The communication can either be synchronous or asynchronous. In the synchronous mode the process that writes to the mailbox will be halted until the message is read by another process. In asynchronous mode the writing process will continue immediately. This mode is used here. The actual message can be associated with different data structures. In our system a message is a line of text. This simplifies the communication between Lisp processes enormously. In Lisp there is no difference between data and programs; they are both stored as lists. This means that if a message is defined to be a line of text then this message can contain an arbitrary Lisp expression. The process that reads the message can, if it contains a function expression, simply evaluate it. This means that the communication interface between EXPERT and IO is totally free. The interface between EXPERT and NUMLIB must, on the other hand, be specified before compilation. A more detailed description of the processes is given below.

NUMLIB

This process consists of a library of numerical algorithms for control, estimation or supervision. The algorithms are coded in a uniform way, with separate procedures for initialisation, execution, parameter changes and abortion. These parts of the code also reflect how the expert system can interact with the algorithms. The structure of the main program looks like:

```
while true do
  begin
  if message-in-box then read-message;
  for all algorithms do
     if algorithm-is-active then execute-algorithm;
  wait-for-next-sample;
  end;
```

NUMLIB can handle four different message types:

`Start` ⟨algorithm⟩ ⟨initial parameter setting⟩
The algorithms is started. Default parameters are set if no initial parameter setting is given.

`Par` ⟨algorithm⟩ ⟨parametername⟩ ⟨value⟩ ...
The parameters of the algorithms are changed.

`Stop` ⟨algorithm⟩
The algorithm is stopped.

`Execute` ⟨algorithm⟩
The algorithm is executed. It results in a call to the execute section of the algorithm. This is only used for algorithms that do not run periodically.

The expert system can change some of the global variables in NUMLIB. Typical examples are sampling period, set point, or the control variable when the controller is in manual mode. This is done with the message

`Global` ⟨parametername⟩ ⟨value⟩ ...

The messages from NUMLIB to EXPERT are sent by the different algorithms when some condition is fulfilled, e.g. some estimation algorithm is ready. Each message to EXPERT has a priority which reflects its importance. Messages from a supervision algorithm, e.g. an algorithm that checks alarm levels, should have higher priority than messages from estimation algorithms. A design goal for NUMLIB has been to make it easy to update old algorithms or to add new ones.

The division between the algorithms and the expert system is very favourable from the point of view of information flow. During normal operation the flow of information goes between NUMLIB and the controlled plant in the form of measured values and controller output. The expert system is not involved until something significant has been detected by an appropriate algorithm and a message has been sent. If an expert system is used on top of an existing control system then the information must flow between the expert system and the controlled process all the time. This means that the expert system must in some way itself extract the useful qualitative information from the signals that it receives. This is a task which in most cases is better handled by numerical algorithms.

EXPERT

This process is the actual expert system written in Franz Lisp and OPS4. OPS4 is a rule based expert system that uses forward chaining in it's reasoning (Winston 1984). It consists of three parts, the working memory, the production memory and the inference engine.

The working memory contains the data operated on by the rules. It can be viewed both as a database and a scratch pad. The working memory is an unordered collection of objects without repetitions. All elements in the memory are constants. Each element has a time tag that tells its age.

The production memory contains the productions or rules of the system. The syntax of a rule looks like:

```
Rulename
     ("condition element" ...
  -->
      "action" ...)
```

The condition elements are patterns which are matched against the contents of working memory. The rule is satisfied if all the condition elements in the rule are matched. The actions can then be performed. The patterns may contain matching variables.

Most actions are executed for the purpose of changing working memory. This is done with the actions (<ADD> "description") and (<DELETE> "description"). There are also actions for writing to and reading from the terminal. OPS4 provides two actions through which the production memory can modify itself. The action <BUILD> adds a new rule and <EXCISE> removes rules. It is also possible to have user written actions.

The inference engine works in a recognize-act cycle with three phases: match, conflict resolution and act. During the match phase, all the rules that are satisfied are found. During conflict resolution one satisfied rule is selected. The actions in the rule are performed during the act phase. The recognize-act cycle and the program ends when no more rules are satisfied. The OPS4 shell can be made to run continuously in real time by using the possibility to add the constant RESTART to the working memory when no more rules are satisfied. A rule with RESTART as its condition element is then satisfied. The action of this rule is a user written function that checks the incoming mailbox. If this mailbox is not empty it reads all the messages and inserts them into an internal queue according to their priority. When the mailbox is empty the first message is read from the internal queue. If both the queue and the mailbox are empty then the process waits for incoming messages. The internal queue is used to simulate mailboxes where the messages are inserted according to priority rather than first-in first-out order. The incoming messages are either elements to be entered in the working memory or Lisp functions to be evaluated. When new elements are inserted into the working memory new rules match and the recognize-act cycle starts again.

The organization of the rules in the production memory is done according to the context or state in which the rule is used. Most rules are only meant to be used when the program is in a certain state e.g. startup or pid-control. This is implemented by having one condition element in these rules that looks like (State is "actual state"). An example of a rule is

```
Rule5
     ((State is startup)
      (Goal is pidcontrol)
      (Pid parameters available)
    -->
      (<DELETE> (State is startup))
      (<ADD> (State is pidcontrol)))
```

Rules that are used independent of state can look like.

```
Rule12
     (((State is =X)
      (Alarm has occurred)
   -->
      (<DELETE> (State is =X))
      (<ADD> (State is alarm) (Oldstate is =X)))
```

The symbol =X denotes a variable that can be matched against any constant.

IO

This is the operator interface to the expert controller. It is implemented in Lisp. Much is gained by this. The interactive environment is achieved automatically and the message format between IO and EXPERT is free. The commands available can be divided into two groups. The first group are the usual commands available to a controller, e.g. commands for changing parameters or commands for switching between different operating modes. These commands are implemented as Lisp functions that send appropriate messages to EXPERT.

The other group consists of commands that have to do with the internal state of the expert system. These are only intended for experienced users. Examples are commands for inspecting the working memory of the expert system, commands that start tracing of the rule execution, etc. A special group of these commands are the commands for adding, deleting and editing of rules on-line. A copy of the rulebase is kept in IO. When rules are added or deleted, the operation is first performed locally. The new rule is then transferred to EXPERT. When a rule is edited, the rule is first deleted in EXPERT. The edited rule is then added. The editing can be done either with a Lisp structure editor or with a screen oriented text editor. There are also commands for searching among the rules with the help of pattern matching. Finally there is the highest level of interaction, namely the possibility to send an arbitrary Lisp function to EXPERT for evaluation.

4. Experiments

The prototype system has been used for experiments with a "smart" PID controller. The controller has auto-tuning and gain-scheduling. The tuning is based on the Ziegler-Nichols auto-tuner (Åström 1982 and Åström and Hägglund 1983). This method uses

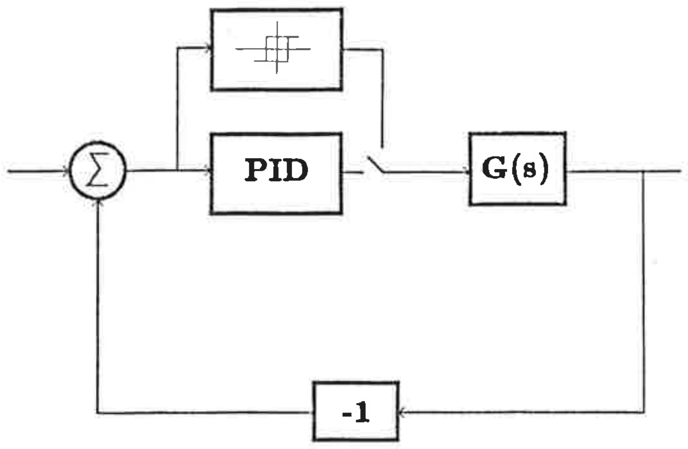

Fig. 2: Ziegler-Nichols auto-tuner. G(s) denotes the controlled process.

relay oscillations to determine the parameters of the PID controller. See Fig. 2. When the control loop is closed with the relay the controlled signal starts to oscillate. The steady state oscillation frequency corresponds to the point where the Nyquist curve of the controlled process crosses the negative real axis. This point gives the critical gain and the critical period which are then used to compute the PID parameters. The algorithms needed for this are a PID algorithm, a relay algorithm, and a oscillation analyzer that measures the oscillation period and amplitude. A noise-estimator algorithm is used to measure the noise level. This information is used when the relay parameters are computed.

From an operational point of view the controller has three modes. In manual mode the process is controlled by the operator. The tuning is performed in tuning mode and the PID controller is run in PID mode. The operator has commands for switching between modes. The operator has also commands for changing the PID parameters and the relay parameters i.e. step size and hysteresis. It is also possible to change the contents of the working memory and the rules the way described in chapter 3.

The rules in the system are grouped according to the context or the state when they are used. The groups are

Noise estimation rules: Contains rule about starting and supervising the noise-estimator algorithm.

Relay oscillation rules: Contains rules related to the relay experiment. These are rules for determining the relay parameters. The relay and the oscillation analyzer are started. Rules are used to adjust the relay parameters and determine when the oscillation is steady. The critical gain and critical period are computed.

Parameter computation rules: Rules for computing the PID parameters from the oscillation measurements.

PID rules: The PID algorithm is started and supervised. Rules are used in the gain-scheduling.

Command decoding rules: Contains rules to take care of operator commands. Examples are rules for mode switching and rules for parameter changes.

The working memory contains a gain-scheduling table whose entries are set points and the corresponding PID parameters. When the set point is changed new PID parameters are sent to the controller. These parameter values are obtained by tuning the regulator at different operating points. Gain scheduling is useful when the dynamics of the controlled process changes with some variable e.g. operating point. Building up and storing information in tables is an effective way to represent knowledge about the controlled process. This method can be extended. Examples of different uses of tables are given in (Åström and Anton 1984). The working memory is used for knowledge about the plant that has been derived during operation. It is also used as a local memory for the different rule groups.

The experiment is started with the system in manual mode. The operator controls the plant manually until the system is in steady state at the desired operating point. The tuning is started by changing to tuning mode. When the tuning is finished the operator has the option to accept or neglect the new PID parameters. If they are accepted the system changes to PID mode. The operator can then change from PID mode to manual mode or tuning mode.

A laboratory model of a tank system was used in the experiments. The control objective was to keep the level of the tank at a reference value and to change it on demand. The measured output is the water level in the tank and the control input is the voltage to a pump that determines the inlet flow. As an alternative to control a physical object a simulation program Simnon (Elmqvist 1975) has been interfaced to the system. This program can simulate nonlinear differential or difference equations. The program has been modified so that it simulates in real time. This arrangement is very convenient for experimentation since it gives a simple way to test many different cases.

5. Experiences

The experiments performed with this system have been promising. The expert control approach gave a cleaner auto-tuner implementation than previous implementations based on Pascal and PLM (Åström 1983). A major benefit was derived from the clear separation between algorithms and logic. Another benefit was the ease of changing the logic in the system. An example of this is that the addition of gain scheduling to the system required only the addition of about 10 new rules. The expert control approach is thus very promising in a testbench for rapid prototyping.

The rulebase of the first experiment contained approximately 70 rules. This may seem large. It can be explained in two ways. One is that a controller like this with several different operating modes contains more logic than is believed at first sight. The second and more important reason is that the expert system shell OPS4 is not ideal for expert control. Expert control contains a strong planning element that is not supported by OPS4. An exampel is the tuning phase of the auto-tuner. It can roughly be described as a sequence of algorithms applied to the controlled process. First a noise estimator algorithm is used to collect information about noise properties of the system. This information is then used to set the relay parameters. When a steady state oscillation is obtained the PID parameters are determined from the wave form of the oscillation. The rules are divided into groups for each stage of this plan. In OPS4 there is no possibility to collect rules into groups. The only way to achieve sequencing is to explicitly program it into the rules i.e. use different contexts. A result of this is that approximately half of the number of rules in the system are dominated by the control knowledge i.e when to apply the rules instead of the actual domain knowledge.

Another disadvantage with OPS4 is that it only allows forward chaining. The data driven computation in forward chaining is suitable in the planning stage of expert control where data in the form of significant events are sent to the expert system from the numerical algorithms. In the monitoring phase, however, the problem can be stated as a diagnosis problem. Backward chaining is more appropriate for this type of problems.

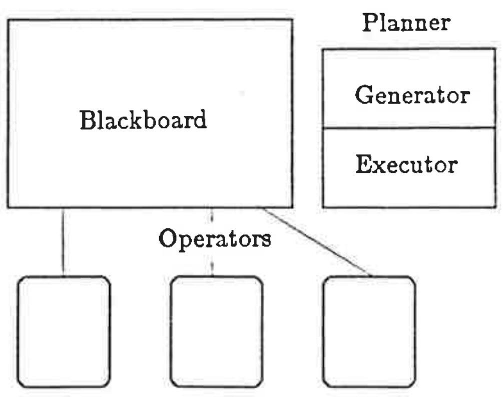

Fig. 3: An expert control architecture.

Further OPS4 has no explanation facilities and no certainty calculus.

The most important drawback with OPS4, which it shares with most of the other existing expert system shells is that it is not designed for real time operation. Real time expert systems is still an area where much research is needed before a true expert control system can be built.

An expert control architecture

A possible expert control architecture might be built around a blackboard architecture (Erman et. al. 1981) and a planning module. The blackboard contains the database that is available to several rule groups or knowledge sources. These knowledge sources could be thought of as operators with a goal and certain preconditions that have to be satisfied for the operator to be applicable. The operators contains the domain knowledge for a certain task of the expert controller. An operator can be associated with an algorithm. Examples of tasks are computation of PID parameters from oscillation measurements or supervision of the oscillation analysis algorithm. The planning module can be divided into two parts; a plan generator and a plan executor. The plan generator generates a plan by comparing the final goal with each operator goal in a backward fashion. This plan will contain both operators in sequence and in parallel. The plan executor, e.g. the agenda mechanism in a blackboard system, then executes the plan step by step. The structure is shown in Fig. 3. This planning problem does not satisfy all the requirements needed for an existing domain-independent planner system (Sacerdoti 1977, Wilkins 1983 and Cohen and Feigenbaum 1982). It is primarily the so-called STRIPS assumption that is violated. This assumption implies that the goal is always achieved when an operator is applied. This is not always the case when an operator is applied in noisy and uncertain environment such as in expert control. A result of this is that the plan executor and the plan generator have to interact after each step to replan if necessary.

This architecture has features of a common real time operating system. The operators are the equivalents of concurrent processes and the plan executor is the equivalent of a scheduler. In a real time operating system the processes can wait a certain time or for a certain event. This can be handled in the expert control architecture as well by assigning a state to each operator that has the values running, active or waiting. When an operator calls the function waittime in the action part of a rule it will be marked waiting by the plan executor. The operator is activated when the time over. This means that the expert system will in a way be able to take time into account in its reasoning. An operator can also wait for a specified element to occur in the database, e.g. wait for a certain message from the algorithm part. A real time expert system architecture like this requires the solution of several difficult AI issues to be generally applicable. Examples of these issues are nonmonotonous and temporal logic (McDermott and Doyle 1980 and McDermott 1982) and planning in time (Allen 1984 and Vere 1983). In the expert control application however, this architecture is promising. Implementation of a system along these directions is in progress. The system will be used for experiment with an "intelligent" PID controller. This controller should be able to decide if the plant can be controlled by a proportional controller only or if integral and derivative actions are needed. The controller should also be able to judge when a PID controller is not sufficient and a more complex controller is needed.

6. Conclusions

A new kind of controller can be developed and implemented by using the expert control approach. The separation between numerical algorithms and logic results in a flexible and transparent system. The system can be used as an interactive testbench for new control ideas. The expert system technique provides a new type of user interface where it is possible to include the domain knowledge of experienced process operators. A prototype implementation of an expert control system has been built. The experiments has showed that the basic ideas hold. They have also showed that existing expert system shells are not suitable for a true expert controller. An architecture is proposed that better suits the application.

This project is part of the Computer Aided Control Engineering Project at the Department of Automatic Control, Lund Institute of Technology. It is supported by STU, the National Swedish Board for Technical Development.

7. References

Allen, J.F. (1984): Towards a General Theory of Action and Time. *AI Journal* **23** pp. 123-154.

Åström, K.J. (1982): A Ziegler-Nichols Auto-tuner Report TFRT-3167. Department of Automatic Control, Lund Institute of Technology.

Åström, K.J. (1983): Theory and applications of adaptive control. *Automatica* **19**, pp. 471-486.

Åström, K.J. (1983): Implementation of an Auto-Tuner using Expert system Ideas. Report CODEN: LUTFD2/TFRT-7256. Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Åström, K.J. and J.J. Anton (1984): Expert Control, Proc. 9'th IFAC World Congress, Budapest Hungary 1984.

Åström, K.J. and T. Hägglund (1983): Automatic tuning of simple regulators. Preprints IFAC workshop on Adaptive systems in control and signal processing, San Francisco, CA.

Åström, K.J. and B. Wittenmark (1973): On self-tuning regulators. *Automatica* **9**, pp. 185-199.

Clark, D.W. (1984): Implementation of adaptive controllers. In Harris and Bildings. (Eds), *Self-tuning and Adaptive Control.* Peter Peregrinus, U.K.

Cohen, P.R. and E. Feigenbaum (1982): STRIPS and ABSTRIPS. Chapter 15B of *The Handbook of Artificial Intelligence* Vol. 3, William Kaufman Publishing, Los Altos, Ca.

Elmqvist, H. (1975): SIMNON, An Interactive Simulation Program For Nonlinear Systems, Report TFRT-3091, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Erman, L.D., P.E. London and S.F. Fickas (1981): The design and an example use of HEARSAY-III. Proc. IJCAI 7 pp. 409-415.

Foderaro, J.K., K.L. Sklower and K. Layer (1983): The Franz Lisp Manual.

Forgy, C.L. (1979): OPS4 User's Manual. Tech. Rep. CMU-CS-79-132, Department of Computer Science, Carnegie-Mellon University, USA.

Isermann, R. (1982): Parameter adaptive control algorithms – A tutorial. *Automatica* **13**, pp. 513-528.

Kashtan, D.L. (1982): EUNICE: A system for porting UNIX programs to VAX/VMS, Artificial Intelligence Center, SRI Internationl, Menlo Park CA.

McDermott, D. (1982): A temporal logic for reasoning about processes and plans. *Cognitive Science* **6** pp.101-157.

McDermott, D. and J. Doyle (1980): Non-monotonic logic I *AI Journal* **13**.

Moore, R.L., L.B. Hawkinson, C.G. Knickerbocker and L.M. Churchman (1984): Expert systens applications in industry. Proc. ISA Int. Conf. Houston.

Nii, H., E. Feigenbaum, J.J. Anton and A.J. Rockmoore (1982): Signal-to-Symbol transformation: HASP/SIAP case study. *The AI Mag.* Vol. **3** No. **2** 1982.

Sacerdoti, E.D. (1977): *A Structure For Plans and Behavior.* New York: American Elsevier.

Sanoff, S. P. and P. E. Wellstead (1985): Expert Identification and Control. Proc. IFAC Identification and System Parameter Estimation pp. 1273 - 1278, York, UK.

Vere, S. A. (1983): Planning in Time: Windows and Durations for Activities and Goals. *IEEE Trans. on PAMI,* Vol. **5**, No. **3**, pp. 246-267

Wilkins, D. (1983): Domain-Independent Planning: Representation and Plan Generation. Tech. Note 266R, Menlo Park, CA: SRI International.

Winston, P.H. (1977): *Artificial Intelligence,* Addison-Wesley, Reading, Mass.

Yamada, N. and H. Motoda (1983): A Diagnosis Method of Dynamic System Using the Knowledge on System Description. Proc. Eighth IJCAI pp. 225-229, Karlsruhe, W. Germany.