CHAPTER 2. LITERATURE REVIEW

2.1 Real-Time Expert Systems

Early work in expert systems was done with the preferred platform of the artificial intelligence community: mainframe computers, and the LISP language. This legacy can be seen today in such widely-used expert systems as OPS5, KEE, Kappa, Nexpert, and Personal Consultant Plus, to name just a few. The mainframe computer has been replaced by a LISP machine, a workstation, or a high-end PC, but massive processor power and slow performance remain the norm. However, in the last decade, there has been increasing interest in real-time[1] expert systems for embedded processors (Laffey et. al. 1988).

One of the first such systems was Hexscon, whose design goals included

> "(1) a capacity of 5000 rules in a microcomputer system with 512K memory, (2) a response time of 10 ms to 100 ms, (3) the ability to handle many objects (about 1000), and (4) the ability to continue functioning despite a lot of uncertainty" (Wright et. al. 1986).

Hexscon uses "progressive reasoning" to produce a quick approximate solution with conventional logic, followed by up to three successive stages of knowledge-based refinement as time allows. IF-THEN rules are compiled into a more efficient form for evaluation; procedural code can be referenced by the knowledge base. Both forward- and backward-chaining are employed by the inference engine. Uncertainty is supported by "belief" and "confidence" values, which are also used to resolve conflicting chains of reasoning. A simple "frame" mechanism allows rudimentary temporal reasoning about "past," "present," and "future," although past knowledge is stored off-line (on disk). Hexscon was implemented in Pascal on 8085 and 8086 microprocessors; when configured for maximum speed on the 8086, knowledge-based response times on the order of 0.25 - 0.5 seconds were achieved.[2]

MUSE is another package intended for real-time applications on embedded hardware (Reynolds

1. In the context of expert systems, "real-time" has been used to mean either "fast" or "deterministic." For the purposes of this survey, no distinction between the two is necessary.

2. It is not known how many Logical Inferences per Second this represents, but about 250 rules were used.

1988). It is written in PopTalk, an object oriented variant of POP-2; the resulting object code for the PopTalk virtual machine is interpreted on the target hardware (currently, a 68000 single-board computer). MUSE implements a forward-chaining inference engine similar to OPS, using a modified Rete (Forgy 1982) pattern matcher. A backward-chaining inference engine is also supplied. Procedural code may be freely used to qualify patterns, express rule firings, and perform input and output.

REAL-OPS is a real-time implementation of the forward-chaining OPS5 production language. It is written in Forth rather than LISP, and uses an improved "synchronous" garbage collection scheme to avoid unpredictable delays, for a substantial speedup:

> REAL-OPS running on a HP236 (8 MHz MC68000 cpu) runs a seven-tower problem in 25.7 s, while the LISP-based OPS5 takes 2 min on the MicroVAX II, 60 s on a VAX 11/780, and 18.9 s on Texas Instruments' Explorer. (Dress 1986)

This amounts to approximately 10 inferences per second. REAL-OPS includes provisions for external events (interrupts), input and output, and calls to procedural code.

Another OPS5 variant, OPS83, was used in the Embedded Inference Engine (EIE). OPS83 is a forward-chaining commercial package written in C, which is "five to 10 times faster than the equivalent LISP-based production system," and allows linkage to procedural C routines (Skapura 1989). EIE applies the concepts of concurrent programming to OPS83, to create "cooperating production systems," each of which works on a different subproblem (albeit coexisting a single CPU). This partitioning of the problem, combined with the performance of OPS5, yield in EIE benchmark speeds on a 10 MHz PC/AT which are 8.76 times faster than OPS5 on a VAX.

FORPS is a forward-chaining production system that was developed for a 68000 microprocessor.[3] It also uses compiled knowledge, with the ability to call procedural code from within rules. An explicit priority mechanism resolves rule conflicts. FORPS has been benchmarked at up to 221 inferences per second, but since the inference engine operates by exhaustive testing, speed is inversely proportional to the number of compiled rules in the knowledge base: "a system with more than 700 rules would require at

---

3. Implementations also exist for the 8086.

least .5 seconds per inference" (Matheus 1986).

At the performance extreme, speeds of 1095 Logical Inferences per Second (LIPS) have been attained with FORTES, a forward-chaining system for the IBM PC (Redington 1986).

An expert system described by (Lewis 1986) is noteworthy because it pioneered the use of a linked list for all rules dependent on a given antecedent[4] (as well as a list of all rules contributing to a given consequent). This provides fast forward- and backward-chaining. This system also used a compiled rule base, with procedural code.

The Prolog language is often used to implement rule-based systems. One commercial product, PDC Prolog, allows routines in a procedural language (C, Pascal, or assembler) to be called from Prolog, and allows Prolog predicates to be called from the foreign language (Murphy 1993, Elder 1993).

The Super expert system (Morizet-Mahoudeaux 1996) addresses almost the entire "wish list" of (Laffey et. al. 1988) for a real-time expert system.[5] Its backward chaining inference engine is capable of managing "rule interruptions, focus of attention, asynchronous event or input handling, multitasking, and temporal reasoning." Super translates a knowledge base to C++ source code, which can then be compiled with procedural C or C++ code for an embedded processor. Super is integrated with a real-time multitasking operating system; inputs and interrupts are communicated as messages to the inference engine. These can start the inferencing process, or rules can be scheduled for periodic activation. Events which occur during an inferencing cycle can suspend and restart the inferencing process. This process is quite fast:

> In the worst case, an Intel 80486/25-MHz microprocessor-based computer can completely analyze a compiled networked knowledge base of fifty thousand rules in less than 0.6 seconds. (Morizet-Mahoudeaux 1996)

Super is noteworthy in that knowledge is explicitly expressed as a logic network, in the manner of (Siddall 1990); this network is also used to enforce data consistency. A limited set of temporal relations is recognized by the system: current time, duration, frequency, and temporal correlation (succession of

4. This is equivalent to the "dependency list" described in this thesis.
5. See Chapter 3.

events).

Time Constrained Reasoning

Hexscon (described previously) uses "progressive reasoning" to achieve a solution within a given time constraint. This technique, also called "any time" search[6] (Lesser 1990), has also been used by (Broeders et. al. 1989), to successively apply up to five rule bases of increasing sophistication to a problem. Their process control expert system also achieves several other goals of a real-time expert system, including mixing of procedural and inferential code, a compiled rule base, direct input and output, and moderately high speed (e.g. response times ranging from 55 to 430 msec).

Rather than evaluate and then discard approximate solutions, an alternative technique attempts to decide in advance the best solution strategy for a given time constraint. This reasoning about the reasoning approach (or "meta-reasoning") may elect several approximations:

> (1) approximate search strategies explore a smaller portion of the search space than would be the case during normal processing, (2) data approximations provide an abstract view of data resulting in a simpler space being searched, and (3) knowledge approximations simplify the knowledge being applied in the system so that the search space can be explored quickly. (Lesser, Pavlin, and Durfee 1988)

Coping with these approximations requires a number of new knowledge representations, e.g., "filters" to eliminate unprofitable hypotheses and strategies (Decker, Lesser, and Whitehair 1990). This system was very effective in meeting specified deadlines for the Distributed Vehicle Monitoring Testbed (DVMT).

Another system which tries to produce a single, best solution within a given time constraint is RUM. Rules are written in KEE, using "both procedural and declarative knowledge" (Bonissone and Halverson 1990). The static rule set is then translated into a directed acyclic graph for efficient execution (in the same manner as Super, described above); the translator also computes static performance metrics for the network. The target (embedded) hardware receives this rule network, forward- and backward-chaining inference engines, and a metacontroller. The metacontroller uses the performance metrics to evaluate the best solution strategy for a given time constraint. The knowledge base is also statically

---

6. So called because a valid result can be obtained at any time after the first, approximate solution.

partitioned into subsets, allowing focus of attention. Multiple tasks with different priorities and deadlines are managed through an agenda, and interrupts can supersede the executing task and update the knowledge base. Although RUM is a single-processor implementation, its authors note that the delivered software is suited to distributed execution over multiple processors.

A different approach to meeting real-time constraints is taken by (Hayes-Roth 1990). The reasoning process is divided into small, well-defined operations. An agenda manager selects a subset of operations which are candidates for execution, depending on, for example, current deadlines or the occurrence of high-priority events. A scheduler then executes the highest priority operation from the (possibly incomplete) agenda. Scheduling and execution are well-bounded; thus the problem of controlling real-time performance becomes one of devising a well-bounded agenda manager.

Fuzzy Logic

By far the most explosive growth in real-time knowledge-based systems has been in the realm of fuzzy logic (Kosko 1997). A recent survey observed:

> For the past few years, particularly in Japan, approximately 1,000 commercial and industrial fuzzy systems have been successfully developed. . . . The most successful domain has been in fuzzy control of various physical or chemical characteristics such as temperature, electric current, flow of liquid/gas, motion of machines, etc. (Munakata and Jani 1994)

The authors point out that most, but not all, fuzzy systems are knowledge-based systems.

When fuzzy rules are used for control, they are generally applied to very simple and limited problems, such as temperature control or automatic camera focusing, which can be solved with at most a few dozen rules. Typical performance of such systems might be

> [An OMRON FP-3000] dedicated-hardware fuzzy processor takes 650 microseconds to process a system of 20 rules with five antecedents and two consequents. A similar system would take up to 8 milliseconds on a 2 MHz 68HC11 . . . (Sibigtroth 1991)

Like the aforementioned FORPS system, the software fuzzy inference engine will usually just cycle through all of the rules in its knowledge base. This implies that inferencing time is inversely proportional to the number of defined rules. Perhaps for this reason, complex fuzzy systems are still rare (Sperry

1993).

2.2 Expert Systems in Control

Most of the aforementioned real-time expert systems were intended for some real-time application, such as data analysis or process control, but their published description focused on the expert system. What follows are descriptions of control applications which have used expert systems. This is but a sampling of the very large number of real-time control applications which have been subjected to AI techniques.

(Laffey et. al. 1988) tabulate 43 applications of real-time knowledge-based systems, including 14 process control applications.

Even a rudimentary expert system can outperform a human operator at simple control tasks. (Zhang and Grant 1990) describe a nine-rule system for the control of an inverted pendulum,[7] and compare its performance to that of human test subjects.

Starting from a rule-based solution to the inverted pendulum, Sammut and Michie evolved a rule-based attitude controller for an orbiting satellite. The final strategy involved 45 rules (15 rules for each control axis), and required no knowledge whatever of the dynamic model of the satellite. In tests with a "black-box" simulator,

> . . . the amount of propellant used to bring the system under control compared favorably with the propellant use of a controller developed using traditional means and having full knowledge of the differential equations used in the model . . . (Sammut and Michie 1991)

To study the feasibility of expert systems for process control, (Francis and Leitch 1985) devised a nonlinear test problem consisting of two coupled water tanks: "a system which, although controllable, is surprisingly difficult to predict." The Artifact expert system, with a knowledge base of 21 rules, performed comparably to a PID feedback controller.

MCM is a material composition management system for chemical manufacturing, responsible for process monitoring, situation assessment, action planning, and plan monitoring. It is based on HCVM, a

---

7. A rigid pole hinged to a cart; the pole is to be balanced upright by moving the cart.

special version of Schemer, a real-time, event-driven, object-oriented expert system. Goals and plans are managed, and events are monitored, using an architecture very similar to a blackboard. MCM executes on a Lisp machine and communicates directly with separate (non-inferential) process control computers. It has successfully diagnosed and treated manufacturing process problems, but in an environment that admittedly does not demand great speed: "Admissible response times of the process management system . . . are on the order of hours." (D'Ambrosio et. al. 1987)

Shallow knowledge (heuristic reasoning) is unable to cope with unforeseen circumstances, so many plant controllers also incorporate deep knowledge (model-based reasoning). MCM, just described, is one such system. Another, to control power plants, is described by (Suzuki et. al. 1990). Its Shallow Inference Subsystem (SIS) is a straightforward rule-based system; while the Deep Inference Subsystem (DIS) uses a frame-based Operation Generator to plan corrective actions, and a fuzzy-logic qualitative simulator to predict the likely outcome:

> A qualitative simulation based on a qualitative model is useful because qualitative modeling is easier and because the qualitative reasoning process better matches to the thinking process of a skilled human operator. (Suzuki et. al. 1990)

Unlike a quantitative simulation, a qualitative simulation cannot provide data suitable for numerical plant control; but it can provide information to a heuristic controller such as the SIS.

EXPRESS is a commercial process control system using a rule-based inference engine (Rather 1993). One successful application has been the automation of a lime kiln (Anway 1994).

Expert systems using progressive reasoning have been applied to environmental control (Kim and Zeigler 1990).

2.3 Distributed Expert Systems

Blackboard Systems

Most expert system research has involved single CPUs, although recently, distributed expert systems have become of interest. The prevailing model for multi-processor experts, by far, is the "blackboard" (Dodhiawala et. al. 1989). In such a system, the blackboard is readable and writeable by all

of the individual experts. Each can post hypotheses for other experts to examine, or results for other experts to use. (Corkill 1991; Englemore, Morgan, and Nii 1988)

One such system is the Distributed Vehicle Monitoring Testbed, mentioned previously. The DVMT is an artificial test problem:

> . . . the purpose of building the testbed is to evaluate alternative distributed problem-solving network designs rather than to construct an actual distributed vehicle monitoring network . . . (Lesser and Corkill 1988)

Tests were performed with a network of nodes[8], each running the Hearsay-II system. Hearsay-II allows multiple "knowledge sources" to contribute to a problem. For DVMT, "communication knowledge sources," which exchange hypotheses and goals with other nodes, were added.

A simple three-processor system has been developed by Park for real-time applications:

> The concurrent EXPERT-5 consists of two EXPERT-5 shells running on separate M68000-based co-processors mapped into the memory structure of a third EXPERT-5 based blackboard controller running under MS-DOS on a pc. (Park 1986)

This is a master-slave hierarchy using tightly-coupled processors, with the blackboard being managed by the master. No performance figures were published for the three-processor combination, but it was noted that each 68000 processor achieved standalone speeds of 9000 LIPS using the EXPERT-5 shell.

Blondie-III uses five cooperating blackboard systems to solve a telecommunications configuration problem (van Liempd, Velthuijsen, and Florescu 1990). Each agent has a separate blackboard, but all of the blackboards are globally visible. A set of distributed nodes is simulated on a Sun-3 workstation by making each node a separate Unix process, and confining all communication to interprocess messages.

(Larner 1990) describes DOSBART, a distributed real-time control system, one of whose goals is "to provide a single blackboard abstraction for control system design that can be partitioned into constituents running on multiple processors." Knowledge is distributed: stored where it is used most, but globally available to all nodes. "Data Server" objects,[9] in addition to storing knowledge, can perform dependent functions such as input, output, timed execution, and history recording. Interrupts are

---

8. In the test system, this network was simulated on a single computer.
9. Analogous to the "fact" objects described in this dissertation.

supported, as are "Trigger Activities" which cause forward-chaining, and "Theorem Provers" which implicitly backward-chain. Implemented with the Common Lisp Object System, DOSBART requires LISP processors or high-end workstations (e.g. SPARCstations).

Non-Blackboard Systems

(Green 1987), in discussing the limitations of blackboard systems, observes that the control task (scheduling knowledge sources based on the current state of the blackboard) is a limiting factor of system performance. His Activation Framework (AF) system uses messages, rather than a blackboard, for communication between many inferencing units, called AF Objects (AFOs).[10] Messages may be sent automatically (forward chaining), on demand (backward chaining), or "suggestively" (to accumulate evidence in the absence of complete data). An "activation level" gives priority to AFOs having high-priority messages or many lower-priority messages. AF also has some ability to reason temporally (sequence and relative timing of events), and can reason with uncertain data. The distributed AF system is written in C, and is simulated on a single multitasking CPU, using interprocess messages for communication.

The Distributed AI Shell (DAIS) operates on a network of MS-DOS PCs. It dispenses with the control mechanism of blackboard systems:

> The DAIS network is a collection of AI applications interconnected by DAIS primitives. . . . DAIS does not provide coordination (central or distributed) among agents; instead, it provides primitives for interaction. Each agent can interact with any other DAIS agent driven by individual need. (Kannan and Dodrill 1990)

Each PC can support a single DAIS "agent." Communication between agents is via connectionless network messages, although a request-response mechanism is supported under this protocol. DAIS is a frame-based system implemented in C; both facts and rules are represented as objects. DAIS applications "can manipulate the entire KB[11] distributed over a network," however,

---

10. The individual experts may internally use a blackboard architecture, however.
11. Knowledge Base.

> Object behaviors are actually functions (sets of instructions), and exchanging such machine-dependent units present very tricky problems -- including compatibility at the machine instruction level, and dynamic loading of compatible functions.

Since rules (being objects) can be modified dynamically, the usual pattern-matching optimizations (e.g., Rete) are not applicable. Performance measurements for DAIS were not published.

The Distributed AI Toolkit (DAIT) supports intercommunication between independent reasoning agents, by defining languages and protocols for data sharing. These protocols are based on the Product Data Exchange Specification (PDES), the ISO specification language Express, and a new Agent Manipulation Language (AML). DAIT knowledge sources use the CLIPS expert system, and run on Sun 3s, 4s, and VAX workstations. "DAIT employs predicates for real-time control, distributed processing, and fault tolerance . . . " (Goldstein 1994) No applications were described.

A system intended for embedded microprocessors is described by (Hendtlass 1991). Knowledge is distributed across nodes of a network, and each node operates an expert system which can use local or global knowledge. Each node's expert system may use fuzzy knowledge, and may also employ a subordinate neural network. Rule "subsets" allow focused attention on part of the rule base for improved evaluation speed. Inter-node messages are sent as text (specifically, source code for an application-specific language), allowing a heterogeneous network of mixed CPUs.

HESCPC is a hierarchical society of experts which advise the operator of a nuclear power plant, and also advise the designer of process control hardware. (Ionescu and Trif 1988) It is implemented using an expert system shell, Reshell, which has been "parallelized" on a VAX cluster consisting of one VAX 11/780 and eight MicroVAX's. Reshell supports both frames and production rules, and allows declarative knowledge to be written in M-Prolog, and procedural knowledge in FORTRAN.

The production system OPS5 has been implemented on parallel processors (Acharya, Tambe, and Gupta 1992). This is an attempt to "parallelize" a single computationally-intensive program, rather than to create a society of independent experts. Nevertheless, it does illustrate one mechanism for distributing an expert system.

2.4 Temporal Reasoning

Temporal logic as a field of mathematics is not new, and several books have been devoted to the subject. Its application to computer science is more recent, and mostly devoted to specification of computer programs and systems.

(Galton 1987) cites several examples of temporal logic in Artificial Intelligence, such as the "Time Specialist" of Kahn and Gorry. However, all of the examples cited represent attempts to explore models of temporal reasoning; that is, they were research vehicles and not applications.

BLOBS is an object-oriented blackboard system which addresses some of the requirements for a temporal reasoner: the ability to reason continuously with a continuous flow of data, the ability to obsolete old data, and the ability to reason within a time limit. BLOBS also allows rudimentary reasoning *about* time, by providing a system clock and allowing actions to be scheduled for specific times. There is no explicit representation of mathematical temporal logic. BLOBS is written in POP-11, and has been applied to the real-time problem of radar analysis. (Zanconato 1988)

A more complete temporal reasoning capability has been added to LES, the Lockheed Expert System shell. This frame-based expert system stores attribute values (data) with time tags. Rule antecedents can be written to include the temporal relations ANYTIME, ALLTIMES, or JUST, using the time qualifiers AT, DURING, BEFORE, and AFTER an absolute or relative time. Adding temporal reasoning to LES caused modest increases in data storage requirements and processing time:

> . . . storage time increased by about 20 percent. Retrieval time increased by a factor of eight, and total time increased by a factor of two (for monitoring that uses temporal reasoning). (Perkins and Austin 1990)

Temporal LES has been used to diagnose real-time telemetry data from the Hubble space telescope.

The MOLTKE expert system has been extended to handle a specific temporal problem, Temporally Distributed Symptoms (TDSs) in diagnosis. (Nökel and Lamberti 1991) MOLTKE's representation language allows declaration of expected temporal relationships between events. The Temporal Matcher then examines the input data (possibly in response to test stimuli generated by the

expert system) until one such pattern is satisfied, in much the same manner as the match-resolve cycle of many expert systems. This can diagnose success or failure in a time- or sequence-sensitive test. The non-temporal functions of MOLTKE remain fully usable, with no degradation of performance.

(Tétreault, Marcos, and Lapointe 1992) describe a limited temporal facility for a Qualitative Simulation (QS) system. Simulation variables can be constrained at particular times or over time intervals; testing for temporal consistency reduces the number of branches in the simulation. While not strictly an expert system, the QS algorithm is similar. For example, this system is coded in Prolog.

Hexscon, AF, and to a greater degree the expert system Super (all described above), provide limited support for temporal reasoning.

2.5 Accelerator Control Systems

General Accelerator Controls

Accelerator laboratories have long used minicomputers and microcomputers for data acquisition and analysis. In recent years, computers have taken a more active role, i.e., control. Of particular interest is the increasing use of distributed processors to perform this real-time control task.

(Castellano et. al. 1989) describe a distributed control system for a superconducting LINAC accelerator. This system is perhaps novel in that it does not use a local area network; instead, interprocessor communication is by direct memory access between VME processor crates. At the time of writing, only manual control adjustments were supported, although a "model-driven" automatic control was anticipated.

A similar distributed system for cyclotron control (Weehuizen et. al. 1992) employs PC/386 computers linked by Ethernet. This too appears to be a manually controlled system, although a reference to "setpoints" implies at least a limited use of feedback control.

Except for a few systems noted below, control systems for accelerators have used only "conventional" techniques: analog or digital feedback loops, occasionally with adaptive control.

Tandem Accelerator Controls

Most computer control systems for Tandem accelerators could better be termed "control multiplexers" or perhaps "adjustment systems," since their primary function is to allow a human operator to manipulate a multitude of adjustments from a central location. A secondary goal is rapid data acquisition. The computer itself rarely provides active control. (McKay, Gingell, and Baris 1986; Keitel and Wilson 1987; Laycak 1989; Greiner and Caswell 1992)

A example of the "state of the art" in computerized Tandem control is described by (Kumar et al. 1994). This system employs an 80386-based IBM PC, interfaced to CAMAC accelerator modules, as a control multiplexer, allowing many parameters to be manually adjusted from a single control point. The computer does not automatically control the accelerator, although it does provide alarms and safety interlocks, a logging database, and computational aids for accelerator parameters.

A similar control multiplexer has been used at Chalk River Laboratories for their "TASCC" Tandem-Accelerator-Superconducting-Cyclotron (Greiner and Caswell 1992). This older system was based on a PDP-11, serving about 2500 CAMAC channels. Like the previous system, the TASCC control system was strictly manual.

Chalk River Laboratories has experimented with automatic computer control of beam steering in the Tandem accelerator (Davies and Howard 1993), but this was strictly a "conventional" feedback control with no heuristic elements. Similar experiments have been performed at McMaster University's FN Tandem accelerator (Wong 1986; Wong and Poehlman 1986; Lingarkar 1988). McMaster University has also implemented computer control of ion sources (Pollock 1985; Poehlman, Pollock, and McNaught 1985; Bokhari 1993) and the analyzing magnet (Lind and Poehlman 1994).[12]

Sweden's Svedberg Laboratory reports upgrading a model EN Tandem accelerator and its ion source to use microprocessor controls (Possnert, Carlsson, and Widman 1992), but no mention is made of the control methodology, and it must be presumed that this is a conventional control system.

A recent survey article (Lutz and Marsaudon 1994) summarizes the requirements for new

12. This research concerned the analyzing magnet of a model KN Van de Graaff accelerator, but the principles are directly applicable to the model FN Tandem accelerator.

Tandem control systems (including the need for distributed processing and optical communication). The authors note that the adoption of "modern controls" in small electrostatic accelerators (such as the Tandem) lags far behind the state of the art found in large high-energy installations. Surprisingly, their vision of a future Tandem control system suggests only one use for heuristic systems: off-line "operators' assistants" for design and repair.

Heuristic Advisory Systems

Expert systems have been developed which do not attempt active process control, but instead act as advisor or assistant to a human operator (Stark and Poehlman, 1988). While this does not eliminate the need for trained operators, it allows expert knowledge to be easily replicated, without placing stringent real-time requirements on the expert system.

The Accelerator Operator's Companion (Poehlman and Stark 1989; Berg 1989; Tam 1989) is an operator's assistant for the McMaster FN Tandem accelerator. In addition to heuristics for routine operation and fault diagnosis, it provides a database (logbook) of past runs and a spreadsheet calculator. The first prototype was developed in C, and the second using Personal Consultant Plus, both on an IBM PC.

(Clearwater et. al. 1990) describe a diagnostic expert system to monitor the performance of a data acquisition trigger subsystem. For this application it was found necessary to have a "fast" inference engine, written in C, to analyze trigger events in the allotted 4 msec. A "slow" expert, written in LISP, performs detailed diagnosis of the trigger hardware. Both experts executed on a 20 MHz 80386 PC.

Advisory systems are not limited to particle accelerators; for example, (Garland et. al. 1989) describes a heuristic advisory system for nuclear power plants. This is a problem whose complexity is far greater than the Tandem accelerator, but whose real-time requirements are similar.[13] Another such system is DIAREX (Naito et. al. 1987), a specialized operator's assistant for the diagnosis of power plant faults.

---

13. As evidenced by the fact that both use human operators.

Heuristic Control Systems

No Tandem accelerator to date has employed heuristic techniques for operations control. Indeed, heuristic controls are rare in accelerators of any description. In 1989 it was observed that

> "Expert Systems, though discussed for some years, have not yet found many significant [control] applications. So far only fairly trivial cases have been attempted." (Gurd and Crowley-Milling 1989)

and in 1994

> "In recent years there has been expectations that AI techniques would be applied widely to accelerator operations, but these expectations have not been fulfilled. While there have been a few uses of expert or knowledge-based systems reported, these have not become widespread." (Crowley-Milling and Busse 1993)

The "fairly trivial cases" of rule-based control usually involve very limited problem domains. One reported example (Perreard and Wildner 1994) employed fuzzy logic in a control loop to condition high voltage cavities, a problem similar to that of conditioning the high-voltage terminal in a Tandem accelerator. The cavity-conditioning system used ten rules written in OPS5, with fuzzification and defuzzification performed by a separate module.

A popular problem for expert system solutions is beam line steering adjustments or "tuning." This is perhaps because

> "There are characteristics common to beam lines. Most have a large number of a few types of interacting devices. Each device has several related devices, the effects of which are not always well understood. While a mathematical model of the theoretical beam line may have been used to design the beam line initially, often that model is not precise enough to predict accurately what a given change will do to the real beam." (Schultz and Brown 1990)

The TRIUMF laboratory in Vancouver created a beam line expert using first NEXPERT, and then KEE, on a VAXstation 3200 (Schultz and Brown 1990). In trials this system was not entirely successful, as it was "unable to find an acceptable solution during the allotted time." Lawrence Livermore Laboratories, for the Advanced Test Accelerator, had greater success with a custom inference engine (Lager et. al. 1990).

A somewhat more ambitious application of expert systems is the SETUP system of (Daems et al 1994), which automates the cold-start and run-time testing of components in the CERN Proton

Synchrotron. This object-oriented expert system represents the knowledge base as "goal procedures" in directed graphs, can chain forwards or backwards, and is able to backtrack to previous goals if a subgoal cannot be achieved.[14] The application is essentially a diagnostic expert system, and does not replace the existing control system. The authors do suggest possible future uses of the expert system, including automatic run-time control.

Of greatest relevance to this work is the automation of the McMaster University model KN Van de Graaff accelerator (Stark et. al. 1992; Lind et. al. 1993). This control system used an expert system operating on a PC/386, communicating with an embedded 8051 single-board computer (SBC) that performed simple I/O functions. The 8051 also incorporated a simple neural network to analyze beam status. Two commercial expert system shells were tried in the PC, but yielded inadequate performance; the final system used a custom-developed expert system in Turbo Pascal (DeMooy and Poehlman 1991; DeMooy 1991; Lind et. al. 1991; Lind and Poehlman 1992). Accelerator startup, steady-state operation, accelerator shutdown were all performed automatically by the expert controller; in addition to fault detection and diagnosis.

The prior work with the McMaster KN accelerator has strongly influenced the goals of this project. The inadequacy of commercial expert system shells for real time control suggested the need for a new inference engine. The KN control system was limited to the inferencing power of a single CPU, and the I/O processing power of a single SBC; thus the desire for a distributed system with multiple inferencing agents and multiple I/O controllers.

---

14. This backtracking, similar to that used in pattern matching, can in fact be emulated by a goal-driven system such as that implemented in this project; see for example (Rodriguez 1989).