Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

TITLE: TOWARD AUTOMATIC CONTROL OF PARTICLE ACCELERATOR BEAMS

LA-UR--88-1200

DE88 009140

AUTHOR(S): D. E. Schultz & R. R. Silbar

SUBMITTED TO: Third International Conference on
Applications of Artificial Intelligence in Engineering
Stanford, CA
August 8-11, 1988

DISCLAIMER

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

TOWARD AUTOMATIC CONTROL OF PARTICLE ACCELERATOR BEAMS

David E. Schultz and Richard R. Silbar

Los Alamos National Laboratory, Los Alamos, New Mexico 87545

ABSTRACT We describe a program aiming toward automatic control of particle accelerator beams. A hybrid approach is used, combining knowledge-based system programming techniques and traditional numerical simulations. We use an expert system shell for the symbolic processing and have incorporated the FORTRAN beam optics code TRANSPORT for numerical simulation. The paper discusses the symbolic model we built, the reasoning components, how the knowledge base accesses information from an operating beamline, and the experience gained in merging the two worlds of numeric and symbolic processing. We also discuss plans for a future real-time system.

INTRODUCTION

There has been much activity lately in applications of artificial intelligence technology to knowledge-intensive domains. We have recently begun to evaluate knowledge-based, or expert, systems for solving problems in accelerator control. Because there have already been successful attempts using these techniques to control other complicated processes (e.g., Brand and Wong [1], Sachs *et al.* [2]), one has good reason to hope for success in this venture.

Knowledge-based system programming often contrasts sharply with traditional computing approaches. One technique that we use is object-oriented programming. This encourages developing a computer model which closely resembles the real-world problem. Using object-oriented programming it is easier to debug, explain, and verify the model. It is also easier to add a new feature (object) to the model, because a new object can inherit most of its characteristics and behavior from exist-

ing objects. The new object then needs only slot-value changes to reflect what is different about that particular device. Because of the partitioning of program behavior—the essence of object-oriented programming—this new object will not interfere with other objects.

Another technique we use is symbolic modeling. Such a model has causal relationships built in, so actions leading up to an event can be easily described (Brown *et al.* [3]). In contrast, the relationships in a numerical model are often structurally opaque. That is, intermediate steps are not transparently related to the underlying physical world, and this makes cause-and-effect explanations difficult. There are many problems, however, that have an elegant and efficient solution using traditional numerical algorithms. For these problems one should not even consider knowledge-based system techniques. Control of beam optics appears to be somewhere between the extremes of purely algorithmic and purely symbolic approaches. Thus (as in Clearwater [4]) the project described here uses a numerical model based on a Fortran code (operating in the LISP environment) to simulate beam line behavior. With the results of the numerical simulation always available to the inference engine, much of the decision-making in our model is symbolic in nature.

Using the Knowledge Engineering Environment (KEE) system, we have built a prototype knowledge base describing the characteristics and the relationships of about 30 devices in a typical beam line. Each device is categorised generically and pertinent attributes for each category are defined. Specific values representing static and dynamic characteristics for each device are assigned to slots in frames. These slot values are constrained by data type and any limitations or restrictions on the range of the data.

Relationships between the various beamline devices are modeled using rules, active values, and object-oriented methods. Our knowledge base provides a framework for analysing faults and offering suggestions to assist in tuning, based on information provided by the accelerator physicists (domain experts) responsible for designing and tuning the beamline. There is a general-purpose mechanism for determining device and beamline status based on device-specific critical parameters. This approach simplifies knowledge acquisition by allowing the domain expert to concentrate on what is important about a particular device without getting bogged down in trying to specify rules for it.

A powerful graphical interface allows the operator to "mouse" on an icon for a particular item in the schematic of the beam line and obtain

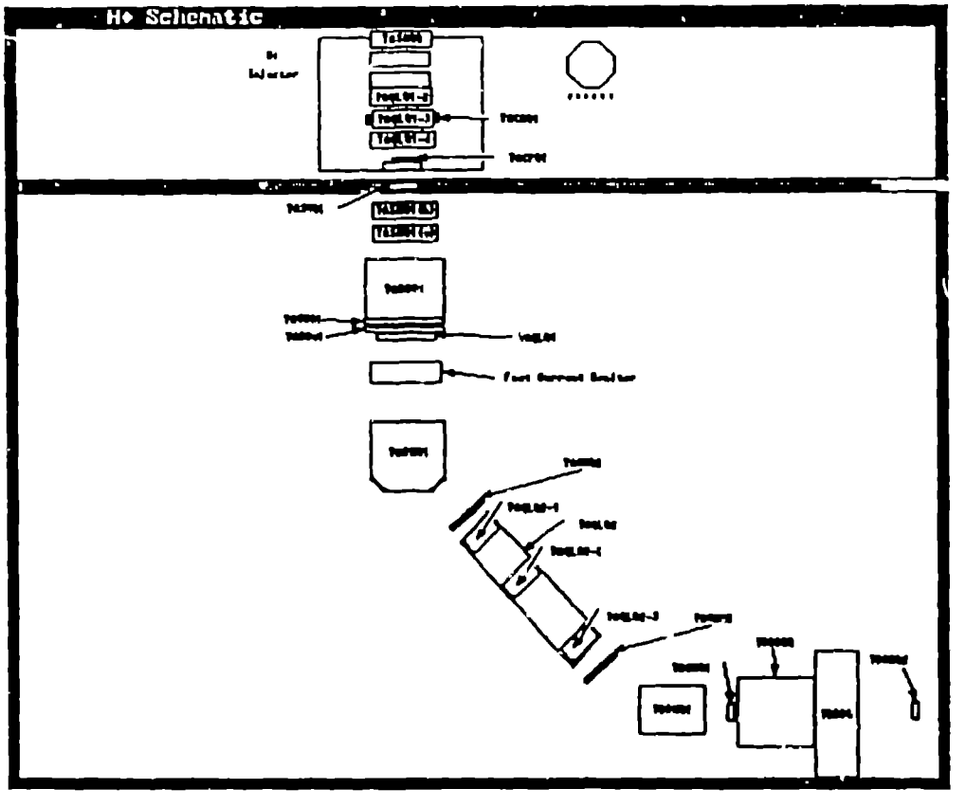

Figure 1. H^+ beamline schematic.

device-specific information and control over that device. The beam optics code TRANSPORT is used to model the beam line numerically, and any changes induced by the operator (or perhaps, by the accelerator itself) are automatically updated on the operator's display. Other numerical techniques will be used to take the raw input wave forms and extract features so that the inference engine has qualitative descriptors with which to reason.

Preliminary indications from using our knowledge base are that artificial intelligence techniques and traditional methods of numerical simulation can, and probably soon will, be successfully combined to provide a powerful tool for control of accelerators.

DESCRIPTION OF THE PROBLEM

The problem chosen is the tuning of the first 30 devices in the H^+ beamline, Fig. 1, of the Clinton P. Anderson Meson Physics Facility at Los Alamos National Laboratory (LAMPF). This low-energy beam transports protons from the Cockcroft-Walton ion source to the first emittance-measuring station (Schultz *et al.* [5]). This relatively small beam line is of

an appropriate size and complexity for a first prototype of a knowledge-based accelerator control system. The tuning goals are to minimize the emittance growth of the beam, to steer it, and to match the output emittance to the acceptance of the next section of the accelerator beam line. These constraints define a small region in the transport phase space which will provide an acceptable tune. Each time the ion source changes, the beam-transport parameters must be re-tuned to accommodate the slightly different characteristics of the new source.

To indicate the complexity of the problem under discussion, the major hardware in the H^+ beamline includes: 1) two bending magnets, 2) six steering magnets, 3) eight quadrupole magnets, 4) a beam deflector, 5) an RF pre-buncher, 6) four current monitors, 7) an emittance measurement device, 8) a beam-profile harp, 9) two phosphor viewing screens, 10) a beam scraper with four jaws, and 11) an adjustable aperture. There are numerous other connecting and support devices, minor but vital to the correct operation of the beamline.

THE TRADITIONAL APPROACH TO BEAM TUNING

Manual tuning of a beam line is an iterative process involving many steps: steering, adjusting quadrupoles, steering again, bringing deflector plates, jaws, and apertures to the edge of the beam, and then repeating the process. The data obtained from the diagnostic devices that measure beam characteristics are analysed by Fortran programs running on the LAMPF VAX/VMS control system. The analyzed data are then used to generate beam envelopes and predict new tunes with a first-order optics code.

This information is available in a graphical or tabular form to the person tuning the beamline. Working with the correlations that can be identified in this data, along with knowledge of the desired "design tune" and use of particle-tracing codes, the operator seeks to find a solution that focuses and steers the beam from one end of the beamline to the other. This procedure works relatively well, but it is time-consuming and labor-intensive, requiring close attention by highly trained personnel. The operator must judge whether the successive iterations are converging to an acceptable solution. It is not uncommon to find that the converged-upon solution space is unacceptable; in such cases the work must be abandoned and the procedure started over. The "better" experts find many fewer unacceptable solutions, i.e., somehow know how to avoid the pitfalls (local minima in the parameter space). Perhaps some beamline tuners are especially adept at extracting nuances from graphical data and exploiting

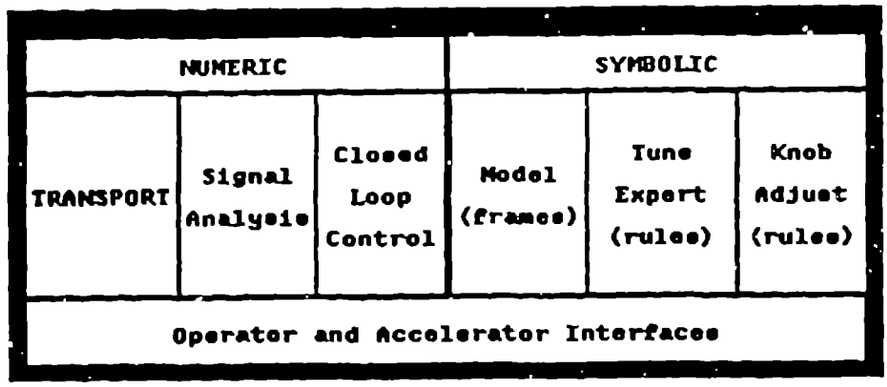

Figure 2. Overview of system organization.

them.

KNOWLEDGE-BASED SYSTEM APPROACH

We have chosen to use a hybrid of Model-Based Reasoning coupled with the beam optics code TRANSPORT (Brown *et al.* [6]) to tackle the beamline tuning problem. We represent the beamline using a symbolic model embedded in a KEE knowledge base. It describes the characteristics of and the relationships among the devices in the beam line. We categorize each device and define pertinent attributes for each category. Specific values are assigned in the knowledge base to represent each actual device. Relationships between devices are modeled using the techniques of rules, active values, and object-oriented methods. The knowledge base can be used to:

1. Simulate the devices in the beamline,
2. Identify faulty devices for repair,
3. Monitor progress in a complex tune procedure,
4. Advise on tuning actions ("What do I do now?"),
5. Explain advice given, and
6. Identify faulty devices as they affect tune procedure.

The basic technique chosen is shown in Fig. 2. It combines the strengths of numeric and symbolic computation and uses the best technique that will work on a given part of the problem. Due to some anticipated performance requirements the preferred order is numeric solution, rules using captured operator expertise, and, only as a last resort, heuristic search. The numeric processing consists of closed loop control for maintenance of a steady state, TRANSPORT for simulation of the optics of the beam-

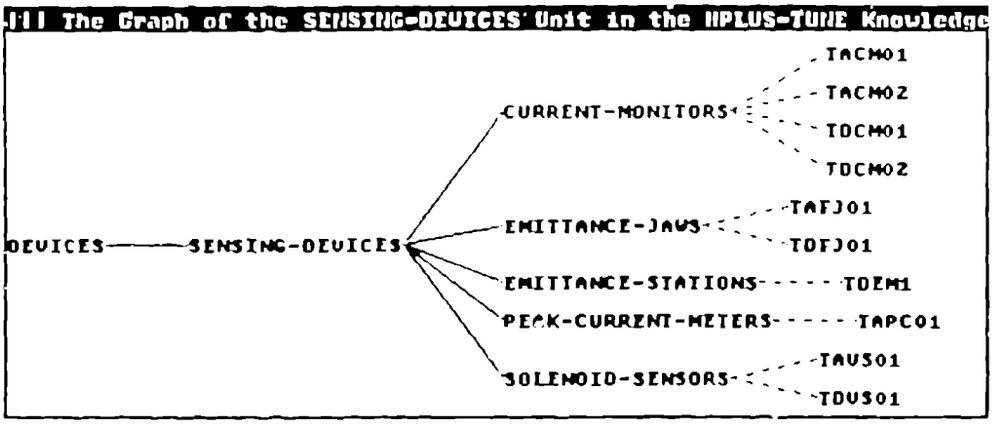

Figure 3. Device hierarchy tree.

line, and signal analysis of the raw wave forms to extract features that can be converted to qualitative states or values. This qualitative information can be used by the rule system to determine where the device is in the tune-up procedure, and when the device is ready for the next stage of tuning. The operator's expertise determines what to do at each step in the tune-up recipe. Only when both the numerical approach and the recipe fail does the system resort to the heuristic search (knob twiddling). Finding a suitable tune by varying parameters based on the known associations between input and output values is the most inefficient of the three methods and is by no means guaranteed to lead to an acceptable solution. However, even such a last ditch attempt may solve a few problems, and it will surely help the operator develop his intuition about the operating modes of the accelerator.

Objects

Figure 3 shows a tree of the sensing devices in the portion of the H^+ beamline we are addressing, together with their class/sub-class relationships (denoted by a solid line). Dotted lines denote particular members of a class. The class SENSING-DEVICES has sub-classes CURRENT-MONITORS, EMITTANCE-JAWS, EMITTANCE-STATIONS, PEAK-CURRENT-METERS, and SOLENOID-SENSORS. Each type of sensing device has members (specific instances) such as the particular emittance jaw labelled TAFJ01. All sensing devices have certain characteristics in common. These class-wide characteristics are inherited by the individual members of the class. The values of the attributes in the display of a member's slots reflect the state of that particular device.

Active Values

Active values model some of the actions of the devices in the beamline.

For example, there is a method, or procedure, associated with the active value attached to any slot (attribute) on any device that has a set-point and a tolerance. Active values are daemons such that, every time the value of the slot changes, the method is automatically invoked. The LISP code in this method could say, for example, that if the value about to be stored differs from the set-point value by more than the tolerance, then this device is a candidate for being the cause of any problem. Other evidence will be needed to narrow down the problem to a specific device.

Reasoning

Rules capture heuristic knowledge. One rule (in KEE syntax), which runs through and checks the status of all the devices of a given type, is

```
(IF  ((IN.CLASS ?BEAM BEAM-DESCRIPTION) AND
      (THE DEVICE.OF.INTEREST OF ?BEAM IS ?DEVICE) AND
      (OR (THE STATUS OF ?DEVICE IS OK)
          (THE CONDITION OF ?DEVICE IS BROKEN)))
 THEN
      (THE DEVICE.OF.INTEREST OF ?BEAM IS
          (THE NEXT.SAME.DOWNSTREAM OF ?DEVICE)))
```

Thus, for example, if the devices of interest are current monitors, the rule says that if you find a current monitor reading that is within its expected range (or that monitor is known to be broken), then go on to consider the status of the next current monitor downstream.

A closely related rule is

```
(IF ((IN.CLASS ?BEAM BEAM-DESCRIPTION) AND
     (THE DEVICE.OF.INTEREST OF ?BEAM IS ?DEVICE) AND
     (THE STATUS OF ?DEVICE IS N-OK) AND
     (THE CONDITION OF ?DEVICE IS WORKING)))
 THEN
     (CHANGE.TO (A SUSPECT.DEVICES OF ?BEAM IS
         (THE DEVICES.THAT.AFFECT OF ?DEVICE)))
```

This rule takes care of the case where some current monitor indicates that the beam has disappeared. All devices upstream from that point are candidates for being the cause of the problem. Subsequent rules reason about the value of SUSPECT.DEVICES to narrow down the specific device at fault. Note the "wild-card" variables beginning with "?" in the rules. Even without knowing the syntax of the KEE rule system, it is relatively easy for a casual reader to determine what the rules mean.

We have developed general-purpose rules, such as those given above, for two reasons. First, the specific detailed rules for some situations have not yet been determined. Second, general-purpose rules simplify knowledge acquisition. For example, domain experts are well aware of the critical parameters of each device but it is often difficult to extract from them specific rules about those devices. A device attribute that is a critical

parameter is given facets reflecting the expected value, relationships, and resultant state value. This framework supports CHECK-FOR-GOOD and CHECK-FOR-BAD functions. CHECK-FOR-GOOD simply does an AND of all the critical parameters for a given device. Each critical parameter must match the good relationship between expected and actual values. Likewise, CHECK-FOR-BAD is an OR of all critical parameters that are out of range. If any critical parameter has a value that falls into a bad relationship with the expected value, then that device is designated as having a bad status.

One problem that often arises in tuning an accelerator is knowing what data you can believe. Does the data reflect a real problem with the beam or is the instrumentation giving misleading indications? We have attempted to address that uncertainty in cases where we have a cross check on the data and some experience in the type of expected failures. Current monitors are a good example. It is impossible to have a proper current monitor reading in a downstream current monitor if an upstream current monitor really shows no current. The most likely failure mode of current monitors is to read zero. With this information we can use the following heuristic. If a current monitor reads zero and the next current monitor downstream has a legitimate value, it is very likely that the first current monitor is broken. In that case, it is safe to note that fact in the knowledge base and to look further downstream to determine where (or if) the beam really disappears. The premise (`THE CONDITION OF ?DEVICE IS BROKEN`) in the first rule shown above skips over broken current monitors.

Interactions with the Real World

Our knowledge base was originally developed on TI Explorer and Symbolics 3600-series LISP machines using the KEE development system. It has also been ported to a micro-VAX AI workstation that has the ability to communicate with the LAMPF control computer. For the present, however, the knowledge base is not connected to all the accelerator real-time values.

Some data can be obtained from the accelerator for use in rules to determine actions to take. Figure 4 shows the flow of information that is required to get data attached to various parts of the accelerator. The programming mechanism we use here is an active value. When an attribute is fetched, for example, in the premise of a rule, the active value causes the data to be read from the actual device. Including this capability pointed out to us another important characteristic of devices. Some devices have values that change very slowly, others change frequently

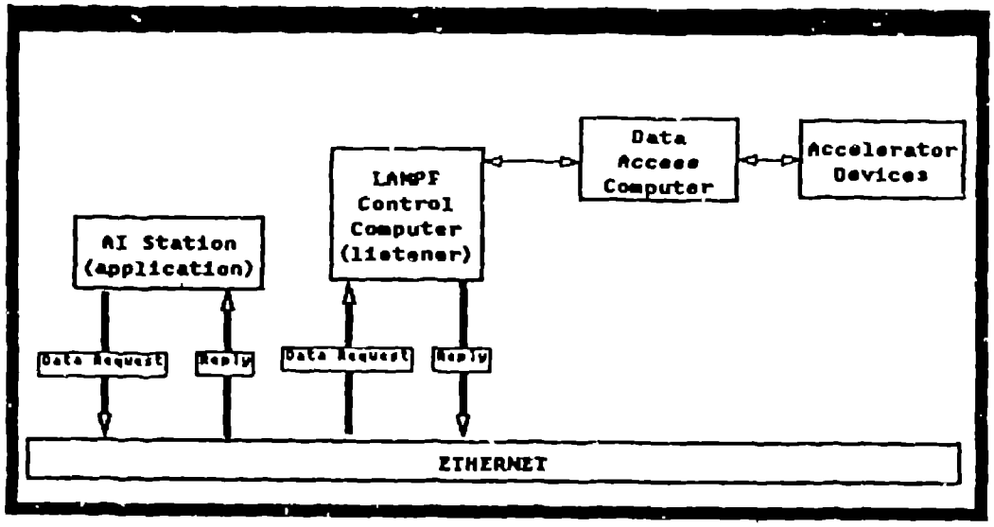

Figure 4. Network information flow for data access.

and rapidly. To avoid the overhead of getting data from a real device that changes slowly, two new characteristics of each device were defined. They are the time interval that a piece of data from this device is likely to remain unchanged and a time stamp of when the actual data was last obtained. The active value checks to see if new real data is needed, i.e., the current value was obtained too long ago. If so, a request is made for new data and the time stamp is updated. If the data is current, the old data is used.

Until more confidence is gained in the accuracy and co. .pleteness of the model it seems prudent to use simulated data rather than real data. For that reason, we designed a mouseable schematic to allow operators to select a device of interest and display a related image panel which displays the values of the interesting parameters for that device. Figure 5 shows a typical panel for changing or viewing simulated values of a steering magnet. The operator can then enter data for test purposes. He selects and changes a (simulated or real) value by positioning the mouse cursor on its icon or image panel and clicking a button on the mouse.

Integration with Numerical Simulations

To tune the device (or to display changes in the tuning when a fault occurs), we make frequent use of the beam optics program TRANSPORT. When a beam-device parameter changes, an active value associated with that parameter updates that value in a file representing the "TRANSPORT input deck". Another method then re-runs TRANSPORT, which writes its results to a standard output file. This file is then parsed by

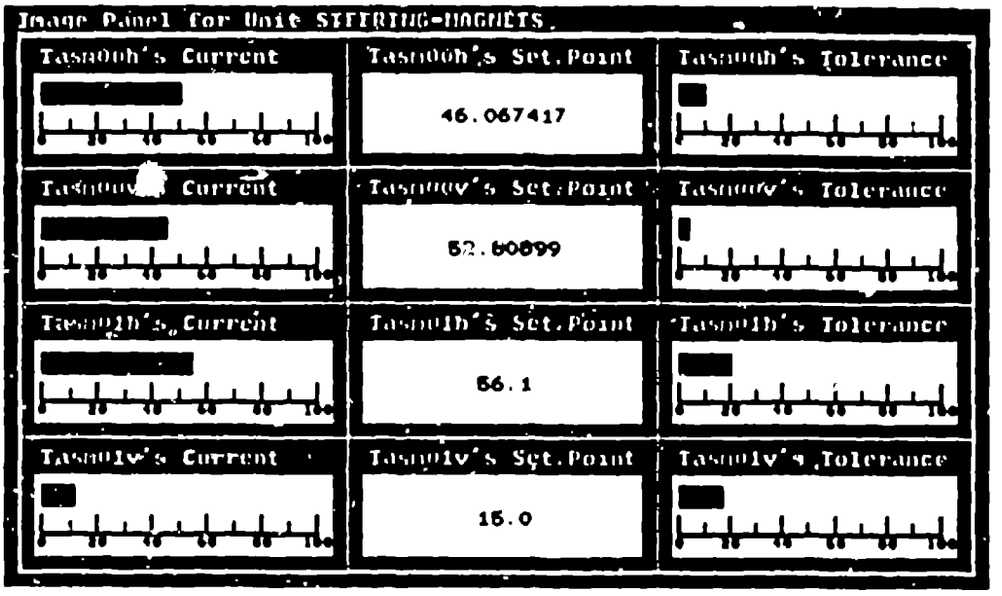

Figure 5. Image panel for steering magnet control.

a third method, which extracts the new beam positions and envelopes, transfer matrices, and phase space ellipses and updates them. In a nutshell, if the operator mouses on a beam element to change a parameter, the changes in the beam are automatically re-computed and re-displayed.

Figure 6 shows an example of what part of the screen seen by the operator looks like. (This beam is not the H^+ beamline discussed above but a simpler example consisting of two quadrupole triplets.) The whole process of recomputing a beamline typically takes ten or twenty seconds. Most of the CPU time involved appears to be spent on setting up the FORTRAN process and accessing files, not on the TRANSPORT calculation.

Numerical outputs from the TRANSPORT program provide important pieces of knowledge needed to select the proper course of action during a tuning procedure. For example, if TRANSPORT reports that the beam is off-axis at some point, we know that only devices upstream from that point can be the cause of the alignment error.

Following the Manual Tune Procedure

The procedure used to set the H^+ beam to some desired state can be described as iterating through eight major steps in a tuneup recipe. Each major step has from 2 to 15 sub-tasks that must be successfully completed before the step is considered done. During each pass through the major steps, a decision is made on the quality of the resulting beam. If the quality is satisfactory, i.e., matches desired characteristics, the tuneup procedure is complete. If the beam quality is less than desired, then the

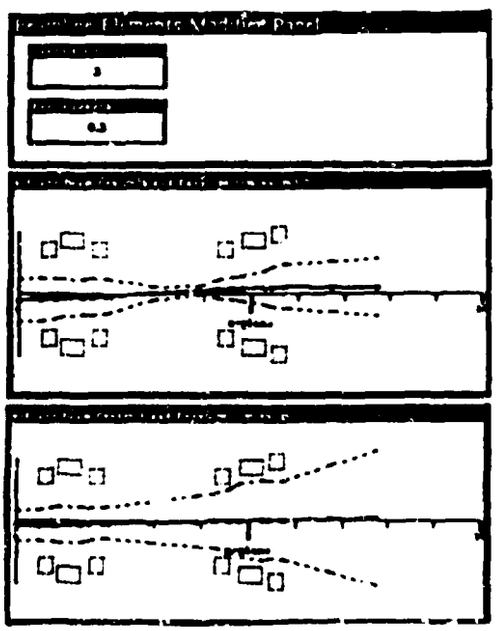

Figure 6. Beamline element knobs (upper panel) available to the operator for changes by clicking with the mouse and entering new values. The lower two panels show the actively updated x- and y- plane beam centroids and envelopes, together with positions of the six quadrupole lenses.

major steps are repeated.

In an ideal world, the tuneup would be a simple straight-forward following of the recipe. Unfortunately, it never is. Devices break, wires are broken or changed, vacuum is lost, or software does not meet requirements. All these cause deviations from the desired plan. The task at hand is to determine what can be done, if such problems exist, that is productive. That is, given the devices and components that are (thought to be) working, what steps/tasks can be done that won't have to be redone when the currently malfunctioning components are fixed.

A set of rules is being developed to advise the operator on what to do next. These rules use two types of information about each item within a step to make the selection. There is fixed (predetermined) information about each task, such as the devices needed to do it, the next task to do if successful, the next task to do if unsuccessful, the expected value (success criteria), and required pre-conditions and resulting post-conditions. Included in this fixed information is global data, i.e., criteria for step completion, step-to-step sequences, and tasks that can be done in parallel.

The other information used to determine if a particular task can be done varies from time to time. It includes what devices are known not to be working or whether a general (or a specific) problem has been found. Before a task is started, a test is made to see that all devices needed to accomplish it are available. Once a task is performed, the program checks to see if the result meets the success criteria. If it failed, an attempt is made to identify the specific problem that caused the failure. The job of identifying the specific problem is delegated to the step-related rule set [as opposed to the task rule(s)]. Other evidence is used to narrow the problem down to one or more specific problems. A specific problem may require adjustment to devices or it may indicate that a certain device is not working. If not enough information is available at the task level to determine a specific problem, a general problem category is assigned and stored for future use.

There are two modes of operation for the tune recipe rule set. The first is called "manual mode". In that mode the rule set displays advice on what needs to be done next. In this mode the system can be used as an advisor. The operator makes the final decision on whether the advice is adequate and whether it should be followed. This mode is useful for testing and to build the operator's confidence in the system. The second mode is automatic. In this mode, as the advice is generated, the system

performs the (simulated) suggested action with no intervention from the operator. The ultimate goal is to turn on the real beam automatically with no operator in the loop.

CURRENT STATUS AND FUTURE WORK

The following components of our knowledge base have been built and tested:

1. A static model containing most of the information about the beamline and the characteristics of each device in it.
2. Image panels that allow simulation of test data.
3. A few rules using current monitor information to identify candidates for causing failure.
4. Active values that propagate device errors to affect the proper current monitor.
5. General rules to identify device status based on critical parameters.
6. Demonstration image panels that allow activation of the rule set and display of the conclusions reached.
7. Conversion of TRANSPORT to run in the LISP/KEE world.
8. Knob panels allowing the operator to change values of the initial beam parameters and up to six beamline elements of his choice.
9. Active-image panels displaying phase space ellipses and beam envelopes as calculated by TRANSPORT, either when a knob is tweaked or when the operator calls for an update.
10. A general menu for modifying the beamline, adding or deleting elements, moving them around, etc. This also can be done from the mouseable schematic of the beamline blueprint.
11. A "beamline spreadsheet" that allows for alternate input and changes of the beamline.
12. The ability to save and read previously-saved beamlines.

The symbolic model part of the prototype was developed in about 3 man-months. One man-month was spent on converting and getting the Fortran TRANSPORT code to run in the LISP environment. The interface between KEE and TRANSPORT and the user has taken an additional 4 man-months. While the parts of the knowledge base we have built so far handle only a very small fraction of the total problem and have yet to be fully integrated with each other, they nonetheless demonstrate the power of the software development environment to create useful models quickly.

The next steps in our development of the knowledge base are to include:

1. Additional descriptions of the tuneup procedure.
2. Additional rules for determining device failure that use diagnostic

tools besides the current monitors.

3. The identification and specification of the critical parameters for all appropriate devices.

4. Refinement coming from addition of more specific rules to handle less obvious and less frequent problems.

5. Other uses of the TRANSPORT code, such as the optimization facility.

CONCLUSIONS

The initial results of our work have shown that knowledge-based systems can be successfully combined with traditional methods of numerical simulation. While only a small part of the problem has been modeled so far, the interpretations given by the hybrid system match those of human operators. It is expected that additional rules will be able to capture more fully the expertise used by a beamline physicist. Automatic operation of future accelerators may well depend on the proper merging of symbolic reasoning and conventional numerical algorithms.

ACKNOWLEDGEMENTS

We thank Peter Clout for introducing us to the problem of automatic accelerator control and Stan Brown, Jim Friar, Mary Fuka, Ed Heighway, Earl Hoffman, Jim Hurd, Walter Lysenko, Tom Mottershead, and Roselle Wright for support, advice, and encouragement. This work has been supported by the U.S. Department of Energy.

REFERENCES

1. Brand, H., and Wong, C. (1986). Application of Knowledge-Based Systems Technology to Triple Quadrupole Mass Spectrometer (TQMS), *AAAI '86 Proceedings*, pp. 812-818.

2. Sachs, P., Paterson, A., and Turner, M. (1986). ESCORT: an expert system for complex operations in real time, *Expert Systems* 3, pp. 22-29.

3. Brown, J., Burton, R., and deKleer, J. (1982). Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III, *Intelligent Tutoring Systems*, Academic Press, London, pp. 227-282.

4. Clearwater, S. (1986). Developing a Hybrid Expert System for use at a Particle Accelerator Facility, *Los Alamos National Laboratory Report*, LA-UR-86-1516.

5. Schultz, D., Hurd, J., and Brown, S. (1987). An Artificial Intelligence Approach to Accelerator Control Systems, *Proceedings of the International Workshop on Hadron Facility Technology*, pp 430-440.

6. Brown, K. L., Rothacker, F., Carey, D., and Iselin, Ch. (1977). TRANSPORT—A Computer Program for Designing Charged Particle Beam Transport Systems, *Stanford Linear Accelerator Report*, SLAC-91, Rev. 2, UC-28.