LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although portions of this report are not reproducible, it is being made available in microfiche to facilitate the availability of those parts of the document which are legible.

3

LA-UR -88-1058

LA-UR--88-1058

DE88 007821

TITLE AUTOMATION OF PARTICLE ACCELERATOR CONTROL

AUTHOR(S) Richard R. Silbar and David E. Schultz

SUBMITTED TO To be presented at the ASME Computer in Engineering Conference, July 31-August 3, 1988, San Francisco, CA

DISCLAIMER

Los Alamos Los Alamos National Laboratory

AUTOMATION OF PARTICLE ACCELERATOR CONTROL

Richard R. Silbar and David E. Schultz
Los Alamos National Laboratory, Los Alamos, New Mexico 87545

Abstract

We have begun a program aiming toward automatic control of charged-particle beam optics using artificial intelligence programming techniques. In developing our prototype, we are working with LISP machines and the KEE expert system shell. Our first goal was to develop a "mouseable" representation of a typical beamline. This responds actively to changes input from the mouse or keyboard, giving an updated display of the beamline itself, its optical properties, and the instrumentation and control devices as seen by the operator. We have incorporated the Fortran beam optics code TRANSPORT for simulation of the beam. The paper describes the experience gained in this process and discusses plans to extend the work so that it is usable, in real-time, on an operating beamline.

INTRODUCTION

There has been much activity lately in applications of Artificial Intelligence technology to knowledge-intensive domains (e.g., Rich, 1983; Hayes-Roth et al., 1983; Harmon and King, 1985). We have recently begun to evaluate such Expert Systems techniques for solving problems in accelerator control. Because there have already been successful attempts using AI to control other complicated processes (Brand and Wong, 1986; Lacky et al., 1986), one has good reason to hope for success in this venture.

AI programming techniques often contrast sharply with traditional computing approaches. One AI technique that we use is object-oriented programming (e.g., Kabrow et al., 1986; Moon, 1986). 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 existing 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., 1982). 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-effect explanations difficult. There are many problems, however, that have an elegant and efficient solution using traditional algorithms. For these problems one should not even consider AI techniques. Control of beam optics appears to be somewhere between the extremes of purely algorithmic and purely symbolic approaches. Thus, as in Clearwater's work (1986), 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 categorized 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 beam-line devices are modeled using rules, active values, and object-oriented methods. Our knowledge base provides a framework for analyzing faults and offering suggestions to assist in tuning, based on information provided by the accelerator physicists (domain experts) responsible for designing and tuning the beam line. There is a general-purpose mechanism for determining device and beam-line 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 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 eventually, by the accelerator itself) are automatically updated on the operator's display. 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.

ESCRIPTION OF THE PROBLEM

The problem chosen is the tuning of the first 30 evices in the H^+ beam line of the Clinton P. Anderson eson Physics Facility at Los Alamos National Laboratory (LAMPF). This low-energy beam transports protons rom the Cockcroft-Walton ion source to the first mittance-measuring station (Schultz et al., 1987). his relatively small beam line is of an appropriate ize and complexity for a first prototype of an AI-ased accelerator control system. The tuning goals are o minimize the emittance growth of the beam, to steer t, and to match the output emittance to the acceptance f the next section of the accelerator beam line. These onstraints define a small region in the transport ase space which will provide an acceptable tune. Each me the ion source changes, the beam-transport parameters must be re-tuned to accommodate the slightly ifferent characteristics of the new source.

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

E TRADITIONAL APPROACH TC BEAM TUNING

Manual tuning of a beam line is an iterative process volving many steps: steering, adjusting quadrupoles, ering again, bringing deflector plates, jaws, and rtures to the edge of the beam, and then repeating process. The data obtained from the diagnostic vices that measure beam characteristics are analyzed Fortran programs running on the LAMPF VAX/VMS con-l system. The analyzed data are then used to erate beam envelopes and predict new tunes with a st-order optics code.

This information is available in a graphical or ular form to the person tuning the beam line. king with the correlations that can be identified in s data, along with knowledge of the desired "design e" and use of particle-tracing codes, the operator ks to find a solution that focuses and steers the m from one end of the beam line to the other. This cedure works relatively well, but it is time-suming and labor-intensive, requiring close attenn by highly trained personnel. The operator must ge whether the successive iterations are converging an acceptable solution. It is not uncommon to find t the converged-upon solution space is unacceptable. such cases the work must be abandoned and the proce-s started over. The "better" experts find many fewer ceptable solutions, i.e., somehow know how to avoid pitfalls (local minima in the parameter space) aps some beam-line tuners are especially adept at racting nuances from graphical data and exploiting

AN AI APPROACH TO BEAM TUNING

We have chosen to use a hybrid of Model-Based Reasoning coupled with the beam optics code TRANSPORT (Brown et al., 1977) to tackle the beam-line tuning problem. We represent the beam line 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 beam line,
2) Identify faulty devices for repair,
3) Monitor progress in a complex tune procedure,
4) Advise on tune actions ("What do I do now?"),
5) Explain advice given, and
6) Identify faulty devices as they affect tune procedure.

Objects

Figure 1 shows the magnetic 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. For example, the class MAGNETS has sub-classes STEERING-MAGNETS and QUADRUPOLE-MAGNETS. Each type of magnet has members (specific instances) such as the particular device labelled TASM01H. All magnets 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.

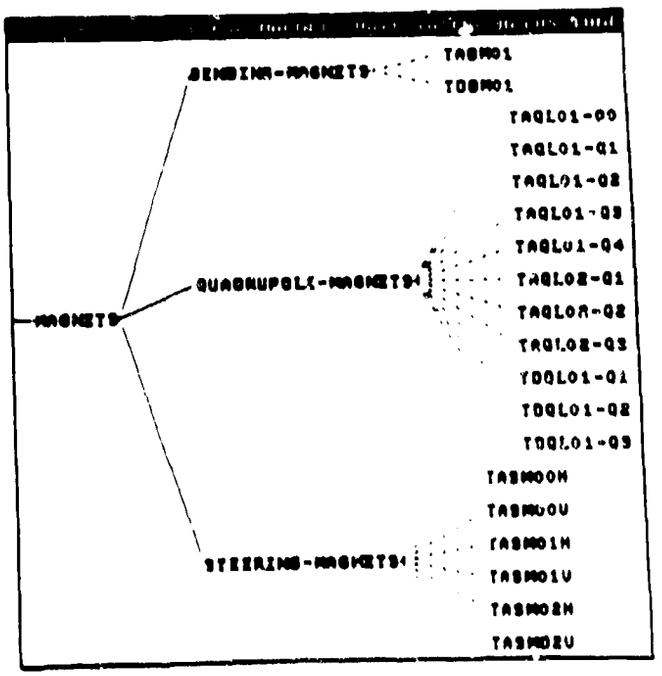

Fig 1 Device hierarchy tree

ctive Values

Active values model some of the actions of the vices in the beam line. For example, there is a thod, or procedure, associated with the active value tached to any slot (attribute) on any device that has set-point and a tolerance. Active values are daemons ch. that, every time the value of the slot changes, e method is automatically invoked. The LISP code in is method could say, for example, that if the value out to be stored differs from the set-point value by re than the tolerance, then this device is a candidate for being the cause of any problem. Other idence will be needed to narrow down the problem to a cific device.

soning

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

```
(IN.CLASS ?BEAM BEAM-DESCRIPTION) AND
(THE DEVICE.OF.INTEREST OF ?BEAM IS ?CM) AND
(OR (THE STATUS OF ?CM IS OK)
    (THE CONDITION.OF.DEVICE OF ?CM1 IS BROKEN)))
EN
THE DEVICE.OF.INTEREST OF ?BEAM IS
    (THE NEXT.SAME.DOWNSTREAM OF ?CM))))
```

s, for example, in the case of current monitors, the s says that if you find a current monitor reading t is within its expected range (or that monitor is m to be broken), then go on to consider the status the next current monitor downstream. Note the ld-card" variables beginning with "?" in the rule. n without knowing the syntax of the KEE rule system, s relatively easy for a casual reader to determine the rules mean.

We have developed general-purpose rules, such as given above, for two reasons. First, the specific iled rules for some situations have not yet been rmined. Second, general-purpose rules simplify ledge acquisition. For example, domain experts are aware of the critical parameters of each device it is often difficult to extract from them specific s about those devices. A device attribute that is a ical parameter is given facets reflecting the ex- ed value, relationships, and resultant state value. framework supports CHECK-FOR-GOOD and CHECK-FOR- functions. CHECK-FOR-GOOD simply does an AND of all critical parameters for a given device. Each criti- parameter must match the good relation-ship between cted and actual values. Likewise, CHECK-FOR-BAD is R of all critical parameters that are out of range. ny critical parameter has a value that falls into a relationship with the expected value, then that e is designated as having a bad status.

e problem that often arises in tuning an accel- r is knowing what data you can believe. Does the reflect a real problem with the beam or is the umentation giving misleading indications? We have pted to address that problem in cases where we a cross check on the data and some experience in type of expected failures. Current monitors are a example. It is impossible to have a proper current or reading in a downstream current monitor if an eam 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 OF ?CM IS BROKEN) in the rule shown above allows the skip over broken current monitors.

Simulating Real Data

Our knowledge base was originally developed on TI Explorer and Symbolics 3600-series LISP machines using the KEE development system. Recently it has also been ported to a microVAX 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. The mechanism that is used 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 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 completeness 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. 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 invokes a method that changes 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 a third method, which extracts the new beam positions and envelopes, transfer matrices, and phase space ellipses. These are used by the active images of the knowledge base. 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 redisplayed.

Figure 2 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 is spent on setting up the FORTRAN process and accessing files, not on the TRANSPORT calculation.

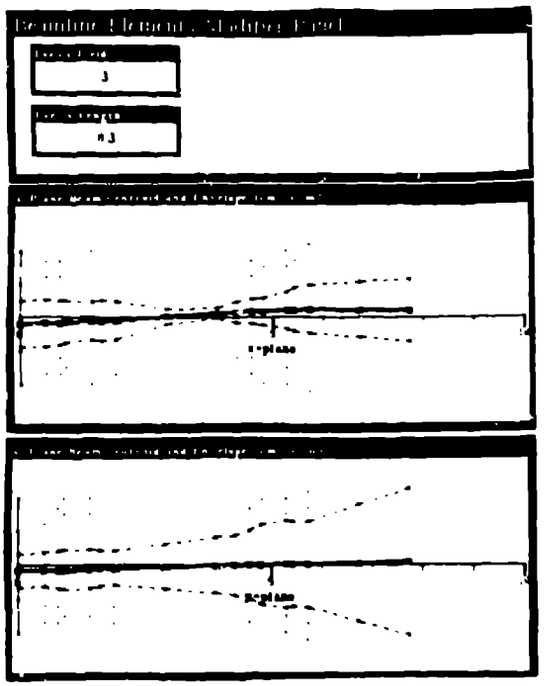

Fig. 2. Beamline element knobs (upper panel) available to the operator for changes by clicking with the mouse and entering new values. The lower two panels show the actively updated x- and y-plane beam centroids and envelopes, together with positions of the six quadrupole lenses.

Numerical outputs from the TRANSPORT program provide important pieces of knowledge needed to successfully 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 is 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 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 clogged, 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 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, non-global 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 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.

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 beam line 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 artificial intelligence techniques 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 beam-line physicist. Successful and efficient 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 this problem and Stan Brown, Jim Friar, Mary Fuka, Ed Lughway, Earl Hoffman, Jim Hurd, Walter Lysenko, Tom Mottershead, and Rozelle Wright for advice and encouragement. This work has been supported by the U.S. Department of Energy.

REFERENCES

Boorow, D., et al., 1986. "Common Loops--Merging Lisp and Object-Oriented Programming". OOPSLA '86 Proceedings, pp. 17-29.

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

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.

Brown, K. L., Rothacker, F., Carey, D., and Iselin, Ch., 1977. "TRANSPORT--A Computer Program for Designing Charged Particle Beam Transport Systems", Report SLAC-91, Rev. 2, UC-28.

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

Harmon, P., and King, D., 1985. Expert Systems, John Wiley and Sons Inc, New York.

Hayes-Roth, F., Waterman, D., Lenat, D. et al., 1983. Building Expert Systems, Addison-Wesley Publishing Co. Inc., Reading, Massachusetts.

Moon, D., 1986. "Object-Oriented Programming with Flavors", OOPSLA '86 Proceedings, pp. 1-8.

Rich, E., 1983. Artificial Intelligence, McGraw-Hill, New York.

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

Schultz, D., Hurd, J., Brown, S., 1987. "An Artificial Intelligence Approach to Accelerator Control Systems". Los Alamos National Laboratory Report LA-UR-87-739.