Klein, W.B., Stern, C.R., Luger, G.F., and Pless, D. "Teleo-Reactive Control for Accelerator Beamline Tuning". In *Proceedings of the IASTED International Conference*. Zurich: IASTED/ACTA Press (2000).

TELEO-REACTIVE CONTROL FOR ACCELERATOR BEAMLINE TUNING

WILLIAM B. KLEIN
Intelligent Programming, LLC., U.S.A.
2403 San Mateo Blvd. NE, Suite W-2
Albuquerque, NM 87110, U.S.A.

CARL R. STERN
SandiaView Software, Inc.
1009 Bradbury Dr. SE, Suite 7
Albuquerque, NM 87106, U.S.A.

GEORGE F. LUGER
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131, U.S.A

DAN PLESS
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131, U.S.A

ABSTRACT

Over the past five years we have designed, built, and tested a portable control architecture for accelerator beamline tuning. To date our software has been tested against software models of accelerators and against actual beamlines at Brookhaven and Argonne National Laboratories. In this paper we briefly describe our object-based, hierarchically organized portable architecture. The main focus of this paper, however, is to describe our extensions to [1] teleo-reactive (TR) planner to make it suitable to handle the complex plan execution tasks of accelerator beamline control. Extensions to Nilsson's original system include the addition of actions for targeted data gathering. We extend the rules of the TR planner with the addition of new types of rules that adjust intermediate goals and that adapt to context changes by loading and unloading rule sets. We give some examples of our own extended teleo-reactive control framework. We conclude by briefly discussing tests of our system in beamline tuning as well as in diagnosing beamline misalignments.

KEYWORDS

Intelligent control, teleo-reactive control, distributed architectures.

OVERVIEW

Over the past five years we have designed and constructed a portable, intelligent software system for accelerator beamline control. Our design has been general in its approach in that our software is configurable to a wide variety of particle beam accelerator facilities and extensible to other related problems. Our control architecture has a multi-layer, hierarchical organization in which knowledge-based decision making is used to dynamically reconfigure lower level optimization and control algorithms. An object-based physical access layer in conjunction with a high speed data bus supports an abstraction layer that hides the lower level details of hardware measurement and manipulation, signal processing, and system synchronization. High level control is carried out by knowledge-based executives that implement a modified version of the teleo-reactive algorithm described in [1] and [2].

In this paper we first discuss the teleo-reactive control system proposed by Nilsson and Benson. Next, we present the accelerator tuning problem including its real-time requirements, its data access constraints, and the problem of organizing interacting subgoals. Then, we present our extensions to the original teleo-reactive system designed to address these control needs.

Our control system has been tested against models of particle beam accelerators, as well as on actual accelerators: the ATF facility at Brookhaven and the ATLAS line at Argonne National Laboratory [3], [4].

A DESCRIPTION OF THE CONTROL ARCHITECTURE

We have developed a comprehensive architecture supporting distributed knowledge-based problem solving and control. This architecture is intended to provide a general framework for achieving distributed control of complex systems. The architecture has been described in numerous papers including [1], [5], and [4] and is described briefly here.

A defining feature of the architecture is the combination of a "top-down" approach typical of hierarchical control architectures such as 3-T [6] with an agent-based or "bottom-up" approach. The result is a system that can accomplish complex problem solving through the

coordination of simple, task-specific agents. The justification for this approach is that simple agents have the advantage of working in smaller and more constrained problem spaces. The higher level controller, on the other hand, can make more global decisions about the entire system, for example how the current results of a focusing magnet can effect the problem of beam steering further down the beamline. (See Figure 3 for an example of an implemented control hierarchy.)

A primary strength of this framework is the ability to integrate a wide variety of representations, both analytic and knowledge-based, into a single control framework. At the core of the architecture is a group of knowledge-based controllers which apply control from a local scope. These controllers are hierarchically organized in a structural/functional hybrid design. Controllers are responsible for making decisions about how control actions will be performed, what those actions will be, when they will occur, and how their performance will be measured. Controllers are also responsible for reasoning about system state, diagnosing errors in control solutions, decomposing goals into tasks and actions, and initiating any necessary human interaction. Controllers typically control the operations of lower level software components, called solvers, that implement general purpose control algorithms including conventional closed loop, neural, or fuzzy control.

Because we use a symbolic system for reasoning about system control, raw data produced by diagnostic elements is rarely appropriate for direct manipulation by controllers. The same is true in reverse; a uniform (low-level) abstraction for manipulation of control elements is usually inappropriate. For this reason we have developed an object-oriented Physical Access Layer (PAL) for providing an abstraction mechanism between controllers and the underlying control system. The PAL is composed of a number of Physical Layer Objects (PLOs) which are abstract representations of a control element or collection of control elements. These objects can be as simple as single steering magnets, or as complex as non-linear tuning knobs which manipulate a series of magnets. PLOs communicate directly with a high speed software data bus. We use Vsystem, an off the shelf commercial control bus, as the software connection between the PAL and control hardware.

TELEO-REACTIVE CONTROL

The intelligent controllers comprising the core of our architecture are based a new adaptation of a robot control technology called teleo-reactive control [1]. Teleo-reactive control combines aspects of feedback-based control and discrete action planning. TR programs sequence the execution of actions that have been assembled into a goal oriented plan.

Unlike more traditional AI planning environments, no assumptions are made that actions are discrete and uninterruptible and that every action's effects are completely predictable. To the contrary, teleo-actions are typically sustained over an indeterminate (but strictly controlled) period of time until their goal condition is achieved. Typically, teleo-actions are executed as long as the action's preconditions are met, the associated goal has not yet been achieved, and incoming data shows that acceptable progress is being made. On the other hand, a short sense-react cycle ensures that when some critical condition in the environment changes, the control actions also change to match the new state.

TR action sequences are represented by a data structure called a TR tree. A TR tree is described by a set of condition-action pairs (or production rules):

$$C0 \rightarrow A0$$
$$C1 \rightarrow A1$$
$$C2 \rightarrow A2$$
$$\ldots\ldots$$
$$Cn \rightarrow An$$

where the Ci are conditions and the Ai are the associated actions. We refer to C0 as the top level goal of the tree and A0 as the null action, indicating that nothing further needs be done once the top level goal is achieved. At each execution cycle of the teleo-reactive system, each Ci is evaluated from the top of the rules to the bottom (C0, C1, C2, ... Cn) until the first true condition is found. The action associated with this true condition is then performed. The evaluation cycle is then repeated at a rate that reflects the desired degree of reactivity.

The Ci $\rightarrow$ Ai productions are organized in such a way that each action Ai, if continuously executed under normal conditions will eventually make some condition higher in the rule tree true. A tree reflecting this rule ordering may be seen in Figure 1. TR tree execution may be seen as adaptive in that if some unanticipated event in the control environment reverses the effects of previous actions, TR execution will fall back to the lower level rule condition that reflects that situation. From that point it will restart its work toward satisfying the higher level goal. Similarly if something "good" inadvertently happens, TR execution is also opportunistic in that control can then automatically shift to the action associated with that true condition.

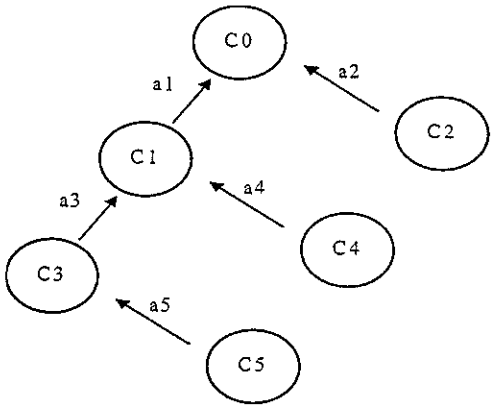

Figure 1. A simple TR tree

TR trees can be constructed with a planning algorithm that employs common AI goal reduction methods. Starting with the top level goal, the planner searches over actions whose effect includes achievement of that goal. The preconditions of those actions generate a new set of subgoals, and this procedure recurses. Termination is achieved when the preconditions of the (leaf) nodes of the tree are satisfied by the present state of the environment. Thus the planning algorithm regresses from the top level goal through goal reduction to the current state. Actions, of course, often have side effects and the planner must be careful to verify that an action at any level does not alter conditions that are required as preconditions of actions at a higher level. Goal reduction is thus coupled with constraint satisfaction, where a variety of action reordering strategies are used to eliminate possible constraint violations.

TR planning algorithms are used to build plans whose leaf nodes are satisfied by the current state of the environment. They usually do not build complete plans, that is, plans that can start from any world state, because such plans would generally be too large to store and execute efficiently. This final point is important because sometimes an unexpected environmental event can shift the world to a state in which no action preconditions in the TR tree are satisfied and some form of re-planning is necessary. This invokes reactivation of the TR planner.

We have found that the teleo-reactive planning mechanism provides a useful framework for representing and implementing accelerator tuning algorithms for several reasons:

- Accelerator beams and their associated diagnostics are typically dynamic and noisy.
- The achievement of accelerator tuning goals is often affected by stochastic processes such as RF breakdown or oscillations in the beam source.
- Actions required for tuning are often of indefinite duration. This is generally true of tweaking and optimization operations that need to be continued until specific criteria are met.
- TR trees have been found to offer an intuitive framework for encoding tuning plans acquired from accelerator physicists.

We have identified several important issues that arise in applying the teleo-reactive framework to industrial control systems. First, the assumption that parameter values referenced in the preconditions of TR rules are always accessible does not hold in many environments. In accelerator control systems, for example, it is often necessary to insert monitors at specific locations to read beam size and position. Monitor insertion, however, can be destructive, interfering with beam parameter readings further down the beamline. Thus, the acquisition of certain types of data requires actions and sequencing decisions that need to be woven into the overall plan. To handle this, we have added data acquisition actions and action sequencing decisions into the teleo-reactive control framework.

In addition, we have identified the need for adaptivity in determining parameter values for intermediate subgoals. These target values sometimes need to be adjusted after a sequence of actions is performed and its effects evaluated. Thus, we have introduced meta-rules with the effect of adjusting intermediate target values.

In the following sections we describe in more detail the extensions to teleo-reactive control that were added to make the TR framework an effective instrument for accelerator control.

PROBLEMS IN ACCELERATOR TUNING

A particle accelerator beamline is a device that is used to transport highly energetic charged particles from a source (the accelerator) to a target. The beamline consists of a number of elements designed to either change beam characteristics (direction, size, shape, etc.) or to monitor those characteristics in some way. The purpose of the beamline is to steer, focus, and otherwise modify the beam such that it is transported through the beam pipe to

a specified location while maintaining its characteristics within an acceptable range. The final beam should reach the target with a very specific set of characteristics, as determined by the experiment or work being done. Figure 2 shows a simple accelerator beamline which includes trim magnets for steering, quadrupole magnets for focusing, Faraday cups and stripline detectors for measuring current, and profile and popup monitors for measuring size and position.

Accelerator beamlines are designed by placing various components along the beam pipe to produce specific effects in a known way. A good design will minimize the number of components necessary to maintain acceptable beam conditions while still allowing enough freedom of control to achieve a range of target conditions. Unfortunately, real systems rarely work exactly as they are designed. Problems arise from imperfect beam production, remnant magnetic fields, poorly modeled beam behavior, misplaced or flawed control elements, and changes to the design or use of the facility after it has been built. Beamline designers consider these problems as well, and build diagnostic components into the beamlines. Profile monitors and current detectors are used to measure beam parameters throughout the line to provide information for verifying or correcting beam characteristics. Even so, imperfect detectors, unknown system errors, and noise can cause beamline control to be difficult at best.

Although most modern accelerator facilities have some capability for automatic beamline tuning, normal day-to-day operation still requires considerable human intervention. Human operators are particularly necessary for generating an initial tune of the beam at a target. In fact, less experienced operators may find an initial tune, and then pass control to an expert (often a physicist) for fine tuning. The manual process of generating a proper tune can be lengthy, often taking many hours and in some cases many days.

Generally, tuning a beamline is a sequential process. Human operators start using a predetermined set of known or calculated settings intended to generate a measurable beam at a downstream location. Once this baseline tune has been found, an operator will adjust various magnet settings between the source and the measurement location to produce another set of predetermined conditions that usually allow the beam to be transported to another location further downstream. When the beam can be measured at the next location, it is again tuned to a set of predetermined conditions. If the conditions cannot be achieved, the operator will back up and attempt to re-tune a prior section in an attempt to produce acceptable pre-conditions for achieving the downstream tune. The operator will then iterate between the upstream and downstream sections until the desired downstream conditions are achieved. The process continues in the same fashion: achieving intermediate conditions at a target location, tuning to a downstream location, and backtracking to achieve better pre-conditions for downstream tunes. The process is complete when the operator achieves a proper set of beam characteristics at a final target location.

Conventional approaches to the sequential tuning problem have failed for a number of reasons. The first stems from the iterative nature of the problem. Although tuning may at first appear to be solvable by following a predetermined script, such an inflexible solution cannot deal with changes in upstream goal conditions arising from downstream tuning. This is especially true when the path from a current set of target conditions to a new set is non-linear. The problem is also difficult because iterative problem solving is often necessary within major sections, where intermediate goals may need to be set by the problem solving system itself.

For example, a simple beamline tuning problem involves producing a symmetric beam at the center of a downstream monitor. This problem involves two sub-problems: 1) focusing the beam in both the x and y axis to produce a small, circular shape, and 2) steering the beam through the beam pipe so that it arrives not only on center, but also aimed along the longitudinal axis of the beam pipe. This problem is made difficult because of interactions between steering and focusing elements. If a beam is not properly steered through a focusing element, changes to the focusing element will re-steer the beam. Operators normally alternate between steering and focusing, first moving the beam to center, then focusing slightly, and then moving the beam back to center.

It is also important to note that such problems have an

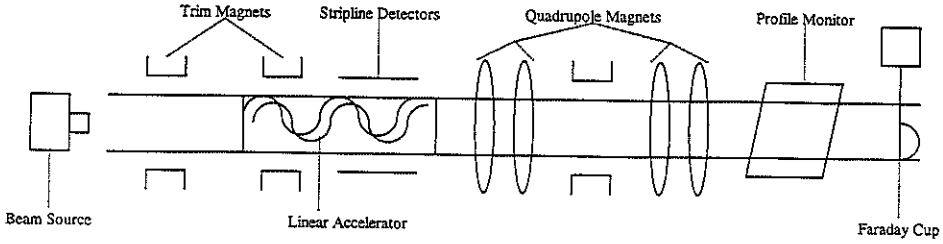

Figure 2. A simple accelerator beamline.

significant knowledge-based aspect. The steering and focusing problem is simplified knowing that a proper on-center, on-axis steer will minimize the steering effect of the focusing elements. This information is especially useful when attempting to produce an off-axis position at a target. A smart system may steer on-axis while focusing, and then adjust the final position only after focusing is complete. Although this example is greatly simplified, it illustrates the use of system-defined goal states and the need for a knowledge-based approach.

A second reason that conventional scripting solutions fail is the need for dynamic identification of intermediate targets and goal conditions. For example, a common tuning goal is achieving a small round beam spot at the end of the beamline. During the early stages of the tuning process, there may be no measurable spot at the end of the beamline. An obvious solution is to pick short range intermediate targets and tune to those, one after another until the beam is tuned to the end. Even if a beam spot does appear at a downstream monitor, width measurements may be inaccurate because of various upstream effects, yielding an incorrect tuning model. It is often necessary to iterate tuning the entire beamline to gain a proper model and a correct tune.

It is important to note that in beamline tuning, the use of *predefined* intermediate goals is inadequate for achieving a final solution. It is usually impossible to know whether an intermediate tune is "good enough" without testing beamline tunability downstream. The tuning process itself thus provides valuable data for building a model of beamline behavior that can be used in subsequent tuning.

Conventional scripting fails primarily because it lacks the flexibility to determine subgoals interactively based on conditions in the current environment. The teleo-reactive approach provides the required flexibility in plan execution. The TR approach allows 1) intermediate goals. 2) automatic backtracking. 3) goal jumping when we get good enough results, and 4) robust behavior when the environment is poorly modeled and results of actions are uncertain. By using a hierarchical approach to TR planning, we also get 5) intermediate goal refinement for macro-level backtracking.

EXTENSIONS TO THE TR FRAMEWORK

Controllers are the primary knowledge-based components of our control architecture. We chose the TR representation for encoding control and domain specific information in controllers. In order to make TR work in our control system, however, we made the following changes:

The original TR representation of Nilsson/Benson used simple condition action pairs to encode rules. While we found this representation to be appropriate for limited scope agents, we needed to expand the capabilities of actions to support a distributed hierarchy. We extended the original formulation by allowing actions that invoke subroutine-like behavior. This is implemented in the following ways: 1) an action may delegate control to another controller with its own TR rule set, 2) an action may trigger swapping the current TR tree with another tree within the same controller, or 3) an action may cause sets of TR rules to be loaded or unloaded. This type of behavior is especially useful in circumstances where the integration of a complete set of TR rules in a single system would cause unnecessary complexity and possibly unpredictable behavior.

A closely related modification to the TR architecture is the extension of the "action' of TR rules to include the invocation of problem solving algorithms as opposed to simple actions. This more complex problem solving occurs through independent control structures (i.e., solvers in our architecture). For instance, rather than simply modifying a control point in response to a monitored condition, the TR tree may also initiate a standalone control algorithm. This "delegation" of control is done in a manner similar to the "subroutining" described above. In either case, special monitoring conditions are set so that the TR tree regains control whenever a higher level rule becomes active, or when the control algorithm finishes execution.

One of the difficulties in applying TR rules to real data is determining a set of rule preconditions which will work in a dynamic system. A third modification we made to the original TR architecture is the use of variable ranges rather than single point goals. In fact, because we used PROLOG to encode our rule sets, any PROLOG clause could be used as a valid goal precondition. For beamline tuning, we used this flexibility to specify ranges of valid monitor readings to indicate general beam characteristics (e.g., a width from 10 to 16 mm is a good "small" beam.)

As stated previously, one of the restrictions of more traditional approaches is an inflexibility in setting intermediate goals, working on downstream sections, and then backtracking using a new set of goals. A fourth modification we made is the use of meta-rules which modify target values. These meta-rules can be used to change the behavior of a TR tree without modifying the structure or ordering of the rule set itself. For instance, the first pass of a tuning algorithm may generate a very rough, on-axis tune at a downstream monitor. After further downstream tuning, a new set of target conditions, along with narrower tolerances, can be used to retune the initial section. Meta-rules are also useful for producing iterative behavior within a tree. For example, we used meta-rules to modify both precondition and target settings for pairs of competing rules. The meta-rules caused the competing rules to alternate firing until a higher level precondition was satisfied. This alternating behavior is especially

useful in situations where "we don't know how close we can get until we try."

A final modification to the original TR architecture is the addition of explicit observation actions. In using TR trees to control accelerators, we realized that the observations necessary to form an accurate picture of the control environment, and therefore necessary to evaluate TR conditions, are not always automatic or cost free. For example, in order to determine beam loss at a particular location in the beamline, a Faraday cup may need be inserted to measure beam charge. This action is called a "destructive" action, in that all beam is lost beyond the insertion point. Furthermore, the use of many such monitoring devices can cost time, since these devices must be mechanically inserted or retracted.

A more difficult problem arises when certain conditions (rule preconditions) cannot be detected except through the use of a complex control sequence. For instance, determining whether a beam is exhibiting a "waist" condition requires modifying focusing elements and observing the beam at two or more profile monitors. In such cases, the entire set of TR preconditions cannot be reevaluated during every TR cycle. Instead, additional TR rules are used to indicate when and if observation actions are used to reacquire costly information.

AN EXAMPLE FROM ATLAS

As part of our research, we constructed software control systems for tuning several particle accelerator beamlines. A primary goal was to automatically tune the PII beamline of the ATLAS facility at Argonne National Laboratory. ATLAS is a facility for experimenting with heavy isotopes. Ions are generated on a high voltage platform and accelerated by the PII accelerator. The PII beamline connects the high voltage platform to the accelerator. The purpose of the line is to transport the beam through a 180° achromatic bend, chop and bunch the beam into packets, and then prepare the beam for entrance into the accelerator. The standard tuning procedure used by human operators breaks control into four sequential parts: tuning the PII0 line, tuning the PII1 achromat, tuning the PII2 beamline, and then refining the entire tune to produce proper beam conditions at the exit of the accelerator.

Our control system, constructed first for the ATF beamline at Brookhaven National Laboratory and then ported to ATLAS, is a distributed, hierarchical control system capable of executing stored teleo-reactive procedures. Distributed control is implemented through a set of knowledge-based controllers, each of which is an "expert" in controlling a section of the beamline. The controllers may perform reasoning, execute actions for directly controlling the beamline, monitor the state of the system, or delegate actions to lower-level controllers or to solvers. Solvers are reusable components that implement general purpose optimization or control algorithms, such as hill-climbing optimization, fuzzy logic or neural network-based feedback control, conventional control loops, etc. All control actions are sent to the Physical Access Layer (PAL) which is responsible for converting higher-level commands into implementation specific control signals. Figure 3 illustrates the basic layout of the control hierarchy for ATLAS.

The following code fragments are parts of the original TR trees used to control the PII beamline at ATLAS. They are included to illustrate the various modifications we made to the TR architecture.

PII - TRANSVERSE

```
rule(7,
  'This rule is for tuning pii1',
  [tuned(pii0), not(tuned(pii1)), not(inserted('FCP001'))],
  delegate(pii1, pii1, [model, nul, tune_pii1]),
  [transmission('FCP201')],
  [set(tuned(pii1))],
  50).

rule(8,
  'Retract the FCP001 before tuning pii1',
  [tuned(pii0), not(tuned(pii1))],
  pal_set([['FCP001', position, out, 0]]),
  [],
  [unset(inserted('FCP001'))],
  50).
```

Rule 7 is a simple example of a TR rule which delegates responsibility of tuning the PII1 beamline to another TR

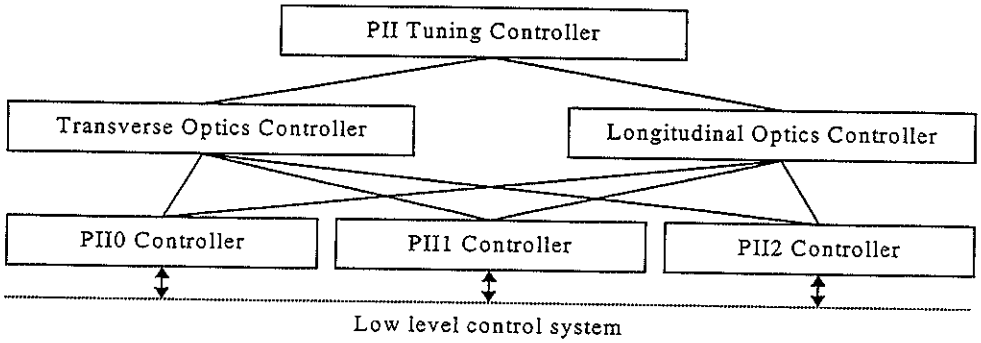

Figure 3. Control hierarchy for ATLAS

controller. The rule states that "if pii0 is tuned and pii1 is not tuned and the FCP001 Faraday cup is retracted" then "delegate control to the pii1 controller" with the goal "tune_pii1". Delegation is often as simple as this rule, but can also include more complex messaging and additional information. The keyword "model" is used to pass a reference to a globally available model, although in rule 7 the model is null.

Rule 8, ordered lower in the TR rule list, is used to retract the FCP001 Faraday cup in preparation for firing rule 7 (guaranteeing one of the higher level preconditions.) Pairs of rules like 7 and 8 are common in our TR control algorithms for performing simple control steps in preparation for more complex ones.

```
rule(3,
  'This rule handles an ask for tuning pii0',
  [agoal(tune_pii0)],
  delegate(pii0, pii0, [model, pii1_preconditions, tune_pii0]),
  [],
  [reply(tune_pii0, ready, yes)],
  50).
```

A more complex version of PII – Transverse includes rules like rule 3 above. Our control system is both knowledge-based and distributed. Information regarding most preconditions for tuning PII1 are stored in the PII1 controller, not in the higher level PII Transverse controller. Rule 3 allows the PII1 controller to ask its hierarchical parent, PII Transverse, to tune PII0. When the ask precondition [agoal(tune_pii0)] occurs in PII Transverse, rule 3 can fire and initiate tuning in the PII0 controller. In this case, model information containing PII1 preconditions can be transferred to the PII0 controller as a new set of goals.

PII – O

```
rule(7,
  '',
  [well_focused, well_steered],
  ask(parent, help, tune_pii1),
  [answer],
  [test(yes, tuned('PII1'))],
  100).

rule(8,
  'Steer after well focused',
  [well_focused],
  delegate('1d_hill', nul, [ctrl_objs,
   ['STP001', 'STP001', 'STP002', 'STP002'], ctrl_params,
   [current_x, current_y, current_x, current_y],
   snapshot_mode, off,
   diag_objs, ['FCP001'],
   diag_params, [current], weights, [10000], exponents, [10000],
   opt_mode,maximize, step_size, 5000]),
   [transmission('FCP001')],
   [set(well_steered)],
   100).

rule(9,
  '',
  [tolerance_or_more([transmission('FCP001'),
goal(transmission('FCP001')),
  tolerance(transmission('FCP001'))]),
   focused_for_transmission],
   delegate('2d_focus', nul, [ctrl_objs, ['QDP001', 'QDP001'],
ctrl_params,
   [current_x, current_y], diag_obj, 'PMP001', tx,
goal(sigma_x('PMP001')),
   ty, goal(sigma_y('PMP001')), step_sizes, [10000, 10000]]),
   [sigma_x('PMP001'), sigma_y('PMP001')],
   [set(well_focused)],
  100).

rule(10,
  '',
  [tolerance_or_more([transmission('FCP001'),
goal(transmission('FCP001')),
  tolerance(transmission('FCP001'))]),
  inserted('FCP001')],
  delegate('1d_hill', nul, [ctrl_objs, ['QDP001', 'QDP001'],
ctrl_params,
   [current_x, current_y], snapshot_mode, off, diag_objs,
['FCP001'],
   diag_params, [current], weights, [10000], exponents, [10000],
opt_mode,
   maximize, step_size, 10000]),
  [transmission('FCP001')],
  [set(focused_for_transmission)],
  100).

rule(11,
  '',

[fired(rule14),significantly_greater_than([transmission('FCP001'),
default('FCP001', current, actual)]),
  inserted('FCP001')],
  delegate('1d_hill', nul, [ctrl_objs, ['STP001', 'STP001'],
ctrl_params,
   [current_x, current_y], snapshot_mode, off, diag_objs,
['FCP001'],
   diag_params, [current], weights, [10000], exponents, [10000],
opt_mode,
   maximize, step_size, 10000]),
  [transmission('FCP001')],
  [],
  100).
```

Rules 7 – 11 above come from the TR tree used for tuning the PII0 beamline. Rule 7 illustrates how a child controller can ask its hierarchical parent to perform a service in preparation for its own continued tuning. In this case, the PII0 controller will need to use a Faraday cup at the exit of the PII1 beamline as a diagnostic element for final PII0 tuning. The PII0 controller will block and wait for a response from the PII Transverse (parent) controller before continuing with tuning.

Rule 8 illustrates how the TR tree can perform a complex action by passing control to a solver. If the TR tree has determined that the PII0 line is "well focused", it will use simple hill-climbing to optimize steering to maximize beam transmission.

Rules 9 and 10 show how ranges may be used rather than strict preconditions. In these rules, "goal()" and "tolerance()" are PROLOG functions which are bound at the time the rule is evaluated. Meta-rules may modify the values associated with these functions in order to change the desired target or accuracy necessary for the rule to fire. Rule 11 also shows how a function,

"significantly_greater_than()" may be used like a fuzzy variable to determine whether the precondition is satisfied. Again, the actual behavior of the function may change at runtime in response to other TR rules.

Rules 10 and 11 also show how iteration may occur in the TR tree. Rule 11 activates steering to achieve maximum transmission and, in this case, to also reduce steering effects during focusing. Once steering has been optimized, rule 10 can fire, focusing in order to minimize beam size at a downstream monitor. If steering is badly affected during focusing, the transmission preconditions will cease to be satisfied, and the steering rule will fire again. These two rules will alternate until a proper combination of steering and focusing is achieved.

RESULTS

We have used our TR-based control system successfully at both the Brookhaven and Argonne accelerators. At the Brookhaven ATF, we were able to assist in diagnosing misalignments in quadrupole magnets. The normal method for doing this is shutting down the accelerator and then manually measuring the placement of each magnet. The lead physicist at the ATF, Xijie Wang, suggested an algorithm which could be automated for diagnosing the misalignment through a series of tuning and testing steps. Our system first tuned the beamline, and then manipulated focusing and steering elements to correctly identify a misaligned quadrupole.

We also achieved significant success tuning Argonne's ATLAS. Using the architecture described previously, we successfully tuned the beam from the high voltage platform, through PII0, around the PII1 achromat, and through PII2, achieving a proper beam spot at the exit to the PII Linac. The TR representation worked well, achieving tunes equal to or better than benchmark tunes set by ATLAS operators. We found TR sequencing to be particularly robust, continuing to function even during periods of instability in the beam source. The controller would adapt to such circumstances by reverting to rules at the bottom of the rule set. Once steering and focusing corrections were made, the controller would jump back to higher level tuning rules and continue.

CURRENT STATUS

Developing of TR rule sets for ATLAS and the ATF was a lengthy process requiring significant testing, tweaking, and reworking. We have developed an automatic planning mechanism for generating TR trees at run-time based on intermediate goals generated during tuning. This automatic planner has not yet been integrated with our control system or tested on an actual beamline. The planner will require a significant knowledge-based component containing considerable heuristic and experiential information relevant to the facility being controlled. In addition, the entire system will have access to a model of the beamline that will be used to predict effects of proposed actions, diagnose problems, and to perform model-based control.

Future efforts will also be directed toward utilizing the distributed nature of the architecture to perform simultaneous control of multiple beamline sections. Higher level controllers will act as real-time supervisors and mediators for lower-level controllers operating over individual beamline sections. The use of various agent communication techniques will be crucial to prevent controllers from working against each other or, even worse, putting the beamline in unstable or unsafe states.

ACKNOWLEDGMENTS

This work was supported by a DOE SBIR contract (#DE-FG05-95ER81897) to Vista Control Systems, Inc. We greatly appreciate the efforts of Richard Pardo for helping us design our algorithms, and Floyd Munsen for helping us connect our architecture to the ATLAS control system.

REFERENCES

[1] Nilsson, N.J., Teleo-Reactive Programs for Agent Control. *Journal of Artificial Intelligence Research*, 1, 1994,:139-158.

[2] Benson, S., Inductive Learning of Reactive Action Models. *Machine Learning: Proceedings of the Twelfth International Conference* (San Francisco: Morgan Kaufmann, 1995).

[3] Klein, W., *A Software Architecture for Intelligent Control* (Doctoral Dissertation, Department of Computer Science. University of New Mexico, Albuquerque, NM., 1997).

[4] Stern, C., Olsson, E., Kroupa, M., Westervelt, R., Luger, G., Klein, W. A Control System for Accelerator Tuning Combining Adaptive Plan Execution with Online Learning, *Proceedings of ICALEPCS'97*, 1997.

[5] Klein, W., Stern, C., Luger, G., Olsson, E.. An Intelligent Control Architecture for Accelerator Beamline Tuning, *Proceedings of AAAI 97*, 1997.

[6] Ansaklis, P.J., Passino, K.M., and Wang, S.J., Towards Intelligent Control Systems: Architectures and Fundamental Issues. *Journal of Intelligent and Robotic Systems*, 1, 1989, 315-342.