A Control System for Accelerator Tuning
Combining Adaptive Plan Execution with Online Learning

C. Stern, E. Olsson, M. Kroupa, R. Westervelt
Vista Control Systems Inc.
1137 18th Street, Los Alamos, NM 87544
G. Luger, W. Klein
Dept. of Computer Science, University of New Mexico
Albuquerque, NM 87131

Abstract

Vista Control Systems, Inc. is developing a portable system for intelligent accelerator control. Our design is general in scope and is thus configurable to a wide range of accelerator facilities and control problems. The control system employs a multi-layer organization in which knowledge-based decision making is used to dynamically configure a lower level optimization and control algorithms. An object-oriented physical access layer (PAL) supported by the Vsystem control database allows abstraction from the lower level details of hardware manipulation, signal processing, and synchronization. A teleo-reactive interpreter [1] is used to sequence control actions. To this framework we have been adding planning, diagnostic, and learning capabilities.

1 An architecture for knowledge-based control

The Intelligent Controls Group at Vista Control Systems has been developing a general purpose, portable, intelligent control architecture. To date this control architecture has been tested at the Brookhaven National Laboratory ATF facility as well as the ATLAS line at Argonne. In this paper we describe briefly the concept of hierarchical distributed knowledge-based control that guides our development. We refer the reader to our other publications for more details [2,3].

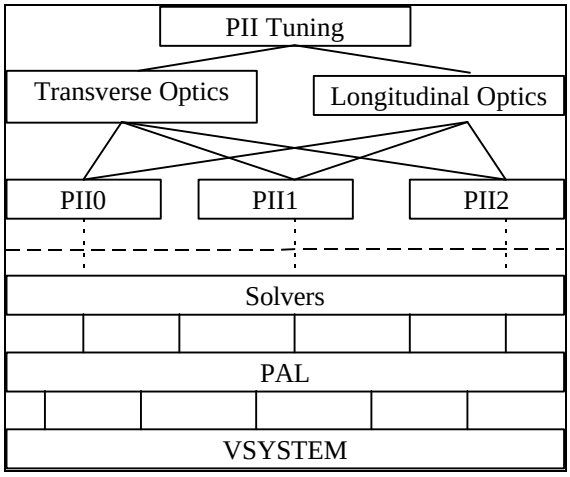

Figure 1. A control hierarchy for ATLAS.

Distributed control is implemented through a set of knowledge-based controllers, each of which is an "expert" in controlling some section of the environment or in performing some function over that environment. Knowledge-based controllers use planning, diagnosis, and learning, as well as knowledge acquired from human domain experts to select, sequence, and configure control actions. Controllers are also responsible for reasoning about the system state, diagnosing errors in controller actions, decomposing goals into tasks and actions, and initiating human intervention as necessary. Controllers are hierarchically organized in a structural-functional hybrid design, an example of which is shown in Figure 1.

Controller actions include delegating tasks to other lower level controllers or calling the services of a simpler type of control component called a *solver*. 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. Individual solvers are generally assembled and coordinated so as to perform more complex procedures. For example, locating the magnetic center of a quadrupole is performed by connecting a hill climbing minimization solver that scans the magnetic surface of the quadrupole with a solver that measures the quad steering, i.e., the displacement at a monitor associated with tweaking the quad.

Because the control system is knowledge-based, raw data is rarely appropriate for direct manipulation by the controllers. The opposite is also true: a low level interface for manipulation of control elements is usually inappropriate. For these reasons we developed an object-oriented physical access layer or PAL as an abstraction mechanism between controllers and the underlying control system. The PAL offers a number of advantages.

First, the PAL provides a mechanism for hiding unimportant implementation details about the domain hardware and provides a uniform interface for control access. Second, resource conflicts can be handled, at least initially, at a low level, and when appropriate, mediated at the controller level. Third, controllers can pass filtering instructions to the PAL to allow preprocessing of data into a representation expected by a controller. This can happen, for example, by giving the PAL fuzzy sets for classifying data, or by passing a neural net encoding to the PAL. Finally, the PAL is highly portable. By abstracting

underlying control elements, control algorithms can be written in a generic manner. The same control framework is usable in a variety of environments by merely updating and exchanging the PAL objects.

2 Advantages of a distributed agent-oriented design

Our architecture for intelligent control is intended to be general purpose, with application to any domain where complex nonlinear subsystems must be controlled to perform specific tasks. As a result, it combines the top-down approach of current hierarchical control architectures, such as 3-T [4] with a bottom-up, "agent based" approach. The result is a system that can accomplish complex problem solving through the coordination of simpler, task specific agents.

The justification for this design philosophy can be summarized as follows. Simple, task specific agents have the advantage of operating in a smaller, more constrained problem space, which simplifies the tasks of learning and adaptation. Capabilities and performance improvements acquired through learning in this smaller problem space propagate up the hierarchy by providing a wider range of basic capabilities that can be assembled into larger units. Functional composition thus allows the system to leverage performance improvements at lower levels into performance improvements at all levels of the hierarchy.

We have taken several approaches to learning and adaptaton. We have experimented with low level learning in the form of pattern recognition and feature detection in the PAL through the use of adaptive filters attached to feedback objects. Neural networks, fuzzy classifiers and other adaptive tools have been used to identify and store information regarding system state.

A second locus of learning and adaptation is at the controller level. Our teleo-reactive execution regime, described in the next section, provides a variety of opportunities for adaptation. Adaptive execution has played an important role in our field tests at Brookhaven and Argonne [2]. We are currently working on extending our teleo-reactive execution environment with action model learning and adaptive planning. A description of this work is the main focus of the remainder of this paper.

3 Teleo-reactive control

Controllers in our architecture use an intelligent executive mechanism called teleo-reactive (TR) control [1]. TR control occupies a region between feedback-based control and traditional discrete action planning. TR programs sequence the execution of actions that have been assembled into a certain kind of goal-related plan. Unlike traditional planning environments, no assumption is made that actions are discrete and uninterruptible and that an action's effects are completely predictable. On the contrary teleo-actions are typically durative and are executed as long as the action's preconditions hold and its goal has not yet been achieved (unless some other action closer to the final goal becomes activated.) A short sense-react cycle ensures that when the environment changes, the control action changes to fit the new state.

The assumptions under which TR plans are assembled differ from the assumptions of conventional discrete action planning. The primary differences are:

i) Actions can be either discrete or continuous.

ii) Actions are not guaranteed to achieve their goals. Continuous or durative actions are expected to achieve their goals when executed for an indefinite period of time under normal conditions. Repeated failure to achieve its goal under normal conditions can result in retraction or modification of the action operator.

iii) Actions can be interrupted in response to changes in the environment.

iv) Like the primary effect or goal of an action, other effects (side effects) are not guaranteed to result. The only guarantee is that if the action is performed continuously for some period of time, known side effects will occur stochastically with some probability.

TR action sequences or plans are represented in a data structure called a TR tree. A TR tree can be described as a set of condition-action pairs:

$C0 \rightarrow A0$
$C1 \rightarrow A1$
…
$Cn \rightarrow An$

where *Cs* are conditions and *As* are the associated actions. *C0* is typically the top level goal of the tree and *A0* is the null action, i.e., do nothing if the goal is achieved. At each execution cycle the *Ci*s are evaluated from top to bottom until the first true condition is found. The associated action *Ai* is then performed. The evaluation cycle is then repeated at a frequency that simulates the reactivity of circuit based control.

TR trees are constructed so that each action *Ak*, if continously executed under normal conditions, will eventually make some condition higher in the tree true. This then ensures that under normal conditions the top level goal, *C0*, will eventually become true. TR tree execution is adaptive in that if some unanticipated event in the environment reverses the effects of previous actions, TR execution will typically fall back to some lower level condition and restart its work towards the top level goal. On the other hand, if something "good" happens, TR execution is opportunistic: when some higher condition unexpectedly becomes true, execution shifts to the action associated with that condition.

We have been employing teleo-reactive execution in our accelerator control system for over six months with considerable success. It has now been tested at both Brookhaven and Argonne. It has been particularly effective in adaptively resteering in the middle of focusing and linac tuning operations to correct for incidental orbit changes caused by those operations.

The teleo-reactive mechanism provides a useful framework for representing and implementing accelerator tuning algorithms for several reasons:

i) Accelerator beams and their associated diagnostics are typically dynamic and noisy.

ii) Achievement of tuning goals is often affected by stochastic processes such as RF breakdown or oscillations in the source.

iii) Many of the actions used for tuning are durative. This is often true of tweaking and optimization operations: they are typically done until specific criteria are met.
iv) We have found TR trees to be an intuitive framework for encoding tuning plans acquired from accelerator physicists. Richard Pardo of Argonne, for example, encoded, with relatively little assistance, a TR plan for tranverse tuning of the PII beam line.

3.1 Teleo-reactive planning

TR trees can also be represented as graphs (Figure 2) encoding the relationships between goals, actions, and subgoals. At each execution cycle, the action associated with the highest true condition in the tree is selected for execution. In Nilsson's original formulation [1], when the highest active level contains more than one action with satisfied preconditions, some arbitrary probabilitistic method is used for choosing between possible actions. We have modified this by dynamically maintaining numerical confidence levels associated with actions and choosing actions with the highest associated confidence.

Construction of TR trees can be accomplished through a planning algorithm that is a modification of common AI goal reduction planning algorithms. Starting from the top level goal, the planner searches over actions whose effect includes achievement of the goal. The preconditions of the action generate a new set of subgoals, and the procedure recurses. Termination is achieved when the preconditions of one of the leaf nodes of the tree is satisfied by the current state of the environment. That is, the planning algorithm regresses from the top level goal through goal reduction to the current state. Actions, of course, generally 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 must consequently be combined with constraint satisfaction, where a variety of action reordering strategies are employed to eliminate constraint violations.

TR tree planning algorithms typically build plans whose leaf nodes are satisfied by the current state of the environment. They do not build *complete* plans, that is, plans that can start from any world state, because such plans would generally be too large to efficiently store and execute. This is an important point because sometimes an unexpected environmental event can shift the world to a state in which no action preconditions in a TR tree are satified. In that case, replanning is necessary.

3.2 *Teleo-operators*

Durative actions require a reformulation of traditional STRIPS-style action operators [5]. During the continuous execution of a durative action, there can be a sequence of intended and incidental (side effect) state changes in the world, all of which are consistent with continuing the execution of the action until its goal condition is produced. For this reason, use of a durative action to achieve a goal is associated with a range of potential starting states from which the action can be effective. Instead of the precondition-based action operators used in traditional

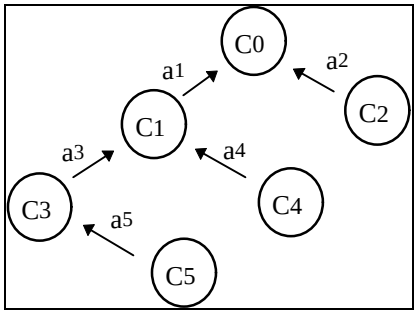

Figure 2. A TR plan graph

STRIPS-style planning [5], teleo-reactive planning must rely on a concept borrowed from robot motion planning: the concept of a preimage. A preimage π is the regression of a goal condition through an action. It is defined as the condition or constraint which an initial state must satisfy in order that continuous performance of the action will, under normal conditions, produce the goal condition.

A preimage π typically defines a region of potentially effective action. This can perhaps best be understood by considering an example from the motion planning domain. Suppose the goal is to move the robot to some location (x,y). The action is to move NE, i.e., 45^{o}. The preimage π that results from regressing the goal, move to (x,y), through the action, move NE, is the 45^{o} line segment extending SW from (x,y). Note that the action, move NE will pass through the point (x,y) iff it is started from a point on this specific 45^{o} line segment.

A goal-action pair and its associated preimage are called a teleo-operator (TOP). Preimages for TOPs are often difficult to compute with precision because it is difficult to anticipate the full range of conditions under which an action will fail (or succeed). One method proposed by Benson [6] is to use approximate preimages and then learn more accurates preimages from experience. This involves keeping traces of the TR tree execution and then using inductive concept learning to induce preimages from the trace data. If preimages are represented as conjunctions of condition literals, induction of preimages is the problem of finding the right concept, i.e., the right conjunction of literals that is satisfied in all the cases in which the TOP was used successfully but unsatisfied in all the unsuccesful cases. A variety of machine induction algorithms, including Inductive Logic Programming (ILP) [7] can be applied to this task.

On the other hand, the use of the TOP representation in a highly constrained, well-modeled domain such as particle accelerators presents other alternatives. There are certain obvious ways in which to derive TOPs from accelerator models. Assuming the goal is to produce a specified beam conditon *Cb* at some location *L*, one method is to start from the beam condition at some upstream location L_0 and ask the model to fit for magnet settings (for appropriate magnets) that give the condition *Cb* at *L*. In other cases, one could make repeated calls to a fitting algorithm to find the *range* of upstream beam conditions at L_0 from which

Cb can be produced. Another possibility is to use simulation to experiment with potential actions and establish the preimages of each.

4 Diagnosis, replanning and learning in the TR model

Execution failure can occur in a TR tree in three ways:
i) the TR tree enters a state in which no node is active,
ii) the TR tree is stuck in a cycle involving two or more nodes, or
iii) the tree is stuck in a cycle involving a single node.

The first case arises because the control plan is incomplete and does not include an action responding to some unexpected state. In the second case some node repeatedly terminates execution before its parent node becomes active (because it makes it makes its own preimage condition false). In the third case a single node repeatedly executes without making its parent active.

In the first case, the plan fails because the controller has entered a state for which it does not have an appropriate action. This state might have arisen as an unanticipated consequence of the TOP action or as a result of some external perturbation. In either case, failure occurs because the plan is incomplete and needs to be extended to cover the new unanticipated state.

Following Benson's suggestion [6], the way in which we handle this case is through replanning. Rather than build a new plan from scratch, replanning extends the current plan by goal reduction from top level goals to the new state, using the same method as the original planning algorithm. This will, in effect, add a new branch to the tree in addition to the original branches. We expect that after some long term period of application and repeated replanning, the tree will be extended to handle all of the commonly occurring states in its environment.

The second and third types of failure, multi- and single-node cycling, are either related to a mechanical failure in an actuator or sensor or result from an inaccurate action model (TOP). Our control system relies on solver routines for automatic magnet and BPM calibration to distinguish mechanical malfunctions from TOP failures.

TOP failures result either from a preimage condition π that is either too large or too small. In the former case, the starting condition for the action is not tightly enough constrained to ensure that the action will produce its intended effect. In the latter case, the preimage condtion is too tightly constrained and becomes false before the action finishes execution, even though it would have achieved its intended goal.

Our current development path includes the automatic derivation of action models or TOPS from accelerator models. When TOPs that are derived from model fitting are diagnosed as having incorrect preimage conditions, it follows that there is a corresponding error in the accelerator model. This suggests the possibility of using the diagnosis of TOP failures to detect errors in accelerator models. We are considering possible development of an automated tool for analysing model inaccuracies based on this approach.

5 Summary

We have described a distributed hierarchical architecture for accelerator control with adaptation and learning capabilities. This hybrid architecture integrates a variety of methodologies, including teleo-reactive trees for dynamic exception handling and replanning. Preliminary tests [2] indicate the potential for equaling the performance of skilled human operators. Continuing research includes extending the diagnostic, planning and learning components of the system.

Acknowledgements

This work was supported by a DOE SBIR grant (#DE-FG05-95ER81897) to Vista Control Systems, Inc.

References

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

[2] Klein, W., Stern, C., Luger, G., and Olsson E. 1997. An Intelligent Control Architecture for Accelerator Beamline Tuning. *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, Cambridge MA: MIT Press.

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

[4] Ansaklis,, P. J., Passino, K. M., and Wang, S. J. 1989. Towards intelligent control systems: Architectures and fundamental issues. *Journal of Intelligent and Robotic Systems*, 1: 315-342, 1989.

[5] Fikes, R. E. and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2, 189-208.

[6] Benson, S. 1995. Inductive Learning of Reactive Action Models in *Machine Learning: Proceedings of the Twelve International Conference*, San Francisco CA: Morgan Kaufmann.

[7] Muggleton, S. and Feng, C. 1990. Efficient induction of logic programs in *Proceedings of the First Conference on Algorithm Learning Theory*, pp. 368-381.