Experimenting with Control in the DVMT

Keith Decker Marty Humphrey Victor Lesser [1]
Computer and Information Science Department
University of Massachusetts

COINS Technical Report 89–85
August 16, 1989

Abstract

In order to operate effectively in real-time situations, or to integrate into a single control architecture a variety of control regimes appropriate for different classes of problems or stages of a problem, the low-level control loop of a blackboard-based problem solver must change significantly. For example, in a real-time system the low-level control loop must support the ability not to do exhaustive analysis of blackboard data and not to apply all applicable knowledge. Similarly, at different stages of problem solving, it may be appropriate for a system to operate with data-, goal-, or plan-based control regimes.

A modular control architecture is proposed to answer all of these needs. Its approach is based on re-engineering the basic blackboard control loop to be highly parameterized — not just in the area of agenda management, but in each area of the basic control cycle. The details of this parameterization will be presented along with examples of how this control architecture allows the expression of a range of control regimes and the reduction of overhead for real-time problem solving.

[1]This work was partly supported by the Office of Naval Research under a University Research Initiative grant, number N00014-86-K-0746, NSF-CER contract DCR-8500332, and ONR contract N00014-89-J-1877.

Introduction

The Distributed Vehicle Monitoring Testbed (DVMT) [12] is a blackboard-based system with a rigid and complex control structure. This structure allows both data-directed and goal-directed problem solving[4]. Extensions added over time have allowed sophisticated goal processing and more intelligent KSI scheduling[10].

More recently, researchers [6,1,2,9,5] have proposed plan-based control systems for the DVMT domain. Implementing these systems, however, required either partially bypassing the existing control mechanism or completely rebuilding the control component. Other researchers [13,9] have been exploring how real-time constraints effect problem solving in the DVMT domain. Real-time techniques such as approximate processing [11,13] require reasoning beyond the traditional data-directed/goal-directed dichotomy. There has also been the recognition that meta-level reasoning about how much time is spent in control versus domain processing should also be allowed.

In order to support these diverse lines of inquiry, the DVMT low-level control loop has been streamlined and parameterized: KSIs produce hypotheses that produce goals that produce more KSIs (see Figure 1). This new loop can be characterized as evaluating the blackboard to decide:

1. What potential work can be done (hyp-to-goal mapping)
2. Relating potential work to existing goals (goal merging and subgoaling)
3. Determining what goals are important to achieve (goal filtering)
4. Deciding how to go about achieving them (goal-to-KSI mapping)
5. Choosing which ones of these to actually do (managing the agenda, or "KSI filtering").

It is controlling each of these activities dynamically that is the key to efficient real-time specialization for specific situations, as well as allowing data-, goal-, and plan-based control regimes. Not only does this control architecture give us the flexibility to dynamically reconfigure the system for different control regimes and to exploit approximate data and knowledge for real-time processing, but it also enables us to reduce the low-level control processing overhead. It ensures that all of the KSIs that appear on the KSI agenda are potentially *useful* to run (as opposed to merely *runnable*). Finally, allowing control over all of the phases makes control strategies clearer and easier to implement by reducing the amount and type of data that must be considered at each point. This conceptual clarity allows rapid prototyping of control strategies.

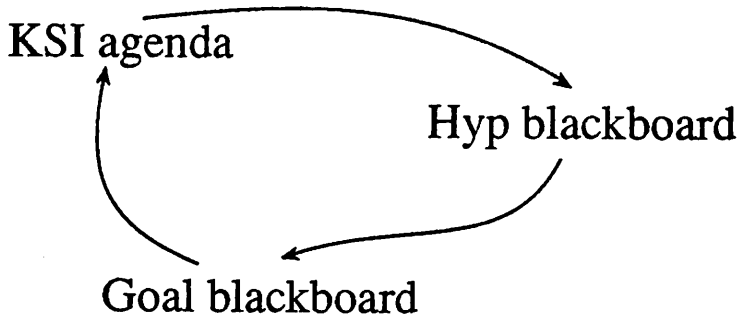

Figure 1: Simple View of the Low-level DVMT Control Loop

In the initial implementation, this low-level loop is controlled by a BB1-style control mechanism[8], where all aspects of the low-level control loop can be modified — what activities are placed on the agenda, why they get there, and the amount of effort involved in making these decisions. The ability to dynamically modify the low-level control loop is an extension of ideas developed originally in BB1 for dynamically specifying the predicates used to evaluate activities on the agenda in order to impose different high-level strategies. In more recent work, B. Hayes-Roth has also proposed extensions similar in character to some developed in this paper for controlling other aspects of agenda maintenance[7].

1 Architecture

Between each major data structure (KSI agenda, hypothesis blackboard, goal blackboard) there is a mapping — hypotheses to goals, goals to KSs (KS triggering), KSIs to hypotheses (KSI execution). These mappings can be changed by the meta-control component to indicate what potential work a hypothesis represents, or what methods should be considered in trying to achieve a goal. Subgoaling can also be considered as a mapping of goals to other goals.

For both real-time and plan-based control, the meta-controller needs to limit the hypotheses being considered by location, belief, blackboard level, and so on. The same holds for goals and KSIs. A filter located before each mapping allows the meta-controller to reduce the amount of data being considered at any point, either to reduce overhead (in real-time situations) or distraction (in plan-based coordination situations). Data that is blocked by the filter is stored so that when the filter changes the blocked data may be refiltered if desired.

The meta-controller can also control how hypotheses, goals, and KSIs are grouped and merged into larger units to avoid duplication of effort or to

reduce the amount of data being considered. This process is invoked after each mapping.

Of course the meta-controller still has access to traditional control methods — it may reason about and insert hypotheses, goals, and KSIs into the pipeline on its own, control the KSI agenda queue cluster width, and it can control the rating of KSIs on the agenda to express preferences in the order of processing. Knowledge sources still have preconditions, and these preconditions return a cost and benefit rating that is used to choose the best KS for a goal.

Finally, it is postulated (though not yet implemented) that the low-level control mechanism can run asynchronously with respect to the meta-controller. The meta-controller is notified of events of interest (for example, an empty KSI agenda) and observes the performance of the low-level mechanism. It may then modify that mechanism dynamically during problem solving. It could also examine the KSI queue after a KSI is chosen but before it is run, so that a different KSI could be executed if needed [3].

1.1 Parameterizing the DVMT

To examine how this approach works, let us step through the cycle in the DVMT (see Figure 2). When a DVMT node is executed, the KSI filter chooses a KSI (or set of KSIs in the multiprocessor case) to execute in the usual way — the first KSI in the queue cluster is chosen, and if the queue cluster is empty then all the KSIs within some distance of the highest rated KSI on the agenda are grouped into a new queue cluster and the highest KSI is chosen. The executed KSI is rated based on the expected costs and benefits (computed by the KSI precondition, see below) including the goal or goals it was intended to satisfy.

KSIs produce hypotheses (or sometimes received goals) which are merged into the hypothesis (or goal) blackboard. If the goal of a KSI was not data-directed[1], then a test for goal satisfaction is made, and the goal is marked as either satisfied or failed. The expected and actual costs (such as time) and benefits (such as goal satisfaction) are recorded in the KSI execution record. KSs have been modified so that when triggered by data-directed goals, they may produce all the data they can, but when triggered by goal-directed goals they try to produce only enough output data to satisfy that triggering goal-directed goal. Triggering goals are called *stimulus goals.*

Hypotheses on the hypothesis blackboard are filtered, and the hypotheses not blocked by the filter are passed through the hyp-to-goal mapping to produce data-directed goals. The hypotheses that are blocked are collected

[1] A *data-directed* goal is one that is created by the hyp-to-goal mapping, as opposed to a *goal-directed* goal that is created by subgoaling or a *received* goal that was received from another node.

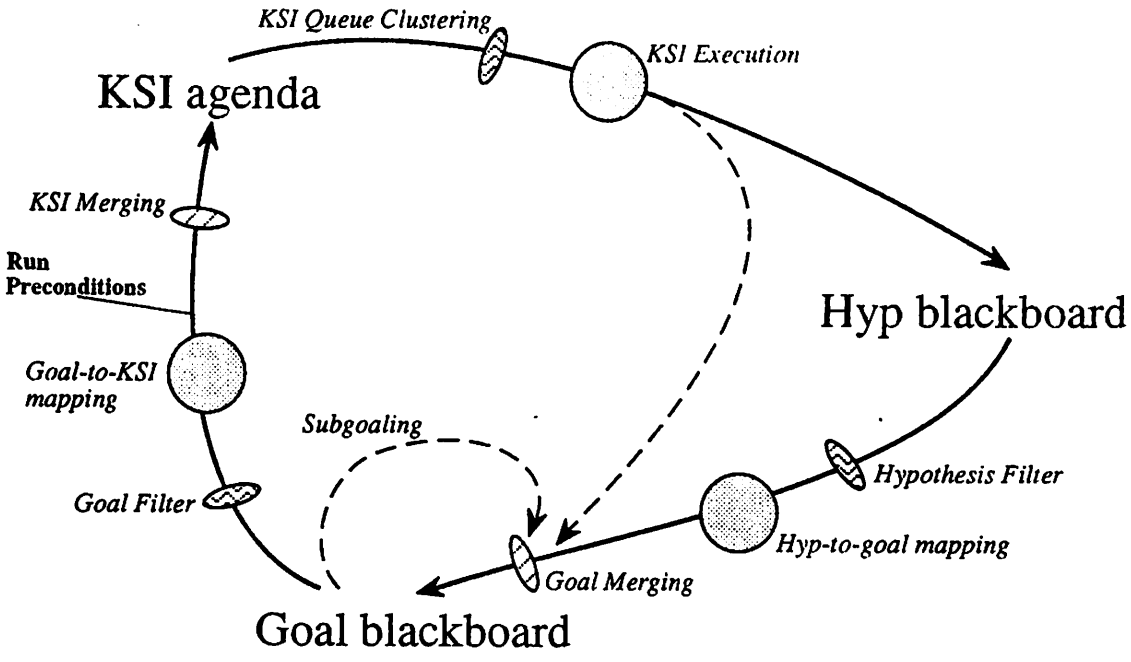

Figure 2: The New Low-level DVMT Control Loop

and saved. The resulting goals are merged with the current set of data-directed goals, received goals (from communication with other nodes), and goal-directed goals produced by subgoaling. All of the filters and mappings keep statistics for use by the meta-controller in making decisions. Examples of these statistics include the average number of objects blocked by the filter or the ratio of input objects to output objects at the mapping. Events may be generated if a number of objects above a certain rating are blocked that might indicate a possible problem with the control strategy.

Elements of the goal blackboard that pass the goal filter are run through the goal-to-ksi mapping, which produces a set of triggered KSs that may accomplish that goal. The preconditions of these KSs are run, which results in a set of costs (such as estimated time) and benefits (such as an estimated output set) for each triggered KS. One KS is chosen based on this data and its instantiation is merged into the runnable KSI queue. A final filter allows some KSIs to pass through to the agenda (for example, low-rated KSIs might be filtered out as runnable but unlikely to produce interesting results). An event may be generated if the KSI agenda is empty to indicate a shift in control strategy may be needed.

Here is a summary of the modifiable DVMT parameters:

Hypothesis Filter: The hypothesis filter takes as input the hypotheses created by a KSI execution and outputs hypotheses to be processed by the hyp-to-goal mapping. Hypotheses can be filtered by time, region, event class, blackboard level, or belief. Filtering a hypothesis means that it

will not create data-directed goals, and therefore will not stimulate any KSs. For example, if a DVMT node decides it will only track objects of class $\mathcal{A}$ the hypothesis filter can be set so that only hypotheses that have class $\mathcal{A}$ event classes are processed by the hyp-to-goal mapping. Hypothesis filtering allows the reduction of low-level control processing overhead since the filtered hypotheses do not create goals, KSIs, or cause any of the processing associated with goals and KSIs. In general, the hypothesis filter is used to prune the raw data by class, time, or location. Filtered hypotheses (those blocked by the filter) are stored and can be re-filtered if necessary. All hypotheses, filtered or not, are stored on the hypothesis blackboard and are available for supporting KSI executions or for examination by a planner.

Hyp-to-Goal Mapping: The hyp-to-goal mapping takes as input a hypothesis that passed the hypothesis filter and outputs goals to be merged into the existing goals on the goal blackboard. The mapping is specified by a set of *goal templates* that specify a pattern that a hypothesis must match and a transformation of that hypothesis into a goal. The hyp-to-goal mapping allows the control of what types of problem solving tasks the system should be concerned with by controlling what goals are created, and therefore what KSs are potentially triggered. For example, if only hypotheses at time n need to be clustered, then only hypotheses at the signal level at time n will generate a cluster goal (all hypotheses at the same time will then be merged into a single cluster goal for that time). Hyp-to-goal mapping is used to control the rough character of problem solving (types of methods applied to the data) by the creation of the proper data-directed goals.

Subgoaling: The subgoaling mechanism takes newly created or updated (merged) goals and produces goal-directed goals. For example, a vehicle level goal creates a group-level subgoal, which in turn creates a signal-level subgoal. The mapping is specified by a set of patterns between blackboard levels. Subgoaling allows us to make the system run in a goal-directed manner, where the system works on producing the parts of a known high-level goal rather than by piecing together existing parts into whatever they might fit. For example, when the most appropriate control is data-directed, subgoaling is turned completely off. If a planner wanted to work on a specific solution (whatever the reason — perhaps as part of a partial global plan) it would place that solution as a goal on the goal blackboard and invoke subgoaling on it to produce the goals of building the components necessary for producing that solution. Subgoaling is used in the balance of data-directed and goal-directed processing.

Goal Filter: The goal filter takes newly created or updated (merged) goals

of any type and outputs goals that will be used to trigger KSs. Goals can be filtered by time, region, event class, blackboard level, belief, or type. The goal filter reduces the number of goals that might trigger KSs, and therefore the number of KS preconditions that must be run. For example, if the control strategy is to produce vehicle level data before creating vehicle tracks, vehicle track goals that are created (by the hyp-to-goal mapping, subgoaling, a planner, or received from other nodes) can be filtered so that none of them trigger (stimulate) track creation/extension KSs (and do not subsequently run KS preconditions or create KSIs). The goal filter is used to avoid triggering and running the preconditions for a class of KSs that is not currently desired but will be in the future (otherwise the goals would not have been created in the first place).

Goal-to-KSI Mapping: The goal-to-KSI mapping takes goals and produces a set of KSIs that attempt to satisfy those goals. It is specified as a table that matches goal types to KSs that can satisfy those types of goals. The preconditions of these triggered (or stimulated) KSs are then run, and the best KS is chosen based on the costs and benefits that the precondition returns (see the KSI precondition choice parameter below). This KS is then instantiated with its context, including the stimulating goal. The goal-to-KSI mapping is used to fine-tune the precise method or algorithm used to satisfy a goal. In the DVMT, the goal-to-KSI mapping is rather coarse, because KSs tend to be written in a general manner rather than tailored for a very specific situation. For example, one may satisfy a vehicle level goal by either the synthesis KS from the group level to the vehicle level, or a level-hopping KS that synthesizes data directly from the signal level to the vehicle level (skipping any intermediate processing). The goal-to-KSI mapping tends to be simple in the DVMT because we have only a few classes of KSs.

KSI Rating: The new or merged KSIs must then be rated and inserted on the KSI agenda. The rating can be computed from a set of heuristics and a focus combining function, as in BB1. Currently a single rating function is specified by the meta-controller to be active at a given time. Any KSI that makes it to this point should be run in the active control strategy, so the rating function acts as a preference for what order in which to do things that must all be done. For example, a preference can be given to run KSs that work on highly believed stimulus hypotheses first since they are likely to produce highly believed results.

Goal Merging: Goals that are produced by hyp-to-goal mapping, received from other nodes, or created by subgoaling may need to be merged. Currently the dynamic modification of goal merging is not implemented,

and goals are always merged if they have equivalent event classes, time-regions, and output types. Merging goals controls the number of overlapping KSIs that are scheduled. Not merging two similar goals means that there may be a KSI for each of them on the agenda that will produce a very focused result rather than one KSI that will produce a slightly less focused result but potentially satisfy both goals at once.

KSI Merging: Sometimes it turns out that two goals that were not merged will stimulate KSIs that will do very similar work, for example if one of the goals subsumes the other. KSI merging allows us to control our scheduling granularity by allowing us to group several processes into one KSI for scheduling or to leave them separate. Currently the dynamic modification of KSI merging is not implemented, and KSIs are merged when they are the same type and one has at least the same set of stimulus hypotheses as the other.

KSI Precondition Choice: If the goal-to-KSI mapping indicates several KSs are triggered, then the preconditions of each are run and return the costs and benefits of running their associated KS. From among these, the KSI precondition chooser picks one to be executed. It seems appropriate to modify the choice function to reduce the effort spent in picking a KS (for example, by taking the first KS whose precondition is satisfied). Each KS could have multiple preconditions that analyze the costs and benefits of running the KS at different levels of detail — a quick analysis might only examine the goal, while a detailed precondition might examine the hypothesis blackboard. The KSI precondition chooser could then choose among these precondition effort levels. Currently this is not implemented, and the system chooses a KS for a goal by choosing the first KS whose precondition returns a positive benefit measure.

KSI Queue Cluster Width: When a KSI is executed on the agenda, all the KSIs near the top of the agenda (those within the KSI queue cluster width) will all execute before looking at the agenda again. By setting the width to 1, a depth-first search occurs while wider settings give a more breadth-first character to the search. A narrow setting allows fine control over what is executed, while a wider setting is appropriate when there is uncertainty over how to proceed and thus prevents the early choice of a preferred solution path. The normal setting is to cluster all KSIs that are rated the same as the highest-rated KSI.

After changing a parameter, the meta-controller may re-run the low-level control loop from any point — often from just before the point that was changed. The meta-controller has the option to reintroduce filtered data at this time as well, from either the hypothesis or goal filter or both.

Various schemes have been discussed for the storage of blocked hypotheses and goals that would make the re-filtering very efficient, but none have been implemented (blocked data is simply re-run through the filter). For example, blocked objects can be divided into classes, i.e., objects to be saved and objects to be permanently removed from consideration. Blocked objects can be stored according to how they were blocked, so that when a filter changes the data that needs to be refiltered (or that passes the new filter) can be retrieved efficiently.

1.1.1 The DVMT meta-controller

In our initial implementation of these ideas in the DVMT, a BB1-style meta-controller with appropriate control knowledge sources and a control blackboard (GBB1) was used to parameterize the low-level control loop. Control knowledge source preconditions examined the current state of the low-level control loop, usually the contents of the agenda and the blackboards. BB1 prescription KSs are used, and strategy and focus goals also examined the agenda and blackboards. **Heuristic control KSs were free to modify any of the low-level control loop parameters, not just the agenda rating mechanism.**

2 DVMT Examples

One meta-controller that has been experimentally evaluated was for real-time vehicle monitoring. The approach used by this meta-controller was to use approximate knowledge sources to control the amount of time it takes to build vehicle tracks, trading off precision for execution time through various approximations such as 'level-hopping', 'partial-support', and 'time-frame-skipping'. These algorithms are beyond the scope of this paper (see [11,13]). The meta-controller decides to switch processing modes (from precise to approximate) when either external information or its own monitoring of the system state shows that a deadline will not be met. It accomplishes this change through the modification of the low-level control parameters. The meta-controller also uses the filtering and mapping mechanisms to reduce the amount of unnecessary hypothesis and goal processing that is performed.

Another meta-controller paradigm that has been considered but not yet built is an incremental planner for control[6]. In this example, the hypothesis filter only passes hypotheses that are posted by the node sensors or external hypotheses. Hypotheses that pass the filter are then mapped into data-directed base cluster goals, which the subgoaling mechanism develops (or 'clusters') into a set of 'alternative goals', representing all the tracks possible in the data. Alternative goals are filtered out and do not trigger KSIs themselves. The incremental planner builds a plan to achieve the alterna-

tive goals, and when expanding the plan produces 'intermediate goals' that are posted to the goal blackboard and which do pass through to the goal-to-KSI-mapping. The incremental planner can also use the KSI execution record to perform plan monitoring. Details on the planning mechanism itself are beyond the scope of this paper.

2.1 Real-Time Vehicle Monitoring

The real-time DVMT meta-controller was developed to control experiments in soft real-time approximate processing — using approximate knowledge and data effectively. Two BB1-style strategies were developed: a goal-directed strategy and a data-directed clustering strategy. The GBB1 prescription KSs were used to move between foci when the goal of a focus was satisfied. A pictorial representation of the goal-directed strategy in shown in Figure 3.

The initial control KS sets all filters to open, the hyp-to-goal mapping to normal (signal level to group level to vehicle level to vehicle tracks to pattern tracks), the goal-to-ksi mapping to normal (the regular, non-approximate KSs), subgoaling off, and KSI rating tied to the rating of the stimulus goals and hypotheses.

2.1.1 Goal-directed Strategy

This strategy is invoked when its precondition determines that there are multiple vehicles and that there is good sensor data. This is determined by testing that the initial data at time 1 is spatially separated. This strategy tries to determine what might be out there, whether it is important to track, and then tracks only what it finds to be important. It has three foci: *find initial vehicles, approximate short tracks,* and *pattern directed processing.*

Find Initial Vehicles: This focus concentrates on the careful (non-approximate) data-directed analysis of initial data. It consists of two heuristics. This focus is over when the KSI agenda is empty.

Consider only time 1 hypotheses: This heuristic sets the hypothesis filter to allow only hypotheses from time 1 through and to block all others. Only time 1 hypotheses, then, pass through to the hyp-to-goal mapping.

Create no tracks nor patterns: This heuristic sets the goal filter to block any track (ST, GT, VT, PT) and pattern (PL, PT) level hypotheses. Thus the system will only work up to the vehicle level.

Approximate Short Tracks: This focus concentrates on quickly building up an idea of what possible patterns are present in the system. It does this

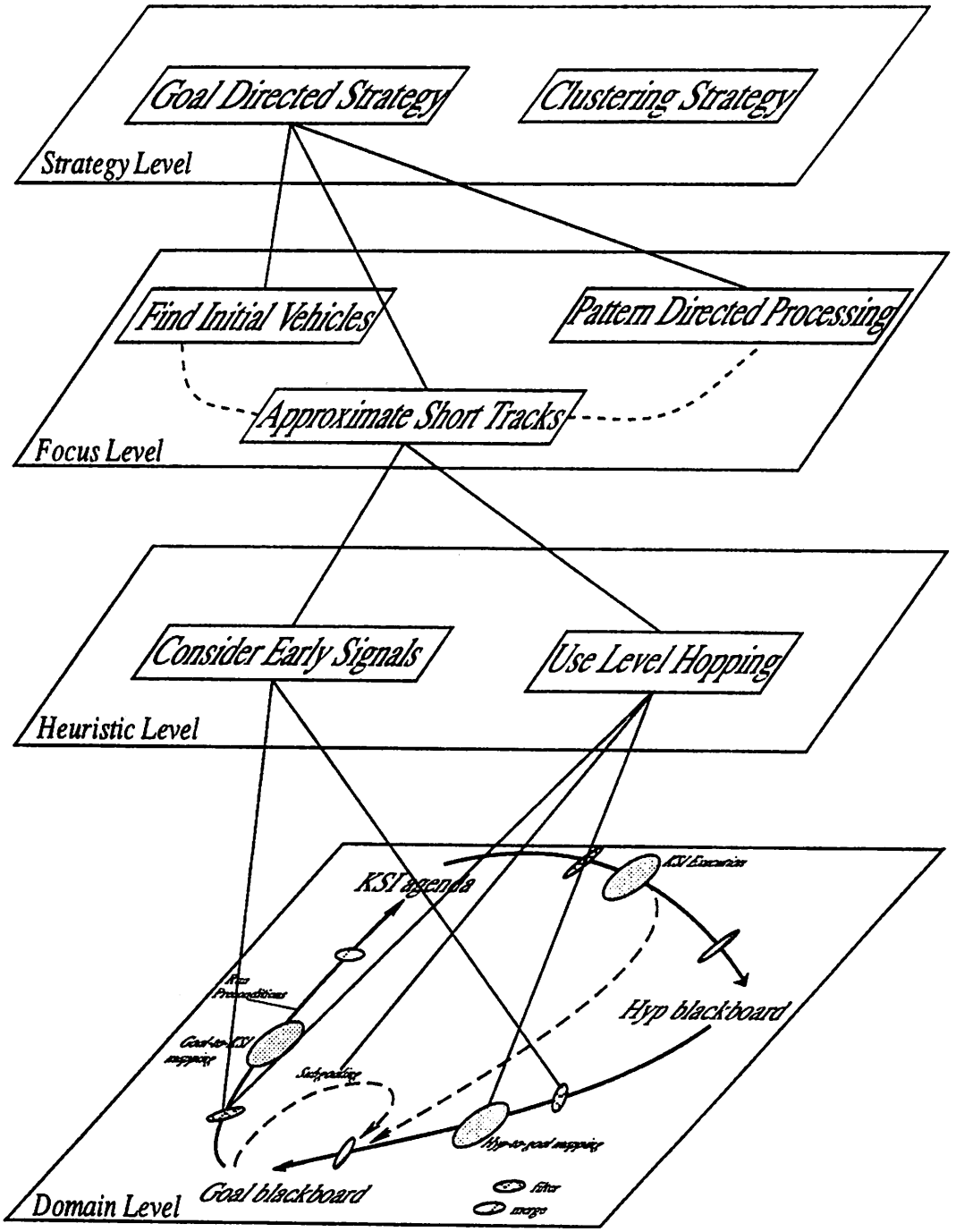

Figure 3: The DVMT Goal-directed Strategy

so that our process resources can be concentrated on what are viewed as important patterns. This focus is completed when all pattern track hypotheses have reached a minimum length at which a decision about their importance can be made.

Consider early signals: This heuristic sets the hypothesis filter to only allow hypotheses from time 1 through 5 through. Hypotheses previously blocked are refiltered. Thus only early hypotheses are considered for further processing.

Use level-hopping: This heuristic changes several parameters to set the system up for level-hopping. Level-hopping is used to approximate vehicle level data directly from signal level data by compression of the signal/group/vehicle grammar. The hyp-to-goal mapping is set to create vehicle level goals from signal level data, subgoaling is turned off, and the goal-to-KSI filter is set to block signal and group goals (since the creation of vehicle level hypotheses is desired).

Pattern Directed Processing: This focus concentrates on developing 'important' tracks at the expense of mostly ignoring 'unimportant' tracks. The importance of a pattern is resolved by a pattern track KS that ran at the end of the approximate short tracks focus. The pattern directed processing focus remains active until the end of problem solving. In pattern directed processing, rather than work in a data-directed manner, we configure the system to work in a goal-directed manner.

Work on important patterns: This heuristic sets up several patterns to process important patterns. The hypothesis filter is set to only allow vehicle level hypotheses through from the important patterns, because we will work below the vehicle level only in a goal-directed manner. Subgoaling is invoked on the important patterns to create goals for extending the important pattern and building it up from the signal level (this creates a partially ordered plan for processing the important tracks). The KSI queue cluster width is set to only execute 1 KSI at a time, so that there is precise control over each invocation.

Approximate unimportant patterns: This heuristic deals with the patterns that are not deemed to be important. The hypothesis filter is set to allow only signal and vehicle level hypotheses through from unimportant patterns, and the hyp-to-goal mapping is set to create level-hopping goals for approximating these unimportant patterns (in the future, we may use time-frame-skipping, which tracks an object intermittently, instead). The KSI rating is reduced on level-hopping KSs, so the system prefers to work on the important patterns instead.

2.1.2 Cluster-directed Processing Strategy

The cluster-directed processing strategy is invoked when its precondition determines that there is highly errorful sensor data; this can be determined by testing that the data at time 1 is not spatially well-separated. This strategy assumes that there is too much data to process individually, and so it clusters data at each time and then processes the clusters in a data-directed manner. It has two foci, *cluster initial data* and *process clustered data.*

Cluster Initial Data: This focus concentrates on clustering the data at the signal level of the blackboard. The focus is completed when there are no more clustering KSs to run.

Turn on clustering: This heuristic sets the hyp-to-goal mapping to map signal level hypotheses at time n to a clustering goal for time n. Subgoaling is turned off; this strategy works in a completely data-directed manner.

Process Clustered Data: This focus concentrates on traditional data-directed processing of the signal-level clustered data. It remains active until the end of processing.

Turn off clustering: This heuristic sets the hyp-to-goal mapping back to its regular set of templates. Subgoaling stays off.

3 Performance Evaluation

To assess the viability of our design, a scenario was constructed which was applicable to real-time vehicle monitoring. The DVMT was run twice; in the first run, the control that was used by the DVMT utilized the design presented in this paper. In the second run, we simulated controlling the domain processing using only an elaborate evaluation function with no parameterization of the low-level control loop, i.e., at each point in the second run, the KSI that was executed in the first run at that point was forced to execute. Each system thus ran the same domain KSIs, in the same order.

The comparison of the runs illustrates the computational benefits of a parameterized low-level control loop. A rough description of the domain processing is presented in Appendix A, not so much as an illustration of real-time problem solving in the DVMT, but as background information for the run analysis.

	Node 1		Node 2	
KSI Execute	3170	(26%)	14436	(72%)
Goal Create	2961	(24)	4331	(22)
Goal Merge	1	(< 1)	4	(< 1)
KSI Create	2569	(21)	1094	(5)
KSI Merge	3507	(29)	128	(1)

Table 1: Summary of Run Times under New Parameterized Architecture

3.1 Run Comparison

The environment file for the run consisted of two nodes, each responsible for a different physical region. Node 1 had good, spatially distributed data, so it chose the goal-directed strategy. Node 2 was given noisy data; thus, it chose the cluster-directed processing strategy.

The various problem-solving activities for each of the two nodes involved in the runs were monitored. From analysis of this data, there were three major categories in which processing time differed. These were: the KSI execution time, the hyp-to-goal mapping, and the goal-to-KSI mapping. In addition, to present a more detailed analysis, the hyp-to-goal mapping was broken down into two subcategories: creating goals and merging goals. Likewise, the goal-to-KSI mapping was broken into two subcategories: KSI creation and KSI merging. For the run using the new parameterized architecture, the absolute time and the percentages of time each node spent in each of these five categories relative to the others are presented in Table 1. Similarly, the data for the non-parameterized run, controlled using only an evaluation function, are presented in Table 2[2].

The major conclusion is that overhead for low-level control is significantly reduced: in node 1 by 33% and in node 2 by 80%. Node 2 came out especially well because many signals are never processed by the low-level control loop in the parameterized system. This also shows up in Table 2 for the non-parameterized architecture where node 2 creates more low-level goals and KSIs that are never executed. This is because the unneeded goals (and the KSIs they stimulate) are never created in the parameterized system, which focuses on clustering hypotheses.

Other relevant statistics include:

- Total Blackboard access time was 7:13 for the new architecture, 8:32 for the other.

[2]Because run 2 *simulated* a BB1-style evaluation function, there was no way to measure the high-level time spent in dynamically creating the appropriate evaluation function that would execute the desired KS. The absolute time in the first, parameterized run that was spent in high-level control was 8139 for node 1 and 11246 for node 2.

	Node 1		Node 2	
KSI Execute	3022	(18%)	14271	(34%)
Goal Create	5573	(34)	15924	(38)
Goal Merge	5	(< 1)	23	(< 1)
KSI Create	3784	(23)	9397	(23)
KSI Merge	4076	(25)	1882	(5)

Table 2: Summary of Run Times using Evaluation Function Only (Non-parameterized)

- Total KSIs created (parameterized / non-parameterized): node 1 - (232/280), node 2 - (222/391)
- Total goals created (parameterized / non-parameterized): node 1 - (221/438), node 2 - (248/831)
- Total executed KSIs in both runs: node 1 - 76, node 2 - 86
- Filtering hyps and goals took < 1% of the total processing time for each node in the parameterized run. Likewise, passing objects back through their respective filters took < 1% of the time. Objects were passed back through the filters each time there was a change in focus.

In summary, the parameterization of the low-level control loop permits a larger percentage of a node's processing time to be used in executing KSIs.

4 Summary and Future Work

The system described in the paper is fully implemented (except where indicated) in the DVMT. The strengths of its design lie in the ability to easily customize the low-level control loop via a set of parameters distributed along the loop (rather than in agenda management only), and in the framework it gives for the meta-control component to examine and make decisions about low-level control.

The short term goals of this effort are to do further analysis of the benefits of this approach by analyzing this system using a model of KSI invocation cost (domain problem-solving cost) and control costs (involving the ratio of potential to actual data filtered or mapped, control KSI costs, etc.) It has also been recognized that a more complex and flexible mechanism for expressing control plans (as opposed to the linear orderings used by the prescription KSs) is needed. Additionally, it is important to work on how to dynamically move between strategies and build sets of foci (a control plan)

on the fly. Another aspect of the system to be expanded is to introduce a model of time in the system, in order to investigate more hard real-time problem-solving tradeoffs.

Finally, there is a desire to build a general control shell for GBB around these ideas. The user would specify the spaces to be used in the hypothesis and goal blackboards, and generic filtering processes could be created that filter objects along GBB dimensional indexes. Goal templates could still be used to map hypotheses to goals, and either KS templates could be created or KS definition would contain the template information.

References

[1] N. Carver. Evidence based plan recognition. COINS Technical Report 88-13, University of Massachusetts, 1988.

[2] N. Carver and V. Lesser. Planning for the control of an interpretation system. COINS Technical Report 89-39, University of Massachusetts, April 1989.

[3] A. Collinot. Revising the bb1 basic control loop to control the behavior of knowledge sources. In *Proceedings of the Second AAAI Workshop on Blackboard Systems*, pages 19–36, 1988.

[4] D. D. Corkill, V. R. Lesser, and E. Hudlicka. A framework for distributed problem solving. In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 143–147, Pittsburgh, August 1982.

[5] K. S. Decker, E. H. Durfee, and V. R. Lesser. Evaluating research in cooperative distributed problem solving. In M. N. Huhns and L. Gasser, editors, *Distributed Artificial Intelligence, Vol. II.* Pitman Publishing Ltd., 1989. Also COINS Technical Report 88-89, University of Massachusetts, 1988.

[6] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a time-constrained, blackboard-based problem solver. *IEEE Transactions on Aerospace and Electronic Systems*, 24(5), September 1988.

[7] B. Hayes-Roth. A multi-processor interrupt-driven architecture for adaptive intelligent systems. Technical report KSL-87-31, Knowledge Systems Laboratory, Stanford University, June 1987.

[8] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[9] V. Lesser. Coordination in distributed problem solving networks. DARPA proposal, COINS, University of Massachusetts, 1989.

[10] V. R. Lesser, D. D. Corkill, R. C. Whitehair, and J. A. Hernandez. Focus of control through goal relationships. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, August 1989.

[11] V. R. Lesser and J. Pavlin. Performing approximate processing to address real-time constraints. COINS Technical Report 87-126, University of Massachusetts, 1988.

[12] Victor R. Lesser and Daniel D. Corkill. The distributed vehicle monitoring testbed. *AI Magazine*, 4(3):63–109, Fall 1983.

[13] R. Whitehair and V. Lesser. Approximate processing in the DVMT. Working paper, COINS, University of Massachusetts, 1989.

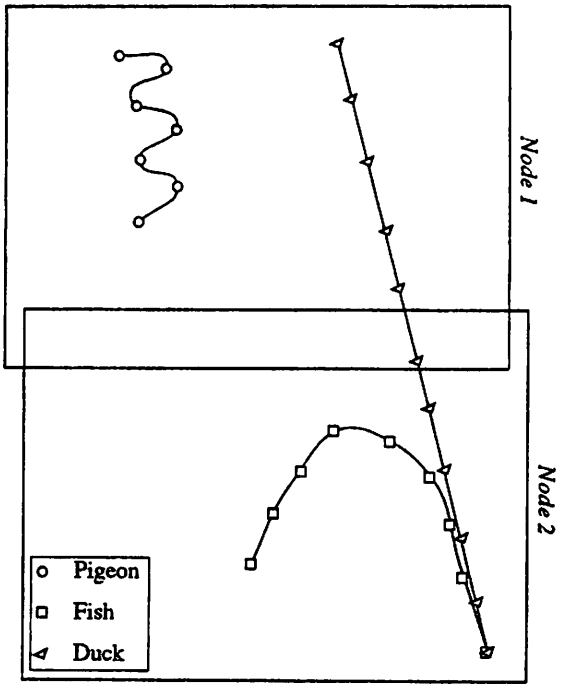

Figure 4: A DVMT Scenario

Appendix A: Environment File and Domain Processing

The scenario used for the experiments consisted of three "vehicles": a pigeon meandering on a path roughly from [10,90] to [20,60], a fish swimming around the point [40,40], and a duck starting at [47,92] and moving on an intercept pattern with the fish (see Figure 4). There are two problem-solving nodes, and their job (before seeing the data) is to save all fish by identifying and tracking ducks moving in an attack pattern toward their general location. Node 1 has the job of tracking data in the region [0,50] to [50,100],and has been given 1 sensor which covers the entire region accurately. Node 2 tracks data in the region [0,0] to [50,50], but its one sensor produces many points of data for every true sound it senses.

Initially, node 1 assesses its raw sensor data and chooses a goal-directed strategy. The "find initial vehicles" focus begins at node time 1, by considering only hyps from time 1, while blocking all others at the hypothesis filter, in an attempt to identify the objects in its sensed region. By node time 61, node 1 has assessed that the object at [47,92] is probably a duck (though it could be a pigeon) , and the object at [17,93] is probably a pigeon (though it could be a duck). At this point, node 1 changes its focus to approximate short tracks, in an effort to identify roughly what the objects are doing. By node time 225, node 1 believes that there is a duck attacking from [47,92] toward [42,70], and a pigeon wandering from [17,93] to [42,70]. Node 1 then sends a request to node 2 to look for something that a duck might attack, in the projected time-region route of the duck. Node 1 then switches to a

pattern-directed processing focus, which consists of tracking the duck more closely, while still monitoring the alleged pigeon in case it suddenly starts showing signs of being a duck, at which point node 1 would have to devote more time to tracking it.

Node 2 initially selects the cluster-directed strategy because an cursory scan of its data suggests that the sensors might be at fault. Its first focus is thus to cluster its data. At node time 25, it has clustered all of its data and switches to the "process clustered data" focus. This focus continues for the duration of the run.

At node time 236, node 2 receives the request to look for something which a duck might attack. Its processing suggests that the duck is attacking the fish which is around [40,40]. At this point, node 2 would attempt to inform the fish that a duck is approaching, and processing would continue from that point, presumably with each node continuing to help the fish escape while still monitoring its sensed area for other objects. However, this interaction with the sensed environment has not been implemented yet, so we induce termination at node time 325.