Class of Hierarchical Controllers and Their Blackboard Implementations

T. L. Skillman,* W. Kohn,† D. Nguyen,‡ C. Ling,‡ and R. Dodhiawala§
The Boeing Company, Seattle, Washington

This paper describes ongoing research dealing with a new representation of generic single-agent hierarchical control systems. The strategy is to generalize the model-following paradigm and standard control system architecture, together with a recursive definition of hierarchy, to represent goal-driven controllers. The suggested hierarchical representation, which is based on the definition of a canonical representation of each of the levels, allows for feedback control based on general data representations. The structure of the hierarchy is illustrated with a simple mobile robot control example. Mapping of the proposed hierarchical control systems into the blackboard computing paradigm is discussed.

Introduction

THIS paper reports on work currently in progress. It proposes a specific model of hierarchical control, based on extensions of traditional control theory. The long-range objective is to develop a system for automatic generation of controllers based on the declarative specifications of control goals and plant dynamics.

Research in autonomous multivariate control of large-scale systems typically addresses hierarchical representation and control.[1-3] A classical multilevel hierarchy is shown in Fig. 1. The hierarchical levels are organized from bottom to top as a function of sensory data abstraction and from top to bottom as a function of control specificity. In general, the higher a particular level, the slower is its update rate. A natural consequence of the hierarchical organization is that, in each higher level, the representation of the dynamics becomes increasingly abstract (symbolic), control actions change from simple signals to complex data structures, and the control processing becomes more complex, often declarative rather than algorithmic. The top of the hierarchy is the man-machine interface, where the goals are set for the system and status is returned. The lowest levels are the interface to the real world via sensors and actuators and represents the only way in which the system can affect the world. Today, system designers are responsible for including design features such as layer definition and functionality, data representation, computational approach, system stability, reliability, robustness, and accuracy. It is our contention that a controller can be defined for each level of the hierarchy as a function of the top-level goal specification and the bottom-level plant description, which includes sensors, actuators, and plant dynamics.

Proposed Concept

For the purposes of brevity, our discussion will be carried out at the conceptual level. The first task in the formulation of the proposed hierarchical control concept is to define a standard controller architecture that represents the dynamics at each level of the hierarchy.

In traditional state feedback control systems, the control signals are constructed as a function of the error between the desired plant output signals and the actual estimated plant output signals. The actual output signals are casually generated by the plant as a result of command signals from the controller. The state estimator uses a model of the plant to generate an updated estimate of the state of the plant, as a function of current sensors readings, control signals, and the previous state estimate of the plant. The control law is determined by a prespecified policy (e.g., optimization) and the control specifications (e.g., overshoots, stability margins, etc.).

We propose to enhance the operation of this control structure by including in it a variation of the model-following paradigm.[4] This paradigm has two elements: an ideal model

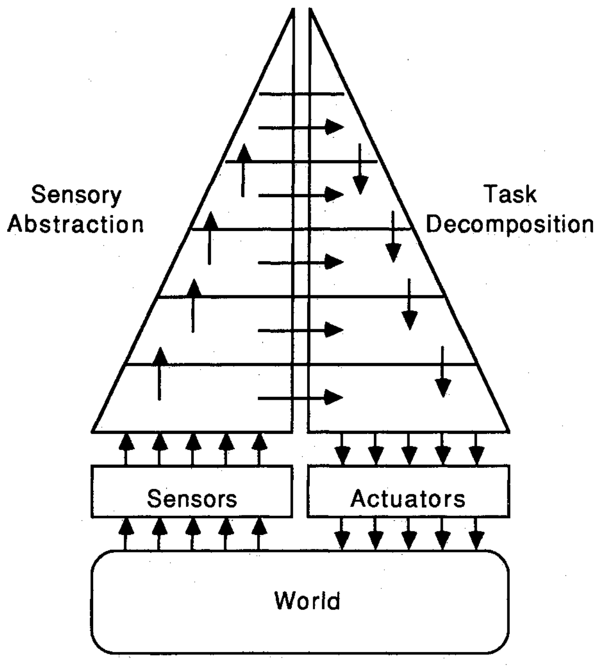

Fig. 1 Generic hierarchical control system.

Received Aug. 6, 1987; presented as Paper 87-2820 at the AIAA Guidance, Navigation, and Control Conference, Aug. 17-19, 1987; revision received March 18, 1988. Copyright © 1987 American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*AI Specialist, Information Processing Laboratory, Boeing High Technology Center. Member AIAA.

†Chief Researcher, Artificial Intelligence, Engineering and Scientific Services, Boeing Computer Services Company. Senior Member AIAA.

‡Research Engineer, Information Processing Laboratory, Boeing High Technology Center.

§AI Specialist, Knowledge Systems Laboratory, Boeing Advanced Technology Center; currently, AI Specialist, FMC Artificial Intelligence Center, San Jose, CA.

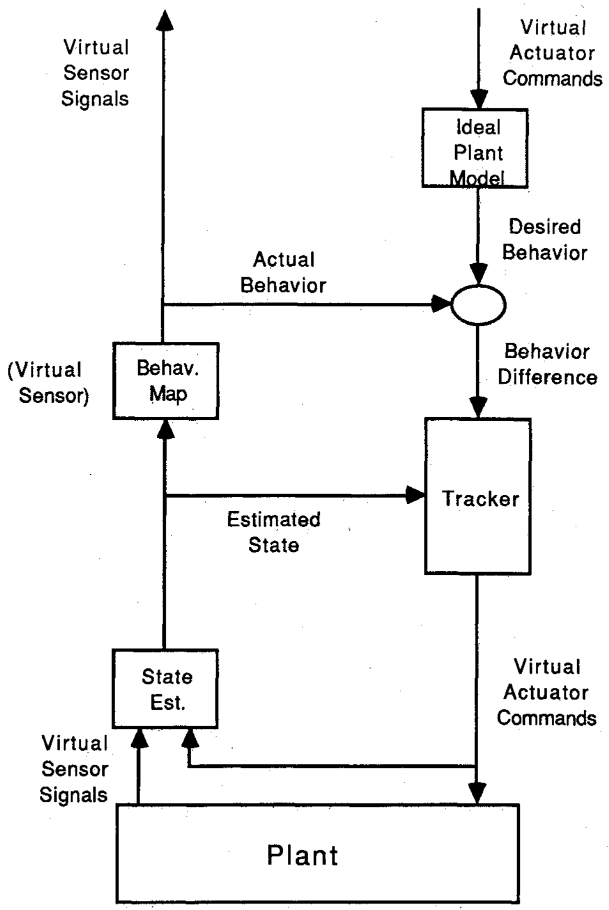

Fig. 2 Proposed canonical form.

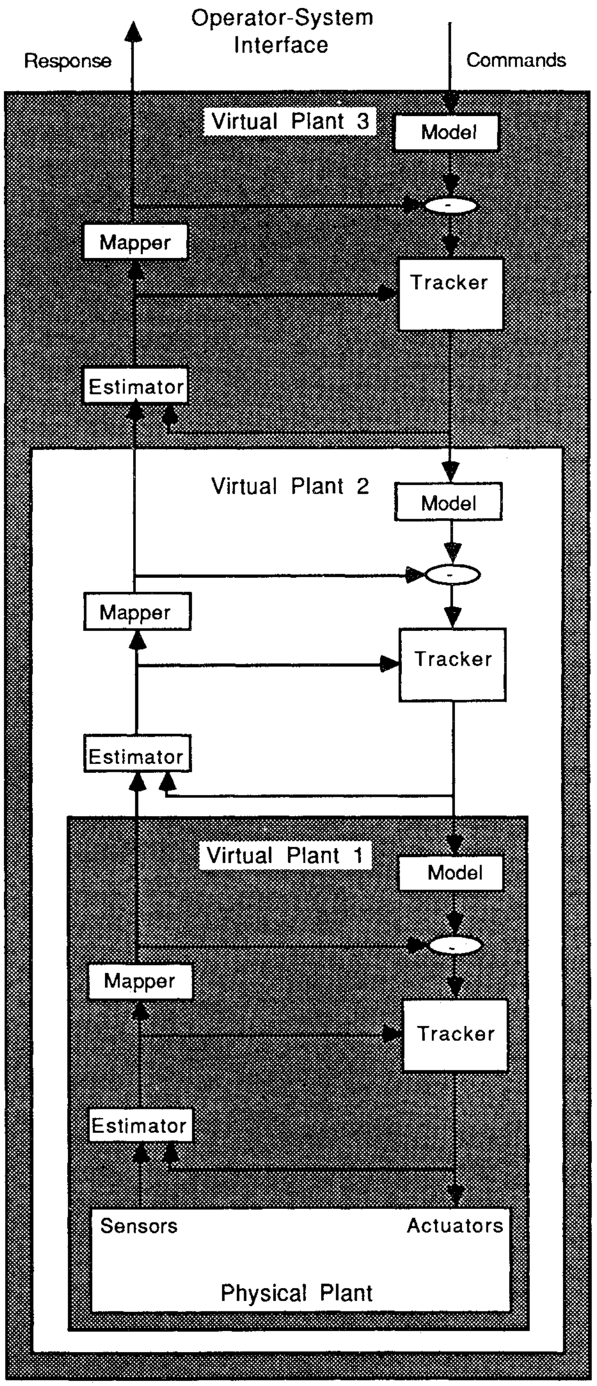

Fig. 3 Recursive application of canonical form.

and a tracker. The model generates desired behavior and the tracker generates commands to reduce the difference between the measured and desired behavior. This paradigm has several crucial advantages over the standard control scheme discussed in the previous paragraph. It admits an implementation that is declarative, decoupling, and flexible. It is declarative because it explicitly includes the control requirements in the ideal model. The model-following paradigm is decoupling because the ideal-model module is concerned only with the control and goal specifications, and the tracker module is concerned only with the control strategy. Finally, the paradigm is flexible by allowing the ideal model to vary dynamically over a range of models within the class of models that meets the control specification.

An additional characteristic in our concept is that the state estimate is fed to a module—the behavior map—which maps the state space into the behavior space. The behavior map is designed to generate a homomorphic version of the state estimate, resulting in a more abstract representation of plant state features, making them compatible with the next higher level in the hierarchy.

Figure 2 shows a block diagram of the functional characteristics of the controller, as defined earlier. We will refer to it as the canonical form of the controller. Figure 3 shows how the canonical form can be applied recursively at each level of a hierarchy. The plant and controller at one level can be viewed in entirety as a virtual plant to be controlled by a controller defined at the next higher level of abstraction. The controller's actuations at one level become the command to the next lower level. The next lower level's behavior mapping becomes the sensed data to the higher level's state estimator. In general, the operation of each level is asynchronous with respect to the other levels. This model can now be extended to apply not only to classical numerical control loops but also to symbolic control loops by carefully defining the idealized plant model, the state estimator, the behavior map, the difference operator, and the tracker.

The operation of this hierarchy is depicted in Fig. 4. This diagram characterizes the relationship between the concepts of causality and measurability in the framework of our proposed hierarchical controller. The diagram is referred to as the "causality cone." The horizontal axis is time, increasing to the right, and the vertical axis is levels of hierarchy, increasing upward. The intersection of the axes represents the current time relative to the plant. The left-hand side of the V-shaped curve represents measurability, the time at which the most recent sensed information from the plant is available to the corresponding level of the hierarchy. The right-hand side of the "V" represents causality, the earliest time at which an actua-

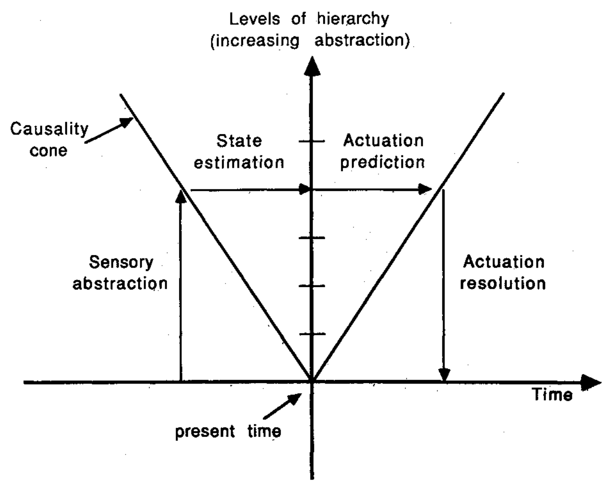

Fig. 4 Relation of causality to nondeterminism.

tion will take place, as planned by the corresponding level of hierarchy. These curves need not be linear, but must be monotonic. It can be seen that, as you move up the levels of hierarchy, the controller must deal with increasingly long time scales, receiving information further from the past and planning for actuations further into the future. This implies that the planned actuation from a level of the hierarchy must be sufficiently nondeterministic so that the lower levels can adapt to sensed changes on shorter time scales. For the proposed hierarchical scheme, it can be shown that nondeterminism is a direct consequence of causality and measurability.[5] It also can be shown that a necessary condition for being well posed is that the control actuation, at any time, be based on the projected estimate of the plant state calculated from the sensed data history. This means that, for the best control, a level of the hierarchy must maintain a model of the plant that it can use to project the current sensed state of the plant forward to the current plant time.

Canonical Form

Following is a detailed discussion of each of the elements of the proposed canonical controller form. The inputs, outputs, and functionality of each will be explained. Figure 2 will be useful in understanding the relationships between the elements. When referring to the "plant," we mean either the real physical plant, at the lowest level of abstraction, or any of the virtual plants defined by the hierarchy of controllers.

Ideal Plant Model

The function of the ideal model is to calculate the desired behavior based on the commanded action. It is a simulation of the desired behavior of the subplant. The model represents an idealization of the subplant and simplifies planning at higher levels by hiding details of the dynamics of the lower levels. This is related to the computer science notion of information hiding. The desired behavior will be compared to the current behavior, and the error between the two will be used by the tracker to plan actions. It is the tracker's job to make the subplant behave in a way that matches, as closely as possible, the ideal model.

State Estimator

In general, one cannot measure directly the part of the plant being controlled. By having a model of the interactions of the plant elements, it is possible to estimate the state of the plant through indirect measurements. This process is called state estimation. The state estimator maintains a current estimate of the state of the plant being controlled. The state estimator must have, or build, a model of the real plant, not the idealized plant. The state estimator has access to the sensed data from the subplant and also the current command to the subplant. From this it can enhance its model of the subplant in two ways: 1) adaption, which is the process of parameter estimation, and 2) learning, which is the process of determining and adding new structures to the state estimate. The group method of data handling (GMDH)[6] is an example of learning. Another function of the state estimator is to reduce uncertainty and measure its strength.

Behavior Map

The behavior map operator maps the estimated state from the state estimator into a higher level of abstraction and/or representation. This map performs a homomorphic transformation on the state information. Its output can be viewed as virtual sensory data to the next higher level of control. Its output is also used by the difference operator to compare the current behavior with the desired behavior as it is specified by the ideal model. For this reason, the definition of the behavior map and the ideal model are closely coupled because they both output the same representations of the behavior of the plant, one desired the other estimated. Therefore, it can be seen that the ideal model is used to calculate desired behavior given some actuation, and the behavior map estimates the actual behavior based on the state estimate.

Difference Operation

The function of the difference operation is to calculate the error between the desired behavior, as supplied by the ideal model, and the current behavior, as supplied by the behavior map. The difference is provided to the tracker element as the current error in behavior. The tracker's job is to reduce this error to zero. In traditional systems, the difference operation is subtraction applied to integers or real numbers. To generalize, the difference operation must be defined over more abstract data structures, such as integers, reals, strings, lists, graphs, automata, etc. The output of the behavior map must be in the same level of abstraction as the output of the ideal model so that a meaningful difference operation can be defined.

Important criteria in the selection of the structures of the behavior space and the ideal model output space are as follows:

1) The ideal output space is a proper subspace of the behavior space.

2) A difference operation is defined with the standard monotonicity conditions.

3) The behavior space is closed with respect to the difference operator.

4) The behavior space topology is chosen so that this difference operation is a continuous function.

Tracker

The tracker consists of three elements: 1) planner, 2) verifier, and 3) executor. The tracker takes the behavior error and the current state estimate and generates a command, a virtual actuation, to the subplant. The command is selected to change the state of the subplant such that the state estimate will change, and the mapped behavior will more closely equal the goal behavior. The other input to the tracker is the current control policy. This sets the constraints that will be used in planning, i.e., optimization of energy, time, etc.

The planner takes the behavior error from the difference operator, current state information from the state estimator, and knowledge about the virtual subplant, and generates a plan. When the plan is executed, virtual actuations are generated to the subplant, which results in changes in the actual behavior, reducing the behavior error. The subplant knowledge may be explicitly or implicitly represented for use by the planner. The planner then generates an appropriate plan for the current error, state, and policy. The plan is a tracker that describes a family of possible solutions for reducing the behavior error. The time scale of the plan is related to the planner's level in the hierarchy. Typically, the higher the level, the longer the time frame of the plan, and the greater the nondeterminism in the plan.

Because the planner plans based on a policy and functional constraints, the plan generated may not meet certain system operational constraints. The verifier must further constrain the plan based on these operational limits. An example of a functional constraint would be minimum-impulse character for the jets in a shuttle-type vehicle. An operational constraint would be to not fire primary jets in terminal phase of rendezvous. The final output of the verifier is a plan (control law) that is a function only of state estimate information.

The executor is then responsible for executing the plan over its time duration. This may require many cycles through the sense/calculate/act cycle. This continues until a new command is issued by the next higher level. When that occurs, a new desired behavior is generated and a new error is calculated. This new error then triggers a new planning and verification step, leading eventually to a new control-law execution. The fact that the control-law execution typically has a higher update rate than the incoming commands introduces the varying computational time scales as one moves up and down the hierarchy. Lower levels have higher update cycle rates but less computations. The higher levels have lower update cycle rates but more computation. The result is a fairly uniform computational load across the entire hierarchy.

This concludes the discussion of the elements of the canonical form. Note that the symmetry between the left and right halves of Fig. 2 corresponds to the duality between control and estimation, which is well known in the literature of linear system control.[7]

Important System Characteristics

In this section, we discuss some fundamental properties of the proposed architecture. Only plausibility arguments for their validity will be given. Formal proofs will be presented in a future paper.[8] The combination of the nested hierarchy with the model-following paradigm gives the following benefits.

Robustness

Our proposed controller architecture has advantages with respect to parameter robustness and singular perturbation. It can be shown that a nested hierarchical implementaion of a controller reduces the performance sensitivity due to parameter variations.[9] This is because the variation due to a given parameter is localized to the level where that parameter appears. Overall sensitivity to singular perturbations is decreased because of the ability of a higher level of the hierarchy to detect singularities at the immediate lower level and then modify, accordingly, the model to be followed by the tracker at the lower level. As an example, consider a joint controller in a robot manipulator with rate (tachometer) control at one level and position (shaft encoder) at the next higher level. If the tachometer fails, the upper level can detect that fact at its tracker module and generate a command to drive the lower level with an estimate of rate generated from the sensed position history.

Adaptability

The model-following paradigm, which is used in each of the hierarchical levels, allows the system to derive a model to be tracked at the immediate lower level. This model is tuned to the current state of the system at that level. This implies that at each level the controller is structurally adapted to the current plant behavior at that level and the current goal.

Disparity

For a particular control problem, there may be great structural differences between the dynamic presentation of the goal, perhaps a semantic network with an arbitrary world domain, and the plant, a set of differential or difference equations evolving on a vector space. This disparity, which includes a multiplicity of time-scale dynamics, is characteristic of almost every robot control design problem. To tackle the design problem, our approach uses a nested hierarchical architecture for the structure of the controller. This allows for the definition of a chain of levels such that the disparity of the data representation of adjacent levels is sufficiently small so that the amount of computation for translation is manageable.

Controller Complexity

The controller complexity is reduced as a result of the nested hierarchical approach. The controller dynamics are spread out across the levels of the hierarchy and the controllers at each of the levels can be analyzed more easily. The representation of the state estimate is simplified because at each level the representation has to cover only the current level of abstraction. Many of the details of the state are left to lower levels.

Divide and Conquer

In principle, given the specifications, goal class, and plant dynamics, it is possible to formulate a single-level controller to accomplish the task. However, the amount of computation required would make a real-time implementation infeasible. The proposed hierarchy reduces the computation required for determining the actuator signals by distributing it across the hierarchies. The different levels of the hierarchy, as well as the major functional modules of the canonical form of the controller, can compute in parallel.

Structural Dependency

Consider the following aerospace design scenario[10]: A six-degree-of-freedom manipulator is to be used to change modules in a satellite on orbit. For this problem, the plant of the system is characterized by the manipulator and actuator

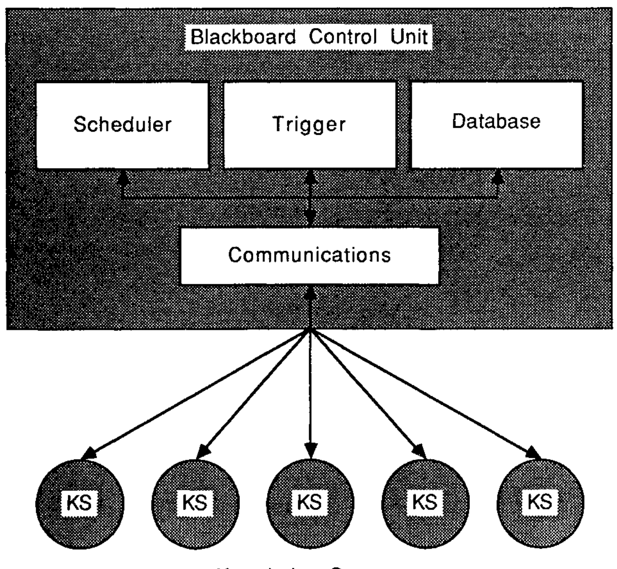

Fig. 5 Generic blackboard architecture.

dynamics, i.e., a set of coupled second-order differential equations on a given vector space. The goal is specified by a semantic net that contains the steps in the manuever (e.g., remove left and right screws, remove lid, locate module, etc.) and the strategies for malfunction and subgoal failure recovery. For this type of application and control specification, a nested hierarchical controller is a natural choice. However, it is our contention that this hierarchy should be derived from the design data and not be imposed, arbitrarily, a priori.

Blackboard Integrating Architecture

A benefit of the canonical representation is that it clearly delineates where large grain parallelism exists in the control system. Not only can each layer of the hierarchy be run in parallel, but each of the five elements of the canonical form can run in parallel. It would be desirable, therefore, to identify an integrating architecture that will support the multiple levels of data abstraction and the parallel execution of large grain parallelism. The blackboard paradigm provides these features.

The generic form of a blackboard architecture is shown in Fig. 5. Reference 11 is a blackboard survey article that explains the paradigm in detail. The key elements are as follows: 1) shared global data base, 2) modules of executable code called knowledge sources (KS), 3) triggering mechanism that enables KS's for execution, 4) scheduler that selects and initiates KS execution.

The data base, trigger, and scheduler can be grouped together and referred to as the blackboard control unit (BCU). This structure has similarities to conventional system control architectures, with executive, data base, and tasks, however, there are some important differences: 1) the data base is an integral part of the system and is designed to handle complex data structures and general data query operations; 2) the KS's may execute on one or multiple processors and are not allowed to communicate with each other directly (only as a result of interactions with the data base); 3) the idea of a software-generated interrupt is extended to allow complex trigger patterns to be defined in terms of the current data base contents (referred to as data-directed invocation); and 4) the scheduler supports the implementation of both algorithmic- and heuristic-based KS dispatching. A heuristic is a general rule or guideline, which is simple to compute and provides a good result most of the time.

The two classes of problems to which this architecture has been applied are sensory perception[12,13] and robot control.[14,15] In the first application class, the KS's tend to be small, numerous, and operate in parallel. The data base contains large abstract data structures. The trigger conditions are complex and a KS that is triggered may be disabled again without ever executing because of changes made to the data base by other KS's. The KS's are scheduled for execution based on heuristics[16] because, in general, there is no closed-form solution for which KS is best—most opportunistic—to run next.

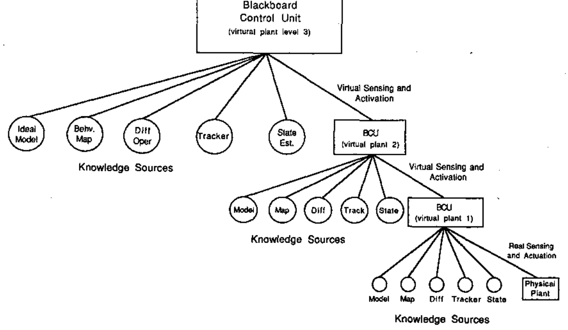

Fig. 6 Mapping of control hierarchy to blackboard.

The second class of applications for blackboards is robot control. The data base generally contains simpler data structures. The KS's are larger and tend to have different computational tasks. Computations of the KS's are synchronized through blackboard-triggering events. The scheduling of trigger events is typically by a simple algoritm, such as first in, first out, and the KS's are typically distributed across multiple processors.

Some recent mobile robot control systems that have been implemented based on the blackboard paradigm[17,18] make use of the blackboard's ability to support multiple levels of data abstraction, data-directed invocation, parallel processing, and heterogeneous software and hardware computing environments.

The hierarchical representation of the control system and the breakdown of each level into a canonical form allow us to define the mapping of the control system into a blackboard-based implementation. The elements of the canonical form—the ideal plant model, the difference operator, the tracker, the state estimator, and the behavior map—are implemented as knowledge sources. (See Fig. 6.) Their interconnections are implemented as blackboard-triggering conditions. Actuator output and sensory input are implemented as a communications KS, which interacts with an independent blackboard system to support the next lower level of the hierarchy. At the lowest level, the actuation and sensory KS is actually the interface to the hardware actuators and sensors. The use of the blackboard in this way is similar to a graph reduction architecture.[19]

It is expected that the perception-type blackboard will find its application within the state-estimator functional block. Here, for example, you may have to determine the location of an object in the world from visual data. Heuristically guided perception has proven successful in this domain.[20]

A very large scale integrated chip set, called the Blackboard Control Unit (BCU), is currently under development for real-time embedded control system applications.[21] The BCU will be responsible for supporting the shared data base, triggering, and scheduling functions of a blackboard system. The chip-set architecture exploits parallelism and pipelining in the execution of these functions to increase the BCU throughput.

Example: Mobile Robot Control

A mobile robot, ZOD, is a zeroth-order device for delivery in an office environment. It is being used as the test bed for the evaluation of our hierarchical control concepts. Five layers of the hierarchy have been defined and are currently being developed. The selection of the layers was based on a retrospective analysis of an ad hoc control system built for the robot. The application is to manuever around the corridors of our office building performing various pickup and delivery tasks, such as collecting time cards. The system is to be voice-commanded and is to plan and execute tasks autonomously. Figure 7 shows the five layers of the hierarchy and an example of the types of information that is being passed between the elements. This is an elementary example. Its purpose is to illustrate the notions presented previously.

This example shows the state of the hierarchical controller midway through the execution of a task to collect time cards from a group of employees. The layers have been named dialog, task, goto, move, and base. The decreasing level of abstraction can be seen as the hierarchy is traversed from top to bottom. The need for increasingly faster update rates also can be seen as the commands go from "ZOD, please collect the time cards," which may take 15 min, to "(move_base 116)," which may take only 3 s. The state estimator at each level is responsible for maintaining the state of an internal model of the subplant below it. At the base level, the state estimator understands about the geometry of ZOD, such as the location of its wheels and the relationship of wheel rotation to transla-

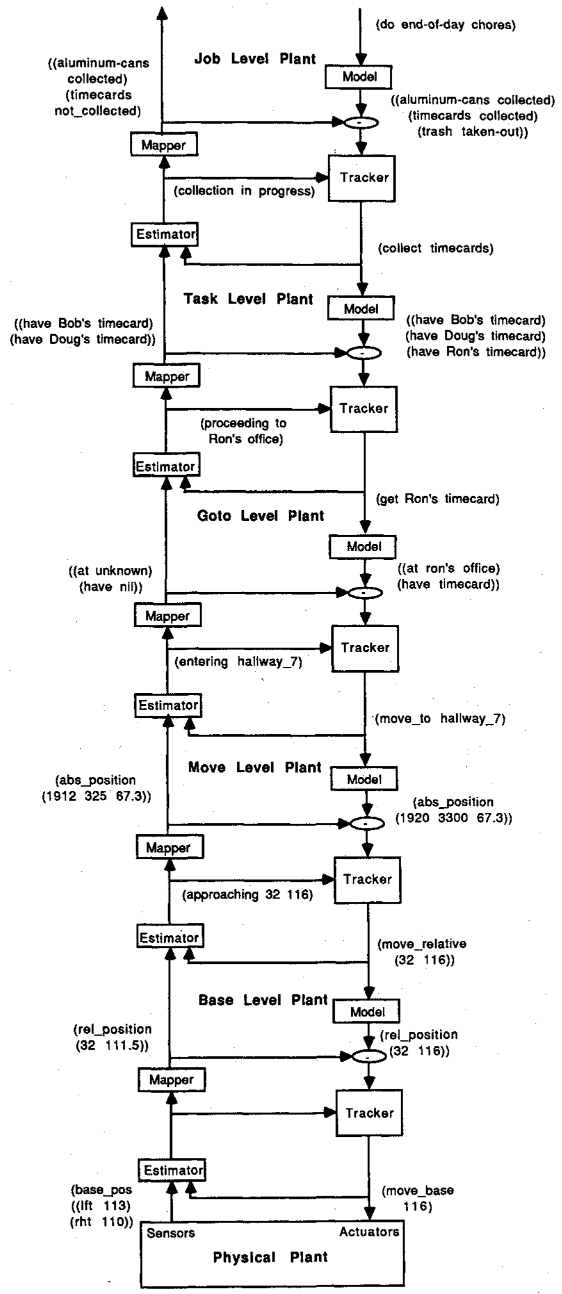

Fig. 7 Example of nested hierarchical controller applied to a mobile robot pickup and delivery robot.

tional motion. The ideal model creates an abstract view of the plant, which may ignore many lower-level details but is computationally efficient. The ideal model at the goto level understands the "get" command. It knows that to get something you must first be near it and then grasp it. The operation of the "get" on the ideal model therefore results in a desired behavior of being "at Ron's office" and "having time card." The behavior map is responsible for abstracting the current state estimate. In the move level, the state estimator generates relative position information. The behavior map then abstracts that information into an absolute reference frame, which is useful for the global path planning operation. The tracker at the task level sees the difference between the desired behavior and the sensed and abstracted behavior as "have Ron's time card." It issued the "get Ron's time card" command and, because the current state estimate for this level is "proceeding to Ron's office," it will continue asserting this command until it is completed.

Conclusions

A canonical form of a controller that can be used at different levels of abstraction within a hierarchical control system was presented. The way this approach addressed the issues of controller robustness, adaptability, disparity, and complexity was discussed. The model-following paradigm, which is part of the canonical form, was shown to allow for explicit declaration of control requirements in terms of a model that drives the system. Model following was also shown to decouple the goal specification from the design requirements during implementation. The computational complexity of each of the controller levels was shown to be simplified by implementing abstract operators on abstract data objects. The real-time execution performance was analyzed in terms of the parallelism that became explicit in the proposed hierarchical control scheme. A related control hardware and software architecture, the blackboard, was presented as a feasible implementation for embedded systems. Finally, this work, as a whole, sets the foundations of a controller decomposition algorithm that will be based on goal specifications, performance requirements, and plant dynamics.

Acknowledgments

The authors would like to acknowledge Robb Graham of Boeing Electronics Company and Dr. Kjell Carlsen of Boeing Computer Services Company for their support during this research; Dr. Andre Goforth of NASA Ames for his useful comments during the preparation of this paper; and the AIAA reviewers for their valuable criticism. This work was performed at Boeing's High Technology Center.

References

[1]Jamshidi, J., *Journal of Large Scale Control Systems*, Springer-Verlag, New York, 1986, Chap. 1-4.

[2]Mesarovic, M. D., Macro, D., and Tashahara, Y., *Theory of Hierarchical Multilevel Systems*, Academic, New York, 1970.

[3]Brady, M. et al. (eds.), *Robot Motion, Planning and Control*, Massachusetts Inst. of Technology, Cambridge, MA, 1982.

[4]Kohn, W. and Van Valkenburg, J., "Real-Time Manipulator for a Robot Manipulator in Simulated Microgravity," *Proceedings of Robots and Expert Systems Conference*, Instrument Society of America, Research Triangle Park, NC, pp. 31-43.

[5]Kohn, W. and Skillman, T., "Intelligent Hierarchical Control—Theoretical Foundations," The Boeing Co., Seattle, WA, D905-10108-1, Apr. 1988.

[6]Farlow, S. (ed.), *Self-Organizing Methods in Modelling: Group Method of Data Handling Type Algorithms*, Marcel Dekker, New York, 1984.

[7]Astrom, B., *Optimal Stochastic Control*, Academic, New York, 1980, Chap. 2 and 3.

[8]Kohn, W., "A Declarative Theory for Rational Controllers," *Proceedings of the IEEE Control and Decision Conference*, Inst. for Electrical and Electronics Engineers, Piscataway, NJ, Dec. 1988.

[9]Cruz, M. (ed.), *Feedback Control*, McGraw-Hill, New York, 1981, Chap. 4.

[10]Kohn, W. and Campbell, R., "Online Task Interpretation for Astrobot," *Proceedings of the IEEE Joint Artificial Intelligence and Process Control Conference*, Houston, TX, June 1983.

[11]Nii, P. H., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *The AI Magazine*, Vol. 7, No. 2, Summer 1986, pp. 38-53.

[12]Fennel, R. D. and Lesser, V. R., "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay II," *IEEE Transactions on Computers*, Vol. C-26, No. 2, Feb. 1977, pp. 98-111.

[13]Nii, P. H., Feigenbaum, E. A., Anton, J. J., and Rockmore, A. J., "Signal-to-Symbol Transformation: HASP/SIAP Case Study." *The AI Magazine*, Vol. 3, No. 2, Spring 1982, pp. 23-36.

[14]Skillman, T. L., "Distributed Cooperating Processes in a Mobile Robot Control System," *Proceedings of the Conference on Artificial Intelligence for Space Applications*, Univ. of Alabama, Huntsville, AL, Nov. 1986, pp. 327-338.

[15]Elfes, A., "A Distributed Control Architecture for an Autono-

mous Mobile Robot," *Artificial Intelligence*, Vol. 1, No. 2, Oct. 1986, pp. 99–108.

[16]Dodhiawala, R. T., Jagannathan, V., and Baum, L. S., "Erasmus System Design: Performance Issues," *Proceedings of Workshop on Blackboard Systems Implementation Issues*, American Assoc. of Artificial Intelligence, Seattle, WA, July 1987.

[17]Shafer, S., Stenz, A., and Thorpe, C., "An Architecture for Sensor Fusion in a Mobile Robot," *IEEE International Conference on Robotics and Automation*, Inst. for Electrical and Electronics Engineers, Piscataway, NJ, 1986.

[18]Harmon, S. Y. and Solorzano, M. R., "Information Processing Architecture for an Autonomous Robot System," *Proceedings of the Oakland Conference on Artificial Intelligence*, Rochester, MI, Apr. 1983.

[19]Hoare, C. A. R. (ed.), *The Implementation of Functional Programming Languages*, Pt. 2, Prentice-Hall International Series in Computer Science, Englewood Cliffs, NJ, 1987, Chap. 10–17.

[20]Pearl, J., *Heuristics—Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, MA, 1984, Chap. 1 and 2.

[21]Ling, C., Dodhiawala, R., Nguyen, D., and Skillman, T. L., "A Parallel VLSI Architecture for Blackboard Systems," *Informal Proceedings of Workshop on Blackboard Systems Implementation Issues*, AAAI, Boeing Advanced Technology Center, Seattle, WA, July 1987.