Intelligent Agent-Based Control

David H. Scheidt

Control agents are effective mechanisms for autonomous control of complex distributed systems. This article describes the Open Autonomy Kernel (OAK), a control agent architecture that is based on a unique combination of model-based reasoning and software agents. The discussion includes the design and testing of OAK agents on Navy auxiliary systems, the need to consider the control network as a part of the system being controlled, and the way in which control networks may be made more robust by allowing them to "flinch" prior to anticipated damage events. We conclude by showing how control agents based on Markov blankets may be used to develop effective, extremely lightweight control of large chemical and biological surveillance sensor grids.

DEFINITIONS

Embedding controllers within autonomous software agents, called *control agents*, provides a basis for autonomous control of distributed sensor/actuator systems. By sharing information and, in special cases, reasoning, control agents can collaboratively employ the resources of the controlled system in a way that addresses global system tasks. In this article we investigate some useful techniques for controlling special cases of distributed sensor/actuator systems, specifically *connection systems* and *spatially oriented systems*. We illustrate these techniques by describing several recent, current, and planned research projects in a variety of domains. Along the way we discuss the use of control agents for the control of Navy ship auxiliary systems, computer networks, societal infrastructure, HVAC (heating, ventilation, and air conditioning) systems, and regional chemical and biological surveillance sensor grids.

We define connection systems to be systems that consist of two or more identifiable, related components that influence each other along one-dimensional connections. Influences may be discrete or continuous in nature. Many real-world systems may be described as connection systems, including household plumbing, computer networks, and electrical systems.

We define spatially oriented systems as systems consisting of elements that act in three-dimensional space. Environmental and system phenomena and their influences on one another may be effectively organized as a three-dimensional matrix. Real-world systems that may be described spatially include fire moving through a ship and chemical plumes.

Control agents are *autonomous* in that they do not require human supervision or monitoring in order to function. However, autonomy does not imply that control agents are oblivious to humans; rather, control agents are responsive to information provided by operators as needed and react to operator demands and objectives.

Control agents employ a cyclic process to accomplish the desired behavior for the system being controlled. The control cycle (Fig. 1) consists of three steps.

1. *Estimation* uses observations to compute a belief function over the state space of the system under control. A belief function describes the possible component states within the system that are consistent with current and historical observations. The state space of a system is the set of possible states for the system.
2. *Planning* uses this belief function, in conjunction with system goals and constraints, to produce a plan of action.
3. *Execution* translates an action plan into actuator commands.

Intelligent control agents control systems whose state space cannot be completely enumerated through traditional discrete and/or continuous means, necessitating the use of artificial intelligence reasoning techniques. Such techniques may be required for estimation, planning, and (rarely) execution. Intelligent planning is usually necessitated by the size and complexity of the system, whereas intelligent estimation is driven by insufficient observability, for example, in systems with sparse or unreliable sensors.

SHIP CONTROL

An early application of control agents was the control of Navy ship auxiliary systems. The electrical, chilled water, low-pressure air, and fire suppression systems are the main auxiliary systems for Arleigh Burke–class Aegis destroyers. These systems are complex, interdependent, and distributed. Currently high-level control of these systems is the primary responsibility of dozens of sailors on each of the Navy's capital ships. Effective automation of these systems would allow the Navy to significantly reduce the number of sailors in harm's way and drastically reduce costs.

Auxiliary systems on Arleigh Burke–class destroyers contain thousands of sparsely instrumented, interdependent, controllable nodes. For example, the chilled water distribution system consists of a dozen complex machines such as pumps and chiller plants, approximately 400 valves, and 23 service loads.

The behavior of components within an auxiliary system depends on the behavior of those components to which they are connected, often recursively. The auxiliary systems themselves are interdependent; e.g., the chilled water system runs on electrical power provided by the electrical system while the electrical system is kept cool by the chilled water system. These interdependencies—which extend to other ship systems such as the HVAC, combat, and fire support systems—combine to generate a complex supersystem, no portion of which may be effectively controlled in isolation. Yet the complexity of the combined system prevents the effective use of traditional control system methodologies.

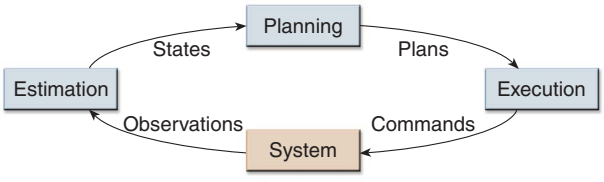


Figure 1. The control cycle.

Technology

The intelligent control agents built by APL for the control of Arleigh Burke auxiliary systems are based on a few key technologies that combine to provide autonomy, distributed intelligence, and real-time performance. *Hierarchical control architectures* allow systems to support both real-time performance constraints and computationally intensive deliberative reasoning. *Software agents* are used to provide autonomy and support distribution, while *model-based reasoning* is an effective reasoning technique for providing intelligence in large "connected" systems.

Hierarchical Control Architectures

Hierarchical control architectures decompose control along the lines of component abstraction, i.e., larger, more abstract representations of the system are controlled by increasingly higher levels of interacting controllers. Our hierarchical control architecture is similar to subsumption architectures[1] in its decomposition of control and allocation of responsibility among controllers. It differs from subsumption architectures in that our higher-level controllers modify the fitness criteria of lower-level controllers, thereby indirectly changing the control actions of the lower-level controllers, whereas subsumption architectures directly modify the control function of the lower-level control architecture.

Hierarchical control architectures provide effective control of complex systems that have multiple hierarchical goals, multiple sensors, and a need for robustness. These architectures decompose control into two or more layers (Fig. 2).

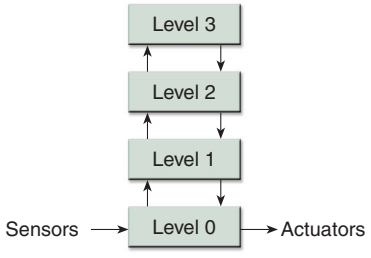


Figure 2. A hierarchical control architecture.

The zeroth layer of control is the only layer permitted to communicate with the system's sensors and actuators. It is an unsophisticated closed loop controller capable of real-time monitoring of the sensors and actuators. Above the zeroth level is the first level control. This layer is aware of the data derived by the zeroth level and is capable of reasoning with those data at a higher level of abstraction. Further, the first level is permitted to unilaterally modify the fitness criteria used by the zeroth level. The second level uses level 1 data to reason at a higher level of abstraction, suppressing the data flow of the first level.

Additional levels may be constructed as necessary, with each level modifying the criteria used by the subordinate level. For example, consider an intelligent control system for a household electrical system. A level zero controller might be a rheostat being used to regulate the flow of power to an electrical lightbulb. A first level controller might be responsible for regulating illumination in a specific room; a second level controller might be responsible for determining which rooms in the house require light; a third level controller might be concerned with power management among lighting (electrical), cooling, and major appliances; and a fourth level controller might be concerned with managing power needs between the house and the hospital across the street.

Software Agents

Software agents are software processes that can reason about and act upon their environments. The software agents used to control ship auxiliary systems are intrinsically permanent and stationary, exhibit both reactive and deliberative behavior, and are declaratively constructed. Agents are reactive in their ability to reconfigure the systems within their control in the context of an existing plan and deliberative in their ability to create a plan in response to observed states and defined goals. Our agent's extrinsic characteristics include proximity to the controlled system, social independence, and both awareness of and cooperativeness with the goals and states of other agents. These agents are environmentally aware, and behavior of the environment is predictable through each agent's model.

Systems of agents consist of nearly homogeneous agents that are independently executed yet contain unique models of the systems for which they are responsible. Homogeneity of construction enables the widespread reuse of code, thus simplifying controller construction and lowering the cost of maintenance. Independent execution provides for the distribution of control (as compared to single threaded control implemented on distributed processors). Distributed control improves efficiency (control agents are never required to wait on another's execution) and enhances survivability (damaged controllers never prohibit surviving controllers from managing their subsystems).

Agents are fundamentally permitted two types of relationships: peer-to-peer and parent-to-child. In building control agents for ship control, the parent-child relationship is used as a mechanism for implementing a topology of agents that support a shipwide hierarchical control architecture.

Model-Based Reasoning

Model-based reasoning is an overloaded term. The model-based reasoning used in the Open Autonomy Kernel (OAK), discussed in detail below, refers to a "reasoning from first principles" approach to diagnosis.[2] A declarative description of the physical system is formulated to explain how parts of a system are functionally interdependent.[3]

The theoretical basis for model-based reasoning is discrete event system theory, specifically, partially observable Markov decision processes (POMDP).[4] In a model-based representation of the physical system, a behaviorally based POMDP model represents a component (e.g., Fig. 3), and each component has a set of possible states. Components within the model are assumed to be in one of a number of discrete states. The state of the entire system being modeled is the union of the states of the components. The necessary conditions for a component to be in a particular state are specified with propositional logic, and transitions among states are expressed by extending this propositional logic with a limited set of temporal operators.

Associating the attributes from different components generates multicomponent systems. For example, the OPEN state of a simple valve is defined in part by a propositional statement equating the inbound flow into the valve with the outbound flow; the CLOSED state is defined in part by associating the inbound and outbound flow with zero. A simple system consisting of two valves in series may be generated by instantiating two valve models, Valve_1 and Valve_2, and establishing a propositional statement equating the inbound flow of Valve_2 with the outbound flow of Valve_1. A small open loop system containing two valves in series is shown in Fig. 4.

This representation scheme is beneficial when constructing large, complex systems. By encapsulating

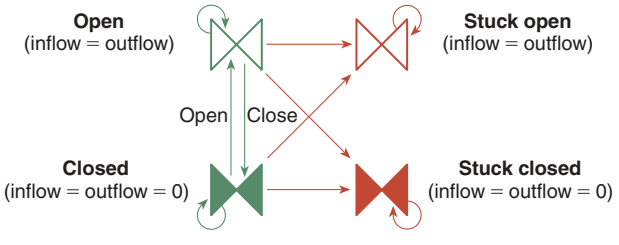


Figure 3. Valve model.

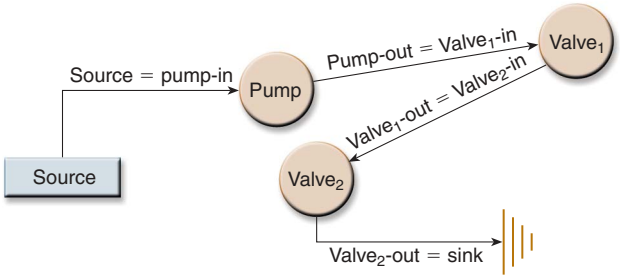


Figure 4. A small open loop system using associated component models.

component behavior within a single logical model and constraining components through context-independent attributes, the components themselves become independent models. This enables model replication and reuse. Attribute associations may be considered context-independent because they are based on the first principles structure of the system. The intended use, or context, of the system is not germane to model construction.

A model-based reasoning inference engine (MBRE) is used to find likely state transition trajectories without having to generate all of the possibilities. The MBRE propagates constraints, linking components' states efficiently, and applies probabilities, both backward and forward in time, to identify probable state trajectories. The process of state identification is the dual of the problem of generating a trajectory to a desired state so the same MBRE can be used to generate a trajectory to a desired state. This property is useful as an interface to a planner.

Reasoning from the model involves correlating observations of the actual physical system with the model and identifying the likely states of the system. This mechanism can then be used to diagnose or identify unexpected observations. A naive approach to solving this problem would generate all possible states and determine which entail the observations. Clearly this is intractable. Even for small systems, there are an impractical number of possible states.

In practice, model-based reasoning is a three-step process.

(1) Propagation of control action effects through a system. Propagation generates a predicted state for the system, including controlled and uncontrolled components. Observations of system behavior are compared to the predicted state; if observations match predictions then the system is assumed to be behaving nominally. The existence of conflicts between observed and predicted states indicates the existence of one or more failures within the system.

(2) Identification of candidate failure scenarios that most effectively resolve the conflicts. The strategy used to resolving these conflicts is *conflict-directed best first search* (CBFS), based on de Kleer and Williams' general diagnostic engine[5] and described in detail in Ref. 4. In CBFS, partial hypotheses that conflict with observations are identified (e.g., a lightbulb being on and the belief that the light switch is off), and those scenarios that include the conflict are removed from the search space (e.g., having identified the conflict between the switch and the lightbulb, in attempting to determine the possible states of the switches and circuit breakers in a house, we will now ignore all solutions containing our switch in question being off).

(3) Selection of the most probable scenario from the identified candidates. The fitness criteria used by the CBFS to select the most probable solution is a system-wide candidate probability based on the probabilities of individual component states. In practice, failure probabilities are arbitrarily assigned by system modelers. In production, these probabilities would be assigned based on historical or engineering failure analysis. By using a general diagnostic algorithm, implementation effort is limited to constructing the model upon which the algorithm operates. System design and maintenance are limited to changes in the model, which is interpreted at runtime, and do not require modifications to compiled software. In addition, because of the encapsulation of the component models, model maintenance is limited to those components modified in the controlled system and any components to which they are physically attached.

THE OPEN AUTONOMY KERNEL

Goals and State Changes

The control agent system devised for ship auxiliary system control is the OAK, a distributed, multi-agent system. OAK consists of a five level hierarchical control architecture (Fig. 5),[6] the top three levels being control agents. Process level control consists of the manufacturer's device drivers and closed loop process control to monitor sensors and the execution of instructions. Each logical device within the system maintains its own

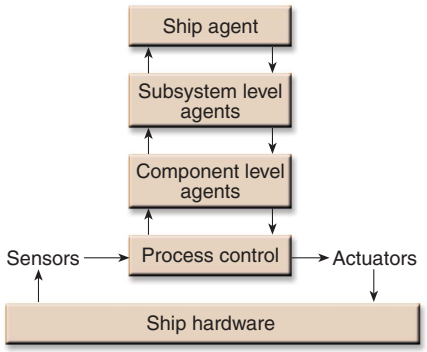


Figure 5. The hierarchical control architecture implemented in the Open Autonomy Kernel.

process level controller. Process level controllers do not consider the effect on/from other devices in the system except through instructions subsumed from the component level. Component level control agents control small groups of devices. Subsystem level agents control subsystems such as the ship's electrical or chilled water system. A single ship control agent manages shipwide control.

One of the major use-cases of OAK is to react to goals entered by an external actor. These are system level goals (such as "battle stations," or "cruise" directives that shift system priorities from efficiency to survivability) that potentially impact the goals for the entire multi-agent system.

Goals that are entered from an operator are sent directly to the ship control agent using a *goal message*. The ship agent develops a plan with goals that apply to the domains of its "child" agents. Goals have a priority associated with them, which is used for goal preemption.

After the ship control agent develops a plan and directs a goal to one of its child agents, the child control agent for processing receives the goal. This control agent develops a plan to implement the goal. The plan includes goal messages for the control agent's own "children." This propagation continues until actual hardware commands are sent to the appropriate actuators.

Successful goal implementation implies a state change. Control agents wait for reactions from child agents—or in the case of process level controllers, reactions from sensors—to indicate that the command has been successful. The agent is then free to pass out goals that were order-dependent on the goal just implemented. State change events are transmitted through the use of a *fact message*, which contains a representation of the knowledge contained in an agent. When states change, all subscribed agents are informed and propagation of state changes begins. Note that since many agents may subscribe to an event, state changes may be propagating in several subtrees at any given time.

To appropriately handle changes in the state of the system, OAK uses the *L2* MBRE[7] developed by NASA/Ames Research Center. OAK's control agents are able to determine the state of the model, compare that state to a knowledge base, and plan reactively. Thus, a control agent can autonomously control its domain until an agent that is higher in the hierarchy (or in the case of the ship control agent, the operator) preempts its control.

OAK agents must be able to plan in order to achieve specified goals. The plan format is an ordered sequence of fragments. Each fragment consists of one or more subgoals. The idea is that, within a fragment, each subgoal may be accomplished in parallel, while subgoals in a prior fragment must be completed before the current fragment can be attempted. OAK executes developed plans by transmitting each subgoal at the appropriate time to the appropriate agent or piece of hardware and then waiting until those subgoals are accomplished or fail.

Different planners may be appropriate for different agents, depending on the domain being planned. Therefore, the planner is instantiated at runtime differently for each agent from a group of developed planners. OAK uses two planners, a general *scripted planner* and the specialized *graph-based planner.*

The scripted planner matches on the incoming goal and a propositional logic expression about the current world state, producing a predefined response. Different propositional expressions, and therefore plans, may be associated with each incoming goal. Also, since the scripts are checked in a specific predefined order, a simple priority of plans can be imposed.

The graph-based planner was written specifically for the test domain described below. Planning consisted of determining how to move flow from a source to several sinks through a dynamic pipe network having many operational constraints. The problem representation was a digraph, with weights on each edge according to the constraints. The planner operated by performing Prim's minimum spanning tree algorithm[8] on the graph to determine how to get flow to as many of the desired sinks as possible. The planner determined the actions that each agent would need to take and generated a plan based on the actions determined.

An example of OAK reasoning on a simplified system is given in the boxed insert.

OAK has been implemented on the chilled water Reduced Scale Advanced Demonstrator (RSAD, Fig. 6) at the Naval Surface Warfare Center, Philadelphia. The RSAD is a reduced-scale model of two zones of the Arleigh Burke chilled water system. It contains 4 pumps, 2 chiller plants, 2 expansion tanks, and approximately 100 controllable valves. In-line tanks containing controllable heaters simulate equipment that is directly cooled by the chilled water system. The units of equipment cooled by the chilled water system are known as *loads*. The RSAD includes 16 simulated loads.

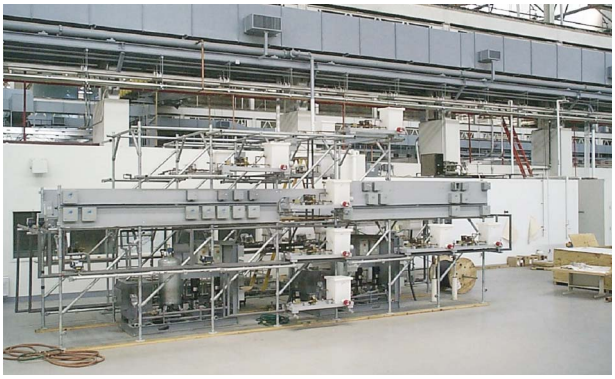


Figure 6. The chilled water testbed.

AN EXAMPLE OF GOAL DECOMPOSITION AND ESTIMATION USING OAK

Consider the simple plumbing system in the figure. The purpose of the system is to provide flowing water to a water-cooled radar system (the chiller plants required to cool the water are removed to simplify the example). The entire system has only one metric—the existence of coolant flow through the radar. This metric is provided by a single flowmeter attached to the outflow of the radar. The four pumps and the radar are controllable in that they can be commanded ON or OFF. The valves are controllable in that they can be commanded to OPEN or CLOSE. The check valves are not controllable and are only included to indicate the direction of flow. α, β, and χ are labeled connections between the assemblies.

Three control levels are provided: seven process control agents shown in pink, four assembly control agents in light blue, and a single ship agent in orange. Decomposition of goals might occur as follows:

Intitial states: All components are OFF.

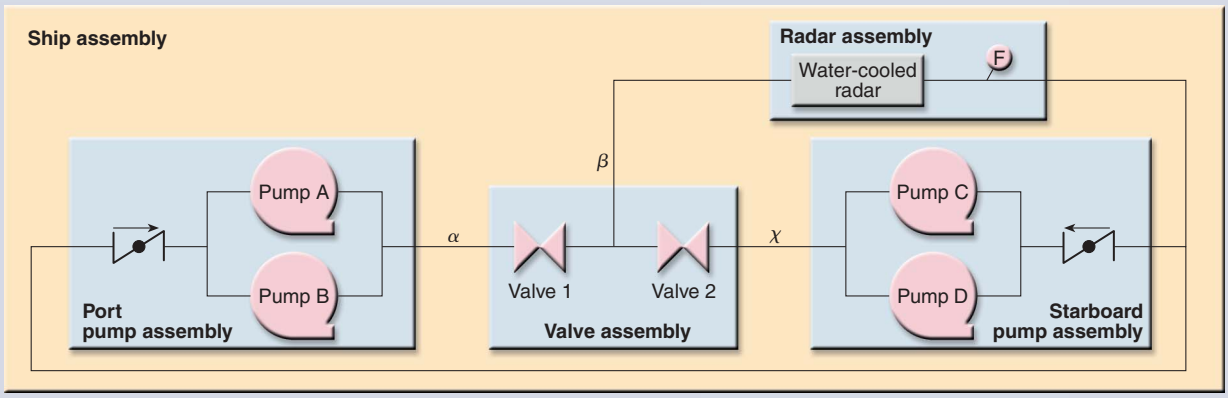
- A goal is given to the ship agent from an operator to provide radar coverage. Knowing the states of the assemblies (but not the components), the ship agent generates three goals for its subordinate assemblies. The goals are to "turn ON" the radar (sent to the radar assembly), connect α to β (sent to the valve assembly), and provide flow to α (sent to the port pump assembly).
- The pump assembly, knowing the states of the pumps, generates a plan that consists of the goal to turn ON for Pump A. The valve assembly generates a plan consisting of the "opening" Valve 1.
- The process control agents send actual commands to the hardware implementing these goals.

Let us examine how the agents would reason in the event that Valve 1 becomes stuck CLOSED. Having completed a planning and execution cycle to implement the initial states, an estimation cycle occurs. During this cycle we observe that there is no flow measured by the flowmeter. This generates a fact that "there is no flow at the radar (β)." This fact is sent to the valve assembly which, believing that Valve 1 is OPEN, infers a new fact that there is no flow at α. This fact is sent to the pump assembly, which infers that there is no flow coming out of Pump A or Pump B. This conflicts with our belief that Pump A is ON (the result of being turned on earlier).

The model-based reasoning engine(s) identifies three possible solutions to this conflict: Valve 1 is stuck CLOSED, Pump A is broken, or the radar is broken (for the purpose of this illustration we do not consider sensor or control network failures). The predicted mean times to failures of our components are 10^6 s for Valve 1, 10^5 s for Pump A, and 10^7 s for the radar. The most probable solution to our conflict is that Pump A has failed. Our belief state for Pump A is updated accordingly. The port pump assembly now generates a new plan to achieve its goal. This new plan consists of commanding Pump B to turn ON.

After Pump B has been commanded to turn ON, a second estimation cycle is performed, again with no flow being recorded by the flowmeter. Recall that we previously concluded that Pump A had failed, so for our current belief state, three possible solutions are identified: Pumps A AND B have failed, Pump A and Valve 1 have failed, and Pump A and the radar have failed. The truth maintenance system used in our model-based reasoning system is non-monotonic, however, in that beliefs are considered tentative, alternative explanations to observations. In our example, a previously ignored possibility, that of Valve 1 failing, accounts for all observed conflicts and is more probable than the derivative solutions of our currently adopted belief. The truth maintenance system now replaces its previous conclusion that Pump A failed with the conclusion that "Valve 1 has failed."

Using our new belief state, the ship control agent generates a plan for the assemblies with two new goals: turn OFF the port pump assembly, and turn ON the starboard pump assembly and connect β and χ. After these goals are decomposed and executed, flow is observed and the current plan is accepted.



A simple plumbing system.

The RSAD control prototype uses 20 OAK agents to control its pumps, plants, and valves. Each agent contains its own diagnostic engine, planning engine(s), and execution managers and has the ability to receive and propagate goals and facts.

OAK Testing

Three test sets were performed for the RSAD implementation of OAK. Four types of test scenarios were performed during each test set. The first three scenarios were designed to provide basic coverage of the primary

OAK capabilities. The final portion of testing was *ad hoc* and gave the testers an opportunity to game the system. During the second and third test sets, hardware faults were instigated by either physically disconnecting a component from its power supply or by physically disconnecting a component from the control network.

The first test scenario was used to demonstrate OAK's ability to reconfigure the RSAD based on high-level operator goals. During this scenario the RSAD was given consecutive commands from a command and control simulator to move from one "ship state" to another. The four specified ship states each had a unique combination of desired states and fitness criteria.

The second test scenario entailed inducing a series of sequential component failures that initially forced OAK to reconfigure the system in order to satisfy the high-level goals and eventually degrade the RSAD so that its stated goals were no longer achievable. This scenario was also specifically designed to test the non-monotonic capability of the reasoning engine, which is non-monotonic in that it makes tentative decisions regarding the belief function. If future states conflict with the tentatively selected state, the reasoning engine changes its current and historical belief function. An improbable component failure that was observationally indistinguishable from a probable failure was generated, resulting in a misdiagnosis. Subsequent failures generated observations that reinforced the correct belief state and caused a change of hypothesis within the inference engine.

The third scenario consisted of inducing simultaneous failures to multiple components within the RSAD.

Throughout the testing, OAK consistently demonstrated the ability to plan, execute, and propagate facts and goals among agents. All three of the test sets were successfully completed. In total, 14 separate multistage tests were conducted during which OAK performed 34 diagnose-plan-execute cycles. It was able to identify the most probable failure scenario when insufficient observables presented multiple, indistinguishable situations. Also, OAK demonstrated the ability to retroactively update its belief state when evidence was provided to support what had been a less probable candidate solution.

At times, equivalent "best fit" reconfigurations were available. In these cases the observing mechanical engineers noted that OAK's reconfiguration was occasionally "unusual" or "not what I would have selected." However, upon inspection, the selected reconfiguration was always consistent with the reconfiguration goals and fitness criteria, and was considered reasonable by the observing engineers.

In addition to planned testing, twice during the third test set the RSAD experienced unexpected hardware failures. One failure occurred when a chiller plant unexpectedly failed to the OFF state. Another failure occurred when a valve unexpectedly failed to STUCK_SHUT. During both instances, OAK correctly diagnosed the failures and successfully reconfigured the RSAD.

OAK's planning and execution capabilities succeeded in performing a successful reconfiguration of the RSAD in all test cases. When a complete solution was available, a solution was found. When multiple solutions were available, OAK was able to determine and select the solution deemed optimal in accordance with the fitness criteria expressed in the script-planner rule base. When no complete solution was available, the best fit partial solution was identified and executed.

NETWORK CONTROL

OAK considers the state of the ship's auxiliary systems and the state of the auxiliary system controllers when estimating, planning, and executing control of the ship. It does not consider the state of the control network. To improve robustness of the control system as a whole, the state of the control network must be considered. For example, network failure may limit the ability of a controller to send commands to subordinate controllers or actuators and may limit a controller's ability to obtain sensed information. The limitations placed on the estimation and controllability of the system by network failures should be considered when performing estimation and planning of a controlled system. Furthermore, if the control network is redundant, network resources may require reconfiguration.

Modeling Control Networks

To consider control network status when performing system control, we must be able to represent the network state in a manner that is consistent with the representation of the system being controlled. For a ship auxiliary system we may accomplish this by representing the network as a connection system. This allows us to use model-based reasoning for network/system estimation purposes.

At a component level we extend our component representation to include models of the component, controller, and network (Fig. 7). However, our architecture requires modification to incorporate control networks. Although components and subcomponents were tightly coupled throughout the hierarchy, the connections within a control network need not be (and frequently are not) tightly coupled with respect to the control system. For example, a network router may not be directly associated with an identifiable component or subsystem within the controlled system, yet the router's operational status may have a profound impact on the system's controllability. We must therefore treat the control network not as an aspect of the control system but as a separate subsystem within an even larger system

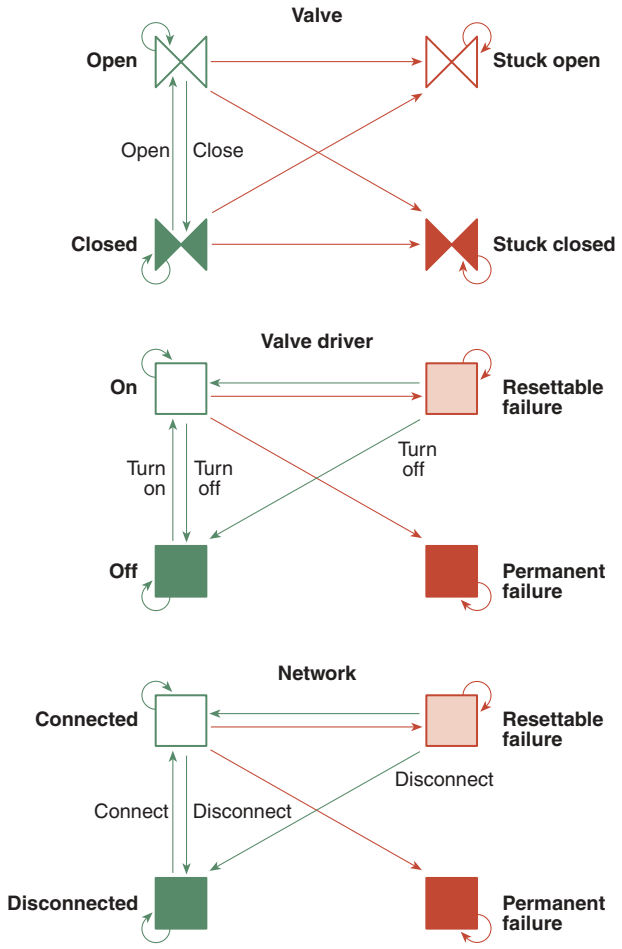


Figure 7. The three component models that determine a valve's behavior.

(Fig. 8) in which component states in the system being controlled are influenced by the control network. To account for the effect of the control network on physical components, new network-conscious states must be added to the POMDP model of the physical system (Fig 9).

The Impact of Network Outages on Low-Level Control Cycles

The ability to perform even the most basic control actions in a distributed control system depends on the "network reachability" of the component being controlled. Network reachability, i.e., the ability of the control system at large to send goals and receive facts from a component of the system, is defined by the state of the control network. The necessity to understand the controllability of a component when making a plan is ubiquitous in a layered control system. The use of computationally intensive control techniques such as model-based reasoning to estimate network state becomes problematic when network outages prevent high-speed process level control loops from functioning.

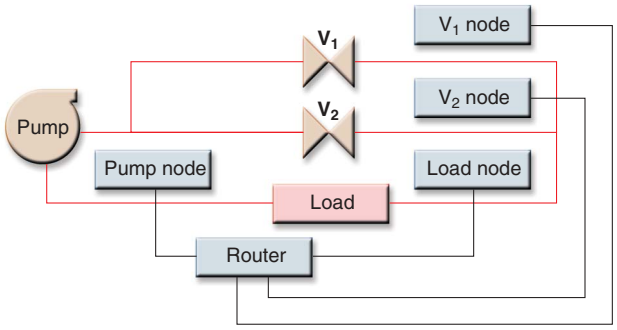


Figure 8. Even though the behavior of individual components may be tightly coupled, the structure of topologies of the control system and the control network may be completely orthogonal. In addition, components may exist in one system that are not tightly coupled with components in the other system.

Latency between a network outage and the recognition of the outage by higher control levels produces periods of time during which the lower levels continue to attempt to control components that cannot receive the control signals.

FLINCHING

Two approaches to mitigating the impact of this "ill informed" time period present themselves: either reduce the time between event and diagnosis, or preconfigure the system to limit the risk of reconfiguration based on an out-of-date diagnosis. While improved performance may reduce the duration of incorrect control, it cannot eliminate it. Correct preconfiguration increases the probability of continuous, robust control if reliable predictors of network outages can be provided. The causes of network damage are often spatial phenomena such as fires propagating through ships, missile damage, and loss of buoyancy. Models exist for spatial phenomena that directly impact the control network. The same spatial phenomena impact both the physical subsystem and the

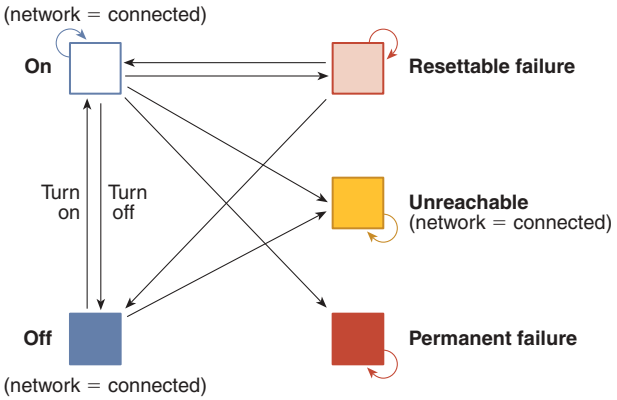


Figure 9. The valve driver model is extended to incorporate the fact that the driver cannot be set when the node is "unreachable." Network reachability is determined by the network model, which is a separate subsystem.

controllers; however, the scope of our funded research today is limited to the control network.

Combining the effects of model-based estimation of connection systems and estimation of spatial systems using cellular automata may be accomplished by modifying our architecture (Fig. 10). Levels that use model-based predictions replace the levels that used model-based diagnosis to infer states and plans for connection systems. Model-based predictions use the same underlying model-based theory previously described, with the noted addition that predicted behavior is construed rather than historical. Simply put, seminal events are provided to the model-based prediction system over time.

The effects of these events on the connection-based system are generated by propagating the effects of the state changes throughout the POMDP representation of the system. The result of the predicted system is a series of predicted system states referenced by time. Plans may be generated for the predicted state prior to the actual event. If the amount of time the system requires to reconfigure itself is represented as Δt, then the predicted system state may be provided to the reconfiguration engine/planner at the referenced (time – Δt) in order to provide "just in time" reconfiguration. In other words, the system can "flinch," thereby protecting itself from an anticipated blow. Because model-based diagnosis and model-based prediction share the same underlying representation, a model-based reasoning engine may be used to estimate actual system states, allowing the built-in non-monoticity of model-based reasoning to select candidate predicted scenarios based on actual observation.

Atop the model-based prediction levels we place spatial reasoning levels that are used to predict the effects of spatial phenomena. Spatial phenomena may be event driven (e.g., a missile hit) or may represent a continuum of effects (e.g., the movement of fire throughout a ship). We can translate the effects of these spatial phenomena on a connection-based system because each component within a connection system may be located in time-referenced geometric space. Accordingly, if we have a probabilistic understanding of the effect a phenomenon might have on a component (e.g., a fire's effect on a router), then we can cast that effect into connection system space. Conversely, if we understand the effect a connection-space component has on geometric space (e.g., a functioning sprinkler head on a fire), then we may predict the effect the connection system may have on the spatial phenomena.

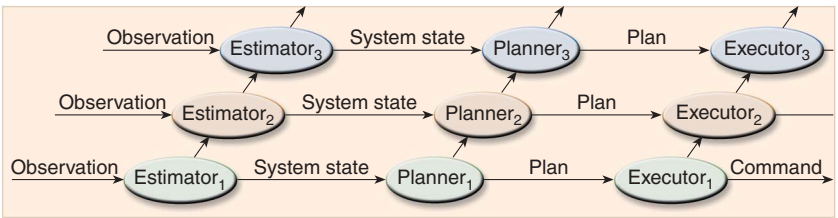


Figure 10. Control and spatial axes as a Markov blanket.

CONTROL OF EXTREMELY LARGE CHEMICAL AND BIOLOGICAL DETECTION GRIDS

The monitoring of public environments such as buildings, subway systems, or entire cities for the presence of chemical or biological agents that may indicate a terrorist attack is a timely problem domain. It would be desirable to field a system that could reliably monitor, map, warn of, and limit the effects of a chemical or biological attack. Sensors that are effective for the detection of these agents must often be situated within the local environments that they are observing. This implies sensors that are spatially distributed throughout the monitored area.

The number of distributed sensors required is proportional to the size of the monitored system, divided by the mean sensor coverage area. This number could easily reach into the thousands or even millions. Two types of systems are of particular interest: systems to detect and manage chemical-biological agents within a large building HVAC system and citywide chemical-biological detection systems. These problems share a number of characteristics that imply requirements for any feasible solution approach.

The envisioned sensor grids are autonomous, massively parallel, fault prone, and dynamic and imply the following requirements and constraints:

- The need for distributed, networked sensors and actuators
- Coordinated planning over the network to accomplish optimal sensor/actuator use (through flexible task-driven teams) and resource allocation (to husband expendable sensor/actuator resources such as power supply and expendable sensors)
- Avoidance of centralized data fusion and planning to stay clear of the burden of high data rate communications requirements, the increase in reaction time (due to communication delays and computational complexity), and the danger of a single point of failure
- Unavailability of a useful system design based on fixed, mode-based control laws or traditional adaptive control
- The need to support the control of system components of varying levels of awareness
- The need for fault tolerance (driven by the sheer size of the system and the potential costs

and difficulties of static, redundant components and real-time component replacement)

As the envisioned chem-bio detection networks become increasingly large, the ability of networks based on current technology to provide reliable, robust communication decreases proportionally. Redundant networking protocols and topologies (e.g., large TCP-IP networks such as the Internet) may be used to improve reliability; however, redundancy incurs a proportional increase in cost. *In situ* networks often have tight power consumption and bandwidth constraints and cannot maintain highly redundant networks.

One approach to providing robust communications without maintaining redundant communication paths is the use of *ad hoc networking*, where network topologies are generated on the fly. In addition to allowing for robustness for fixed-node networks, *ad hoc* networking has been shown to be effective in supporting efficient networks across mobile nodes.[9] In *ad hoc* networking, network elements "discover" each other at runtime, determine each other's relative position and networking capabilities and, armed with this information, collaboratively establish a network topology.

EMERGENT CONTROL

The use of *ad hoc* networks allows control networks to communicate between control elements in novel ways that are tailored to network-element orientations not conceived of at design time. These systems undergo change due to the removal, addition of, or movement of system components, controllers, and/or control network elements and require adaptive network topologies. As these systems become increasingly large and long-lived, and as the nature of the system being controlled changes, an adaptive control mechanism will be required as well.

We believe that *ad hoc* control may be provided through the provision of a ***framework for "emergent control" via self-organizing systems of network-distributed control agents***. Such a framework avoids the pitfalls of centralized control functions while retaining the ability to provide sophisticated control via intelligent agents that collaborate through carefully designed protocols for communication and self-organization.

Emergent control refers to a control strategy based on flexible collaborations among distributed control agents that implement local control. *Locality* is defined along multiple "control axes" that orthogonally decompose the control domain, at various levels of abstraction, and support the appropriate encapsulation and hierarchical layering of control functions. Changes in the configuration and structure of the controlled system are visible at an appropriate level of abstraction at each encapsulation boundary, and protocols are defined for the autonomous, local adaptation of control agent interfaces at each boundary.

The first thesis in our approach is that semantic descriptions of control functions may be formed and used as the basis for agent communication and self-organization. Semantically described control agent behavior is based on the concept of a *control generator*. A control generator is specified semantically as a function $Y = G(X)$, where Y is its information *products*, G is its *behavior*, and X is its information *influences*. The generator abstraction encapsulates the fundamental components from which our control system is built.

The simplest control generators are (1) those that directly ingest sensor readings and provide them as their information product, with behavior descriptions of the form $Y = G(\{\})$, and (2) those that accept actuator command lists and directly issue corresponding hardware commands to a device, with behavior descriptions of the form $\{\} = G(X)$. More complex generators encapsulate diverse approaches for control: a Kalman filter may infer the state of a component from sensor-produced influences, a connection-based finite-state automation may produce a system state estimate from component state estimates, a rule-based expert system may use a system state estimate as the basis for goal generation, and system goals may influence a Petri-net planner in high-level action for the control system.

The second thesis in our approach is that the control steps of an intelligent controller for a single device may be viewed as a Markov chain, with each step providing information to its successor.

It is easy to imagine how one would create a set of three agents that collaboratively control our target device by building one agent for each step in the Markov chain. However, if our device is adjacent (in terms of a spatial model) to other devices, then each step in our device's Markov chain is potentially influenced by the equivalent steps in Markov chains of those adjacent devices (Fig. 10). That is, their respective Markov chains are correlated. Hoffmann[10] named such structures "Markov blankets" and has suggested them as a method for building very large inference nets that are more computationally efficient than neural networks.

To control a system, we must consider the possible states of our control system observers and controllers, in addition to the possible states of the "world" that they measure and affect. To estimate the states of our observers and controllers, we must in turn consider the network over which we obtain information and issue commands. For example, when a failed actuator reduces system controllability, effective control should identify the actuator failure when generating a plan. In systems that contain redundant controllers, the overall world plan may require a reconfiguration of the controllers prior to an attempt to control the world. Network faults reduce observability and controllability, and

may also be addressed through reconfiguration.[11] Control of the observers, the controllers, and the control network extends our Markov blanket into the domain axis (Fig. 11).

The ontology for any given control domain will support the definition of many roles along the various control axes. For example, if we consider a spatial control axis for a connection-based spatial model, every component of the controlled system could potentially have an estimator (i.e., there is an estimator role for it). While the framework supports and can use any or all of these roles, most of them will remain unfilled in any real control network implementation. To the extent that roles are not filled, local observability and controllability for a given portion of the controlled system may be reduced. This increases the burden on the distributed control system and potentially decreases the maximum performance that can be attained, but it can be compensated through collaborative, intelligent control.

When the product of a generator, or portion thereof, satisfies the semantic definition of an influence of a second generator, then that product is said to be *useful* to the second generator. Both products and influences may be polyamorous, that is, a product may satisfy multiple influences, and multiple products may satisfy a single influence. Two control agents are *relevant* to one another if at least one generator in one of the agents supplies products that are useful to a generator in the other agent. To support fault tolerance, all generators must implement default behaviors when any combination of influences is in a "state of no knowledge" (not being satisfied by products). This implements a form of default reasoning, as in Ref. 12.

We wish to create control information flows over a network of information relevancy relationships by creating communications paths among agents according to relevancy. To accomplish this, the relevancy relationships among control agents are associated with communications links, or *channels*, connecting relevant agents. Channels are event driven and asynchronous, with events occurring on a channel when the value of a generator product supplying the channel changes. Scheduled control between temporally cognizant control agents may be implemented through a timer control generator that creates timing events on the product. However, in this case the underlying framework remains asynchronous, and timed completion is not guaranteed.

To minimize communications costs (bandwidth and latency), we prefer to implement channels as *direct*, *physical* communications links. This should result in a correlation between the ability to support high-performance control agent communications and the relevancy among the respective control agents. The approach is feasible if and only if the relevancy relationships among control agents correspond to the physical network connectivity among the processing nodes where agents are located. Such a correspondence is implicitly supported by agent semantics with respect to the three control axes. Relevancy relationships on the spatial model control axis will, in general, closely correspond to physical proximity in Euclidean or connection space (control agents with a null spatial model will have no spatial relevancy relationships). Proximity in space, in turn, closely correlates to the ability to support, and hence availability of, high-performance network communications.

Relevancy relationships on the other two control axes tend to show a strong correspondence to abstraction relationships in the control domain ontology. As information flows in along these control axes, we wish to take advantage of abstraction in order to reduce requirements for communication bandwidth and also reduce the tightness of coupling (and hence reaction time) required among control agents.

To accomplish this, we give control agents the ability to delegate their products and influences to control agents that are physically adjacent on the network. Delegates accept responsibility for those products or influences as well as a relevancy relationship with the delegator. Product delegates accept responsibility for supplying the influences of physically adjacent agents to which the product is useful, and influence delegates accept responsibility for seeking products useful to those influences from physically adjacent agents. Delegation is controlled via a broadcast mechanism in which agents announce their ability to supply products or their need to satisfy influences.

The delegation mechanism can be implemented through the concept of *control generator proxies*. A proxy implements a remote interface that encapsulates a communications link between the proxy and the originator of the proxy. This mechanism can implement inheritance and abstraction, in which the delegate inherits a product or influence from the delegator. The inheritance relationships implied by delegation are dynamic and

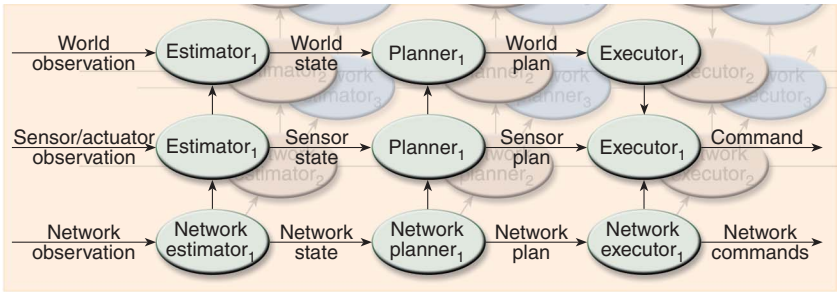


Figure 11. Collaborative controllers as a three-dimensional Markov blanket operating in the spatial, control, and domain axes.

support the ability for control agents to form flexible, overlapping teams. Proxies, in addition to encapsulating a communications link, also hide the implementations and location of the generator supporting the mobility of computation.

To support the above functionality with the minimal required overhead cost, we propose a networking strategy implemented as a simple three-layer network architecture that supports the implementation of channels as participants in a multiplexed direct physical communications link between physical network processing nodes. The bottom two layers are the standard physical and data link layers of the ISO Open System Interconnect (OSI) protocol. By adopting these layers we allow ourselves the use of existing low-level commercial communications devices. We replace the top five layers of the ISO OSI protocol with a single Control Network Protocol (CNP). The CNP is a context-sensitive protocol that is based on a small set of network operations, with context provided by the channel semantics, which is determined by the interface semantics of the control generators communicating over the channel. The CNP supports the formation of a relevance-based communications network in which there is no explicit concept of "routing." In place of routing, we create a network of direct information relevancy relationships (Fig. 7) that implements direct communications between control agent teams dynamically formed via discovery and delegation.

Establishing new channels is based on a service-discovery approach in which control generators are services. Such an approach allows a general, scalable, self-configuring, self-healing, and domain-independent way of building and maintaining large, emerging, and *ad hoc* virtual networks. Discovery is supported in the agent communication operations via the channel-forming operations described above. *Services* can be viewed as offering contracts to potential *clients*. In our framework, a service contract is simply the semantic description of a control generator interface implemented as control generator meta-data.

Control system components, with varying degrees of functionality, various hardware/software implementations, and various communications interfaces, integrate themselves with the control network via the control generator abstraction, whose implementation is a service interface that presents the published semantics of the generator and encapsulates the component implementation of those semantics.

Use of the service-based approach allows us to encapsulate generator implementations, but also any control agent collaborations that generators participate in via delegation. This supports flexible creation and reconfiguration of control system functional abstractions.

Information sharing among control agents will not always be sufficient to satisfy the control needs of all systems. Control agents that produce information that is co-useful (e.g., agent A produces information useful to agent B and vice versa) are called *co-dependent*. Agents whose generators are co-dependent may reach suboptimal conclusions when processing is performed prior to receipt of new information. This problem is commonly called a "race condition," and may be addressed by supporting shared, collaborative generation in addition to information sharing. Generators may collaborate in several ways (e.g., conditional products, shared constraint satisfaction, distributed truth maintenance). We address collaborative generation by treating generator collaboration as a special case of product-influence usefulness. This allows specially designed generators to establish communications channels for collaboration within the general agent framework.

The direct relevancy relationships in which control agents participate implicitly define a global degree-of-relevance measure over the control Markov blanket with respect to each control agent. This implicit measure will be dynamically and adaptively reflected in the delegation relationships (implemented by proxies) that obtain at any given time. These, in turn, define dynamic control abstractions, implemented as adaptive control agent teams, which are formed and re-formed through discovery. The organization of these teams is not defined via fixed, "sharp" interfaces, but is better conceptualized in terms of a fuzzy relevance measure that results from the degree of indirection and abstraction occurring along paths within the network of communications channels. This organization is not the result of a centralized design or a centralized execution-time control algorithm—it is emergent.

REFERENCES

[1]Brooks, R., "A Robust Layered Control System for a Mobile Robot," *IEEE J. Robotics and Automation* **2**(1), 14–23 (1986).

[2]Kuipers, B., *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*, The MIT Press, Cambridge, MA (1994).

[3]Williams, B. C., and Nayak, P. P., "A Model-Based Approach to Reactive Self-Configuring Systems," in *Proc. AAAI-96*, pp. 971–978 (1996).

[4]Kurien, J., *Model-Based Monitoring, Diagnosis and Control*, Ph.D. Thesis Proposal, Brown University Department of Computer Science (1999).

[5]de Kleer, J., and Williams, B. C., "Diagnosing Multiple Faults," *Artif. Intel.* **32**, 97–130 (1987).

[6]Scheidt, D., McCubbin, C., Pekala, M., Vick, S., and Alger, D., "Intelligent Control of Auxiliary Ship Systems," *Conf. on Innovative Applications of Artificial Intelligence*, pp. 913–918 (2002).

[7]Williams, B., and Nayak, P., "A Reactive Planner for a Model-Based Executive," in *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 1178–1185 (1997).

[8]Prim, R. C., "Shortest Connection Networks and Some Generalizations," *Bell Syst. Tech. J.* **36**, 1389–1401 (1957).

[9]Chakrabarti, S., and Mishra, A., "QoS Issues in ad hoc Wireless Networks," *IEEE Communications Magazine* **39**(2), 142–148 (Feb 2001).

[10]Hoffmann, R., *Inference in Markov Blanket Networks*, Technical Report FKI-235-00, Technical University of Munich (2000).

[11]Smith, S., White, K., Dunn, S., and Gupta, S., *Dependable Network Topologies with Network Fragment Healing for C.L.I.D.C.S. for Naval Shipboard Automation*, Florida Atlantic University Technical Report, Final Report for ONR Contract N00167-96-D-0053/0001 (1999).

[12]Etherington, D. W., and Crawford, J. M., "Towards Efficient Default Reasoning," *Proc. AAAI-96*, pp. 627–632 (1996).
[13]Scheidt, D., Vick, S., McCubbin, C., and Pekala, M., "Distributed Model-Based Control System for Engineering Plant Control," in *Proc. 15th Int. Conf. on Systems Engineering*, Las Vegas, NV (2002).

ACKNOWLEDGMENTS: OAK was developed by the author, Michael Pekala, Chris McCubbin, Shon Vick, Richard Avila, and David Alger. OAK research is described in Refs. 6 and 13. Excerpts from these paper have been incorporated into this article. The use of Markov blankets for emergent control was conceived by the author and Steven Langs. OAK work was performed under ONR Contract N00014-00-C-0050. The modeling of control networks was performed initially under APL IRAD X9AVX and subsequently under Department of the Interior Contract NBCH-C-02-0006. The work on network preconfiguration (flinching) is being performed under ONR Contract N00014-01-M-00.

THE AUTHOR

DAVID H. SCHEIDT received a B.S. in computer engineering from Case Western Reserve University in 1985. He joined APL in 1999 and is a member of the Senior Professional Staff in the National Security Technology Department. His current work concentrates on the research and development of distributed intelligent control systems. Prior to coming to APL he spent 14 years in industry performing and supervising the development of over 30 information and control systems. Mr. Scheidt has led various other projects involving a meta-database containing ≈1,000 heterogeneous databases, statewide immunology tracking, locomotive control, railroad dispatching, and distributed multilevel secure information systems. He has twice received the National Performance Review's Hammer Award. His e-mail address is david.scheidt@jhuapl.edu.