From: AAAI-02 Proceedings. Copyright © 2002, AAAI (www.aaai.org). All rights reserved.

Intelligent Control of Auxiliary Ship Systems

David Scheidt, Christopher McCubbin, Michael Pekala, Shon Vick and David Alger
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel Maryland 20723-6099

Abstract

The Open Autonomy Kernel (OAK) is an architecture for autonomous distributed control. OAK addresses control as a three-step process: diagnosis, planning and execution. OAK is specifically designed to support "hard" control problems in which the system is complex, sensor coverage is incomplete, and distribution of control is desired. A unique combination of model-based reasoning and autonomous agents are used. Model-based reasoning is used to perform diagnosis. Observations and execution are distributed using autonomous intelligent agents. Planning is performed with simple script or graph-spanning planners. A prototype OAK system designed to control the chilled water distribution system of a Navy surface ship has been developed and is described.

Introduction

Next generation ship engineering plant designs must incorporate a variety of increasingly sophisticated propulsion and auxiliary subsystems while reducing the overall requirement for human monitoring, maintenance and control. In order to meet these objectives, new levels of subsystem interoperability and autonomy must be achieved. The *Open Autonomy Kernel* (OAK) addresses critical infrastructure requirements for next generation autonomous and semi-autonomous systems, including fault detection and recovery, and goal-directed control. OAK brings together the Artificial Intelligent (AI) technologies of agent-based systems, and qualitative model-based reasoning to enable a new generation of integrated auxiliary subsystem autonomy.

Efforts to automate the control of engineering plants aboard naval vessels have emphasized the infrastructure and diagnostic aspects of plant management, i.e. monitoring vessel subsystems via sensors and presenting sensor data to human operators. Interpretation of and response to the data remain largely manual tasks. This interpretation and response function, especially in damage control scenarios, is a significant factor in determining ship-manning levels. If the incident assessment and response loop can be closed with a reliable autonomous reasoning process, significant relief in overall manning levels can be realized. Successful automation efforts to date have been based on expert diagnostic knowledge in the form of coded rules or procedures that are interpreted by the system at runtime to detect, predict, or diagnose fault conditions. The OAK architecture and working prototype extend this automated reasoning paradigm in a number of important ways. First, OAK uses goal-directed commanding at the system component level. This shifts the control paradigm from one of sending commands to subsystems, to that of sending goals and resources to intelligent subsystem management agents that require no direct operator interaction. Secondly, OAK's subsystem management agents are loosely coupled and distributed across a networked infrastructure. This results in a dynamic and adaptable coordination capability. Finally, OAK uses qualitative model-based reasoning as an extension to current rule- and procedure-based formalisms in the agent control loop. Model-based reasoning is used to perform real-time detection and identification of unanticipated fault conditions.

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Motivating Example

The motivating example used for the existing prototype is the control of auxiliary systems on capital ships. The specific target domain is the electrical, chilled water, low-pressure air and fire main systems on US Navy Arleigh Burke class Aegis destroyers. These systems are complex, inter-dependent, distributed, redundant, embedded, and contain numerous components that are unobservable. Currently high level control is the primary responsibility of dozens of sailors on each ship. Effective automation of these systems will allow the Navy to significantly reduce its manning requirements.

The auxiliary systems on the Arleigh Burke contain thousands of interdependent controllable nodes. One of the auxiliary systems, the chilled water distribution system, consists of a dozen complex machines, such as pumps and chiller plants, approximately four hundred valves, and twenty-three service loads. Behavior of components within an auxiliary system is dependent upon the components to which they are connected, often recursively. The auxiliary systems themselves are inter-dependent; for example, 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. Behavior of the auxiliary systems is also dependent upon other ship systems such as the HVAC, combat, and fire support systems. These dependencies combine to generate a complex super-system; no portion of which may be controlled in isolation. The size of the auxiliary systems, combined with their interdependency, prevents the effective use of traditional control system software. The auxiliary systems are designed with redundant supply and distribution mechanisms for critical resources. This redundancy presents the possibility of multiple valid configurations for a goal system-state pair, introducing control optimization in addition to control satisfaction.

Survivability, the justification for redundancy within auxiliary systems, demands that auxiliary control mechanisms avoid single points of failure. Therefore, the control mechanism must be fault tolerant.

The auxiliary Arleigh Burke control systems are only sparsely instrumented. The vast majority of system components do not have sensors that provide direct feedback on the behavior of the component. Comprehensive sensor coverage incurs additional cost, resource utilization and introduces additional points of failure. Without the availability of complete sensor coverage system states must be inferred from indirectly observable behavior.

Model-Based Reasoning

Model-Based Reasoning is an overloaded term. The Model-based reasoning used in OAK refers to a "reasoning from first principles" approach to diagnosis (Kuipers 1994). The theoretical basis for model-based reasoning is Discrete Event System theory, specifically Partially Observable Markov Decision Processes (POMDP).

Discrete Event System theory shows that systems may be modeled discretely using the automaton $G = (X, E, f, \Gamma, x_o, X_m)$ (Cassandras & Lafortune 1999) in which X is the discrete state space; E is the finite set of events associated with the transitions in Γ; f is the transition function; Γ is the active event function; x_o is the initial state; and X_m is the set of marked states. Control of Γ is provided by a control policy S that includes a set of control actions $S(s)$.

Automata that are memoryless (i.e. all past state information, and how long the process has been in the current state, is irrelevant), are considered *Markov Processes*. Association of control actions for state transitions, cost for such transitions, and transition probabilities, allows us to derive Markov Decision Processes (MDP). MDP are defined by the tuple (X_S, E_A, f_T, R) in which: X_S is the finite set of states of the system being tracked; A is the set of commands; E_A is a finite set of actions; and f_T is a state transition model of the environment which is a function mapping $X_S \times E_A$ into discrete probability distributions over X_S. The actions are non-deterministic, so we write $f_T(x, e)$ for the probability that transition e will occur given the state x. R is the cost of action. Our use of model-based reasoning is limited to diagnosis; therefore we are not concerned with R.

POMDP are MDP that have been extended to include a finite set of observations. POMDP are represented by the tuple M = (X_S, E_A, O, f_T, R) in which O is the observation function that maps the finite set of observations into X_S. The probability of making an observation o from state x is denoted as $O(o, x)$.

The tuple M is capable of representing the behavior of systems that are composed of independent subsystems. This may be generated by creating a supermodel M_S whose states, events and function are the cross product of the subsystem models; $M_S = M_1 \times M_2 \times \ldots \times M_N$. This approach to modeling complex systems is impractical for two reasons: first, the state space quickly becomes unwieldy; second, the majority of complex systems of interest are composed of dependent subsystems. Model-based Control solves this dilemma by modeling the system as concurrent constraint automata, separately enumerated component models that are constrained by shared attributes (Williams & Nayak 1996).

This representation scheme is beneficial when constructing large complex systems. By encapsulating component behavior within a single logical model and by constraining components through context independent attributes the components themselves become context independent models. This provides for model replication and reuse.

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 indicate the existence of one of 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), loosely based on De Kleer and Williams' General Diagnostic Engine (de Kleer & Williams 1987) and described in detail by Kurien (Kurien 2001). (3) Selection of the most probable scenario from the identified candidates. The fitness criteria used by CBFS to select the most probable solution is a system-wide candidate probability based upon the probabilities of individual component states $f_T(X_i, E_i) \forall M$; by using a general diagnostic algorithm, implementation effort is limited to the model upon which the algorithm operates. System design and maintenance do not require software modifications. In addition, because of the encapsulation of the component models, model maintenance is limited to those components modified in the controlled system and their immediate neighbors.

Autonomous Agents

Within the context of OAK, an agent is a software process that can reason about and act upon its environment. Four sets of characteristics that may be used to describe agents and agent systems are: Intrinsic Agent Characteristics; Extrinsic Agent Characteristics; System Characteristics; and Agent and Environment-Agent characteristics. OAK agents are intrinsically permanent, 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. Agents are deliberative in their ability to create a plan in response to observed states and defined goals. OAK agent's extrinsic characteristics include proximity to the controlled system, social independence, and both awareness of and cooperativeness with goals and states of other agents. Systems of OAK agents consist of nearly homogeneous agents, and are independently executed yet contain unique models of the system for which the agent is responsible. OAK agents are environmentally aware and behavior of the environment is predictable through each agent's model.

Application Description

OAK is a distributed, multiagent system. The system can have varying topology based on the application. OAK has two major use-cases that almost fully describe the operation of the system: OAK's reaction to user-input goals; and OAK's reaction to system state change. The primary intelligent components that enable OAK to accomplish these use-cases are the model-based reasoning engine and the planner. In the sections that follow, the OAK application is described in detail.

The Multiagent System

To perform the diagnostic phase of the control cycle, OAK agents continually update their states using the model-based reasoning engine, and pass these state updates to other agents that are interested so that these agents may update their states. In response to these states, or to the system's environment, an external actor or an OAK agent will provide goals to the OAK system, which are distributed for further processing. A hierarchical agent topology was used for testing. However, the OAK architecture does not preclude other topologies.

Agent Communication Framework and Language Each agent has an associated Agent Communication Broker (ACB), which is responsible for handling all of the agent's communication with the Agent Communication Framework (ACF). The ACB maintains a queue of messages coming into the agent. Additionally, each agent that has direct communication with hardware has a control mediator (CM) to handle the hardware level goals that are generated by these agents for the hardware associated with it, and to receive updates about this hardware. These messages are not handled by the ACF.

The ACF of OAK is built using the Control of Agent Based Systems (CoABS) Grid, a "flexible information infrastructure" built by Global InfoTek, Incorporated. CoABS allows OAK to have a dynamic, heterogeneous agent membership, facilitates agent replication to remove single points of failure, and supplies utilities for ACF visualization.

The Agent Communication Language (ACL) of OAK provides several message templates, including messages for queries, state updates, subscription requests, goals, exceptions, and agent coordination. The ACF allows any agent to communicate directly with any other agent. Thus, communication between agents is not restricted to any particular logical framework.

User-determined Goals One of the major use-cases of OAK is to react to goals entered by an external actor. These are system-level goals which have the potential of transitioning the entire multiagent system from one state to another.

Goals that are entered from an external actor, such as a human operator, through this interface are sent directly to the root level agent using a *goal message*. This 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 root node develops a plan and directs a goal to one of its child agents, the goal is received by the child agent's ACB, sorted into its queue, and eventually accepted by the agent for processing. This agent develops a plan to implement the goal. Since this agent is a root of its own tree, the goals developed by the planner are passed to its child agents. This propagation continues until leaf agents receive goals for their specific domains.

Once goals are received at the leaf level, a similar process occurs, in that a plan is developed and goals are passed out of the agent. The only difference is that the goals are now passed to the agent's CM, which translates the goal into commands that a hardware driver can understand. Since the CM is the only component that has direct interaction with the hardware drivers, it is the only component that has to be updated when hardware itself is changed or when hardware drivers are updated.

Successful goal implementation implies a state change, so an agent does not have to set up callbacks with the hardware to confirm that a command was successful. Leaf agents are already required to monitor the hardware they control for changes in order to accomplish the second major use-case of OAK. Therefore, the leaf agents wait for reactions from hardware monitors 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. Since state changes are propagated up the hierarchy, agents at higher levels are also informed that their goals were implemented and they can then pass out goals that had to be put in a wait state. To an implementer of OAK, this means that the incoming goal use case and the state change use case, which comprise the two major functions of OAK, are decoupled.

Reacting to State Changes To appropriately handle changes in the state of the system, OAK uses a model-based reasoning engine (MBRE)(Kurien & Nayak 2000). In this section, we follow the sequence of events that are implied by a state change in OAK.

State change events are transmitted through the use of a *fact message*. This message 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.

One of OAK's strengths is an agent's ability to determine the state of its model, compare that state to a knowledge base, and reactively plan. Thus, an agent can autonomously control its domain until an agent that is higher in the hierarchy (or in the case of the root agent, the external actor's agent) preempts its control. The component of OAK that controls reactive planning is called the *reactive manager*.

There are two types of information in the reactive manager: persistent goals and emergency conditions.

Persistent goals are simply goals that are desired true for the duration of the agent.

We define an emergency condition as a state that cannot be reversed and requires OAK to act immediately to protect the resident system. When OAK detects an emergency, it will preempt the external actor's goal and go to a predetermined goal that will minimize damage to the system being modeled. From this point on, goals from the user are implemented as completely as possible based on the damage to the system.

Planning

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 may be attempted. OAK executes a plan once it has been developed by transmitting each subgoal at the appropriate time to the appropriate agent or piece of hardware and 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 run-time differently for each agent from a group of developed planners. So far, two planners have been developed. The deployed planners are the *Scripted Planner* and the highly specialized *Graph-Based Planner*.

The scripted planner is extremely simple but useful for simple agents, such as leaf agents. 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, with 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 (Prim 1957) algorithm 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 would generate a plan based on the actions determined.

Implementation

OAK has been implemented on the Chilled Water Reduced Scale Advanced Demonstrator (RSAD) at the Naval Surface Warfare Center (NSWC) Philadelphia. The RSAD, seen in Figure 1, is a reduced scale model of the Arleigh Burke Chilled Water system. The RSAD is a physical implementation of two Arleigh Burke chilled water zones using reduced scale equipment. The RSAD contains four pumps, two chiller plants, two expansion tanks and approximately one hundred 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 sixteen simulated loads.

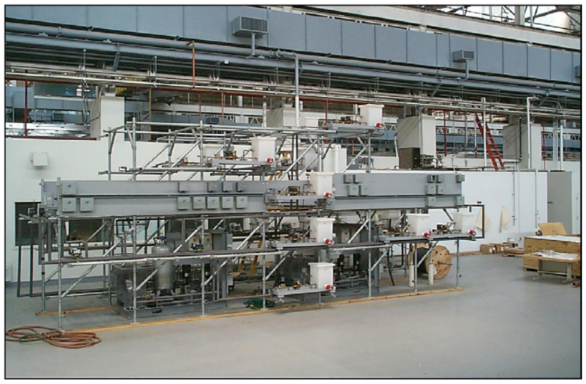
Figure 1: Reduced Scale Advanced Demonstrator (RSAD)

The RSAD control prototype uses twenty OAK agents to control the RSAD's pumps, plants and valves. Each agent contains its own diagnostic engine, planning engine(s), execution managers, and the ability to receive and propagate goals and facts. The agents are organized hierarchically with agents controlling individual hardware components, small aggregations of components, system level abstractions, and the ship.

The *L2* inference engine was used within the OAK agents as the diagnostic engine. L2 is NASA AMES Research Center's second generation model-based reasoning engine. L2, and its predecessor Livingstone, are based on Williams' model-based reasoning approach.(Williams & Nayak 1996) Livingstone was demonstrated as a diagnostic tool for fault analysis of satellites in a 1998 experiment on NASA's Deep Space One(Muscettola *et al.* 1998). Models processed by L2 are written in the Java-Based Model Programming Language (JMPL). L2 provides multiple candidate identification and resolution strategies. The strategy used for the RSAD prototype is CBFS(Kurien 2001). The L2 CBFS implementation uses constraint satisfaction to identify candidate system states. The candidate states are then maintained as hypothetical belief states over time. Each successive observation is used to update the stored belief states and to generate new hypothetical belief states. L2 provides a likelihood rank associated with each belief state. OAK reconfigures the RSAD based upon the current most likely belief state while maintaining other trajectories. The reasoning system is nonmonotonic, as active belief states will be abandoned if future observations provide support to other previously less likely possible belief states.

Each OAK agent contains a JMPL model of the real-world system or subsystem for which it is responsible. The model includes POMDP representations of components within the agent's system or subsystem.

The ship agent's planner accepts high-level goals from the ship's Command Center. The ship agent also accepts inferred facts from the intermediate agents that express the believed state of the ship systems. The ship agent planner generates goals for the intermediate agents.

The Chilled Water agent is the only intermediate level agent currently implemented in the RSAD prototype. The model used for diagnosis in the chilled water agent consists of twenty-one components. Each component represents a small cluster of machinery within the RSAD. The Chilled Water Agent directly controls eleven of these components.

Beneath the Chilled water agent in the hierarchy are eighteen low-level agents. These agents manage the two cooling units with their supporting valves and regional valve-pipe aggregate components within the chilled water system.

Results

Three test sets were performed for the RSAD implementation of OAK. The first round of testing was performed for and by the development team. The second round of testing was performed with control systems engineers from the Naval Surface Warfare Center Carderock Division Advanced Auxiliary Controls and Automation Group (NSWC-CD Code 825) who are familiar with the Arleigh Burke auxiliary systems. The third round of testing was performed for representatives of the ship construction industry. All three sets of testing followed the same format. All tests were performed with the RSAD hardware. 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 provided 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 test scenario the RSAD was given consecutive commands from a Command and Control simulator to move from one "ship state" to another. The four ship states each have a unique combination of desired states and fitness criteria.

The second test scenario consisted of inducing a series of sequential component failures that initially force OAK to reconfigure the system in order to satisfy the high-level goals and eventually degrade the RSAD so that the RSAD's stated goals are no longer achievable. The second scenario was also specifically designed to test the non-monotonic capability of the reasoning engine. An improbable component failure that was observationally indistinguishable from a probable failure was generated resulting in a misdiagnosis. Subsequent failures generated observations that re-enforced 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 between agents. During the first test set, the test team found in OAK the propensity to select a legitimate yet unlikely candidate from a set of possible candidate diagnosis. This problem was addressed by modifying the agent topology for the RSAD implementation. After modifications were made, all three test sets were successfully completed. In total, fourteen separate multi-stage tests were conducted. During these fourteen tests, OAK performed thirty-four diagnose-plan-execute cycles. OAK 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.

In some cases, 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 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 consisted of a chiller plant unexpectedly failing to the OFF state. Another failure consisted of a valve unexpectedly failing to STUCK_SHUT. During both of these unexpected failures, 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. In all cases where a complete solution was available, a solution was found. In cases where 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.

Difficulties Distributing Diagnosis in Strongly Coupled Systems

Each OAK agent maintains the ability to obtain observations and perform diagnostics on components that the agent controls. Individual agents use observations to infer component states within the agent's sphere of influence. Information on component states are propagated throughout the agent community by disseminating facts that represent the belief state of the agent-modeled subsystem. The model and diagnostic engines within individual agents are independent and, by design, capable of performing diagnosis in isolation from other agents. When an observation is made that conflicts with the model's current belief state (indicating a fault within the system), the reasoning engine attempts to resolve the conflict internally. When the observation and faulty component(s) are both contained within the same agent/model, OAK was found to correctly diagnose component failures. However, when a conflicting observation is observed by one agent and the failed component is within the sphere of influence of another agent, and a candidate failure within the observing agent's model exists, the observing agent will attempt to resolve the conflict prior to disseminating observations. Since a candidate solution does exist, the observing agent will resolve that its internal solution is correct and disseminate the belief that its internal component is faulty. This can result in OAK selecting a less likely candidate state for the system over a more likely candidate.

The potential to select unlikely candidate failure states was mitigated by removing the diagnostic portion of the control loop from those agents that did not have access to observables either directly or through subordinate agents. Diagnostic responsibility for components that did not have access to observables was the undertaken by the lowest parent agent in the hierarchy that did have access to the necessary observations. In the RSAD implementation, this involved moving the diagnostic responsibility for the chiller plant agents into the Chilled Water agent. All agents retained the ability to make observations, plans and execute commands. Thus modified, OAK was able to successful complete all three sets of tests.

Future Work

While the principle of OAK has been demonstrated in a real world setting, additional non-AI related steps are necessary for OAK's acceptance into ship construction. A partial list of future non-AI activities include the creation of a mature set of modeling tools, development tools, and testing upon actual ships.

Future AI related activities involve the extension of the modeling capability (and corresponding model-based reasoning) to include more diverse and sophisticated representations. The most straightforward modeling extension involves the detailed modeling of the other auxiliary ship systems; namely the electrical and low pressure air distribution systems. Slightly more sophisticated is the ability to incorporate a model of the control system itself into the ship systems model. By modeling the control network and the computing infrastructure, the control system becomes capable of reasoning with an inability to command otherwise functional components. Particularly interesting is the possibility that operating agents could jointly control a system through observing each other's behavior should their communications become severed.

In addition to extending the modeling capabilities laterally, more comprehensive control may be found by extending the modeling vertically as well. In the RSAD implementation, the ship model above the chilled water system was limited to a simple finite-state model of the basic ship operating modes. Complex models of ship operating practices and policies have been developed. Extending OAK's modeling capability upward to incorporate both fluid control and operational models within a common reasoning system would further improve OAK's effectiveness. OAK may be improved by extending modeling downward as well. JMPL lacks the sophistication to represent detailed fluid systems normally modeled by ordinary differential equations. Hybrid modeling involving discrete and continuous models is a large vibrant community.

Finally, distributed non-monotonic reasoning should be investigated. Recall that while facts and goals are shared between agents, reasoning is performed in isolation. The inability to send out tentative facts limits diagnosis to encapsulated systems that have co-located components and observables.

Conclusions

OAK has successfully demonstrated the ability to autonomously control complex, sparsely observable systems with social distributed agents that use model-based reasoning for system diagnosis. Specifically OAK has been demonstrated on a reduced scale version of a US Navy destroyer's chilled water distribution system. The design of OAK is not ship system specific, and OAK has the potential to improve the autonomous control of real-world systems that exhibit similar traits. Systems OAK should be effective in controlling include electrical utilities, water utilities and communications systems.

Acknowledgments

We conducted this work as part of the Office of Naval Research "Machinery Systems and Automation to Reduce Manning" program under contract N00014-00-C-0050. The RSAD and low-level control software was developed and provided by the Naval Surface Warfare Center Carderock Division Advanced Auxiliary Controls and Automation Group. Don Dalessandro, Brian Callahan, Richard Avila and William Basil provided substantial insight into the Arleigh Burke auxiliary control system design and Naval doctrine. John Bracy contributed to the editing of this paper.

References

Cassandras, C., and Lafortune, S. 1999. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.

de Kleer, J., and Williams, B. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32:97–130.

Kuipers, B. 1994. *Qualitative Reasoning*. The MIT Press, Cambridge MA.

Kurien, J., and Nayak, P. 2000. Back to the future for consistency-based trajectory tracking. In *Proceedings of AAAI-2000*.

Kurien, J. 2001. *Model-Based Monitoring, Diagnosis and Control*. Ph.D. Dissertation, Brown University Dept. of Computer Science.

Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote agent: to boldly go where no AI system has gone before. *Artificial Intelligence* 103:5–47.

Prim, R. 1957. Shortest connection networks and some generalizations. *Bell System Technical Journal* 36:1389–1401.

Williams, B., and Nayak, P. 1996. A model-based approach to reactive self-configuring systems. In *Proceedings of AAAI-1996*.