AGENT-BASED, DISTRIBUTED DIAGNOSIS FOR SHIPBOARD SYSTEMS

Gregory Provan, Yi-Liang Chen
Rockwell Scientific Company,1049 Camino dos Rios, Thousand Oaks, CA 91320, USA
{gprovan,ylchen}@rwsc.com

The ability to reconfigure a ship's engineering plant in response to changing mission or equipment conditions can dramatically increase a ship's capability and survivability. We describe the agent-based, distributed diagnostic aspects of a distributed control architecture that integrates multiple ship systems and provides resource- and diagnostic-driven reconfiguration at multiple system levels, such as mission-level, process-level and component-level. We embed system- and sub-system models in distributed, agent-based components to provide distributed diagnostics. We demonstrate this architecture using a shipboard chilled water system application.

1. INTRODUCTION

Improving the fight-through capability of future combat ships necessitates an integrated ship control system that can monitor all shipboard engineering assets and adaptively reconfigure these assets during battle conditions. For example, if a chilled water supply unit is damaged during combat, a ship control system may temporarily overload another chiller to provide the required cooling capacity or shed non-combat heat loads to reduce cooling requirements. If the chilled water system still cannot satisfy combat cooling demand, the ship control system may allow certain combat systems to moderately overheat or selectively reduce the performance of combat systems (e.g., reduce gun firing rate) to avoid complete system shutdown. To achieve such intelligent ship responses, all the various shipboard subsystems (e.g., chilled water system, fire-main, propulsion, generator, combat system, etc.) must be tied together under an integrated control hardware and software architecture. To maximize fault/damage tolerance, the integrated ship control system must also be a physically distributed architecture with intelligent autonomous behavior embedded in low-level system components.

We use an agent-based approach to guarantee global system properties, such as mission achievement, while distributing inference for fault-tolerance and sub-system autonomy. To this end, we have developed a hierarchical, distributed software architecture based on agent technology. Agents are self-contained software entities capable of communicating and making individual decisions (Wooldridge and Jennings, 1995). They work autonomously, handle goals, maintain beliefs, and cooperate through negotiation to create solutions.

An agent's decision logic is usually implemented using rule-based reasoning similar to an expert system. Rule-based reasoning has two main drawbacks: (1) difficulty of ensuring the completeness of the rule set for handling all situations, and (2) difficulty of maintaining the rule set when the controlled system expands or changes. To overcome these drawbacks, we use a model-based framework for the agents' decision logic. A model not only captures an expert's knowledge, but is a rigorous description of how a system works based on first principles. Given a model, model-based diagnostics algorithms can guarantee soundness and completeness of all diagnoses generated (Darwiche, 1998).

We embed within each agent system and sub-system models to provide diagnostics and assist in system reconfiguration when problems are detected. In the following sections, we describe the autonomous cooperative agent and model-based reasoning methodology, and the integration of these two methodologies within a distributed, reconfigurable control architecture. We use a chilled water system as a demonstration test-bed of this agent-based architecture.

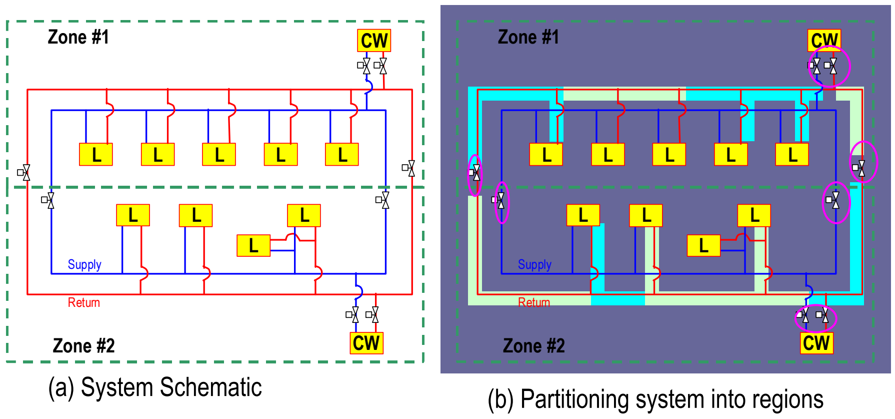

Figure 1: *Figure (a) shows a simplified diagram of chilled water system test bed. L = Load, CW = Chilled Water Production Unit. Individual valves for each load are not shown. Figure (b) shows how we partition regions of the system into functional units. The partitioning of segments of the return piping system are shown .*

2. CHILLED-WATER SYSTEM APPLICATION DOMAIN

In this article we apply our agent-based diagnostics architecture to a chilled water system testbed. This system provides multiple types of services across two physical zones, and contains redundant pumps and valves to allow for control reconfiguration in case of equipment failure. Figure 1(a) shows a highly simplified diagram of the chilled water system. The heat loads (marked by "L") are divided into three types: non-vital, vital, and combat system heat exchanger. The non-vital loads have a single path for supply water and a single path for return water; the vital loads have a single supply path and dual (redundant) return paths, the combat system heat exchangers (critical loads) have either a single or dual supply paths and dual return paths. The loads are also located in two different physical zones. The key objectives of control reconfiguration are to segregate the zones, shut down low-priority services in combat conditions, and to manage the loads upon detection of a chiller machine failure.

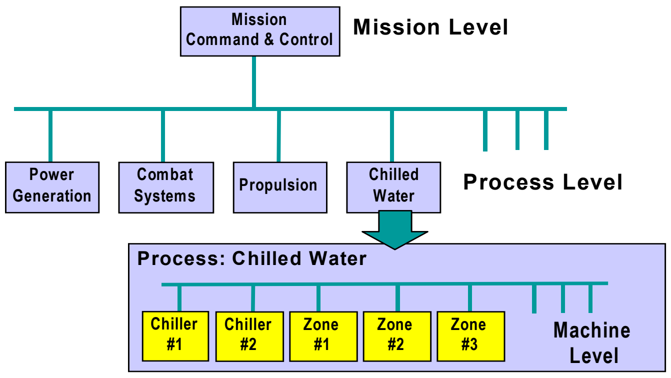

Figure 2: *Control system hierarchy is composed of three levels.*

2. DISTRIBUTED AGENT FRAMEWORK

We break our architecture into three levels, based on the functional requirements and control techniques which must be employed at each level. These levels are:

1. Mission Level – This is the highest level in the control architecture; the mission-level controller sets priorities and performance objectives for all ship engineering systems based on the overall mission goals. For example, in a damage control situation, resources would be shifted from less critical services, such as drinking water production, to fire-fighting systems. Similarly, control and use of the ship's resources for the various system functions can be optimized for dockside, normal steaming, battle stations, and damage control conditions. Inference at this level typically consists of tradeoff analysis of system requirements and optimization of mission-level parameters, such as overall fuel usage.
2. Process Level – Each process-level controller provides a general ship service (e.g., chilled water service, propulsion service, etc.), and will reconfigure its system operation based on the performance objectives set by the higher level (i.e., mission-level) controller. For example, the chilled water service may temporarily overload a water chiller unit when it senses another unit is down, in order to maintain the cooling rate objective set by the mission-level controller. This will result in optimized machine and energy use, and automatic reconfiguration of resources to maintain service availability in the event of faults or damage. Inference at this level typically consists of optimization of process-level parameters, together with discrete-event control.
3. Machine Level – Controllers at this level monitor and control individual machine units (e.g., a chiller) to maintain machine availability and achieve the machine setpoints requested by the process-level controller. For example, if excessive motor vibration at a pump is detected, the machine-level controller may alter the motor speed to avoid exciting a critical frequency and thus avoid damage. Inference at this

level typically consists of optimization of process-level parameters, together with discrete-event and/or continuous control.

In the agent-based control system, each agent (called an Autonomous Cooperative Unit, or ACU) represents a physical process or equipment and coordinates its operation with other agents. Control actions are the result of coordinated decisions among all ACUs.

Figure 3 shows the control software architecture, composed of a collection of mission-level ACU, process-level ACUs, and machine-level ACUs. Each ACU at each level receives performance goals from a higher level ACU, and uses its internal model (whether it be a mission-level, process-level, or machine-level model) to diagnose problems and achieve the desired performance goal.

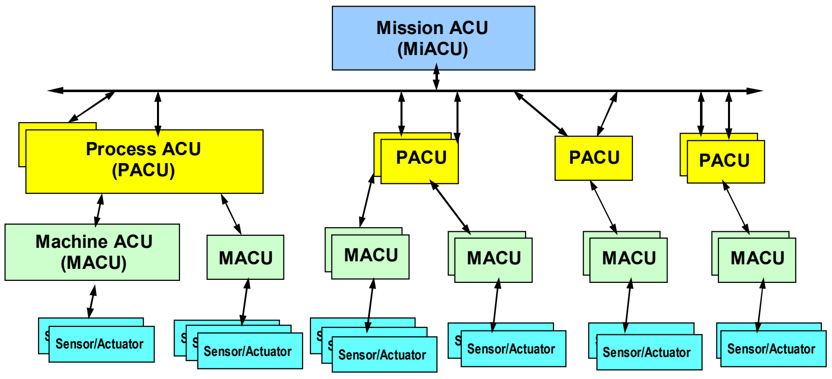

Figure 3: *Hierarchical software architecture showing three ACU levels, corresponding to Mission, Process and Machine. The machine level ACU interacts with sensors and actuators.*

When a requested performance goal cannot be achieved, ACUs initiate distributed problem solving actions to find alternative solutions that best meet the system's goals. The technique is comprised of 2 steps. The first step consists of discovering the physical relationships among Machine ACUs. The second step involves developing feasible plans that satisfy both local and shared constraints.

Machine ACUs contain models that provide self-awareness and self-assessment capabilities. The interaction of Machine ACUs is founded on the creation and dissolution of *virtual control clusters*. Within these clusters, Machine ACUs negotiate to obtain the near-optimal solutions, while considering both local and system conditions. The sequence of communication is as follows: Machine ACUs provide Process ACU with a number of solutions to be prioritized according to goals and constraints. Process ACUs use local (process) goals to select the best settings. In the same way, when a process-level goal cannot be achieved, Process ACUs also negotiate and provide the Mission ACU with a number of possible solutions.

Cooperative software agents permit the creation of highly flexible control systems based on both autonomy and cooperation. The ACU framework is well suited for solving resource allocation problems on distributed hardware. The resultant software is easily scalable to large systems. When a new machine is added to a system, the overall software is updated simply by adding a corresponding agent to represent the new machine. Autonomous capability is directly designed into an agent so that it can continue to operate in the event of failures in the other agents.

To ensure generality of our approach, we have developed a Job Description Language, called JDL (Tichý et al., 2002), for inter-agent communication. This language enables agents to communicate for all activities using a single, unified framework.

3. DISTRIBUTED MODEL-BASED DIAGNOSTICS

This section summarizes our distributed framework for model-based diagnostics that can generate a distributed system model, and then integrate the diagnostics computed by each distributed component into a globally sound and complete system-level diagnosis (Provan, 2002). We first show how we create a distributed model, and then how we use agents to integrate the distributed diagnoses into a system-level diagnosis.

3.1 Distributed Model-Based Diagnosis

A model-based diagnosis approach works as follows. The algorithm uses the system model to simulate sensor values, and compare these with the actual sensor readings. If there is a discrepancy, the diagnostic engine is invoked to determine the most likely cause of this discrepancy.

We have extended the same causal-network diagnostics approach such that it can be used for both diagnostics and for assisting control reconfiguration (Provan and Chen, 1999). For diagnostics, the actual sensor values are propagated through the causal-network model to determine the faulty component and the operational mode(s) that would produce the abnormal sensor values. Conversely, for system reconfiguration, the desired machine or process output values are propagated through the causal network to determine the equipment settings necessary to achieve the desired output (subject to the constraint that certain components are unavailable or are in failure mode). When multiple diagnostic solutions or reconfiguration solutions are found in the model, the best solution can be determined by evaluating probabilities or costs associated with the various component modes or settings.

Our distributed diagnosis technique transforms a centralized model into a distributed model, and then synthesizes the minimal diagnoses computed by the distributed components into a global minimal diagnosis using formally sound principles. We use the topology, both functional and physical, of the system to partition the model into distributed components. For example, for the shipboard chilled-water system shown in Figure 1, we can create components for the major subsystems (e.g., chillers and loads) and piping segments (according to the physical connectivity of pipes).

For each distributed component, we also record the entities that are inputs and outputs. For our chilled water application, for each distributed component we have water (with properties like temperature, pressure and flow) as the inputs and outputs. In addition, each component may have control entities, which include requirements placed *on* the component, e.g., a chiller being required to provide *x* units of chilled water, or requirements placed *by* a component, e.g., a load requesting *y* units of chilled water. For this chilled-water application, we decompose the system into a set of functional units, as shown in Figure 1(b). A functional unit is defined to be a load, valve, chiller, and pipe segment; we then assign an agent to each unit.

Unlike previous approaches, which compute diagnoses using the system observations and a centralized system description (Darwiche, 1998; de Kleer and Williams, 1987), we use a distributed approach. We build a causal network model for the high-level component interactions. A causal-network model consists of interconnected component models in which each component has various operational modes; for example, a valve component may have the following operational modes: normal, stuck open, and stuck closed. An input/output behavior is defined for each mode of the component. For example, if a valve is in normal mode, then the output pressure is equal to the input supply pressure when the valve is open.

We assume that each component can compute a local minimal diagnosis based only on sensors internal to that component and knowledge only of the component system description; note that we place no restriction on the technique used to compute that local diagnosis, e.g., neural network, Bayesian network, etc. Given a set of least-cost or most-likely local diagnoses, we can compute a globally sound, complete and minimal diagnosis for the complete system (Provan, 2002). The algorithm uses a graph-based message-passing algorithm that passes diagnoses as messages and synthesizes local diagnoses into a globally minimal diagnosis. We employ agents to handle the message-passing and diagnostic synthesis. By compiling diagnoses for collections of components (as determined by the system topology), we can significantly improve the performance of distributed embedded systems.

3.2 Agent-Based Diagnostic Synthesis

This section outlines how the agents facilitate the diagnostic synthesis process. Figure 4 shows the internal functions within an ACU. The ACU entity consists of three main components: a data table, middleware, and a set of application components. The data table contains input/output data for sensing/controlling the devices attached to the control hardware as well as data for the embedded applications. The middleware supports the agent services, enabling inter-agent communication across networked hardware. The application components are:

1. *equipment simulator* (Equip. Sim), which simulates the behavior of the associated machine or process;
2. *diagnostics engine* (Diag), which computes the fault status of the associated machine or process;

3. *coordinator* (Coord), which computes optimal control reconfiguration plans whenever a fault condition arises or system goals change;
4. *execution control* module (Exe. Ctrl), which controls the attached devices and can execute the reconfiguration control plans.

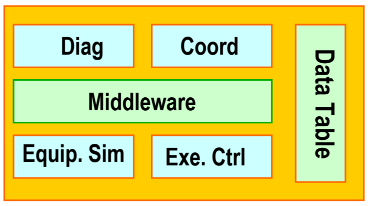

Figure 4: *Internal functions within an ACU*

The equipment simulator and diagnostic engine are both implemented using a causal-network model. The equipment simulator takes equipment settings (or commands) as input, and simulates the machine or process behavior with all components assumed healthy. The equipment simulator thus computes the values of the expected sensor readings by propagating the equipment setting values through the model. These simulated values are stored in the data table, and compared with the actual sensor values retrieved by the agents (and also put into the data table). If there is a discrepancy between the simulated and actual sensor values, the agent invokes the diagnostic engine to update the local machine's health state. Given any local fault, the agents communicate appropriate health status data, and then invoke the diagnosis synthesis algorithm to generate a global system health status. Finally, if a fault prevents the mission from being achieved, the agents invoke the control reconfiguration algorithm to reconfigure the system controls, and then the Execution Control module (via agent messaging) modifies the equipment control settings.

4. DISTRIBUTED DIAGNOSIS OF A CHILLED-WATER SYSTEM

ACU Type	Number	Description
Non-Vital Load	2	Provides low priority cooling that is shut off during combat mode.
Vital Load	9	Provides high priority cooling that needs to be maintained under any operation mode.
Combat Sys. Heat Exchanger	4	Provides high priority cooling during combat mode.
Chilled Water Producer	2	Provides cooling capacity.
Water Transport Pipe	1	Provides water transport path under open and segregated zone conditions.
Chilled Water Service	1	Oversees the chilled water process.
Zone	2	Oversees the areas within a zone.

Table 1: *Agent types for chilled-water testbed*

We have developed parameterized component models for pump, motor, pipe, valve, chiller, and heat load. We also identified 7 types of ACU associated with the chilled water test-bed; these ACU types are listed in Table 1 along with the number of ACUs required for controlling the simplified chilled water system.

For this testbed, we can use this agent-based framework to demonstrate the diagnostics and control reconfiguration capabilities for several scenarios, such as:

1. *Mode changes*: moving from one system mode to another, such as from cruise to battle mode, requires readjustment of chilled water supply due to differences in chilled-water demands, and to differences in system segregation between the different modes.
2. *Faults*: we can simulate the incidence of a fault in the system, such as the failure of a chiller unit, or a pipe fault (blockage or leakage). In such fault scenarios, the fault is first identified by a distributed agent, which then must consult with other agents to compute a global diagnosis.
3. *Control Reconfiguration*: Given a global diagnosis, the agents then must reconfigure the control settings in order to see if the goals (both local goals and chilled-water system goals) can still me met. The coordinator module of each distributed entity then executes the control reconfiguration.

5. SUMMARY

We have described an architecture for using agents to compute diagnoses and reconfigure control for a distributed system. Our approach uses a novel agent language, called a Job Description Language, for the agent interactions, and a novel model-based diagnostics technique for distributing diagnostics models and computing system-level diagnoses. We have applied this approach to a chilled-water test-bed, and have shown how our approach can handle a variety of mode changes and faults within the test-bed.

6. REFERENCES

(Darwiche, 1998) A. Darwiche Model-based diagnosis using structured system descriptions. *J. of AI Research*, 8:165- 222, June 1998.

(de Kleer and Williams, 1987) J. de Kleer and B. Williams. Diagnosis with Behavioral Modes. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1324—1330, August 1989. Morgan-Kaufmann Publishers.

(Provan and Chen, 1999) Provan, G. and Y.-L. Chen, "Model-Based Diagnosis and Control Reconfiguration for Discrete Event Systems: An Integrated Approach," Proc. 38th IEEE Conf. on Decision and Control, pp. 1762-1768, December 1999.

(Provan, 2002) G. Provan. A Model-based Framework for Distributed Embedded Diagnostics. In *Proc. Conf. on Principles of Knowledge Representation*, Toulouse, France April 2002.

(Tichý et al., 2002) Tichý P., Šlechta P., Maturana F., and Balasubramanian S.: *Industrial MAS for Planning and Control*. In: Multi-Agent Systems and Applications II, LNAI 2322, Springer Verlag, Heidelberg, 2002, pp. 280-295.

(Wooldridge and Jennings, 1995) Wooldridge, M. and N.R., Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, 10(2), 115-152, 1995.